33nd International Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2013, December 12–14, 2013, Guwahati, India

Edited by

Anil Seth

Nisheeth K. Vishnoi

Editors

Anil Seth
Dept. of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur 208016
India
seth@cse.iitk.ac.in

Nisheeth K. Vishnoi
Microsoft Research
No. 9 Lavelle Road
Bangalore 560001
India
nisheeth.vishnoi@gmail.com

ACM Classification 1998
D.2.4 Software/Program Verification, F.1.1 Models of Computation, F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic, F.4.3 Formal Languages

ISBN 978-3-939897-64-4

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-64-4.

Publication date
December, 2013

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2013.i

ISBN 978-3-939897-64-4 **ISSN 1868-8969** **www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

www.dagstuhl.de/lipics

Contents

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Session 4B

Session 5A

Session 5B

Session 6A

Session 6B

Session 7A

Session 7B

Preface

The 33rd international conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013) was held at the Indian Institute of Technology, Guwahati, from December 12 to December 14, 2013.

The program consisted of 5 invited talks and 40 contributed papers. This proceedings volume contains the contributed papers and abstracts of invited talks presented at the conference. The proceedings of FSTTCS 2013 is published as a volume in the LIPIcs series under a Creative Commons license, with free online access to all, and with authors retaining rights over their contributions.

The 40 contributed papers were selected from a total of 111 submissions. We wish to thank the program committee for its efforts in carefully evaluating and making these selections. Names of PC members and external reviewers are given in the next few pages. We also wish to thank all those who submitted their papers to FSTTCS 2013.

We are particularly grateful to the invited speakers: Venkatesan Guruswami (CMU), Martin Hofmann (LMU Munich), Subhash Khot (NYU), Stephan Kreutzer (TU Berlin) and Kazushige Terui (Kyoto, Japan), who readily accepted our invitation to speak at the conference.

On the administrative side, we would like to thank to the organizing committee led by Prof. R. Inkulu (IIT Guwahati), who put in many months of effort in ensuring excellent conference arrangements at the IIT Guwahati campus. We also thank the Easychair team whose software has made it very convenient to do many conference related tasks. Finally, we thank the Dagstuhl LIPIcs staff for their coordination in production of this proceedings, particularly Marc Herbstritt who was very prompt and helpful in answering our questions.

Anil Seth and Nisheeth K. Vishnoi
December 2013

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi

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

Conference Organization

Program Chairs

Anil Seth (IIT Kanpur, India)
Nisheeth K. Vishnoi (Microsoft Research, India)

Program Committee

Manindra Agrawal (IIT Kanpur, India)
Arnab Bhattacharyya (DIMACS, Rutgers and IISc, Bangalore)
Elisa Celis (Xerox Research Centre, India)
Anupam Gupta (CMU)
Prahladh Harsha (TIFR, Mumbai)
Swastik Kopparty (Rutgers)
Meena Mahajan (IMSc, Chennai)
Raghu Meka (IAS, Princeton and DIMACS, Rutgers)
Yogish Sabharwal (IBM Research, India)
C. Seshadhri (Sandia National Lab, CA)
Ola Svensson (EPFL)
Madhur Tulsiani (TTI, Chicago)
Laszlo Vegh (London School of Economics)
Thomas Vidick (MIT)
David Williamson (Cornell)

Mohamed Faouzi Atig (Uppsala, Sweden)
Roberto Bruni (Pisa, Italy)
Arnaud Carayol (Institut Gaspard-Monge, France)
Anuj Dawar (Cambridge, UK)
Laurent Doyen (ENS Cachan)
Deepak D'Souza (IISc, Bangalore)
Deepak Garg (Max Planck, Germany)
Dan Ghica (Birmingham, UK)
Hugo Gimbert (LaBRI)
Ugo Dal Lago (Bologna, Italy)
Ranko Lazic (Warwick, UK)
Gopalan Nadathur (Minnesota, USA)
K. Narayan Kumar (CMI, Chennai)
R. Ramanujam (IMSc, Chennai)
Szymon Torunczyk (Warsaw, Poland)

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi

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

Organizing Committee

R. Inkulu (IIT Guwahati), Chair
Purandar Bhaduri (IIT Guwahati)
Santosh Biswas (IIT Guwahati)
Benny George (IIT Guwahati)
Diganta Goswami (IIT Guwahati)

External Reviewers

Faried Abu Zaid
Luca Aceto
Jade Alglave
Antonios Antoniadis
V Arvind
Eugene Asarin
David Baelde
Christel Baier
Paolo Baldan
Suman Bandyopadhyay
Pablo Barceló
Djamal Belazzougui
Dietmar Berwanger
Amey Bhangale
Markus Blaeser
Christopher Broadbent
Didier Caucal
Venkatesan Chakaravarthy
Deeparnab Chakrabarty
On Siu Chan
Karthik Chandrasekaran
Namit Chaturvedi
Thomas Colcombet
Samir Datta
Yuxin Deng
Kashyap Dixit
Andrew Drucker
Alina Ene
Marco Faella
Stephan Falke
Nathanaël Fijalkow
Tamas Fleiner
Vojtěch Forejt
Zachary Friggstad
Andrew Gacek
Juergen Giesl
Serge Grigorieff
Manoj Gupta
Gregory Gutin
Stefan Göller
Peter Habermehl
Matthew Hague
Ichiro Hasuo
Szczepan Hummel
Petr Jancar
Neil Jones
Lukasz Kaiser
Prateek Karandikar
Telikepalli Kavitha
Felix Klaedtke
Bartek Klin
Naoki Kobayashi
Roman Kontchakov
Eryk Kopczynski
Arpita Korwar
Egor V. Kostylev
Amit Kumar
Mrinal Kumar
Dietrich Kuske
Salvatore La Torre
Carl Leonardsson
Jerome Leroux
Paul Blain Levy
Kevin Lewi
Anthony Widjaja Lin
Alberto Lluch-Lafuente
Kamal Lodaya
Michele Loreti
Anand Louis
P. Madhusudan
Yury Makarychev
David Manlove
Amaldev Manuel
Alberto Marchetti-Spaccamela
Arie Matsliah
Richard Mayr
Paul-André Melliès
Julian Mestre
Matthias Mnich
Fabio Mogavero
M. Raj Mohan
K Narayan Kumar
Gonzalo Navarro
Alantha Newman
Vivek Nigam
Yahav Nussbaum

Krzysztof Onak
Debmalya Panigrahi
Soumya Paul
Marcin Pilipczuk
Jean-Eric Pin
Dmtr Plvlgyi
Pavithra Prabhakar
M. Praveen
Eric Price
Alexander Rabinovich
Roman Rabinovich
Jaikumar Radhakrishnan
Steven Ramsay
Baharak Rastegari
B. Ravikumar
Saurabh Ray
Julien Reichert
Fabien Renaud
Sambuddha Roy
Wojciech Rytter
Sushant Sachdeva
Prakash Saivasan
Sylvain Salvati
Arnaud Sangnier
Srinivasa Rao Satti
Nitin Saurabh
Aleksy Schubert
Stefan Schwoon
Shinnosuke Seki
Valerio Senni
Olivier Serre
Jeffrey Shallit
Mahsa Shirmohammadi
Alexandra Silva
Sunil Simon
A V Sreejith
Karteek Sreenivasaiah
B Srivathsan
Jari Stenman
Howard Straubing
S P Suresh
Riccardo Traverso
Ashutosh Trivedi
Emilio Tuosto
Pawel Urzyczyn
Kasturi Varadarajan
Thomas Varghese
Jan Vondrak
John Wright
Yi Wu
Yuan Zhou
Wieslaw Zielonka

Polar Codes: Reliable Communication with Complexity Polynomial in the Gap to Shannon Capacity

Venkatesan Guruswami

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh PA 15213, USA.
guruswami@cmu.edu

Abstract

Shannon's monumental 1948 work laid the foundations for the rich fields of information and coding theory. The quest for *efficient* coding schemes to approach Shannon capacity has occupied researchers ever since, with spectacular progress enabling the widespread use of error-correcting codes in practice. Yet the theoretical problem of approaching capacity arbitrarily closely with polynomial complexity remained open except in the special case of erasure channels.

In 2008, Arikan proposed an insightful new method for constructing capacity-achieving codes based on channel polarization. In this talk, I will begin with a self-contained survey of Arikan's celebrated construction of polar codes, and then discuss our recent proof (with Patrick Xia) that, for all binary-input symmetric memoryless channels, polar codes enable reliable communication at rates within $\epsilon > 0$ of the Shannon capacity with block length (delay), construction complexity, and decoding complexity all bounded by a *polynomial* in the gap to capacity, i.e., by $\mathrm{poly}(1/\epsilon)$. Polar coding gives the *first explicit construction* with rigorous proofs of all these properties; previous constructions were not known to achieve capacity with less than $\exp(1/\epsilon)$ decoding complexity.

We establish the capacity-achieving property of polar codes via a direct analysis of the underlying martingale of conditional entropies, without relying on the martingale convergence theorem. This step gives rough polarization (noise levels ϵ for the *good channels*), which can then be adequately amplified by tracking the decay of the channel Bhattacharyya parameters. Our effective bounds imply that polar codes can have block length bounded by $\mathrm{poly}(1/\epsilon)$. We also show that the generator matrix of such polar codes can be constructed in polynomial time by algorithmically computing an adequate approximation of the polarization process.

1998 ACM Subject Classification E.4 Coding and Information Theory, F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Error-correction algorithms, Linear Codes, Shannon capacity, Martingale convergence, Computational complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.1

Category Invited Talk

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 1–1

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

Computing With a Fixed Number of Pointers

Martin Hofmann and Ramyaa Ramyaa

Ludwig Maximilian University
Munich, Germany
{hofmann,ramyaa}@ifi.lmu.de

Abstract

Consider the P-complete problem HORN which asks whether a given set of Horn clauses is (un)satisfiable. To solve it one keeps a dynamic set of atoms that are forced to be true. Using the clauses one then adds atoms to this set until saturation is reached. It is easy to see that this dynamic set will in general more than constant size even if we allow to discard already proved atoms. Given that we need logarithmic space to store a single atom on a Turing machine tape this seems like a strong intuitive argument for the hypothesis that logarithmic space is different from polynomial time. We thus tried to find formal models of computation in which this intuitive argument can be made rigorous. Thus, we study computational models that can be simulated in logarithmic space and encompass logspace algorithms which manipulate a constant size of objects that require logarithmic space individually such as pointers or graph nodes. The hope is then to be able to show that such models are provably unable to solve P-complete problems. We report in this survey article on our partial results towards this goal as well as the state-of-the-art in general.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Logarithmic space, Jumping graph automata (JAGs), st-connectivity, co-st-connectivity, Cayley graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.3

Category Invited Talk

1 Introduction

Consider the problem HORN defined as follows. We are given a finite set of letters Σ and a finite set of clauses $\mathcal{C}$ each of the form $U \to V$ where U, V are subsets of Σ with $|U| \leq 2$ and $|V| \leq 1$. A *valuation* $\eta : \Sigma \to \{0, 1\}$ *satisfies* a clause $U \to V$ if $\eta(x) = 0$ for some $x \in U$ or $\eta(y) = 1$ for some $y \in V$. E.g. η satisfies $x, y \to z$ if whenever $\eta(x) = \eta(y) = 1$ then $\eta(z) = 1$, too. It satisfies $\to x$ (here $U = \emptyset$) if $\eta(x) = 1$; it satisfies $x \to$ (here $V = \emptyset$) if $\eta(x) = 0$.

For example, the instance $\mathcal{C} = \{\to x;\ x \to y;\ x,y \to z;\ z \to\}$ (where $\Sigma = \{x, y, z\}$) is unsatisfiable. We call a letter x with $\to x$ in $\mathcal{C}$ an *axiom* and we call a letter y with $y \to$ in $\mathcal{C}$ a *goal*. In this terminology an instance is unsatisfiable if one of the goals may be deduced from the axioms with the understanding that if there is a clause $x, y \to z$ and x, y have both been deduced then z may be deduced, too.

The obvious dynamic programming algorithm (grow a set of letters that can be deduced by repeatedly going over the clauses until you reach a goal) places HORN into the class PTIME (polynomial time) and it is well-known and easy to see that HORN is PTIME-complete w.r.t. LOGSPACE-reductions, e.g. by a straightforward reduction to Cook's solvable paths [1].

On the other hand, considering that storing a single letter requires logarithmic space (in the number of letters $|\Sigma|$), we see that this algorithm does not run in logarithmic space on a

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 3–18

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

Turing machine unless we can bound the size of the dynamic set in which we keep already deduced letters by a constant. We might try to improve the algorithm by removing entries from the dynamic set according to some strategy. However, even if we allow nondeterministic selection of the letters to be "forgotten", we will not be able to achieve a constant bound as can be seen from the following pebble argument:

▶ Proposition 0.1. For each $d \in \mathbb{N}$ let $\Sigma_d = \{0,1\}^{\leq d}$ and

$$\mathcal{C}_d = \{\rightarrow w \mid \ |w| = d\} \cup \{w0, w1 \rightarrow w \mid \ |w| < d\} \cup \{\epsilon \rightarrow\}$$

The instance $(\Sigma_d, \mathcal{C}_d)$ is unsatisfiable. Let $A_1, \ldots, A_N$ be a sequence of subsets of Σ such that $A_1 = \emptyset$ and $A_{i+1} \subseteq A_i \cup \{x \mid \ y, z \rightarrow y \in \mathcal{C}_n,\ \{y,z\} \subseteq A_n\}$. If $\epsilon \in A_N$ then $\max_i |A_i| \geq d+1$. The easy proof is by induction on d.

Since $|\Sigma_d| = 2^{d+1} - 1$ this furnishes a logarithmic lower bound on the size of the dynamic set in any algorithm for HORN that uses the dynamic programming strategy and is allowed to forget already established facts according to any strategy however clever it might be. Thus, no such algorithm can be implemented in logarithmic space (to be precise, space $\log(n)^2$ is needed at least). We also remark that even a $\sqrt{n}$ lower bound on the size of the dynamic set can be achieved using a more complicated instance based on Cook's "pyramids" [3].

A simpler yet related problem is st-connectivity in directed or undirected graphs. It can be considered as the special case of HORN where one has only one axiom (the source), one goal (the target), and for every other clause $U \rightarrow V$ has $|U| = |V| = 1$. Those then represent the edges of a graph. In this case, the pebbling argument no longer works and indeed, we can solve st-connectivity with a nondeterministic algorithm that keeps only a constant number of letters (graph nodes) in memory. In fact just one suffices. On the other hand, it is not known whether a deterministic algorithm with the same space bounds exists for this problem.

Indeed, Cook and Rackoff have shown that no deterministic algorithm exists for st-connectivity even on undirected graphs (USTCON) which uses graph nodes only abstractly rather than having access to their machine representation and can only hold a fixed number of graph nodes simultaneously in memory. Of course, one must make precise what is meant by "use graph nodes only abstractly". There are several formalisations which attempt to do so. We describe below in more detail: Cook and Rackoff's Jumping Automata on Graphs (JAGs) for which their original result holds; Deterministic Transitive Closure Logic, an extension of first order logic by a transitive closure operator providing unlimited iteration; and, finally, the PURPLE programming language introduced by the first author and U. Schöpp which extends JAGs with a Java like iterator and subsumes DTC-logic. In [14] we could extend Cook and Rackoff's result to PURPLE and thus also show that DTC-logic cannot define USTCON which was an open question at the time.

Interestingly, however, Reingold's result [19] shows that USTCON *can* be solved in logarithmic space on a Turing machine. However, this algorithm, in addition to storing a constant number of graph nodes stores a logarithmic number of booleans.

At the time, a motivation for Cook and Rackoff's work could have been the attempt to separate PTIME from LOGSPACE (deterministic logarithmic space) or indeed LOGSPACE from NLOGSPACE (nondeterministic logarithmic space). Indeed, before Reingold's result one might be tempted to try to show the non-existence of any LOGSPACE-algorithm by devising an ingenuous abstraction that associates with any run of a deterministic algorithm for USTCON a run of an accompanying JAG. Now, Reingold's result shows that without further restrictions on the nature of such hypothetical algorithm such an argument is bound to fail.

On the other hand, Reingold's result shows that looking at graph nodes or propositional letters as abstract objects provably makes a difference which raises the hope that while

an unconditional separation of, say, LOGSPACE and PTIME is not attainable with current methods one might still obtain a "relativized" separation, for instance, in the form of a rigorous proof that no algorithm for HORN exists that only uses a fixed number of abstract pointers (referring to graph nodes, letters, clauses, or similar).

Of course, neither JAGs nor DTC-logic and even PURPLE are able to solve USTCON, and thus are unable to solve HORN. Thus, for a meaningful result of this kind, one would need an extension of any of these formalisms which does encompass USTCON. In this paper we summarise our efforts at finding such an extension and try to explain why this turned out to be unexpectedly difficult.

The rest of this paper is structured as follows.

The next section describes the basic framework we consider - systems which build upon minimal abstract pointers.

Section 3 describes Cook and Rackoff's jumping automata on graphs, the first, and most basic model for computation with a fixed number of pointers. We also report therein on our current study of the computational strength of nondeterministic jumping automata.

Section 4 describes the programming formalism PURPLE [15] which extends JAGs with an iteration construct that provides access to all nodes of the input structure not only those that are accessible from originally known ones. We present our results on the expressivity of PURPLE and its extensions.

Section 5 is about (deterministic) transitive closure logic known from finite model theory. We show that its strength lies strictly between JAGs and PURPLE.

Section 6 concludes and gives some directions for future work.

2 Minimal Abstract Pointers

Our aim is to define a system contained in LOGSPACE that is provably weaker than PTIME, but powerful enough to make the separation non trivial. Following the intuition outlined above, the systems we consider aim to capture the subclass of "abstract pointer algorithms" within LOGSPACE.

The minimal requirements of such a system is the ability to refer to a node of a graph (or any structured input) using pointer (or pebble) variables, comparing two such variables for equality, assigning the value of one to another, and traversing along an edge. Thus, the concrete representation of the pointers is hidden. The main difference between these systems and Turing machines is that Turing machines have access to a binary encoding of the input graph which embodies a total ordering on the nodes.

In fact, it is easy to see that once a total ordering on the input nodes is available even the most basic systems we consider capture all of LOGSPACE. Such a total ordering can be provided in a number of ways, for instance by assuming that one of the directions $\{1, \ldots, d\}$ threads all the nodes ($\lambda x.x.i$ is a permutation for some i). So, we want to focus on graphs for which such total ordering is neither directly available nor definable.

On the other hand, we typically assume the presence of a local order, in particular an ordering of the edges adjacent to or emanating from a node. A *1-locally ordered graph*, (1LO graph) is a one such that for each node, the edges emanating from it are ordered. A *2-locally ordered graph* (2LO graph) is one such that for each node, the edges coming out of it or leading to it are ordered. For undirected graphs these two coincide.

The encoding of graphs using abstract pointers and the presentation of graphs in the context of JAGs both induce at least a 1-local ordering. It is, however, possible to represent graphs without local ordering as pointer structures, e.g. by using special edge objects having

pointers to their source and target. In the context of first-order logic and extensions thereof, the unordered case may seem more natural but leads to artificial weaknesses.

An interesting intermediary between absence and presence of a total ordering is the ability to count or more generally perform arithmetic till the size of the input, called *counting*. A total order can be utilized to simulate counting. However, this is not the case with local orderings. In the extreme case of discrete graphs, local ordering provides no information at all, and it is not possible to count over these graphs.

The systems we study are: jumping automata on graphs (JAGs), "pure pointer language" (PURPLE language), and Deterministic transitive closure Logic (DTC). It has been shown that without counting, these systems cannot solve connectivity. We describe these systems and several extensions.

3 Jumping automata on graphs

Cook and Rackoff ([2]) introduced Jumping Automata on Graphs (JAGs) in order to study space lower bounds for reachability problems. A JAG is a finite automaton which can move a fixed number of pebbles along labelled edges of input graphs of fixed degree. Thus, JAGs are a nonuniform machine model.

A labelled degree d graph for $d > 1$ comprises a set V of vertices and a function $\rho : V \times \{1, \ldots, d\} \to V$. If $\rho(v, i) = v'$ then we say that (v, v') is an edge labelled i from v to v'. All graphs considered here are labelled degree d graphs for some d. The important difference to the more standard graphs of the form $G = (V, E)$ where $E \subseteq V \times V$ is that the out degree of each vertex is exactly d and, more importantly, that the edges emanating from any one node are linearly ordered. One extends ρ naturally to sequences of labels (from $\{1, \ldots, d\}^*$) and writes $v' = v.w$ if $\rho(v, w) = v'$ for $v, v' \in V$ and $w \in \{1, \ldots, d\}^*$. The induced sequence of intermediate vertices (including v, v') is called the path labelled w from v to v'. Such a graph is undirected if for each edge there is one in the opposite direction $\rho(v, i) = v' \Rightarrow \exists j.\rho(v', j) = i$. It is technically useful to slightly generalise this and also regard such graphs as undirected if each edge can be reversed by a path of a fixed maximum length.

▶ **Definition 1.** A d-Jumping Automaton for Graphs (d-JAG), J, consists of

- a finite set Q of states with distinguished start state q_0 and accept state q_a
- a finite set P of objects called pebbles (numbered 1 through p)
- a transition function δ which assigns to each state q and each equivalence relation π on P (representing incidence of pebbles) a set of pairs $(q', \vec{c})$ where $q' \in Q$ is the successor state and where $\vec{c} = (c_1, \ldots, c_p)$ is a sequence of moves, one for each pebble. Such a move can either be of the form move(i) where $i \in \{1, \ldots, d\}$ (move the pebble along edge i) or jump(j) where $j \in \{1, \ldots, p\}$ (jump the pebble to the (old) position of pebble j).
- The automaton is deterministic if $\delta(q, \pi)$ is a singleton set for each q, π.

The input to a JAG is a labelled degree d graph. An *instantaneous description* (id) of a JAG J on an input graph G is specified by a state q and a function, *node*, from the P to the nodes of G where for any pebble p, *node*(p) gives the node on which the pebble p is placed.

Given an id $(q, node)$ a legal move of J is an element $(q', \vec{c}) \in \delta(q, \pi)$ where π is the equivalence relation given by $p\ \pi\ p' \iff node(p) = node(p')$. The action of a JAG, or the *next move* is given by its transition function and consists of the control passing to a new state after moving each pebble i according to c_i: (a) if $c_i =$ move(j) move i along edge j; (b) if $c_i =$ jump(j) move (jump) it to *node*(j). Any sequence (finite or infinite) of id's of a JAG J on an input G which form consecutive legal moves of J is called a *computation* of J

on G. We assume that input graphs G have distinguished nodes *startnode* and *targetnode*, and that JAGs have dedicated pebbles s and t. The initial id of J on input G has state q_0, $node(t) = targetnode$ and $\forall q \neq t.\ node(q) = startnode$. J accepts G if the computation of J on G starting with the initial id ends in an id with state q_a.

One criticism of JAGs is that they are artificially weak on directed graphs. Since edges can only be traversed in the forward direction, there is no way for a JAG to reach a node without incoming edges, for example. One solution to this is to work with graphs having a local ordering both on the outgoing and on the incoming edges of each node, so that edges can be traversed in both directions [7].

Cook and Rackoff's result [2] shows that even with this modifications, JAGs can only compute local properties:

▶ **Theorem 2** ([2]). *(u)st-connectivity cannot be solved by* JAG*s.*

It is instructive to deduce this result from a generalization due to Schöpp [20] who gave a formal proof in Coq. For any group G define its exponent $\exp(G)$ as the maximum element order, i.e., the least m so that $g^m = e$ holds for all $g \in G$. Note that $\exp((\mathbb{Z}/m\mathbb{Z})^d) = m$.

▶ **Theorem 3.** *Let G be a group. A* JAG *with p pebbles and q states can visit at most $(q \cdot \exp(G))^{d^p}$ nodes in the course of any computation on $CG(G)$.*

This implies that JAGs trivially cannot solve HORN since USTCON is obviously a special case of HORN. It is thus natural to investigate strengthenings of JAGs.

3.1 Counters

The RAMJAG [18] consists of a finite state control together with p pebbles and a fixed number of $O(\log(n))$-bit registers which in total require $O(log(q))$ bits of storage. Its storage is defined as $(plog(n) + log(q))$ bits. on which it can perform the usual RAM operations on the registers and also three special instructions: walk, jump and compare. The instructions $walk(P, j)$ and $jump(P, P')$ are the same as that in a JAG. The instruction $compare(P, P', R)$ checks whether pebbles P and P' are on the same node and stores the result (T or F) in a register R. where n is the size of the input graph. Obviously, using more registers this bound extends to any polynomial in n.

▶ **Theorem 4** ([18]). *Reingold's algorithm for* USTCON *can be implemented with* RAMJAG*s.*

We remark that Beame et al. citeDBLP:journals/siamcomp/BeameBRRT99 showed that JAGs with arithmetic registers ($O(\log n)$ space bounded JAGs in their terminology) are equivalent to LOGSPACE Turing machines without using and in fact prior to Reingold's theorem.

▶ **Corollary 5.** RAMJAG*s can define a total order on connected graphs*

Proof. Choose an arbitrary node as the start node and enumerate all logarithmic length paths from it. The order in which the nodes are visited gives a total ordering on the graph. ◀

▶ **Corollary 6.** JAG*s cannot count*

Proof. JAGs cannot solve USTCON, but RAMJAGs can. This shows that counting cannot be simulated in JAGs. ◀

3.2 Nondeterminism

Regarding nondeterministic JAGs (ND-JAGs) the situation is less clear. For our purpose, ND-JAGs are relevant, for they are stronger than deterministic JAGs and PURPLE since they can solve STCON, so assuming that they are not equivalent to NLOGSPACE it would then constitute and interesting and perhaps accessible open question whether nondeterministic JAGs can solve HORN.

Before, however, attempting that question one should first try to see whether ND-JAGs can solve co-STCON or even just co-USTCON. i.e., whether there exists a nondeterministic JAG with the property that if *startnode* and *targetnode* are not connected then there exists an accepting run whereas in the case where they are connected, all runs reject or abort. It has been left as an open problem in [5] whether or not ND-JAGs are able to decide co-USTCON or indeed whether their computational power equals all of NLOGSPACE.

In a recent as yet unpublished paper [12] the authors have tried to make some progress towards the special case of this question where the graphs under consideration are *Cayley graphs*. Recall that the Cayley graph of a group G with generators $\vec{m}$, written as $CG(G, \vec{m})$ is the labelled degree $|\vec{m}|$ graph whose nodes are the elements of G and where the edge labelled i from node v leads to $m_i v$, formally $\rho(v, i) = m_i v$. Note that since every generator has an inverse the Cayley graphs are undirected in the above relaxed sense.

Cayley graphs are interesting in this contexts because they furnish the hard examples in [2] and [14]. In the former case the underlying group is $(\mathbb{Z}/m\mathbb{Z})^m$ whose Cayley graph resembles an m-dimensional torus of circumference d. E.g. for $m = 480$ and $d = 2$ one obtains a 480 x 480 "screen" with opposite borders identified as is common in some video games. It is hard for a JAG to find its way through such a graph because the close neighbourhoods of all nodes are the same and because repeated moves quickly lead to a repetition. E.g. the order of each cyclic subgroup of $(\mathbb{Z}/m\mathbb{Z})^d$ is $\leq m$ whereas the order of the group itself is m^d. Of course, a JAG does not necessarily stupidly repeat a fixed move and it required Cook and Rackoff an ingenious pumping argument to turn this intuition into a rigorous proof. The result in [14] generalises this using an iterated wreath product of a cyclic group $\mathbb{Z}/m\mathbb{Z}$.

▶ **Definition 7.** A nondeterministic Jumping Automaton for Graphs (ND-JAG) J is a JAG whose transition function is nondeterministic. It accepts an input if there is *some* finite computation starting at the initial configuration that reaches q_a, and rejects an input if *no* such computation does. A $d-$ND-JAG operates on graphs of degree d. Again, we assume that appropriate degree reduction is applied before inputting a graph to an ND-JAG.

It is easy to see that the argument used in [2] for deterministic JAGs which is similar to a pumping argument cannot be adapted easily to ND-JAGs, which can solve reachability (guess a path from *startnode* to *targetnode*). However, it is unclear whether ND-JAGs can solve co-st-connectivity. Since JAGs cannot count, it is reasonable to believe that ND-JAGs cannot implement Immerman-Szelepcsenyi's algorithm, and more generally, cannot solve co-st-connectivity.

▶ **Theorem 8** ([12]). ND-JAG*s are equivalent to* NLOGSPACE *and thus can in particular solve co-*STCON *if the input consists of disjoint copies of one of Cayley graphs where the underlying group is one of the following*

- *an abelian group*
- *a finite simple group*
- *groups obtained from these by direct, semidirect, or wreath product*
- *iterations of the above product constructions*

To us this power of ND-JAGs came as quite a surprise; initially, we believed that even on the (abelian) group $(\mathbb{Z}/m\mathbb{Z})^d$ that were used by Rackoff the ND-JAGs would be strictly weaker than NLOGSPACE.

To give a taste of these results we sketch here a special case:

▶ **Theorem 9.** *There exists an* ND-JAG *that can visit all nodes of* $CG(\vec{m}, (\mathbb{Z}/m\mathbb{Z})^d)$ *in a fixed order where* $\vec{m}$ *comprises the unit vectors* $e_1, \ldots e_d$ *of dimension* d*.*

This shows in particular that co-STCON can be solved if the input consists of several disjoint copies of this Cayley graph.

Proof sketch. Indeed, each element of the group can be uniquely written in the form $\lambda_1 e_1 + \ldots \lambda_d e_d$ where $\lambda_i \in \{0, \ldots, m-1\}$. Moreover, we can order the elements of the group lexicographically using this representation. Now, we can design an ND-JAG that places a "cursor pebble" on all nodes of the Cayley graph in this lexicographic order. Suppose that the cursor pebble is on $\lambda_1 e_1 + \ldots \lambda_d e_d$. Using another pebble we nondeterministically trace a path from *startnode* to the cursor pebble making sure that it has the form "some e_1, then some e_2, etc". By uniqueness of representation this path will repeat the coefficients λ_i or fail to reach the cursor pebble in which case we abort. As we trace this path we can on-the-fly move a third pebble to the lexicographically next position by incrementing the first coefficient that is different from $m-1$ and resetting all the previous ones. ◀

Given these results our current working hypothesis is that ND-JAGs can solve co-STCON on all graphs (in fact, we even believe now that they capture NLOGSPACE on all graphs); the natural next step will be to demonstrate this for arbitrary Cayley graphs. We also note that our results considerably narrow the search for counterexamples to the conjecture and in particular rule out all the known ones.

We also note that any counterexample to our working hypothesis would in particular have to be such that deterministic JAGs cannot solve STCON on them (otherwise we would trivially get co-STCON) so that we would need a new proof of Cook-Rackoff's result with substantially different example graphs for the known ones are thwarted by our current results.

4 Purple language

Rather than as an automaton, we may understand a JAG as a while-program whose variables are partitioned into two types: boolean variables and graph pointer variables. Boolean variables are used to represent the finite state of the JAG and the usual boolean operations are available for boolean expressions. Pointer variables are used to reference graph nodes in the same way that pebbles are used in the automata-theoretic formulation of JAGs. For pointer variables one only has an equality test and, for each constant number i, a successor operation to move a pointer variable along the i-th edge from the graph node it points to. Over unconnected graphs, JAGs may not be able to visit all nodes. For example, the property whether a graph contains a node with a self-loop is not decidable with JAGs for the simple reason that such a witnessing node might be in an unreachable part of the graph.

To address this, the formalism PURPLE (for "pure pointer language"), was introduced by the first author and U. Schöpp [15]. Essentially, it consists of augmenting this programming language-theoretic version of JAGs with a special loop construct (`forall` x `do` P) whose meaning is to set the pointer variable x successively to all graph nodes in some arbitrary order and to evaluate the loop body P after each such setting. The important point is that the order is arbitrary and will in general be different each time a `forall`-loop is evaluated.

A program computes a function or predicate only if it gives the same (and correct) result for all such orderings. The `forall`-loop in PURPLE can be used to evaluate first-order quantifiers and thus to encode DTC- logic (see below) on locally ordered graphs. Moreover, PURPLE is strictly more expressive than that logic. For instance, determining whether the input graph has an even number of nodes is not possible in locally-ordered DTC logic [10], but the following PURPLE-program does this: ($b := \mathit{true}$; `forall` x `do` $b := \mathit{not}(b)$.

PURPLE program are parametrised by a finite set L of labels and a finite set S of predicate symbols. Each predicate symbol p is assumed to have a finite arity $ar(p) \in \mathbb{N}$.

The input of a programs is a pointer structure, which interprets the labels and predicates: A *pointer structure* on L and S ((L,S)-model, for short) specifies a finite set U as a universe, a function $[\![l]\!] \colon U \to U$ for each label $l \in L$ and a set $[\![p]\!] \subseteq U^{ar(p)}$ for each predicate symbol p.

The special case of d-labelled graphs arises when $L = \{1, \dots, d\}$ and $S = \emptyset$.

A program with labels L and predicate symbols S can access its input structure through the following terms for pointers to elements of the universe and for booleans.

$$\begin{aligned} t^U &::= x^U \mid t^U.l \text{ for any label } l \in L \\ t^{\texttt{bool}} &::= x^{\texttt{bool}} \mid \neg t^{\texttt{bool}} \mid t_1^{\texttt{bool}} \wedge t_2^{\texttt{bool}} \mid p(x_1^U, \dots, x_{ar(p)}^U) \text{ for any predicate } p \in S \end{aligned}$$

We call x^U pointer variables. The intention is that $t^U.l$ is interpreted by $[\![l]\!](t^U)$.

The programs themselves are given by the grammar.

$$\begin{aligned} \mathit{Prg} ::= \quad & \texttt{skip} \mid \mathit{Prg}_1; \mathit{Prg}_2 \mid x^U := t^U \mid x^{\texttt{bool}} := t^{\texttt{bool}} \\ & \mid \texttt{if } t^{\texttt{bool}} \texttt{ then } \mathit{Prg}_1 \texttt{ else } \mathit{Prg}_2 \mid \texttt{forall } x^U \texttt{ do } \mathit{Prg} \end{aligned}$$

We write `if` $t^{\texttt{bool}}$ `then` Prg for `if` $t^{\texttt{bool}}$ `then` Prg `else skip`.

A *configuration* $\langle \rho, q \rangle$ consists of a *pebbling* ρ and a *state* q. The pebbling ρ maps pointer variables (which we also call pebbles) to elements of the universe U. The state q is a function mapping boolean variables to booleans. Given a configuration I, we can define an interpretation of the terms $[\![t^{\texttt{bool}}]\!]_I \in \{\texttt{true}, \texttt{false}\}$ and $[\![t^U]\!]_I \in U$ in the usual way.

A big-step reduction relation $\mathit{Prg} \vdash_M I \longrightarrow O$ between configurations I and O on some (L,S)-model M and a program Prg is defined inductively by the following clauses:

- $\texttt{skip} \vdash_M I \longrightarrow I$.
- $\mathit{Prg}_1; \mathit{Prg}_2 \vdash_M I \longrightarrow O$ if $\mathit{Prg}_1 \vdash_M I \longrightarrow R$ and $\mathit{Prg}_2 \vdash_M R \longrightarrow O$ for some R.
- $x^U := t^U \vdash_M \langle \rho, q \rangle \longrightarrow \langle \rho[x \mapsto [\![t]\!]_{\langle \rho, q \rangle}], q \rangle$.
- $x^{\texttt{bool}} := t^{\texttt{bool}} \vdash_M \langle \rho, q \rangle \longrightarrow \langle \rho, q[x \mapsto [\![t]\!]_{\langle \rho, q \rangle}] \rangle$.
- `if` t `then` Prg_1 `else` $\mathit{Prg}_2 \vdash_M I \longrightarrow O$ if $[\![t]\!]_I = \texttt{true}$ and $\mathit{Prg}_1 \vdash_M I \longrightarrow O$.
- `if` t `then` Prg_1 `else` $\mathit{Prg}_2 \vdash_M I \longrightarrow O$ if $[\![t]\!]_I = \texttt{false}$ and $\mathit{Prg}_2 \vdash_M I \longrightarrow O$.
- `forall` x^U `do` $\mathit{Prg}_1 \vdash_M I \longrightarrow O$ if there exists an enumeration $u_1, u_2, \dots, u_n$ of $[\![U]\!]$ and configurations $I = \langle \rho_1, q_1 \rangle, \langle \rho_2, q_2 \rangle, \dots, \langle \rho_{n+1}, q_{n+1} \rangle = O$, such that $\mathit{Prg}_1 \vdash_M \langle \rho_k[x \mapsto u_k], q_k \rangle \longrightarrow \langle \rho_{k+1}, q_{k+1} \rangle$ holds for all $k \in \{1, \dots, n\}$.

When the model M is clear from the context we may omit the subscript.

In order for a PURPLE program to accept (resp. reject) an input, it must do so no matter what enumerations are chosen for its `forall`-loops. This is defined formally in the next definition, in which we use a boolean variable *result* to indicate acceptance.

▶ **Definition 10.** A program Prg *accepts* (resp. rejects) an (L,S)-model M if $\mathit{Prg} \vdash_M \langle \rho, q \rangle \longrightarrow \langle \rho', q' \rangle$ implies $q'(\mathit{result}) = \texttt{true}$ (resp. $q'(\mathit{result}) = \texttt{false}$) for all ρ, ρ', q and q'.

A program Prg *recognises* a set X of (L,S)-models if it accepts any model in X and rejects all others. Note that a program may neither accept nor reject its input, namely if for some

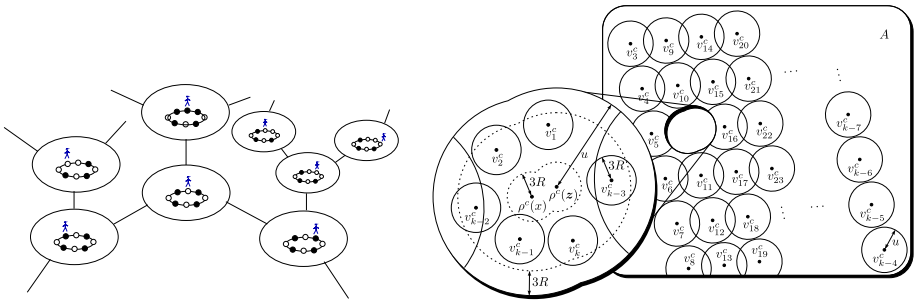


Figure 1 The lamplighter graph $CG(\Lambda(\mathbb{Z}/8\mathbb{Z}))$ and the traversal sequence from [14]

runs it returns `true` and for others it returns `false`. To put it simply, a PURPLE program for some problem X should give the correct answer, be it `true` or `false` for any given input and independent of the run, i.e. the traversal sequences chosen.

▶ **Theorem 11.** *It is undecidable whether a given* PURPLE *program is such that for every input, it either rejects or accepts.*

We also note that predicate symbols, which were not part of the original definition of PURPLE [15], are there just for notational convenience and do not add expressive power. Unary predicates can be modelled with an extra pointer that points to designated nodes for "true" and "false". A binary relation can be modelled by introducing an extra node for each pair of related nodes with pointers *fst* and *snd* pointing to the latter two nodes. One uses a unary predicate to differentiate between the actual nodes and these helper nodes.

4.1 Power of purple

While PURPLE subsumes JAGs and also deterministic transitive closure logic (see below) it is strictly weaker than LOGSPACE. Notice that PURPLE programs can be evaluated on a LOGSPACE-bounded Turing machine.

▶ **Theorem 12** ([15]). *Checking whether the input is a discrete graph with n nodes, where n is a power of two, is possible in* LOGSPACE *but cannot be programmed in* PURPLE.

▶ **Theorem 13** ([14]). USTCON *cannot be decided in* PURPLE.

Proof idea. For any group G one defines the lamplighter group $\Lambda(G)$ by $\Lambda(G) = \{(f, g) \mid f \in 2^G, g \in G\}$ and $(f, g)(f', g') = (\lambda h.f(h) + f'(gh), gg')$ where $+$ refers to addition modulo 2. Given a set of generators $(m_1, \ldots, m_k)$ for G one can generate $\Lambda(G)$ by $(0, m_i)$ for $i = 1 \ldots k$ and the *toggle move* (t, e) where $t(e) = 1$ and $t(g) = 0$ for $g \neq e$. The nodes of the Cayley graph $CG(\Lambda(G))$ can be thought of as states of a system involving one streetlight at each node of $CG(G)$ and a lamplighter situation at one such node. Fig. 1 shows 8 of the 2048 nodes of $CG(\Lambda(\mathbb{Z}/8\mathbb{Z}))$. If G has order n, number of generators m and exponent e and n', m', e' denote those measures for $\Lambda(G)$ then we have $n' = n2^n$, $m' = m + 1$, $e' = 2e$. Thus, repeated application of the lamplighter construction furnishes groups with a very high order yet moderate number of generators and exponent. Nothing beyond these numerical properties is required from the lamplighter construction for our purpose.

Now, for any PURPLE-program P we can find t, m depending on size and nesting depth of loops in P so that when run on $CG(\Lambda^t(\mathbb{Z}/m\mathbb{Z}))$ the effect of P can be simulated (for appropriately chosen traversal sequences of the `forall`-loops) by a very large JAG. Theorem 3 applied to this JAG then shows that some nodes remain unvisited.

To obtain the desired simulation by a JAG assume that all `forall`-loops except for the outermost one have already been eliminated. One thus essentially has a JAG with one special iteration pebble that is supposed to be placed on every node in some arbitrary order. In between such placements the JAG is to be run until it reaches a dedicated state. Now, in an intuitive sense that can be made precise, if the traversal sequence is chosen in such a way that temporally close nodes, in particular successive ones, are sufficiently far apart spatially, then the only nodes of the sequence that the JAG will be able to remember will be those from the beginning and the end of the traversal sequence. Thus, if the traversal sequence is chosen in this fashion then at the end of the traversal all pebbles will be close to the original nodes and it is possible to hardwire the total effect of the `forall`-loop into a very large JAG. The second half of Figure 1 illustrates this traversal sequence $v_1^c, v_2^c, \ldots$. Herein R and u are appropriately chosen large numbers. The radius R is related to the number of nods the JAG representing the body of the loop is able to visit according to Theorem 3, quantity u on the other hand stems from the combinatorial possibility of designing an appropriate sequence. Finally, $\rho^c(\_)$ indicates the initial pebble positions. ◀

This result provides the strongest possible evidence so far that USTCON cannot be solved with a constant number of abstract pointers and that the use of arithmetic in Reingold's algorithm is intrinsic to the problem solved. The result also answers an open question about transitive closure logic, see Section 5 below.

Similar to the case of JAGs, this implies that PURPLE trivially cannot solve HORN since USTCON is obviously a special case of HORN. It is thus natural to investigate strengthenings of PURPLE by various computational devices that stay within LOGSPACE or NLOGSPACE. We describe our efforts in this direction in the following subsections.

4.2 Counters

So, one obvious addition to PURPLE is counting. Though PURPLE subsumes JAGs, we explored whether PURPLE with counting is strictly continued within LOGSPACE. Here we extend PURPLE by counting variables ("counters"), each of which can hold a number from 0 to the size of the input structure's universe, and we extend the terms with arithmetic operations:

$$t^{\texttt{bool}} ::= \cdots \mid \mathit{iszero}(t^{\texttt{count}}) \qquad\qquad t^{\texttt{count}} ::= x^{\texttt{count}} \mid \mathit{max} \mid \mathit{pred}(t^{\texttt{count}})$$

We extend the operational semantics such that the state q now not only maps boolean variables to booleans, but also counting variables to numbers.

PURPLE with counters (PURPLE$_c$) can do arithmetic: the complement of a counter can be computed using *max* and repeated decrement; the increment can be implemented using double complementation and decrement; The rest of the operations follow by repeated applications of these. PURPLE$_c$ can count the number of tuples of nodes satisfying any PURPLE-definable property, and so can simulate counting quantifiers used in DTC or TC logics.

▶ **Lemma 14.** PURPLE$_c$ *captures* LOGSPACE *on graphs with a two-way local ordering, represented as pointer structures as described above.*

Outline. PURPLE captures all of LOGSPACE on ordered graphs. So, it suffices to show that a total ordering can be defined on any input graph.

To this end we note that given any graph node n, PURPLE$_c$ can define a total ordering of the weakly connected component containing n. We use Reingold's algorithm for undirected s-t-connectivity, which checks for connectivity by enumerating all nodes in the weakly

connected component of s and checking if t appears therein. This algorithm be implemented by RAMJAGs [18], and so, by an easy translation, also in PURPLE$_c$. In this way, PURPLE$_c$ can order the nodes of the weakly connected component according to their order of their first appearance in the enumeration.

In their proof that TC-logic with counting captures NLOGSPACE [8], Etessami and Immerman have shown how a total ordering can be defined using counting from such orderings of the weakly connected components. This argument can be adapted to PURPLE$_c$ to complete the proof. ◀

4.2.1 Iterators

Another possibility for strengthening PURPLE that we investigated consists of replacing `forall`-loops by *iterators* that are available e.g. in the Java library for the representation of sets as trees or hash maps. Thus, in addition to Boolean and pointer variables, this extension of PURPLE then has *iterator* variables which store a subset of the nodes of the input graph. The operations allowed on an iterator variable are:

- *Initialize*: The variable is assigned the set of all nodes of the input graph
- *Next*: If the variable refers to a non-empty set, an arbitrary node is removed from it (so the variable now refers to a set without this node), and the node is returned.
- *isNull*: Returns `true` if the variable is empty and `false` otherwise

With iterators in place, the `forall`-loop can be replaced by a plain while loop. Further, we can determine whether the number of nodes with a self loop is equal to the number of nodes without one. Surprisingly, iterators permit the definition of counting and thus render PURPLE with iterators (on locally ordered input) equivalent to full LOGSPACE.

▶ **Theorem 15.** PURPLE *with iterators can count.*

Proof. A counter variable is represented by an iterator variable which is assigned a subset of nodes with the required cardinality. It is direct that the operations of checking for zero and decrement can be performed. To increment a variable x, we use another iterator variable y as follows: Initialize y and keep decrementing both x and y till x is zero; increment y once more and initialize x; finally, decrement both variables till y is zero. Similarly, variable can be copied (assigned to another variable). ◀

4.3 Nondeterminism with counting but without local order

By the asymmetric acceptance and rejection conditions that are used in any nondeterministic definition, it is not clear whether nondeterministic PURPLE is closed under complementation. Since the answer is not known even in the case of JAGs, we add counting as well, allowing us to implement Immerman-Szelepcsenyi's algorithm for complementation in NLOGSPACE to get PURPLE $_{c,nd}$. However, adding counting boosts the power of PURPLE to full NLOGSPACE in the presence of local ordering. So, we consider input graphs without local ordering in this case. These can be presented via primitive predicates or special edge objects.

Counters are added as described above. Adding nondeterminism is not completely straightforward, as we must separate nondeterministic choices from the choices made in the evaluation of `forall`-loops. We would like to allow programs to make nondeterministic choices, while still maintaining that their acceptance behaviour is independent of the enumerations chosen in the `forall`-loops.

PURPLE with nondeterminism (PURPLE$_{nd}$) has a command for nondeterministic choice:

$Prg ::= \ldots \mid$ `choose` Prg_1 `or` Prg_2

To define the semantics of PURPLE with nondeterminism, we amend the notion of configuration so that it now consists of a triple $\langle \rho, q, \sigma \rangle$, where ρ and q are a pebbling and a state as before and σ is an infinite list enumerations of the universe U. This new component σ specifies the runs of all future `forall` loops. Therefore, in the definition of $(\texttt{forall } x^U \texttt{ do } Prg) \vdash \langle \rho, q, \sigma \rangle \longrightarrow \langle \rho', q', \sigma' \rangle$ we do not use an arbitrary enumeration of U, but we take the first one $u_1, \ldots, u_n$ from σ. That is, we require there to be configurations $\langle \rho, q, \mathit{tail}(\sigma) \rangle = \langle \rho_1, q_1, \sigma_1 \rangle, \ldots, \langle \rho_{n+1}, q_{n+1}, \sigma_{n+1} \rangle = \langle \rho', q', \sigma' \rangle$ with $Prg \vdash \langle \rho_k[x \mapsto u_k], q_k, \sigma_k \rangle \longrightarrow \langle \rho_{k+1}, q_{k+1}, \sigma_{k+1} \rangle$ for all $k \in \{1, \ldots, n\}$. For the semantics of the new term, we stipulate $\texttt{choose } Prg_1 \texttt{ or } Prg_2 \vdash I \longrightarrow O$ if $Prg_1 \vdash I \longrightarrow O$ or $Prg_2 \vdash I \longrightarrow O$. In all other cases, σ is merely passed on. E.g. $x := t \vdash \langle \rho, q, \sigma \rangle \longrightarrow \langle \rho[x \mapsto [\![t]\!]_I], q, \sigma \rangle$.

With these provisos, we can make the role of the two kinds of choices precise and define when a nondeterministic program accepts an input:

▶ **Definition 16.** A nondeterministic program *Prg accepts* an (L, S)-model M if for all I there exists O with $Prg \vdash_M I \longrightarrow O$ and $O(\mathit{result}) = \texttt{true}$. It rejects M if for all I and for all O with $Prg \vdash_M I \longrightarrow O$ one has $O(\mathit{result}) = \texttt{false}$.

Thus, in the positive case, M must find, for all traversals of the `forall`-loops, appropriate nondeterministic choices leading to result `true`. In the negative case, however, the program must yield result `false` no matter how the nondeterministic choices are made and how the `forall`-loops are being traversed.

Note that for programs without `choose`, this definition agrees with the one for PURPLE above. For programs without `forall`-loop it agrees with the definition of nondeterminism.

In [15] we have shown that PURPLE can evaluate formulae in DTC-logic. One may expect that with nondeterminism, this result generalises to TC-logic. We obtain the following proposition. Recall that any relational structure M can be understood as a pointer structure.

▶ **Lemma 17.** *For each closed* TC*-formula φ on a relational signature Σ, there exists a program P_φ in* PURPLE $_{c,nd}$ *such that, for any Σ-structure of Σ, $M \models \varphi$ holds iff P_φ recognises M.*

4.3.1 Tree isomorphism

The problem of determining whether rooted, directed trees (unlabeled, without edge labeling) are isomorphic is in LOGSPACE.

▶ **Theorem 18** ([17]). *Tree isomorphism is in* LOGSPACE.

The proof uses an algorithm to canonize such trees. The algorithm uses counting, and depth first search for which it relies on the edge ordering implicit in the encoding of the input tree. Etessami and Immerman [8] were able show that transitive closure logic with counting is unable to define tree isomorphism and thus confirmed that such ordering is intrinsically necessary. The same counterexample albeit with a rather different proof technique shows that PURPLE with counting and nondeterminism cannot decide tree isomorphism either.

▶ **Theorem 19** ([13]). PURPLE $_{c,nd}$ *cannot decide tree-isomorphism.*

Proof idea. The family of trees (AB-trees) used by Etessami and Immerman comprises two non-isomorphic tree structures which are defined by mutual induction to contain a large number of isomorphic subtrees. For any two non-isomorphic trees of the same height, say A_h and B_h, every immediate subtree is one of C_h, D_h or E_h. The only difference between A_h and B_h is how many subtrees of each type are present, i.e., when the immediate subtrees

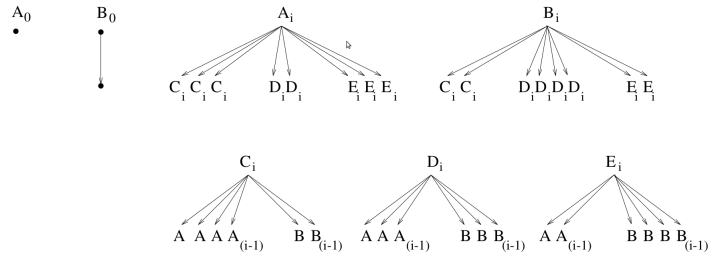


Figure 2 AB-trees

of A_h and B_h are grouped according to isomorphism, the cardinality of the groups differ (though the number of groups is the same). Unlike Etessami and Immerman who use a variant of Ehrenfeucht Fra issé games we rely on a simulation argument between two runs of a given PURPLE program on both A_h and B_h.

Intuitively, to differentiate between t_1 and t_2, PURPLE has to traverse each immediate subtree to group them according to isomorphism and determine the cardinality of each such group. So, reasoning recursively, this would need traversing the trees in a depth first manner. Intuitively, the `forall` -loop provides no additional strength beyond quantifiers here, since it is not required to enumerate the nodes in this order. The main work in the proof goes into showing this rigorously. With no edge ordering, remembering whether a subtree has been traversed or is yet to be traversed involves placing a pebble on the subtree. Given that it has a limited number of pebbles, this cannot be done for arbitrary depths. ◀

4.4 Recursion

PURPLE can be extended with procedures in the expected manner. We restrict the extension to Boolean functions which take a tuple of pointers as input, and allow mutual recursion but disallow global variables or side effects. The semantics of a (mutually) recursive function is defined in terms of least fixed points in the usual way. We do formally allow non-terminating functions but since the set of global states remains bounded such non-termination can be detected and thus we can assume w.l.o.g. that all procedures are total.

This extension does not subsume LOGSPACE since it is unable to count over discrete graphs (cf. Theorem 12).

▶ **Theorem 20.** *No* PURPLE *program with recursion can checking whether the input is a discrete graph with n nodes, where n is a power of two.*

Proof sketch. Non-pebbled nodes of discrete graph are indistinguishable, so the result of a function should be identical on them. This allows one to deduce a bound on recursion depth that is independent of the input size and thus reduce to Theorem 12. ◀

On the other hand, the fact that PURPLE with recursion can hold more than a constant number of pointers in memory albeit according to a stack discipline enables it to solve HORN.

▶ **Theorem 21.** PURPLE *with recursion can solve* HORN *and more generally evaluated formulas in LFP logic.*

5 Deterministic transitive closure logic

In the context of descriptive complexity theory deterministic transitive closure logic (DTC-logic) was introduced as a logical characterisation of LOGSPACE on ordered structures [16]. This logic is parametrized by a relational signature σ. Its syntax extends that of first-order logic with equality over the signature σ by a construct $\texttt{dtc}\varphi\vec{x}\vec{y}\vec{s}\vec{t}$ for deterministic transitive closure. The deterministic transitive closure of a binary relation R is the transitive closure of the relation $R_d = \{\langle x, y\rangle \mid xRy \wedge (\forall z.\, xRz \Rightarrow z = y)\}$ and the formula $\texttt{dtc}\varphi\vec{x}\vec{y}\vec{s}\vec{t}$ expresses that the pair $\langle \vec{s}, \vec{t}\rangle$ is in the deterministic transitive closure R_d of the binary relation on tuples that is defined by $\vec{x}R\vec{y} \iff \varphi(\vec{x}, \vec{y})$.

Note that R_d is a partial function in the sense that $R_d(x, y) \wedge R_d(x, y')$ implies $y = y'$ and that if R itself is a partial function then $R = R_d$.

Deterministic transitive closure logic captures LOGSPACE on finite structures with a total ordering, i.e. where σ contains a binary relation lt that is interpreted as a total ordering, see e.g. [4]. Informally, this is because with a total ordering one can do enough arithmetic in the logic to encode work-tapes of LOGSPACE Turing Machines and thus simulate the computation of such machines using the DTC-operator.

On unordered structures, however, DTC-logic is extremely weak, see [9].

An interesting, yet less studied, middle ground is DTC-logic on d-labelled graphs (called locally-ordered graphs in this context), where the edges emanating from any given node carry a linear order but the nodes themselves do not. This can be formally represented in a number of equivalent ways. On such inputs, DTC-logic can simulated (deterministic) JAGs by taking the transitive closure of the transition relation. On the other hand, PURPLE can evaluate DTC-formulas:

▶ Proposition 21.1. For each closed DTC-formula φ on locally ordered graphs there exists a program P_φ such that, for any finite locally ordered graph G, $G \models \varphi$ holds if and only if P_φ recognises G.

In order to evaluate quantifiers we use the `forall`-loop to search for witnesses or counter-examples. For transitive closure we first use `forall`-loops to find the (uniquely determined) successor of any tuple; the transitive closure can then be simulated using a while-loop.

The converse of this proposition is not true since the parity of the input size is not definable in DTC-logic [6]. The following direct consequence of Proposition 21.1 and Theorem 13 provides an answer to question left open by Etessami & Immerman [6].

▶ **Corollary 22** ([14]). USTCON *in locally-ordered graphs is not definable in* DTC*-logic.*

6 Conclusion and Future Works

Our starting point was the observation that LOGSPACE and NLOGSPACE algorithms operating on graph-like structures and then can hold a constant number of pointers into the input structure in memory. In addition, they can perform arithmetic up to the size of the input and, finally, can access the binary representation of the input nodes in the form of a fixed but arbitrary linear order.

We then considered that it might be possible to obtain "relativized" separation results if some of these features are removed by considering pointers as an abstract datatype.

We surveyed existing systems that are based on this idea namely jumping automata on graphs, transitive closure logic, and the PURPLE programming language. All of these systems are provably below PTIME but somewhat trivially so because they cannot solve USTCON which

lies in LOGSPACE. We thus considered various extensions of these systems with arithmetic, nondeterminism, iterators, recursion and examined the strength of the resulting systems.

In the light of these results, the original motivation of obtaining a "relativised" separation of LOGSPACE from PTIME has become somewhat elusive: most systems considered either coincide with LOGSPACE, NLOGSPACE, PTIME or there exists a LOGSPACE problem that can provably not be decided in them (USTCON, tree isomorphism) and can be reduced to HORN.

A possible way to address this, is to consider a weak system and add to it constructs that will solve LOGSPACE problems. For instance, to strengthen PURPLE with counting and nondeterminism over graphs with no edge ordering can be strengthened with recursion in tree-like structures in order to solve tree-isomorphism. Such a construct has been used in [11], but their motivation differs from ours. However, this might require us to add such constructs for many LOGSPACE complete problems since weaker systems would not necessarily capture the reductions between the different LOGSPACE complete problems.

Another option could be to look for a subset of HORN instances to which known LOGSPACE complete problems such as USTCON or tree isomorphism cannot be reduced yet still require a non-constant number of pebbles to be solved. Some initial progress has been made by restricting graph-theoretical parameters such as tree width of the graph induced by a HORN instance.

References

1. Daniel P. Bovet and Pierluigi Crescenzi. *Introduction to the theory of complexity*. Prentice Hall international series in computer science. Prentice Hall, 1994.
2. Stephen A. Cook and Charles Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM J. Comput.*, 9(3):636–652, 1980.
3. Stephen A. Cook and Ravi Sethi. Storage requirements for deterministic polynomial time recognizable languages. *J. Comput. Syst. Sci.*, 13(1):25–37, 1976.
4. H.D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995.
5. J. Edmonds, C. Poon, and D. Achlioptas. Tight lower bounds for st-connectivity on the nnjag model. *SIAM Journal on Computing*, 28(6):2257–2284, 1999.
6. K. Etessami and N. Immerman. Reachability and the power of local ordering. *Theoretical Computer Science*, 148(2):261–279, 1995.
7. Kousha Etessami and Neil Immerman. Reachability and the power of local ordering. *Th. Comp. Sci.*, 148(2):261–279, 1995.
8. Kousha Etessami and Neil Immerman. Tree canonization and transitive closure. In *IEEE Symp. Logic In Comput. Sci*, pages 331–341, 1995.
9. E. Grädel and G.L. McColm. On the power of deterministic transitive closures. *Information and Computation*, 119(1):129–135, 1995.
10. Erich Grädel and Gregory L. McColm. On the power of deterministic transitive closures. *Inf. Comput.*, 119(1):129–135, 1995.
11. Martin Grohe, Berit Grußien, André Hernich, and Bastian Laubner. L-recursion and a new logic for logarithmic space. In *CSL*, pages 277–291, 2011.
12. Martin Hofmann and Ramyaa Ramyaa. Power of nondetreministic jags on cayley graphs, 2013. arXiv, 835993.
13. Martin Hofmann, Ramyaa Ramyaa, and Ulrich Schöpp. Pure pointer programs and tree isomorphism. In Frank Pfenning, editor, *FoSSaCS*, volume 7794 of *Lecture Notes in Computer Science*, pages 321–336. Springer, 2013.
14. Martin Hofmann and Ulrich Schöpp. Pointer programs and undirected reachability. In *LICS*, pages 133–142, 2009.

15 Martin Hofmann and Ulrich Schöpp. Pure pointer programs with iteration. *ACM Trans. Comput. Log.*, 11(4), 2010.

16 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, October 1988.

17 Steven Lindell. A logspace algorithm for tree canonization (extended abstract). STOC '92, pages 400–404, New York, NY, USA, 1992. ACM.

18 Lu, Zhang, Poon, and Cai. Simulating undirected *st*-connectivity algorithms on uniform jags and nnjags. In *Algo. and Comp.*, volume 3827 of *LNCS*. 2005.

19 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

20 Ulrich Schöpp. A formalised lower bound on undirected graph reachability. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *LPAR*, volume 5330 of *Lecture Notes in Computer Science*, pages 621–635. Springer, 2008.

On Approximation Resistance of Predicates

Subhash Khot

Department of Computer Science
Courant Institue of Mathematical Sciences
New York University
251 Mercer Street
New York NY-10012, USA.
khot@cs.nyu.edu

Abstract

Constraint satisfaction problems are some of the most well-studied NP-hard problems, 3SAT being a prominent example. It is known by Hastad's 1997 result that 3SAT is "approximation resistant" in the following sense: given a near-satisfiable instance, a trivial algorithm that assigns random boolean values to the variables satisfies 7/8 fraction of the constraints and no efficient algorithm can do strictly better unless P=NP!

3SAT is a CSP that corresponds to the ternary OR predicate. In general, a CSP has constraints given by some fixed predicate $P : \{0,1\}^k \mapsto \{\text{True}, \text{False}\}$ (on possibly negated variables) and the predicate is called approximation resistant if, on a near-satisfiable instance, it is computationally hard to perform strictly better than a random assignment.

The quest to understand approximation resistance has played a central role in the theory of probabilistically checkable proofs (PCPs) and hardness of approximation. This talk will give a survey of the topic, including recent work giving a complete characterization of approximation resistance (i.e. a necessary and sufficient condition on the predicate that makes the corresponding CSP approximation resistant).

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Approximation resistance, Hardness of approximation, Probabilistically checkable proofs, Constraint satisfaction problems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.19

Category Invited Talk

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 19–19

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

Characterisations of Nowhere Dense Graphs

Martin Grohe[1], Stephan Kreutzer[2], and Sebastian Siebertz[2]

1 **RWTH Aachen University**
grohe@informatik.rwth-aachen.de
2 **Technical University Berlin**
{stephan.kreutzer,sebastian.siebertz}@tu-berlin.de

Abstract

Nowhere dense classes of graphs were introduced by Nešetřil and Ossona de Mendez as a model for "sparsity" in graphs. It turns out that nowhere dense classes of graphs can be characterised in many different ways and have been shown to be equivalent to other concepts studied in areas such as (finite) model theory. Therefore, the concept of nowhere density seems to capture a natural property of graph classes generalising for example classes of graphs which exclude a fixed minor, have bounded degree or bounded local tree-width. In this paper we give a self-contained introduction to the concept of nowhere dense classes of graphs focussing on the various ways in which they can be characterised. We also briefly sketch algorithmic applications these characterisations have found in the literature.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graph Algorithms, Algorithmic Graph Structure Theory, Finite Model Theory, Nowhere Dense Classes of Graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.21

Category Invited Talk

1 Introduction

Structural graph theory has proved to be a powerful tool for coping with computational intractability. It provides a wealth of concepts and results that can be used to design efficient algorithms for hard computational problems on specific classes of graphs that occur naturally in applications. Examples include polynomial-time (in fact, fixed-parameter linear) algorithms for problems such as computing dominating sets, independent sets, Hamiltonian cycles, 3-colourings and many other problems on classes of graphs of *bounded tree-width*. See e.g. [6, 8, 7, 5, 4] for surveys on tree-width and the huge number of algorithmic applications. Other examples are *polynomial-time approximation schemes* (PTAS) for problems such as vertex cover, dominating sets or Steiner-forests on planar graphs or, more generally, on graph classes of bounded genus or which exclude a fixed graph as a minor [3, 58, 29, 36, 14].

In their monumental work on graph minors [56], Robertson and Seymour developed a very powerful structure theory for classes of graphs excluding a fixed minor which has found a large number of algorithmic consequences, for instance to constant-factor approximations of colouring problems (see e.g. [19, 20]), to polynomial-time approximation schemes ([36, 19, 14, 21]) or for general parameterized algorithms for problems such as dominating sets and many other in form of bidimensionality theory developed in [17] and subsequent papers. See e.g. [16, 18].

Excluding a fixed graph H as a minor yields classes of graphs for which topological methods can be employed to obtain efficient algorithms for computational problems. A

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 21–40

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

different approach is taken in classes of bounded degree, where topology does not play a decisive rôle in the development of algorithms but other approaches succeed in obtaining good algorithms for various problems on input graphs of maximum degree d, for some constant d. Graph classes of bounded degree can be generalised further to classes of *bounded local tree-width* [30, 31], a property where we do not require that the entire graph has small tree-width, but only that every r-neighbourhood of a vertex in the graph has tree-width bounded by some function of its radius r. Again, bounded local tree-width and excluded minors are incomparable concepts.

Whereas many papers provide optimised algorithms for specific, individual problems on certain classes of graphs, another line of research aims at developing general tractability results for a whole and natural class of problems on special classes of inputs. One example is bidimensionality theory as outlined above. Many other examples of such general tractability results, often referred to as *algorithmic meta-theorems*, use definability of computational problems in logical languages to obtain natural classes of problems. The best-known of these results is Courcelle's theorem [12] stating that every algorithmic problem definable in *monadic second-order logic* can be decided in linear time on classes of graphs of bounded tree-width. This includes many common algorithmic problems such as Hamiltonicity, 3-Colourability and many covering problems such as dominating sets. Following Courcelle's result, meta-theorems of various forms have been developed, see e.g. [9, 57, 33, 34, 13, 28] and the surveys [37, 38, 45].

In general, the main goals of this whole line of research described so far, sometimes referred to as *algorithmic graph structure theory*, are the following: we want to understand for natural and important classes of graphs what kind of problems can be solved efficiently on these graphs and to develop the corresponding graph structural and algorithmic techniques; for natural classes of problems we want to understand their general tractability frontier, i.e. the "most general" classes of graphs on which these problems become tractable.

In particular the last aspect has been pursued intensively in research on algorithmic meta-theorems with a quest for finding the largest classes of graphs where problems definable in first-order logic become tractable. First-order definable problems define a natural class of problems including dominating sets, vertex covers, network centres and many others.

As diverse as the examples above of graph classes with a rich algorithmic theory may appear, a feature all these classes have in common is that they are relatively *sparse*, i.e. graphs in these classes have a relatively low number of edges compared to the number of vertices. In fact, classes of graphs excluding a fixed minor can only have a linear number of edges. This suggests that this "sparsity" might be an underlying reason why many problems can be solved efficiently on these classes of graphs, even though they otherwise do not have much in common. This leads to the question how to define a reasonable concept of "sparse classes of graphs".

A first idea to capture the concept of "sparse" classes of graphs is to bound the average degree, i.e. to study classes $\mathcal{C}$ of graphs such that for all $G \in \mathcal{C}$, $\frac{|E(G)|}{|V(G)|} \leq d$ for some constant d. However, given any graph G of order $n := |V(G)|$, we can bound its average degree by "padding", i.e. simply by adding n^2 isolated vertices. While this reduces the average degree below 2, for many problems it does not significantly change the structure of the graph. For instance, adding extra isolated vertices does not really change the problem of evaluating first-order formulas. Hence, this notion is not a satisfactory measure for sparsity in general. Therefore, to prevent padding arguments of this form, we may want to require that sparse graphs remain sparse if we take a subgraph, i.e. that sparse graphs do not have dense subgraphs.

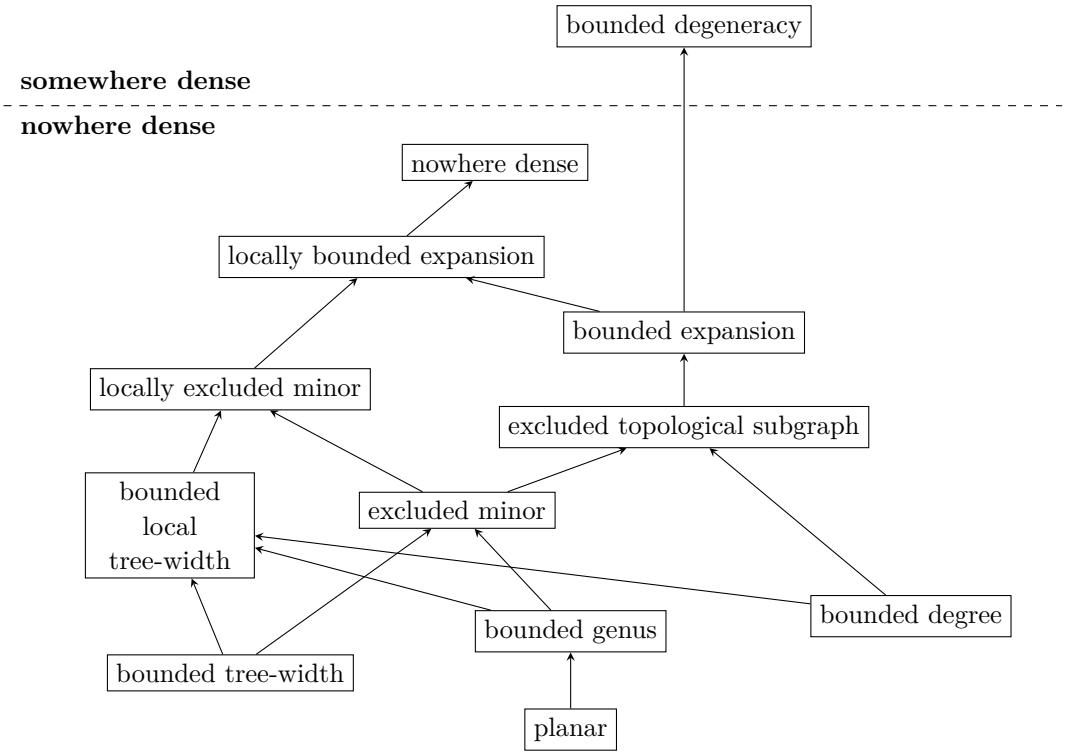


Figure 1 Sparse graph classes.

This leads to the well-studied concept of *degeneracy*. A graph G is d-degenerate if every subgraph of G contains a vertex of degree at most d. In particular, this means (and in fact degeneracy is equivalent to saying) that the average degree of every subgraph is bounded by a constant. But again this concept is not universally satisfactory as we can make every graph 2-degenerate by *subdividing* every edge once. Here, by subdividing an edge we mean the operation of replacing an edge $\{u, v\}$ by a path of length 2. Again, for various problems such as evaluating first-order formulas, this does not change the nature of the graph significantly. Hence, in addition to closure under subgraphs, we may want to require of our notion of sparsity that it should be invariant under such subdivisions or local modifications. The two requirements together exactly yield the concept of *nowhere dense* classes of graphs introduced by Nešetřil and Ossona de Mendez [52]. See [49] for an extensive study of sparse graphs. We defer a formal definition to Section 2, see Definition 2.5.

It turns out that nowhere dense classes of graphs can equivalently be characterised in many different ways which at first sight have very little in common. For instance, nowhere dense classes of graphs can equivalently be characterised by the concept of *uniformly quasi-wideness* (see Section 6), a concept studied in finite model theory; by the existence of *low tree-depth colourings* (see Section 4); by generalised colouring numbers (see Section 3); by a game characterisation (see Section 7.2); or by the model-theoretic concept of *independence*, see [1]. See the individual sections for references. This shows that the concept of nowhere dense classes of graphs is very robust and seems to capture a natural property of graph classes arising independently in many diverse contexts.

By construction, nowhere dense classes of graphs contain all examples of special graph classes mentioned above and thereby form a very general type of graph classes. Figure 1 shows the containment of the various classes mentioned so far. In particular, nowhere dense

classes unify the two incomparable concepts of bounded degree or bounded local tree-width on the one hand and excluded minors or excluded topological subgraphs on the other hand. They therefore provide a unifying framework in which to study sparse classes of graphs.

Following their introduction, nowhere dense classes of graphs have been studied also with algorithmic applications in mind. The fact that they can be characterised in many different ways also has very nice algorithmic consequences, as each characterisation yields different algorithmic techniques. For instance, it has been shown that problems such as network centres and dominating sets can be solved by fixed-parameter algorithms on nowhere dense classes of graphs using uniformly quasi-widesness [15] (see Section 6). Using low tree-depth colourings, Nešetřil and Ossona de Mendez [50] showed that the subgraph isomorphism or homomorphiosm problem is fixed-parameter tractable on nowhere dense classes. Furthermore, using the same characterisation, Gajarsky et al. [35] extended the meta-kernelisation framework of [10] to nowhere dense classes of graphs providing polynomial kernels for a large number of algorithmic problems (see Section 4). On an important subclass of nowhere dense graphs, called classes of *bounded expansion* (see Definition 2.8), even more algorithmic applications are known, for instance in database query answering and enumeration [42], which relies on the concept of augmentations (see Section 5), or in approximating dominating sets [25], which is based on generalised colouring numbers (see Section 3).

As mentioned above, in a series of papers starting in the 1990s researchers have tried to investigate the largest classes of graphs on which all first-order definable graph properties can be decided efficiently, or more formally, on which first-order model checking is fixed-parameter tractable. It turns out that for classes of graphs closed under subgraphs, this limit are exactly nowhere dense classes of graphs. For, it was shown in [45] and [27] that on classes of graphs which are not nowhere dense but closed under subgraphs first-order model-checking cannot be fixed-parameter tractable unless FPT = W[1], a consequence widely believed to be false in the parameterized complexity community (see e.g. [24, 32]). On the other hand, very recently it was shown by the authors of this paper that on nowhere dense classes of graphs, first-order model-checking is fixed-parameter tractable [39]. Hence, this is another indication that nowhere dense classes of graphs form a natural limit for certain types of algorithmic problems to be solved efficiently.

As the exposition above already indicates, the appeal of nowhere dense classes and their applications relies on the fact that they are characterised in many different ways. In this paper we provide a self-contained introduction to nowhere dense classes with a strong focus on their different characterisations. In each of the individual sections of this paper we will present some characterisation of nowhere dense classes of graphs, prove its equivalence to other characterisations, state their main properties and sketch some algorithmic consequences.

Notation. We use standard notation in graph theory and refer, e.g., to [23] for details. In particular, we write $d(G)$ for the average degree of a graph G, $\delta(G)$ for the minimum degree and $\Delta(G)$ for its maximum degree. $\Delta^-(\vec{G})$ denotes the maximum in-indegree of a directed graph $\vec{G}$.

2 Sparse graphs

As motivated in the introduction, we want to find the largest number of edges an n-vertex graph can have such that we still find structure that can be used algorithmically. This is closely related to one of the classical questions of extremal graph theory: given a graph H, what is the maximum number of edges in an n-vertex graph that does not contain H as a

subgraph? This number is known as the Turán number $ex(n, H)$ of H and Turán determined the exact value of $ex(n, K_t)$ for the complete graph on t vertices, K_t. For algorithmic purposes often a more interesting restriction is to exclude a graph as a topological subgraph. A graph H is a *subdivision* of a graph H' if H can be obtained from H' by replacing edges by vertex disjoint paths. H is a *topological subgraph* or *topological minor* of G, denoted $H \preceq^t G$, if a subdivision of H is isomorphic to a subgraph of G. Mader was one of the first who considered Turán's question for topological subgraphs.

▶ **Theorem 2.1** (Mader [46]). *Given $t \in \mathbb{N}$, there exists a constant c_t depending only on t such that every graph G on n vertices with at least $c_t n$ edges contains a subdivision of the complete graph K_t.*

Bollobás and Thomason [11] and independently Komlós and Szemerédi [44] showed that $c_t \leq ct^2$ for some absolute constant c, hence every graph with average degree at least ct^2 contains a subdivision of K_t. More generally, a graph that contains a subgraph with average degree at least ct^2 contains a subdivision of K_t.

Recent results show that even less structural information suffices to solve many problems efficiently. A graph H' is an *r-subdivision* of a graph H if H' can be obtained from H by replacing edges by vertex disjoint paths of length at most $r+1$. H is a *topological depth-r subgraph* or *topological depth-r minor* of a graph G, denoted $H \preceq_r^t G$, if an r-subdivision of H is isomorphic to a subgraph of G. In the proofs of the above results, the edges of the complete graphs that are found as topological subgraphs are subdivided by more than a constant number of vertices. It was hence the next step to ask how many edges an n-vertex graph that does not contain an r-subdivision of a complete graph K_t can maximally have. A first result in this direction is implied by a result of Alon, Krivelevich and Sudakov.

▶ **Theorem 2.2** (Alon, Krivelevich, Sudakov [2]). *Let $t \in \mathbb{N}$ and $\epsilon \geq 1/2$. Every graph G on n vertices with at least $\frac{t^2}{2} n^{1+\epsilon}$ edges contains a 2-subdivision of K_t.*

On the other hand, there are well known classes of graphs of girth at least g and $\Omega(n^{1+1/g})$ edges. Any c-subdivision of K_t, where $t \geq 3$, must contain a cycle of length at most $3c$. Hence there are n-vertex graphs with $\Omega(n^{1+1/(3c+1)})$ edges and no c-subdivision of K_t. Dvořák [26] and Jiang [41] independently answered the question for $0 < \epsilon < 1/2$ by showing the following.

▶ **Theorem 2.3** (Dvořák [26], Jiang [41]). *Given $t \in \mathbb{N}$ and $\epsilon > 0$. There exists $n_0 = n_0(t, \epsilon)$ and $c = c(\epsilon)$ such that all graphs G with $n \geq n_0$ vertices and at least $n^{1+\epsilon}$ edges contain a c-subdivision of the complete graph K_t.*

Jiang [41] provides the best bound for the constant $c(\epsilon)$ known today, $c(\epsilon) \leq \lfloor 10/\epsilon \rfloor$. If we apply this result to infinite classes of graphs, we obtain an interesting dichotomy in terms of edge densities. Note that for every graph G with at least one edge and for all $\epsilon \geq 0$,

$$|E(G)| = |V(G)|^{\epsilon} \iff \frac{\log |E(G)|}{\log |V(G)|} = \epsilon.$$

▶ **Corollary 2.4.** *Let $\mathcal{C}$ be an infinite class of graphs. Then either for all $r \in \mathbb{N}$*

$$\lim_{n\to\infty} \sup \left\{ \frac{\log |E(H)|}{\log |V(H)|} \;\middle|\; G \in \mathcal{C} \text{ with } |V(G)| \geq n, H \preceq_r^t G \right\} \leq 1 \tag{2.1}$$

or there exists $r \in \mathbb{N}$ with

$$\lim_{n\to\infty} \sup \left\{ \frac{\log |E(H)|}{\log |V(H)|} \;\middle|\; G \in \mathcal{C} \text{ with } |V(G)| \geq n, H \preceq_r^t G \right\} = 2. \tag{2.2}$$

Here we take $\frac{\log|E(H)|}{\log|V(H)|}$ *to be* $-\infty$ *if* $E(H)=\emptyset$.

Proof. Note that the supremum always exists, because $\frac{\log|E(H)|}{\log|V(H)|} \leq 2$ for all H. Furthermore, observe that if an r-subdivision of H_1 is a subgraph of G and an s-subdivision of H_2 is a subgraph of H_1, then an $((r+1)s+1$-subdivision of H_2 is a subgraph of G. Assume that for some $r \in \mathbb{N}$, $\lim_{n\to\infty} \sup\left\{ \frac{\log|E(H)|}{\log|V(H)|} \;\middle|\; G \in \mathcal{C} \text{ with } |V(G)| \geq n, H \preceq_r^t G \right\} = 1+2\epsilon$ for some $0 < \epsilon < \frac{1}{2}$. Then there are infinitely many n-vertex graphs H with at least $n^{1+\epsilon}$ edges such that an r-subdivision of H is a subgraph of some $G \in \mathcal{C}$. Let $\mathcal{C}'$ be the class of those graphs. By Theorem 2.3, for all $t \in \mathbb{N}$ there exists $n_0(t,\epsilon)$ and $s := c(\epsilon)$ such that all graphs in $\mathcal{C}'$ contain an s-subdivision of K_t. By our above observation, $\mathcal{C}$ contains $(r+1)s+1$-subdivisions of arbitrary large complete graphs. Then the above limit goes to 2 for $(r+1)s+1$. ◀

The previous corollary was proved by Nešetřil and Ossona de Mendez [52] who showed that the limits defined there form a trichotomy, i.e. that for all classes $\mathcal{C}$ of graphs the lim sup can only take the values $\{0,1,2\}$. Those classes for which the limits is ≤ 1 are called nowhere dense.

▶ **Definition 2.5.** Let $\mathcal{C}$ be a class of graphs. $\mathcal{C}$ is *nowhere dense*, if

$$\lim_{n\to\infty} \sup\left\{ \frac{\log|E(H)|}{\log|V(H)|} \;\middle|\; G \in \mathcal{C} \text{ with } |V(G)| \geq n, H \preceq_r^t G \right\} \leq 1.$$

Otherwise $\mathcal{C}$ is called *somewhere dense*.

We can rephrase the definition in the following ways.

▶ **Corollary 2.6.** *A class* $\mathcal{C}$ *of graphs is nowhere dense if, and only if, for all* $r \in \mathbb{N}$ *and all* $\epsilon > 0$ *there is* $n_0(r,\epsilon)$ *such that all* n*-vertex graphs* $H \preceq_r^t G \in \mathcal{C}$ *with* $n \geq n_0$ *vertices satisfy* $|E(H)| \leq n^{1+\epsilon}$.

▶ **Corollary 2.7.** *A class* $\mathcal{C}$ *is nowhere dense if and only if there is a function* f *such that for all* $r \in \mathbb{N}$ *we have* $K_{f(r)} \not\preceq_r^t G$ *for all* $G \in \mathcal{C}$.

The original interest of Nešetřil and Ossona de Mendez when studying sparse graph classes was the following subclass of nowhere dense classes [47, 53, 48].

▶ **Definition 2.8.** A class $\mathcal{C}$ has bounded expansion if for every r there exists $n_0(r)$ and $c(r)$ such that all n-vertex graphs $H \preceq_r^t G \in \mathcal{C}$ with $n \geq n_0$ vertices satisfy $|E(H)| \leq c \cdot n$.

▶ Remark. Nowhere dense classes and classes of bounded expansion were originally defined in terms of excluded depth-r minors and not in terms of excluded r-subdivisions. In general, minors and topological subgraphs behave quite different. Surprisingly, densities of bounded depth minors and bounded depth topological subgraphs are strongly related. In our presentation we will always work with topological subgraphs and hence we have defined nowhere dense classes accordingly.

In the rest of this section we present a proof of Theorem 2.3. Our presentation follows [49]. A well-known lemma from graph theory states that any graph with high average degree contains a subgraph of high minimum degree.

▶ **Lemma 2.9.** *Let* G *be a graph and* $1/|V(G)| < \epsilon \leq 1$. *Then* G *has a subgraph* H *with*

$$\delta(H) \geq (1-\epsilon)\frac{|E(G)|}{|V(G)|} \qquad \text{and} \qquad |E(H)| \geq \epsilon \cdot |E(G)|.$$

The next lemma will be used to show that in graphs of large minimum degree d we can find a 1-subdivision of H of only slightly smaller minimum degree and such that we can carefully control the order of H. We will use this to show that when we express the minimum degree relative to the order of H, it will in fact grow.

▶ **Lemma 2.10.** *Let A be an n-element set and let $A_1, \dots, A_n \subseteq A$ be a collection of sets of size at least d such that $A = \bigcup_{1 \leq i \leq n} A_i$. Let $d \leq s \leq n$. Then there exists a set $S \subseteq A$ of size s such that $|A_i \cap S| \geq \lfloor s \cdot d/n \rfloor$ for at least $n/2$ of the A_i.*

Proof. We may assume without loss of generality that every set A_i has exactly d elements. For each i, the number of subsets of A of size s including exactly k elements of A_i is $\binom{n-d}{s-k}\binom{d}{k}$ (choose k elements of A_i and $s-k$ elements of $A \setminus A_i$). Hence the number of subsets of A of size s including at least k elements from A_i is $\sum_{k \leq \ell \leq s} \binom{n-d}{s-\ell}\binom{d}{\ell}$.

For $k \leq d$ consider the bipartite graphs G_k, where one part consists of the s-element subsets of A and the other part consists of the sets A_i. We add an edge (B, A_i) if and only if $|B \cap A_i| \geq k$. The degree of B in G_k corresponds to the number of A_i that B shares at least k elements with. As observed above, every A_i has degree $\sum_{k \leq \ell \leq s} \binom{n-d}{s-\ell}\binom{d}{\ell}$. Hence

$$|E(G_k)| = n \cdot \sum_{k \leq \ell \leq s} \binom{n-d}{s-\ell}\binom{d}{\ell}.$$

The average degree of a vertex B in G_k is $\frac{n \cdot \sum_{k \leq \ell} \binom{n-d}{s-\ell}\binom{d}{\ell}}{\binom{n}{s}}$ and there must be one vertex S with at least this degree.

Observe that $\frac{\binom{n-d}{s-k}\binom{d}{k}}{\binom{n}{s}}$ is the probability mass function of a hypergeometric distribution with mean $\frac{ds}{n}$. Hence for $k = \lfloor \frac{ds}{n} \rfloor$ the above sum is greater than $1/2$ and hence $d(S) \geq n/2$ for this k. We conclude that $|A_i \cap S| = k = \lfloor \frac{ds}{n} \rfloor$ for at least $n/2$ of the A_i. ◀

We now show how to find a subgraph with larger minimum degree with respect to its order.

▶ **Lemma 2.11.** *Let $\rho > 1$. There exists $n_0(\rho)$ such that for all graphs G on $n \geq n_0$ vertices with minimum degree $\delta(G) \geq n^{1/\rho}$ there exists a graph H such that a 1-subdivision of H is a subgraph of G and either*

- *H is a complete graph of order $n^{1/(3\rho^2)}$, or*
- *$\delta(H) \geq |V(H)|^{1/(\rho - 1/2)}$ and $|V(H)| \geq \sqrt{n/6}$.*

Proof. Let $\mu := 1/\rho$ and $s := n^{1-\mu+\mu^2/3}$. For each vertex $a_i \in V(G)$ let A_i be the set of neighbours of a_i. Then every set A_i has at least n^μ many vertices, $\bigcup_{1 \leq i \leq n} A_i = A$ and $n^\mu \leq s \leq n$. By Lemma 2.10 there exists a subset $S \subseteq A$ of size s and a set $T' \subseteq V(G)$ of size at least $n/2$ such that every vertex in T' has at least $n^\mu \cdot n^{1-\mu+\mu^2/3}/n = n^{\mu^2/3}$ neighbours in S. Let $T := T' \setminus S$. T has size at least $t := n/2 - s$. Enumerate the vertices of T as $t_1, \dots, t_t$. We construct a sequence $H_0 \subseteq H_1 \subseteq \dots$ of graphs, where H_0 is the empty graph on vertex set S. Assume that graph H_i has already be constructed. Consider all neighbours of t_{i+1} in S. If they induce a complete graph in H_i define H to be this complete graph of order $n^{\mu^2/3} = n^{1/(3\rho^2)}$. Otherwise, we add the edge e_{i+1} to H_{i+1} between two arbitrary not yet adjacent neighbours u, v of t_{i+1} and associate with this edge the path u, t_{i+1}, v of length 2 and continue the construction. After t steps, we define $H' := H_t$. Then $|E(H')| = t$.

Let $d := n^{\mu-\mu^2/3} > 2$ and $\epsilon := (1-1/d)/(2-1/d)$ (hence $1/n < 1/2 < \epsilon < 1$). Note that $n/d = s$. By Lemma 2.9, H' has a subgraph H with

$$\delta(H) \geq (1-\epsilon)\frac{|E(H')|}{|V(H')|} = \frac{1}{2-1/d} \cdot \frac{n/2 - n/d}{n/d} = \frac{1}{2-1/d} \cdot (2d-1) = d = n^{\mu-\mu^2/3}$$

and

$$\begin{aligned}|E(H)| \geq \epsilon \cdot |E(H')| &= \frac{1-1/d}{2-1/d} \cdot (n/2 - n/d) = (1-1/d) \cdot \frac{d}{2d-1} \cdot \frac{d-2}{2d} \cdot n \\ &= (1-1/d) \cdot \frac{d-2}{4d-2} \cdot n \geq (1-1/d) \cdot \frac{n}{10}.\end{aligned}$$

We conclude that

$$|V(H)| \geq \sqrt{2|E(H)|} \geq \sqrt{(1-1/d)\frac{n}{5}}.$$

For sufficiently large n, $1/d \leq 1/6$ and hence $|V(H)| \geq \sqrt{n/6}$.

On the other hand, as $H \subseteq H'$, we have $|V(H)| \leq n^{1-\mu+\mu^2/3}$, and hence $n \geq |V(H)|^{\frac{1}{1-\mu+\mu^2/3}}$. Then

$$\delta(H) \geq |V(H)|^{\frac{\mu-\mu^2/3}{1-\mu+\mu^2/3}} \geq |V(H)|^{\mu+\frac{\mu^2}{2}}.$$

We conclude the proof by observing that $\left(\frac{1}{\rho} + \frac{1}{2\rho^2}\right)^{-1} = \rho - \frac{1}{2} + \frac{1/2}{2\rho+1} > \rho - \frac{1}{2}$. ◀

We are ready to take the final step.

▶ **Lemma 2.12.** *Let $\rho > 1$. There exists $n_0 = n_0(\rho)$ and $\mu = \mu(\rho) > 0$ such that for all graphs G on $n \geq n_0$ vertices with minimum degree at least $n^{1/\rho}$ we find a complete graph of order n^μ as a 9^ρ subdivision of G.*

Proof. We construct a sequence of graphs $G_0, G_1, \ldots, G_k$ such that for each $0 \leq i \leq k$ the graph G_i has order n_i and minimum degree at least $n_i^{1/(\rho-i/2)}$ as follows. Let $G_0 := G$. Iteratively, for each $i \geq 0$, if G_i is not a complete graph we apply Lemma 2.11 to G_i. We get a graph H_i whose 1-subdivision is a subgraph of G_i. If H_i is a 1-subdivision of a complete graph we stop. Otherwise we let $G_{i+1} := H_i$. The process stops after $k \leq 2\rho$ iterations because of the increase of $\delta(G_i)$. We have $n_{i+1} \geq \sqrt{n_i/6}$ and hence $n_{k-1} \geq \frac{1}{6}\left(\frac{n}{6}\right)^{2^{-2\rho}}$. At the next step we find a complete subgraph of size at least $n_{k-1}^{1/(3\rho^2)}$ and we let $\mu > 0$ such that for any $k \leq 2\rho$ we have $n_k \geq n^\mu$. Now every 1-subdivision of a k-subdivision is a $2k+1$ subdivision of the original graph. For simplicity we treat it as a $3k$ subdivision. Hence we find G_k as a $3^{2\rho} = 9^\rho$-subdivision of G. ◀

3 Generalised colouring numbers

As explained in the introduction, degeneracy is another concept to describe sparse graphs. Degeneracy gives rise to an ordering of the vertices of a graph with nice properties in a very natural way. Let d be the degeneracy of G. By induction on the number of vertices we construct an order $v_1 < \ldots < v_n$ such that every vertex v_i has at most d neighbours in $\{v_1, \ldots v_{i-1}\}$. In G, there is a vertex v which has at most d neighbours. $G - v$ is also

d-degenerate and has $n-1$ vertices. By induction, we have an order $v_1 < \ldots < v_{n-1}$ such that every v_i has at most d neighbours in $\{v_1, \ldots, v_{i-1}\}$. Adding v as the largest element to this order gives us an order with the desired properties. This order can for example be used to compute a vertex colouring of $V(G)$ with at most d colours in linear time.

When characterising nowhere dense classes in terms of degeneracy, we have to talk about the degeneracy of topological depth-r subgraphs. For example the class of 1-subdivisions of complete graphs is a class of degeneracy 2 but it is dense at depth 1. The aim of this section is to find a measure which generalises degeneracy and which allows to state that a class is nowhere dense if and only if this measure is bounded by n^ϵ for every sufficiently large n-vertex graph from $\mathcal{C}$. Such generalisations were found by Kierstead and Yang [43] which they called the generalised colouring numbers of a graph. Their theorem is weaker than what they actually proved, as they were not aware of the depth-r minor terminology. Zhu [60] formulated their theorem in terms of topological depth-r subgraphs. The following presentation follows Kierstead and Yang.

For a graph G, let $\Pi(G)$ be the set of all linear orderings of the vertices of G. For $\leq \in \Pi(G)$ and $x, y \in V(G)$, we say that x is *weakly k-reachable*[1] from y if there is a path of length $0 \leq \ell \leq k$ from y to x such that x is the smallest vertex with respect to the ordering. Let $\mathrm{WReach}_k[G, \leq, y]$ be the set of vertices that are weakly k-reachable from y with respect to the ordering. If furthermore, all internal nodes of the path are larger than y in the ordering, then x is called *strongly k-reachable* from y. Let $\mathrm{SReach}_k[G, \leq, y]$ be the set of vertices that are strongly k-reachable from y with respect to the ordering. The *weak k-colouring number* $\mathrm{wcol}_k(G)$ of G is defined as

$$\mathrm{wcol}_k(G) = \min_{L \in \Pi(G)} \max_{v \in V(G)} [\mathrm{WReach}_k(G, \leq, v)]$$

and the *k-colouring number* $\mathrm{col}_k(G)$ of G is defined as

$$\mathrm{col}_k(G) = \min_{L \in \Pi(G)} \max_{v \in V(G)} [\mathrm{SReach}_k(G, \leq, v)].$$

The aim of this section is to present a proof of the following theorem which follows from Kierstead and Yang's result [43] and Zhu's result [60].

▶ **Theorem 3.1.** *A class $\mathcal{C}$ of graphs is nowhere dense if and only if for every $\epsilon > 0$ there is $n_0 = n_0(r, \epsilon)$ such that $\mathrm{wcol}_r(G') \leq n^\epsilon$ for all n-vertex subgraphs $G' \subseteq G$ of a graph $G \in \mathcal{C}$ with $n \geq n_0$.*

The direction from right to left is easy to see. We show that an $r-1$-subdivision G of a complete graph K_t satisfies $\mathrm{wcol}_r(G) \geq t-1$. To see this, fix any ordering of $V(G)$. Let v be the largest vertex with respect to the ordering that corresponds to a vertex of K_t. For every other vertex w which corresponds to a vertex of K_t, v weakly k-reaches either w or some subdivision vertex on the path of length at most $r-1$ between v and w. Hence if $\mathcal{C}$ is somewhere dense, i.e., it contains arbitrary large complete graphs as $r-1$-subdivisions for some r, then the weak colouring numbers are too large.

To prove the other direction, we first show the following connection between weak-colouring number and colouring number.

▶ **Theorem 3.2** (Kierstead, Yang [43])**.** *Let G be a graph. Then $\mathrm{col}_k(G) \leq \mathrm{wcol}_k(G) \leq \mathrm{col}_k(G)^k$.*

[1] Note that weak k-reachability is also known as weak k-accessibility and strong k-reachability is known as k-accessibility in the literature.

Proof. The first inequality clearly holds. For the second inequality let $L \in \Pi(G)$. We show by induction on k that

$$\max_{v \in V(G)} |\mathrm{WReach}_k[G, \leq, v]| \leq (\max_{v \in V(G)} |\mathrm{SReach}_k[G, \leq, v]|)^k.$$

For the base step $k = 1$, observe that $|\mathrm{WReach}_1[G, \leq, v]| = |\mathrm{SReach}_1[G, \leq, v]|$. (Note also that the degeneracy of a graph is equal to $\mathrm{wcol}_1(G) + 1 = \mathrm{col}_1(G) + 1$). Let $k > 1$ and $y \in V(G)$. For each $x \in \mathrm{WReach}_k[G, \leq, y]$ let P_{xy} be a shortest $x - y$-path such that every vertex $z \in V(P)$ satisfies $x \leq_L z$. If $x \neq y$, let w be the first vertex on P_{xy} such that $w <_L y$ and let i be the distance from y to w. Then $w \in \mathrm{SReach}_i[G, \leq, y] - \mathrm{SReach}_{i-1}[G, \leq, y]$ and $x \in \mathrm{WReach}_{k-i}[G, \leq, w]$. It follows that

$$\begin{aligned}
&|\mathrm{WReach}_k[G, \leq, y]| \\
&\leq 1 + \sum_{i=1}^{k} \big|\mathrm{SReach}_i[G, \leq, y] - \mathrm{SReach}_{i-1}[G, \leq, y]\big| \cdot \max_{v \in V(G)} \big|\mathrm{WReach}_{k-1}[G, \leq, v]\big| \\
&\leq \big|\mathrm{SReach}_k[G, \leq, y]\big| \cdot \max_{v \in V(G)} \big|\mathrm{WReach}_{k-1}[G, \leq, v]\big|.
\end{aligned}$$

Thus by the induction hypothesis

$$\begin{aligned}
|\mathrm{WReach}_k[G, \leq, y]| &\leq |\mathrm{SReach}_k[G, \leq, y]| \cdot \max_{v \in V(G)} |\mathrm{WReach}_{k-1}[G, \leq, v]| \\
&\leq |\mathrm{SReach}_k[G, \leq, y]| \cdot (\max_{v \in V(G)} |\mathrm{SReach}_{k-1}[G, \leq, v]|)^{k-1} \\
&\leq (\max_{v \in V(G)} |\mathrm{SReach}_k[G, \leq, v]|)^k.
\end{aligned}$$

◀

We now show that for a nowhere dense class of graphs, for sufficiently large n-vertex subgraphs G' of graphs from the class, $\mathrm{col}_r(G') \leq n^\epsilon$.

▶ **Theorem 3.3.** *There exists a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ which is linear in the first argument such that for all $d, r \in \mathbb{N}$ and all classes $\mathcal{C}$ of graphs, if the class $\{H : H \preceq_r^t G, G \in \mathcal{C}\}$ is d-degenerate, then $\mathrm{col}_k(G') \leq f(d, r)$ for every subgraph $G' \subseteq G$ of a graph $G \in \mathcal{C}$.*

Proof. Let $G' \subseteq G$ for some $G \in \mathcal{C}$. Define f by

$$f(d, r) = \begin{cases} r + 1 & \text{if } d = 1 \\ 2d \cdot f(d, r-1)^{2r^2} & \text{else.} \end{cases}$$

If $d = 1$, then G' is a forest and it is easy to see that $\mathrm{col}_r(G') \leq r + 1$.

If $d \geq 2$, we recursively construct an ordering $L = x_1 x_2 \dots x_n$ of V as follows. Suppose that we have constructed the final sequence $x_{i+1} \dots x_n$ of L (if $i = n$ then this sequence is empty). Let $M = \{x_{i+1}, \dots, x_n\}$ be the set of vertices that have already been ordered and let $U = V - M$ be the set of vertices that have not yet been ordered. Notice that even though we have not finished constructing L, we have determined $\mathrm{SReach}_r[G', \leq, y]$ for any $y \in M$. However, we have not necessarily determined $\mathrm{WReach}_r[G', \leq, y]$. We now have to choose x_i from U. To do this we first define a probability space Ω, where each point in Ω is a graph $H = (U, F)$ such that an r-subdivision of H is a subgraph of G'. For each pair $\{u, v\} \subseteq U$ for which there exists a $u - v$ path of length at most r whose internal vertices are all in M, choose any such path and denote it by P_{uv}. For each vertex $z \in M$ let

$$S_z = \{\{u, v\} \subseteq U : z \in P_{uv}\}.$$

Label each $z \in M$ with a random element chosen from S_z; if $S_z = \emptyset$ then leave z unlabeled. Let F be the set of edges $\{u, v\}$ such that every internal vertex of P_{uv} is labeled with $\{u, v\}$. Then an r-subdivision of H is a subgraph G'. If P_{uv} is defined then the probability that $\{u, v\} \in F$ is

$$\Pr(\{u, v\} \in F) = \prod_{z \in M \cap V(P_{uv})} \frac{1}{|S_z|}.$$

In particular, if $\{u, v\} \in E$ then $\Pr(\{u, v\} \in F) = 1$. Let $E[d_H(u)]$ be the expected value of the degree of u in H. Choose x_i in U such that $E[d_H(x_i)]$ is minimal. We show that $E[d_H(x_i)] \leq 2d$.

Assume towards a contradiction that for all $u \in U$

$$E[d_H(u)] = \frac{\sum_{H' \in \Omega} d_{H'}(u)}{|\Omega|} > 2d.$$

Then

$$|U| \cdot d \geq \frac{\sum_{H' \in \Omega} |E(H')|}{|\Omega|} = \frac{\sum_{H' \in \Omega} \sum_{u \in U} d_{H'}(u)/2}{|\Omega|} > |U| \cdot d.$$

This is a contradiction and completes the construction of L.

We now argue by induction on $s \leq r$ that $|\mathrm{SReach}_s(y)| < f(d, s)$ for all vertices y. The base step $s = 1$ is trivial, so consider the induction step $s = t + 1$. Let $y \in V(G')$. Let U and M be the sets at the step before y was added to the order in the above recursion ($\mathrm{SReach}_s[G', \leq, y]$ is determined at this step). For each $z \in M$ and $\{u, v\} \in S_z$ both u and v are in $\mathrm{WReach}_t[G', \leq, z]$. Thus, by induction hypothesis and Theorem 3.2, $|S_z| \leq |\mathrm{WReach}_t[G', \leq, z]|^2 < f^{2t}(d, t)$. It follows that

$$\begin{aligned} 2d \geq E[d_H(y)] &= \sum_{x \in \mathrm{SReach}_s(y)} \Pr(\{x, y\} \in F) \\ &= \sum_{x \in \mathrm{SReach}_s(y)} \prod_{z \in M \cap V(P_{xy})} \frac{1}{|S_z|} > |\mathrm{SReach}_s[G', \leq, y]| \cdot f(d, t)^{-2t^2}. \end{aligned}$$

So $|\mathrm{SReach}_s[G', \leq, y]| < 2d \cdot f(d, t)^{2t^2} = f(d, s)$. ◀

As an algorithmic application, we close this section by demonstrating how generalised colouring numbers can be used in the design of *sparse neighbourhood covers*. *Neighborhood covers* of small radius and small size play a key role in the design of many data structures for distributed systems. See e.g. [54]. In this section we will show that nowhere dense classes of graphs admit sparse neighbourhood covers of small radius and small size.

▶ **Definition 3.4.** For $r \in \mathbb{N}$, an *r-neighbourhood cover* $\mathcal{X}$ of a graph G is a set of connected subgraphs of G called *clusters*, such that for every vertex $v \in V(G)$ there is some $X \in \mathcal{X}$ with $N_r(v) \subseteq X$. The *radius* $\mathrm{rad}(\mathcal{X})$ of a cover $\mathcal{X}$ is the maximum radius of any of its clusters. The *degree* $d^{\mathcal{X}}(v)$ of v in $\mathcal{X}$ is the number of clusters that contain v. The *maximum degree* $\Delta(\mathcal{X})$ of $\mathcal{X}$ is $\Delta(\mathcal{X}) = \max_{v \in V(G)} d^{\mathcal{X}}(v)$. The size of $\mathcal{X}$ is $\|\mathcal{X}\| = \sum_{X \in \mathcal{X}} |X| = \sum_{v \in V(G)} d^{\mathcal{X}}(v)$.

As proved in [39], nowhere dense classes of graphs admit sparse neighbourhood covers. This follows relatively easily from the characterisation of nowhere dense classes by weak colouring numbers (Theorem 3.1).

▶ **Theorem 3.5** (Grohe, Kreutzer, Siebertz [39]). *Let $\mathcal{C}$ be a nowhere dense class of graphs. There is a function f such that for all $r \in \mathbb{N}$ and $\epsilon > 0$ and all graphs $G \in \mathcal{C}$ with $n \geq f(r, \epsilon)$ vertices, there exists an r-neighbourhood cover of radius at most $2r$ and maximum degree at most n^{ϵ} and this cover can be computed in time $f(r, \epsilon) \cdot n^{1+\epsilon}$.*

4 Low tree-depth colourings

Many local problems can be solved by decomposing a graph into smaller pieces on which the problem hopefully becomes easier to solve. In the previous section, we have seen the concept of small radius neighbourhood covers. In this section we will use a graph colouring to define decompositions of graphs. It has been conjectured by Thomas [59] that for every graph K there is an integer k such that if a graph G excludes K as a minor then G has a vertex partition into two graphs with tree-width at most k. DeVos et al. proved the following stronger theorem.

▶ **Theorem 4.1** (DeVos et al. [22]). *For every graph K and every integer $j \geq 1$, there is an integer k, such that every graph with no K-minor has a vertex partition into $j+1$ graphs such that any j parts form a graph with tree width at most k.*

This result was strengthened by Hell and Nešetřil.

▶ **Theorem 4.2** (Hell and Nešetřil [40]). *For every graph K and integer $j \geq 1$, there is an inter $N(K,j)$ such that every graph with no K-minor has a vertex partition into N graphs such that any $j' \leq j$ parts form a graph with tree depth at most j'.*

The aim of this section is to present a characterisation of nowhere dense classes in terms of low tree depth colourings in the sense of the above theorem. Let us first give the formal definitions of tree depth and low tree depth colourings.

An *elimination tree* of a graph G is a rooted tree Y with vertex set $V(G)$ defined recursively as follows. If $V(G) = \{v\}$ then Y is just $\{v\}$. Otherwise, let $w \in V(G)$ be an arbitrary vertex which is chosen as the root of Y. The branches of Y at w are the elimination trees of the connected components of $G - w$ whose roots are the sons of w in Y. The height of a rooted tree Y is the maximum distance of the root to any vertex of the tree. The *tree depth* of a graph G is the minimum height of an elimination tree of G. It follows that we can give the following recursive characterisation.

Let G be a graph. The tree depth $\mathrm{td}(G)$ is defined as

$$\mathrm{td}(G) = \begin{cases} 0 & \text{if } |V(G)| = 1 \\ 1 + \min_{v \in V(G)} \mathrm{td}(G - v) & \text{if } G \text{ is connected and } |V(G)| > 1 \\ \max_{1 \leq i \leq k} \mathrm{td}(G_i) & \text{if } G_1, \ldots, G_k \text{ are the connected components of } G. \end{cases}$$

For $r \geq 1$, an r-tree depth colouring of G is a colouring such that every nonempty subgraph $G' \subseteq G$ is coloured by at least $\min\{r, \mathrm{td}(G') + 1\}$ colours. Equivalently, an r-tree depth colouring of G is a colouring such that any $r' \leq r$ colour classes induce a subgraph with tree depth at most $r' + 1$. The minimum number of colours of such a colouring of G is denoted by $\text{td-col}_r(G)$.

The aim of this section is to show the following theorem.

▶ **Theorem 4.3** (Nešetřil and Ossona de Mendez [52]). *A class $\mathcal{C}$ of graphs is nowhere dense if and only if for every $\epsilon > 0$ there is $n_0(r, \epsilon)$ such that $\text{td-col}_r(G') \leq n^\epsilon$ for all n-vertex subgraphs $G' \subseteq G$ of a graph $G \in \mathcal{C}$ with $n \geq n_0$.*

As a first step towards the proof of this theorem, we present a result of Zhu [60]. Recall that every nowhere dense class admits an ordering of its vertices such that only few vertices are weakly r-reachable from any other vertex.

▶ **Theorem 4.4** (Zhu [60]). *If G is a graph with $\mathrm{wcol}_{2^{r-2}}(G) \leq m$, then G can be coloured with m colours such that any in connected subgraph $H \subseteq G$ either some colour appears exactly once in H or H gets at least r colours.*

Proof. Let $\leq \in \Pi(G)$ be an ordering of $V(G)$ witnessing $\mathrm{wcol}_{2^{r-2}} \leq m$. Colour the vertices greedily with m colours, using the order L, such that the colour assigned to v is distinct from colours assigned to vertices weakly reachable from v. We claim that this colouring satisfies the desired properties.

Let H be a connected subgraph of G and let v be the minimum vertex of H with respect to L. If the colour $c(v)$ appears exactly once in H then we are done.

Assume $c(v)$ occurs more than once in H. We shall prove that H uses at least r colours. Let $u \neq v$ be a vertex of H with $c(u) = c(v)$ and let $P_0 = v, v_1, \ldots, v_q = u$ be a path in H connecting v and u. We must have $q > 2^{r-2}$, for otherwise v is weakly 2^{r-2}-accessible from u and we should have $c(v) \neq c(u)$. Let $u_0 := v$ and let $P_1 := v_1, \ldots, v_{2^{r-2}}$. Observe that no vertex of P_1 uses colour $c(u_0)$ and P_1 contains 2^{r-2} vertices. Assume $0 \leq j \leq r-2$ and that a vertex u_j of P_j and a subpath P_{j+1} of P_j are chosen such that the following holds. No vertex of P_{j+1} uses the colour of u_j and P_{j+1} contains at least 2^{r-j-2} vertices. We show how to establish this same situation for $j+1$. Let u_{j+1} be the minimum vertex of P_{j+1} with respect to L let P_{j+2} be the largest component of $P_{j+1} - u_{j+1}$. Then u_{j+1} is weakly 2^{r-2}-accessible from each vertex of P_{j+1} and hence no vertex of P_{j+2} uses the colour $c(u_{j+1})$. Moreover, P_{j+2} is a path containing at least 2^{r-j-3} vertices. We repeat this process until $j = r-2$ and obtain vertices $u_0, \ldots, u_{r-1}$ of distinct colours. Hence H uses at least r colours. ◀

The properties of the colouring in the above proof are important enough to give a special name to such colourings.

▶ **Definition 4.5.** An *r-centered colouring* of a graph G is a vertex colouring such that for any connected subgraph $H \subseteq G$, either some colour appears exactly once in H or H gets at least r colours.

It is not difficult to see that indeed such colourings induce low tree depth colourings.

▶ **Lemma 4.6.** *Any r-centered colouring is an r-tree depth colouring.*

Proof. Let c be an r-centered colouring of G. Assume that there is a subgraph $G' \subseteq G$ with $\mathrm{td}(G') = k < r$ which does not get $k+1$ colours. Let G' be minimal with this property. Then G' is connected. As G' does not get $k+1 \leq r$ colours and c is r-centered, there is one colour which occurs exactly once, say this colour is given to vertex v. Then $\mathrm{td}(G'-v) \geq k-1$ and $G'-v$ does not get $k-1$ colours. Hence G' was not minimal. ◀

Let us show how low tree depth colourings bound the edge density of depth-r topological subgraphs.

▶ **Lemma 4.7** (Zhu [60]). *Let G be a graph, $r \in \mathbb{N}$ and $H \preceq_r^t G$. Then*

$$\frac{|E(H)|}{|V(H)|} \leq \binom{td\text{-}col_{r+2}(G)}{r+2}(r+1).$$

Proof. Consider a vertex colouring c of G with $N = \text{td-col}_{r+2}(G)$ colours such that any $i \leq r+2$ colours induce a subgraph of tree depth at most i. For every set J of $r+2$ colours let $G_J := G[c^{-1}(J)]$ and let Y_J be an elimination tree of height $td(G_J) \leq r+2$ of G_J (which is in fact a rooted forest).

Suppose that $V(H) = [k]$. As $H \preceq_r^t G$, there are vertices $h_1, \ldots, h_k \in V(G)$ and mutually internally disjoint paths $P_{ij} \subseteq G$ of length at most $(r+1)$ from h_i to h_j for all edges $ij \in E(H)$. Let J_{ij} be a set of $r+2$ colours such that $c(V(P_{ij})) \subseteq J_{ij}$. Then $P_{ij} \subseteq G_{J_{ij}}$, and there is a vertex v_{ij} on P_{ij} that is a common ancestor of h_i and h_j in the elimination tree $Y_{ij} = Y_{J_{ij}}$. Possibly, $v_{ij} = h_i$ or $v_{ij} = h_j$. We orient the edge $e = ij$ from i to j if $h_i = v_{ij}$ and from j to i if $v_{ij} = h_j$. If neither $h_i = v_{ij}$ nor $h_j = v_{ij}$ then we orient the edge ij arbitrarily. Let $\vec{H}$ be the resulting oriented graph.

To bound the number of edges of H, we bound the maximum in-degree $\Delta^{-1}(\vec{H})$. Let $j \in V(H)$. For every edge $ij \in E(\vec{H})$, the vertex v_{ij} is a proper ancestor of h_i in the elimination tree Y_{ij}, and there are at most $r+1$ such ancestors. Moreover, for distinct edges $ij, i'j$ the vertices v_{ij} and $v_{i'j}$ are distinct, because the paths P_{ij} are internally disjoint. Thus

$$\Delta^-(\vec{H}) \leq \sum_J (r+1) = \binom{N}{r+2}(r+1),$$

where the sum ranges over all sets J of at most $r+2$ colours. It follows that

$$|E(H)| \leq |V(H)| \cdot \Delta^-(\vec{H}) \leq |V(H)| \binom{N}{r+2}(r+1).$$

◀

Note that in order to conclude the proof with Corollary 2.6, we have to express $\frac{|E(H)|}{|V(H)|}$ with respect to $|V(H)|$ and not with respect to $|V(G)|$. For r-subdivisions this is no problem though. Take a minimal subgraph G' of G such that an r-subdivision is a subgraph of G'. Then G' has at most $|V(H)|^2 \cdot r$ edges.

We demonstrate the algorithmic applications of low tree depth colourings by showing that the *subgraph isomophism problem* can be solved in time $\mathcal{O}(n^{1+\epsilon})$ on nowhere dense classes of graphs for any fixed template H. Recall that the subgraph isomorphism problem is the problem, given two graphs H and G as input, to decide whether G contains a subgraph isomorphic to H. For a fixed template H and $\epsilon > 0$, the problem can be solved on any nowhere dense class $\mathcal{C}$ of graphs as follows. Let $h := |V(H)|$ and set $\epsilon' := \frac{\epsilon}{h}$. Let $G \in \mathcal{C}$. To decide whether G contains a subgraph G' isomorphic to H, we first apply Theorem 4.3 to obtain a colouring γ of $V(G)$ with at most $n^{\epsilon'}$ colours such that any h colour classes together induce a subgraph of G of tree depth at most h. Note that while we have only given a proof of the existence of such colouring, such a colouring γ can be computed in time $\mathcal{O}(n^{1+\epsilon'})$ using the concept of augmentations introduced in the next section (see e.g. [53, 47]). Furthermore, it is well-known (and follows, e.g. from Courcelle's theorem mentioned in the introduction) that on any class of graphs of bounded tree depth there is a linear time algorithm for solving the subgraph isomorphism problem for a fixed template H. Hence, to verify whether G contains a subgraph isomorphic to H we can compute all $(n^{\epsilon'})^h = n^{h \cdot \epsilon'} = n^{\epsilon}$ subgraphs $G_{c_1,\ldots,c_h}$ induced by exactly h colour classes $c_1, \ldots, c_h$ with respect to γ and for each test in linear time whether $G_{c_1,\ldots,c_h}$ contains H as an isomorphic subgraph. Together this yields the required running time.

5 Augmentations

For many algorithms it is essential to compute an ordering that witnesses $\mathrm{wcol}_r(G) \leq n^{\epsilon}$ or a td-col$_r$-colouring with at most n^{ϵ} colours. Not surprisingly, the problem of computing an optimal such ordering is NP-complete in general (it is easy to modify the proof of Pothen [55], showing that computing the tree depth of a graph is NP-complete). The question whether

the problem is fixed-parameter tractable (with parameter $\mathrm{wcol}_r(G)$) is an interesting open question, yet even a positive answer would not help in the context of nowhere dense classes of graphs, as the parameter is only bounded by n^ϵ for such classes. Yet there are good approximation algorithms, see e.g. Dvořák [25]. We are going to present another way of approximating $\mathrm{wcol}_r(G)$ in order to present another important method for nowhere dense classes of graphs. The idea is as follows. Assume that an order witnessing $\mathrm{wcol}_r(G) = k$ has been found. We define a directed graph $\vec{H}_r$ on the same vertex set as G by adding an edge from u to v if and only if u is weakly r-accessible from v. $\vec{H}_r$ has the following properties. For all pairs u, v of vertices such that $\mathrm{dist}_G(u, v) \leq r$ one of the following holds. Either there is an edge (u, v) or (v, u) in $\vec{H}_r$ or there is a vertex w such that (w, u) and (w, v) are edges of $\vec{H}_r$ (if the first two cases do not hold, consider the smallest vertex w on the path from u to v). Furthermore, every vertex of $\vec{H}_r$ has indegree at most k.

▶ **Lemma 5.1.** *Let G be a graph and let $r > 0$. Let $\vec{H}$ be a directed graph such that for all pairs u, v of vertices with $\mathrm{dist}_G(u, v) \leq r$ one of the following holds. Either there is an edge (u, v) or (v, u) in $\vec{H}$ or there is a vertex w such that (w, u) and (w, v) are edges of $\vec{H}$ and such that every vertex of $\vec{H}$ has indegree at most d. Then $\mathrm{wcol}_r(G) \leq 2(d + 1)^2$.*

Proof. As $\Delta^-(\vec{H}_r) \leq d$, the underlying undirected graph H is $2d$-degenerate and we can order the vertices of H such that each vertex has at most $2d$ smaller neighbours. Denote this order by $\leq$. For each vertex $v \in V(G)$ we count the number of endvertices of paths of length at most r from v such that the endvertex is the smallest vertex of the path. This number bounds $|\mathrm{WReach}_r[G, \leq, v)]|$.

By assumption, for each such path with endvertex w, we either have an edge (v, w) or an edge (w, v) or there is u on the path and we have edges $(u, v), (u, w)$ in $\vec{H}$. By construction of the order there are at most $2d$ edges (v, w) or (w, v) such that $w < v$. Furthermore, we have at most d edges (u, v), as v has indegree at most d and for each such u there are at most $2d$ edges (u, w) such that $w < u$ by construction of the order. These are exactly the pairs of edges we have to consider, as no vertex on the path from v to w may be smaller than w. Hence in total we have $|\mathrm{WReach}_r[G, \leq, v]| \leq 2d + 2d^2 + 1 \leq 2(d + 1)^2$. (Note that we have to add 1 because $\mathrm{WReach}_r[G, \leq, v]$ contains v which is not reachable by any edges in the above way). ◀

It was shown by Nešetřil and Ossona de Mendez that we can iteratively compute a good approximation to the above on nowhere dense classes of graphs.

▶ **Definition 5.2.** Let $\vec{G}$ be a directed graph. A *tight* 1*-transitive fraternal augmentation* of $\vec{G}$ is a directed graph $\vec{H}$ with the same vertex set, including all the arcs of $\vec{G}$ and such that for all distinct vertices u, v, w

- if $(u, w), (w, v) \in E(\vec{G})$ then $(u, v) \in E(\vec{H})$,
- if $(u, w), (v, w) \in E(\vec{G})$ then (u, v) or (v, u) are arcs of $\vec{H}$ and
- for all $(u, v) \in E(\vec{H})$, either $(u, v) \in E(\vec{G})$ or there is some w such that $(u, w), (w, v) \in E(\vec{G})$ or $(u, w), (v, w) \in E(\vec{G})$.

Let us show that edge densities of depth-r topological subgraphs are relatively stable under tight 1-transitive fraternal augmentations.

▶ **Definition 5.3.** Let H, G be graphs. The *lexicographic product* of G and H is defined as the graph $G \bullet H$ with vertex set and edge set respectively:

$$V(G \bullet H) = V(G) \times V(H)$$

$$E(G \bullet H) = \big\{\{(x, y), (x', y')\} : \{x, x'\} \in E(G) \text{ or } (x = x' \text{ and } \{y, y'\} \in E(H))\big\}.$$

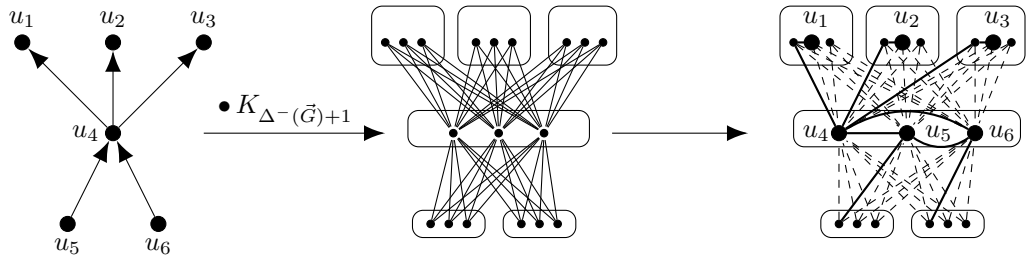


Figure 2 Proof sketch of Lemma 5.5.

The following is not hard to see.

▶ **Lemma 5.4.** *Let G be a graph and let $r, t, m \in \mathbb{N}$. If G has an r-centered colouring with m colours then $G \bullet K_t$ has an r centered colouring with $t \cdot m$ colours.*

The following was shown by Nešetřil and Ossona de Mendez [49, Lemma 7.2].

▶ **Lemma 5.5.** *Let $\vec{G}$ be a directed graph and let $\vec{H}$ be a tight 1-transitive fraternal augmentation of $\vec{G}$. Let $t := \Delta^-(\vec{G}) + 1$ and let G, H be the undirected graphs underlying $\vec{H}$ and $\vec{G}$, respectively. Then a 1-subdivision of H is a subgraph of $G \bullet K_t$.*

We only sketch the proof of the lemma. The main construction is illustrated in Figure 2. We want to show that the tight 1-transitive fraternal augmentation of the graph $\vec{G}$ on the left hand side of the figure is a 1-subdivision of a subgraph of $G \bullet K_3$. $G \bullet K_3$ is illustrated in the middle section of the figure, where rounded rectangles correspond to copies of K_3. The right hand side of the figure shows how the tight 1-transitive fraternal augmentation of $\vec{G}$ can be found as a 1-subdivision. Here, the thick black edges show the edges that are subdivided once. To improve readability we have omitted the edges from u_5 and u_6 to u_1, u_2, u_3 as these are not subdivided. Note that the only purpose of the two edges going to the two rectangles on the bottom is that in this way we can extend the construction to larger graphs $\vec{G}$. See [49, Lemma 7.2] for details.

By Lemma 4.7 we can conclude that H has small degeneracy if G had an r-centered colouring with few colours and if the indegree of $\vec{G}$ was not too large. Both is the case for nowhere dense classes of graphs. We can hence reorient the degenerate graph H to obtain again a small indegree orientation. Note furthermore that $(G \bullet K_t) \bullet K_s \cong G \bullet K_{t \cdot s}$. We can hence iterate the augmentation procedure for r times and obtain as a result a directed graph with the properties required by Lemma 5.1. It was shown by Nešetřil and Ossona de Mendez that this procedure can be implemented in time $\mathcal{O}(n^{1+\epsilon})$ on nowhere dense classes of graphs.

Augmentations are a key algorithmic tool for nowhere dense classes of graphs. Due to space restriction we refrain from giving explicit algorithmic applications here and refer, e.g., to [42] instead, where augmentations are used for query answer enumeration in nowhere dense classes of databases.

6 Quasi-wideness

A set $A \subseteq V(G)$ is called r-scattered if $N_r(u) \cap N_r(v) = \emptyset$ for all distinct $u, v \in A$.

▶ **Definition 6.1.** A class $\mathcal{C}$ of graphs is *uniformly quasi-wide* with margin $s : \mathbb{N} \to \mathbb{N}$ and $N : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ if for all $r, k \in \mathbb{N}$, if $G \in \mathcal{C}$ and $W \subseteq V(G)$ with $|W| > N(r, k)$, then there is a set $S \subseteq W$ with $|S| < s(r)$, such that W contains an r-scattered set of size at least k in $G - S$.

The aim of this section is to show the following theorem.

▶ **Theorem 6.2.** *A class $\mathcal{C}$ is nowhere dense if and only if $\mathcal{C}$ is uniformly quasi-wide.*

Obviously, if $\mathcal{C}$ is somewhere dense then it is not uniformly quasi-wide. We will hence show the other direction.

The first step of showing that nowhere density implies uniformly quasi-wideness is to find large independent sets. We will make use of the following Ramsey Theorem.

▶ **Theorem 6.3.** *For $k \geq 1$, let $n_1, \ldots, n_k \in \mathbb{N}$. There exists a number $R = R(n_1, \ldots, n_k)$, called Ramsey number, which is minimum with the following property. For every complete graph G on at least R vertices with edges coloured by colours $\{1, \ldots, k\}$, there exists some $1 \leq i \leq k$ such that G contains an induced subgraph of size at least n_i such that every edge has colour i.*

The theorem implies that every sufficiently large graph contains either a large independent set or a large complete graph. As nowhere dense classes of graphs do not contain large complete graphs, we conclude that in large graphs from the class we find large independent sets.

To go from 1-independent sets to 2-independent sets, we will use the following lemma which was proved by Nešetřil and Ossona de Mendez. The proof makes use of Theorem 6.3 in it's general form.

▶ **Lemma 6.4** (Nešetřil and Ossona de Mendez [51])**.** *There is a function Θ such that for all $s, k, a, b \in \mathbb{N}$ and all bipartite graphs $G = (A \cup B, E)$ with $|A| \geq \Theta(k, a, b, s)$ at least one of the following holds.*

- *There exists in B a subset of size at most s whose removal leaves in A a 2-independent set of size k.*
- *A includes the branch vertices of a 1-subdivision of the complete graph K_a.*
- *B includes $s+1$ vertices that form one side of a complete bipartite subgraph $K_{s+1,b}$ in G.*

Given a graph G and an independent set $A \subseteq V(G)$ of size at least $\Theta(k, a, b, s)$ we can just construct the bipartite graph with part $B = N(A)$ and establish the situation of the above lemma. For a nowhere dense class of graphs and sufficiently large a, b, s, we again know that we are in the first case.

We now apply the above arguments iteratively for r times. Given a large $2k$-independent set in a graph, we contract the k-neighbourhoods of the elements of the independent set and find large a $2k+1$ independent set. Given a large $2r+1$-independent set, we contract the $2k+1$-neighbourhoods of the elements of the set, and after the deletion of few elements, we find a $2k+2$-independent set. Note that we do not delete contracted vertices but vertices of $V(G)$ and hence we really delete only few vertices.

Algorithmically, the concept of uniformly quasi-wideness is very useful in designing bounded depth search trees for parameterized algorithms for solving problems such as dominating sets and network centres. We refer to [15] for examples demonstrating this technique.

7 Game-theoretic characterisation

As a final characterisation we show that nowhere dense classes of graphs can also be defined in terms of a game, which we call the splitter game introduced in [39].

▶ **Definition 7.1** (Splitter game)**.** Let G be a graph and let $\ell, m, r > 0$. The (ℓ, m, r)-*splitter game* on G is played by two players, "Connector" and "Splitter", as follows. We let $G_0 := G$. In round $i+1$ of the game, Connector chooses a vertex $v_{i+1} \in V(G_i)$. Then Splitter picks a subset $W_{i+1} \subseteq N_r^{G_i}(v_{i+1})$ of size at most m. We let $G_{i+1} := G_i[N_r^{G_i}(v_{i+1}) \setminus W_{i+1}]$. Splitter wins if $G_{i+1} = \emptyset$. Otherwise the game continues at G_{i+1}. If Splitter has not won after ℓ rounds, then Connector wins.

A *strategy* for Splitter is a function f, which associates to every partial play $(v_1, W_1, \ldots, v_s, W_s)$ with associated sequence $G_0, \ldots, G_s$ of graphs and move $v_{s+1} \in V(G_s)$ by Connector a set $W_{s+1} \subseteq N_r^{G_s}(v_{s+1})$ of size at most m. A strategy f is a *winning strategy* for Splitter in the (ℓ, m, r)-splitter game on G if Splitter wins every play in which he follows the strategy f. If Splitter has a winning strategy, we say that he *wins* the (ℓ, m, r)-splitter game on G.

▶ **Theorem 7.2** (Grohe, Kreutzer, Siebertz [39])**.** *Let $\mathcal{C}$ be a nowhere dense class of graphs.*

1. *For every $r > 0$ there are $\ell, m > 0$, such that for every $G \in \mathcal{C}$, Splitter wins the (ℓ, m, r)-splitter game on G.*
2. *Conversely, if for every $r > 0$ there are $\ell, m > 0$ such that for every graph $G \in \mathcal{C}$, Splitter wins the (ℓ, m, r)-splitter game, then $\mathcal{C}$ is nowhere dense.*

References

1 H. Adler and I. Adler. Interpreting nowhere dense graph classes as a classical notion of model theory. *Europ. Journal of Combinatorics*, 2013. to appear.

2 N. Alon, M. Krivelevich, and B. Sudakov. Turán numbers of bipartite graphs and related ramsey-type questions. *Combinatorics Probability and Computing*, 12(5):477–494, 2003.

3 B. S. Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.

4 H. Bodlaender. A partial k-aboretum of graphs with bounded tree-width. *Theoretical Computer Science*, 209:1 – 45, 1998.

5 H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–22, 1993.

6 H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proc. of Mathematical Foundations of Computer Science (MFCS)*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36, 1997.

7 H. L. Bodlaender. Discovering treewidth. In *31st International Conference on Current Trends in Theory and Practice of Computer Science*, pages 1–16, 2005.

8 H. L. Bodlaender. Treewidth: Characterizations, applications, and computations. In Fedor V. Fomin, editor, *Graph-Theoretic Concepts in Computer Science*, volume 4271 of *LNCS*, pages 1–14. Springer, 2006.

9 H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (meta) kernelization. In *50th Annual Symposium on Foundations of Computer Science (FOCS 2009)*, 2009.

10 H.L. Bodlaender, F.V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D.M. Thilikos. (Meta) Kernelization. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 629–638, 2009.

11 B. Bollobás and A. Thomason. Proof of a conjecture of mader, erdös and hajnal on topological complete subgraphs. *Europ. Journal of Combinatorics*, 19(8):883–887, 1998.

12 B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 2, pages 194 – 242. Elsevier, 1990.

13 A. Dawar, M. Grohe, and S. Kreutzer. Locally excluding a minor. In *Logic in Computer Science (LICS)*, pages 270–279, 2007.

14 A. Dawar, M. Grohe, S. Kreutzer, and N. Schweikardt. Approximation schemes for first-order definable optimisation problems. In *Logic in Computer Science (LICS)*, pages 411–420, 2006.

15 A. Dawar and S. Kreutzer. Domination problems in nowhere-dense classes of graphs. In *Foundations of software technology and theoretical computer science (FSTTCS)*, pages 157–168, 2009.

16 E. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, pages 332–337, 2008.

17 E. Demaine, M. Hajiaghayi, F. Fomin, and D. Thilikos. Bidimensional parameters and local tree-width. *SIAM Journal of Discrete Mathematics*, 18:501–511, 2004.

18 E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and -minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

19 E. D. Demaine, M. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *46th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 637–646, 2005.

20 E. D. Demaine, M. T. Hajiaghayi, and K. Kawarabayashi. Decomposition, approximation, and coloring of odd-minor-free graphs. In *SODA*, pages 329–344, 2010.

21 E. D. Demaine, M. T. Hajiaghayi, and K. Kawarabayashi. Contraction decomposition in h-minor-free graphs and algorithmic applications. In *STOC*, pages 441–450, 2011.

22 M. DeVos, G. Ding, B. Oporowski, D. P Sanders, B. Reed, P. Seymour, and D. Vertigan. Excluding any graph as a minor allows a low tree-width 2-coloring. *Journal of Combinatorial Theory, Series B*, 91(1):25–41, 2004.

23 R. Diestel. *Graph Theory*. Springer-Verlag, 3rd edition, 2005.

24 R. Downey and M. Fellows. *Parameterized Complexity*. Springer, 1999.

25 Z. Dvořák. Constant-factor approximation of the domination number in sparse graphs. *Eur. J. Comb.*, 34(5):833–840, 2013.

26 Zdenek Dvorák. Asymptotical structure of combinatorial objects. *Charles University, Faculty of Mathematics and Physics*, 2007.

27 Z. Dvořák, D. Král, and R. Thomas. Deciding first-order properties for sparse graphs. *Journal of the ACM*. to appear.

28 Z. Dvořák, D. Král, and R. Thomas. Deciding first-order properties for sparse graphs. In *51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2010.

29 D. Eisenstat, P. N. Klein, and C. Mathieu. An efficient polynomial-time approximation scheme for steiner forest in planar graphs. In *Proceedings of the 23nd ACM-SIAM Symposium On Discrete Algorithms*, 2012.

30 D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. of Graph Algorithms and Applications*, 3(3):1–27, 1999.

31 D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27:275–291, 2000.

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

33 Jörg Flum and Martin Grohe. Fixed-parameter tractability, definability, and model checking. *SIAM Journal on Computing*, 31:113 – 145, 2001.

34 M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *Journal of the ACM*, 48:1148 – 1206, 2001.

35 J. Gajarský, P. Hlinený, J. Obdrzálek, S. Ordyniak, F. Reidl, P. Rossmanith, F. Sanchez Villaamil, and S. Sikdar. Kernelization using structural parameters on sparse graph classes. In *ESA*, pages 529–540, 2013.

36 M. Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23(4):613–632, 2003.

37 M. Grohe. Logic, graphs, and algorithms. In E.Grädel T.Wilke J.Flum, editor, *Logic and Automata – History and Perspectives*. Amsterdam University Press, 2007.

38 M. Grohe and S. Kreutzer. Methods for algorithmic meta-theorems. *Contemporary Mathematics*, 588, American Mathematical Society 2011.

39 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. submitted, 2013.

40 P. Hell and J. Nešetřil. On the complexity of< i> h</i>-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990.

41 T Jiang. Compact topological minors in graphs. *Journal of Graph Theory*, 67(2):139–152, 2011.

42 W. Kazana and L. Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *PODS*, pages 297–308, 2013.

43 H.A. Kierstead and D. Yang. Orderings on graphs and game coloring number. *Order*, 20(3):255–264, 2003.

44 J. Komlós and E. Szemerédi. Topological cliques in graphs. *Combinatorics, Probability & Computing*, 3:247–256, 1994.

45 S. Kreutzer. Algorithmic meta-theorems. In Javier Esparza, Christian Michaux, and Charles Steinhorn, editors, *Finite and Algorithmic Model Theory*, London Mathematical Society Lecture Note Series, chapter 5, pages 177–270. Cambridge University Press, 2011. a preliminary version is available at Electronic Colloquium on Computational Complexity (ECCC), TR09-147, http://www.eccc.uni-trier.de/report/2009/147.

46 W. Mader. Homomorphieeigenschaften und mittlere Kantendichte von Graphen. *Mathematische Annalen*, 174(4):265–268, 1967.

47 J. Nešetřil and P. Ossona De Mendez. Grad and classes with bounded expansion i. decompositions. *Europ. Journal of Combinatorics*, 29(3):760–776, 2008.

48 J. Nešetřil and P. Ossona de Mendez. Grad and classes with bounded expansion iii. restricted graph homomorphism dualities. *Europ. Journ. of Comb.*, 29(4):1012–1024, 2008.

49 J. Nešetřil and P. Ossona de Mendez. *Sparsity*. Springer, 2012.

50 J. Nešetřil and P. Ossona de Mendez. First order properties of nowhere dense structures. *Journal of Symbolic Logic*, 75(3):868–887, 2010.

51 J. Nešetřil and P. Ossona de Mendez. First order properties of nowhere dense structures. *Journal of Symbolic Logic*, 75(3):868–887, 2010.

52 J. Nešetřil and P. Ossona de Mendez. On nowhere dense graphs. *Europ. Journal of Combinatorics*, 32(4):600–617, 2011.

53 J. Nešetřil and P. Ossona de Mendez. Grad and classes with bounded expansion ii. algorithmic aspects. *Europ. Journal of Combinatorics*, 29(3):777 – 791, 2008.

54 D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 1987.

55 A. Pothen. *The complexity of optimal elimination trees*. Pennsylvania State University, Department of Computer Science, 1988.

56 N. Robertson and P.D. Seymour. Graph minors I – XXIII, 1982 – 2010. Appearing in Journal of Combinatorial Theory, Series B from 1982 till 2010.

57 D. Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 5:505–526, 1996.

58 S. Tazari and M. Müller-Hannemann. Dealing with large hidden constants: engineering a planar steiner tree ptas. *ACM Journal of Experimental Algorithmics*, 16, 2011.

59 R. Thomas. Problem session of the third slovene conference on graph theory, Bled, Slovenia. 1995.

60 X. Zhu. Colouring graphs with bounded generalized colouring number. *Discrete Mathematics*, 309:5562–5568, 2009.

Intersection Types for Normalization and Verification

Kazushige Terui

RIMS, Kyoto University
Kitashirakawa Oiwakecho, Sakyo-ku, Kyoto 606-8502, Japan
terui@kurims.kyoto-u.ac.jp

Abstract

One of the basic principles in typed lambda calculi is that typable lambda terms are normalizable. Since the converse direction does not hold for simply typed lambda calculus, people have been studying its extensions. This gave birth to the intersection type systems, that exactly characterize various classes of lambda terms, such as strongly/weakly normalizable terms and solvable ones (see e.g. [6] for a survey).

More recently, a new trend has emerged: intersection types are not only useful for extending simple types but also for *refining* them [4]. One thus obtains finer information on simply typed terms by assigning intersection types. This in particular leads to the concept of *normalization by typing*, that turns out to be quite efficient in some situations [5]. Moreover, intersection types are invariant under β-equivalence, so that they constitute a denotational semantics in a natural way [1]. Finally, intersection types also work in an infinitary setting, where terms may represent infinite trees and types play the role of automata. This leads to a model checking framework for higher order recursion schemes via intersection types [2, 3].

The purpose of this talk is to outline the recent development of intersection types described above. In particular, we explain how an efficient evaluation algorithm is obtained by combining normalization by typing, β-reduction and Krivine's abstract machine, to result in the following complexity characterization. Consider simply typed lambda terms of boolean type $o \to o \to o$ and of order r. Then the problem of deciding whether a given term evaluates to "true" is complete for n-EXPTIME if $r = 2n + 2$, and complete for n-EXPSPACE if $r = 2n + 3$ [5].

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases simply typed lambda calculus, computational complexity, denotational semantics, intersection types

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.41

Category Invited Talk

References

1 Thomas Ehrhard. Collapsing non-idempotent intersection types. In *Proceedings of 26th CSL*, LIPIcs, Vol. 16, pages 259–273, 2012.

2 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of 36th POPL*, pages 416–428, 2009.

3 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of 24th LICS*, pages 179–188, 2009.

4 Sylvain Salvati. On the membership problem for non-linear abstract categorial grammars. *Journal of Logic, Language and Information*, 19(2):163–183, 2010.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 41–42

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

5 Kazushige Terui. Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In *Proceedings of 23rd RTA*, LIPIcs, Vol. 15, pages 323–338, 2012.
6 Steffen van Bakel. Intersection type assignment systems. *Theoretical Computer Science*, 151(2):385–435, 1995.

Polynomial Kernels for λ-extendible Properties Parameterized Above the Poljak-Turzík Bound

Robert Crowston[1], Mark Jones[1], Gabriele Muciaccia[1], Geevarghese Philip[2], Ashutosh Rai[3], and Saket Saurabh[3,4]

1 **Royal Holloway, University of London, UK,** {robert,markj,G.Muciaccia}@cs.rhul.ac.uk
2 **Max-Planck-Institut für Informatik (MPII), Germany,** gphilip@mpi-inf.mpg.de
3 **Institute of Mathematical Sciences, India,** {ashutosh,saket}@imsc.res.in
4 **University of Bergen, Norway**

Abstract

Poljak and Turzík (*Discrete Mathematics* 1986) introduced the notion of λ-extendible properties of graphs as a generalization of the property of being bipartite. They showed that for any $0 < \lambda < 1$ and λ-extendible property Π, any connected graph G on n vertices and m edges contains a spanning subgraph $H \in \Pi$ with at least $\lambda m + \frac{1-\lambda}{2}(n-1)$ edges. The property of being bipartite is λ-extendible for $\lambda = 1/2$, and so the Poljak-Turzík bound generalizes the well-known Edwards-Erdős bound for Max-Cut. Other examples of λ-extendible properties include: being an acyclic oriented graph, a balanced signed graph, or a q-colorable graph for some $q \in \mathbb{N}$.

Mnich et al. (*FSTTCS* 2012) defined the closely related notion of *strong* λ-extendibility. They showed that the problem of finding a subgraph satisfying a given strongly λ-extendible property Π is fixed-parameter tractable (FPT) when parameterized above the Poljak-Turzík bound—*does there exist a spanning subgraph H of a connected graph G such that $H \in \Pi$ and H has at least $\lambda m + \frac{1-\lambda}{2}(n-1) + k$ edges?*—subject to the condition that the problem is FPT on a certain simple class of graphs called *almost-forests of cliques*. This generalized an earlier result of Crowston et al. (*ICALP* 2012) for Max-Cut, to all strongly λ-extendible properties which satisfy the additional criterion.

In this paper we settle the kernelization complexity of nearly all problems parameterized above Poljak-Turzík bounds, in the affirmative. We show that these problems admit quadratic kernels (cubic when $\lambda = 1/2$), *without using* the assumption that the problem is FPT on almost-forests of cliques. Thus our results not only remove the technical condition of being FPT on almost-forests of cliques from previous results, but also unify and extend previously known kernelization results in this direction. Our results add to the select list of *generic* kernelization results known in the literature.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Kernelization, Lambda Extension, Above-Guarantee Parameterization, MaxCut

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.43

1 Introduction

In parameterized complexity each problem instance I comes with a parameter k, and a parameterized problem is said to be *fixed parameter tractable* (FPT) if for each instance (I, k) the problem can be solved in time $f(k)|I|^{\mathcal{O}(1)}$ where f is some computable function. The parameterized problem is said to admit a *polynomial kernel* if there is a polynomial time algorithm, called a *kernelization* algorithm, that reduces the input instance down to

an instance with size bounded by a polynomial $p(k)$ in k, while preserving the answer. This reduced instance is called a $p(k)$ *kernel* for the problem. The study of kernelization is a major research frontier of Parameterized Complexity; many important recent advances in the area pertain to kernelization. These include general results showing that certain classes of parameterized problems have polynomial kernels [1, 4, 14, 13] and randomized kernelization based on matroid tools [19, 18]. The recent development of a framework for ruling out polynomial kernels under certain complexity-theoretic assumptions [3, 9, 15] has added a new dimension to the field and strengthened its connections to classical complexity. For overviews of kernelization we refer to surveys [2, 16] and to the corresponding chapters in books on Parameterized Complexity [17, 23]. In this paper we give a generic kernelization result for a class of problems parameterized above guaranteed lower bounds.

Context and Related Work. Many interesting graph problems are about finding a largest subgraph H of the input graph G, where graph H satisfies some specified property and its size is defined as the number of its edges. For many properties this problem is NP-hard, and for some of these we know nontrivial lower bounds for the size of H. In these latter cases, the apposite parameterization "by problem size" is: Given graph G and parameter $k \in \mathbb{N}$, does G have a subgraph H which has (i) the specified property and (ii) at least k *more* edges than the best known lower bound? Max-Cut is a sterling example of such a problem. The problem asks for a largest *bipartite* subgraph H of the input graph G; it is NP-complete [25], and the well-known *Edwards-Erdős bound* [11, 12] tells us that any connected loop-less graph on n vertices and m edges has a bipartite subgraph with at least $\frac{m}{2} + \frac{n-1}{4}$ edges. This lower bound is also the best possible, in the sense that it is tight for an infinite family of graphs—for example, for the set of all cliques with an odd number of vertices.

Poljak and Turzík investigated the *reason* why bipartite subgraphs satisfy the Edwards-Erdős bound, and they abstracted out a sufficient condition for *any* graph property to have such a lower bound. They defined the notion of a λ-extendible property for $0 < \lambda < 1$, and showed that for any λ-extendible property Π, any connected graph $G = (V, E)$ contains a spanning subgraph $H = (V, F) \in \Pi$ with at least $\lambda|E| + \frac{1-\lambda}{2}(|V|-1)$ edges [24]. The property of being bipartite is λ-extendible for $\lambda = 1/2$, and so the Poljak and Turzík result implies the Edwards-Erdős bound. Other examples of λ-extendible properties—with different values of λ—include q-colorability and acyclicity in oriented graphs.

In their pioneering paper which introduced the notion of "above-guarantee" parameterization, Mahajan and Raman [20] posed the parameterized tractability of Max-Cut above its tight lower bound (Max-Cut ATLB)—*Given a connected graph G with n vertices and m edges and a parameter $k \in \mathbb{N}$, does G have a bipartite subgraph with at least $\frac{m}{2} + \frac{n-1}{4} + k$ edges?*—as an open problem. This was recently resolved by Crowston et al. who showed that Max-Cut ATLB can be solved in $2^{\mathcal{O}(k)} \cdot n^4$ time and has a kernel with $\mathcal{O}(k^5)$ vertices [7]. Following this, Mnich et al. [21] generalized the FPT result of Crowston et al. to *all* graph properties which (i) satisfy a (potentially) stronger notion which they dubbed *strong λ-extendibility*, and (ii) are FPT on a certain simple class of graphs called *almost-forests of cliques*. That is, they showed that for any strongly λ-extendible graph property Π which satisfies the simplicity criterion, the following problem—called Above Poljak-Turzík (Π), or APT(Π) for short—is FPT: *Given a connected graph G with n vertices and m edges and a parameter $k \in \mathbb{N}$, does G have a spanning subgraph $H \in \Pi$ with at least $\lambda m + \frac{1-\lambda}{2}(n-1) + k$ edges?* Problems which satisfy these conditions include Max-Cut, Oriented Max Acyclic Digraph, Max q-Colorable Subgraph and, more generally, any graph property which is equivalent to having a homomorphism to a fixed vertex-transitive graph [21].

Our Results and their Implications. Our main result is that for almost all strongly λ-extendible properties Π of (possibly oriented or edge-labelled) graphs, the Above Poljak-Turzík (Π) problem has kernels with $\mathcal{O}(k^2)$ or $\mathcal{O}(k^3)$ vertices. Here "almost all" includes the following: (i) *all* strongly λ-extendible properties for $\lambda \neq \frac{1}{2}$, (ii) *all* strongly λ-extendible properties which contain all orientations and labels (if applicable) of the graph K_3 (triangle), and (iii) all *hereditary* strongly λ-extendible properties for simple or oriented graphs. In particular, our result implies kernels with $\mathcal{O}(k^2)$ vertices for Max q-Colorable Subgraph and other problems defined by homomorphisms to vertex-transitive graphs.

We address both the questions left open by Mnich et al. [21], albeit in different ways. Firstly, we resolve the kernelization question for strongly λ-extendible properties, except for the special cases of non-hereditary $\frac{1}{2}$-extendible properties which do not contain some orientation or labelling of the triangle, or hereditary $\frac{1}{2}$-extendible properties which do not contain some labelling of the triangle. Note that for non-hereditary properties, we may expect to find kernelization very difficult, as a large subgraph with the property can disappear entirely if we delete even a small part of the graph. For the cases when the membership of the triangle depends on its labelling, we may expect the rules of kernelization to depend greatly on the family of labellings, and so it is difficult to produce a general result.

Secondly, we get rid of the simplicity criterion required by Mnich et al. Showing that a specific problem is FPT on almost-forests of cliques takes—in general—a non-trivial amount of work, as can be seen from the corresponding proofs for Max-Cut [6, Lemma 9], Oriented Max Acyclic Digraph, and having a homomorphism to a vertex transitive graph [22, Lemmas 27, 31]. Mnich et al. had proposed that a way to get around this problem was to find a logic which captures all problems which are FPT on almost-forests of cliques, and had left open the problem of finding the right logic. The proof of our main result shows that *all* strongly λ-extendible properties—save for the special cases —are FPT on almost-forests of cliques: in fact, that they have polynomial size kernels on this class of graphs. No special logic is required to capture these problems, and this answers their second open problem.

Formally, our main result is as follows:

▶ **Theorem 1.** *Let $0 < \lambda < 1$, and let Π be a strongly λ-extendible property of (possibly oriented and/or labelled) graphs. Then the* Above Poljak-Turzík (Π) *problem has a kernel on $\mathcal{O}(k^2)$ vertices if conditions 0a or 0b holds, and a kernel on $\mathcal{O}(k^3)$ vertices if only 0c holds:*

1. *$\lambda \neq \frac{1}{2}$;*
2. *All orientations and labels (if applicable) of the graph K_3 belong to Π;*
3. *Π is a hereditary property of simple or oriented graphs.*

As a corollary, we get that a number of specific problems have polynomial kernels when parameterized above their respective Poljak-Turzík bounds:

▶ **Corollary 2.** *The* Above Poljak-Turzík (Π) *parameterization of* Max q-Colorable Subgraph, *$q > 2$, has a kernel on $\mathcal{O}(k^2)$ vertices, and the* Above Poljak-Turzík (Π) *parameterization of* Oriented Max Acyclic Digraph *has a kernel on $\mathcal{O}(k^3)$ vertices.*

An outline of the proof. We now give an intuitive outline of our proof of Theorem 1. Our proof starts from a key result of Mnich et al.

▶ Proposition 1 ([21]). Let Π be a strongly λ-extendible property and let (G, k) be an instance of APT(Π). Then in polynomial time, we can either decide that (G, k) is a Yes-instance or find a set $S \subseteq V(G)$ such that $|S| < \frac{6k}{1-\lambda}$ and $G - S$ is a forest of cliques.

Proposition 1 is a classical WIN/WIN result, and either outputs that the given instance is a YES instance or outputs a set $S \subseteq V(G); |S| < \frac{6k}{1-\lambda}$. In the former case we return a trivial YES instance. In the latter case we know that $G - S$ is a forest of cliques and $|S| < \frac{6k}{1-\lambda}$; thus $G - S$ has a very special structure. For $\lambda \neq \frac{1}{2}$, or when all orientations or labels of the graph K_3 have the property, we show combinatorially that if the combined sizes of the cliques are too big then either we can get some "extra edges", or we can apply a reduction rule. We then show that the reduced instance has size polynomial in k. For $\lambda = \frac{1}{2}$, we need the extra technical condition that the property be hereditary, and defined only for simple or oriented graphs. In this case we can show that either the problem either contains (all orientation of) K_3, or is exactly Max-Cut, or that we can bound the number and sizes of the cliques. In any of these cases the problem admits a polynomial kernel.

A block of a graph G is a maximal 2-connected subgraph of G. Note that a block B of G may consist of a single vertex and no edges, if that vertex is isolated in G.

Let G, S be as described above, and let Q be the set of cut vertices of $G - S$. For any block B of $G-S$, let $B_{\mathsf{int}} = V(B) \setminus Q$ be the *interior* of B. Let $\mathcal{B}$ be the set of blocks of $G-S$. A *block neighbor* of a block B is a block B' such that $|V(B) \cap V(B')| = 1$. Given a sequence of blocks $B_0, B_1, \ldots, B_l, B_{l+1}$ in $G - S$, the subgraph induced by $V(B_1) \cup \cdots \cup V(B_l)$ is a *block path* if, for every $1 \leq i \leq l$, $V(B_i)$ contains exactly two vertices from Q and B_i has exactly two block neighbors B_{i-1} and B_{i+1}. A block B in $G - S$ is a *leaf block* if $V(B)$ contains exactly one vertex from Q. A block in $G - S$ is an *isolated block* if it has no block neighbors.

Let $\mathcal{B}_0$ and $\mathcal{B}_1$ be the set of isolated blocks and leaf blocks, respectively, contained in $\mathcal{B}$. Let $\mathcal{B}_2$ be the set of blocks $B \in \mathcal{B}$ such that $B = B_i$, $1 \leq i \leq l$, for some block path $B_0, \ldots, B_{l+1}$. Finally, let $\mathcal{B}_{\geq 3} = \mathcal{B} \setminus (\mathcal{B}_0 \cup \mathcal{B}_1 \cup \mathcal{B}_2)$.

In order to bound the number of vertices in $G-S$ it is enough to bound (i) the number of blocks, and (ii) the size of each block. When $\lambda \neq \frac{1}{2}$ or the property includes all orientation and labellings of K_3, we show (Lemma 20) that all blocks with two or more vertices have positive excess. Using this fact, we can bound the number of vertices in blocks of $\mathcal{B}_1$ or $\mathcal{B}_2$ directly, and it remains only to bound $|\mathcal{B}_0|$. In the remaining case, we have to bound each of $|\mathcal{B}_0|, |\mathcal{B}_1|, |\mathcal{B}_2|$ and the size of each block separately. We bound these numbers over a number of lemmas.

Due to space constraints, many proofs are omitted. See [8] for a full version.

2 Definitions

We use "graph" to denote simple graphs without self-loops, directions, or labels, and use standard graph terminology used by Diestel [10] for the terms which we do not explicitly define. Each edge in an *oriented* graph has one of two directions $\{<, >\}$, while each edge in a *labelled* graph has an associated label $\ell \in L$ chosen from a finite set L. A *graph property* is a subclass of the class of all (possibly labelled and/or oriented) graphs. For a labelled and/or oriented graph G, we use $U(G)$ to denote the underlying simple graph; for any graph property of simple graphs, we say that G has the property if $U(G)$ does: for instance, G is connected if $U(G)$ is. For a (possibly labelled and/or oriented) graph $G = (V, E)$ and weight function $w : E(G) \to \mathbb{R}^+$, we use $w(F)$ to denote the sum of the weights of all the edges in $F \subseteq E$. We use K_j to denote the complete simple graph on j vertices for $j \in \mathbb{N}$, and K to denote an arbitrary complete simple graph. For a graph property Π, we say that $K_j \in \Pi$ if $G \in \Pi$ for every (oriented, labelled) graph G such that $U(G) = K_j$. A connected (possibly labelled and/or oriented) graph is a *tree of cliques* if the vertex set of each block

of the graph forms a clique. We use $\mathcal{C}(G)$ to denote the set of connected components of graph G. A *forest of cliques* is a graph whose connected components are trees of cliques. We use $Q(G)$ to denote the set of cut vertices of graph G; thus G is 2-connected if and only if $Q(G) = \emptyset$.

Mnich et al. [21] defined the following variant of Poljak and Turzík's notion of λ-extendibility [24].

▶ **Definition 3.** Let $\mathcal{G}$ be a class of (possibly labelled and/or oriented) graphs and let $0 < \lambda < 1$. A graph property Π is *strongly λ-extendible* on $\mathcal{G}$ if it satisfies the following properties:

- Inclusiveness $\{G \in \mathcal{G} : U(G) \in \{K_1, K_2\}\} \subseteq \Pi$;
- Block additivity $G \in \mathcal{G}$ belongs to Π if and only if every block of G belongs to Π;
- Strong λ-subgraph extension Let $G \in \mathcal{G}$ and let (U, W) be a partition of $V(G)$, such that $G[U] \in \Pi$ and $G[W] \in \Pi$. For any weight function $w : E(G) \to \mathbb{R}^+$ there exists an $F \subseteq E(U, W)$ with $w(F) \geq \lambda w(E(U, W))$, such that $G - (E(U, W) \setminus F) \in \Pi$.

In the rest of the paper we use $\mathcal{G}$ to denote a class of (possibly labelled and/or oriented) graphs, and Π to denote an arbitrary—but fixed—strongly λ-extendible property defined on $\mathcal{G}$ for some $0 < \lambda < 1$. The focus of our work is the following "above-guarantee" parameterized problem:

Above Poljak-Turzík (Π) (APT(Π))	
Input:	A connected graph $G = (V, E)$ and an integer k.
Parameter:	k
Question:	Is there a spanning subgraph $H = (V, F) \in \Pi$ of G such that $\lvert F\rvert \geq \lambda\lvert E\rvert + \frac{1-\lambda}{2}(\lvert V\rvert - 1) + k$?

Let $G \in \mathcal{G}$. A Π-*subgraph* of G is a subgraph of G which is in Π. Let $\beta_\Pi(G)$ denote the maximum number of edges in any Π-subgraph of G, and let $\gamma_\Pi(G)$ denote the Poljak-Turzík bound on G; that is, $\gamma_\Pi(G) = \lambda|E(G)| + \frac{1-\lambda}{2}(|V(G)| - |\mathcal{C}(G)|)$. The *excess of* Π *on* G, denoted $ex_\Pi(G)$, is equal to $\beta_\Pi(G) - \gamma_\Pi(G)$. We omit the subscript Π when it is clear from the context. We use $ex(K_j)$ to denote the minimum value of $ex(G)$ for any (oriented, labelled) graph G such that $K_j = U(G)$. Thus, for example, if $ex(K_3) = t$ then any graph G with underlying graph K_3 has a Π-subgraph with at least $\gamma(G) + t$ edges, regardless of orientations or labellings on the edges of G. We say that a strongly λ-extendible property *diverges on cliques* if there exists $j \in \mathbb{N}$ such that $ex(K_j) > \frac{1-\lambda}{2}$. We say that a simple connected graph $\widetilde{K}$ is an *almost-clique* if there exists $V' \subseteq V(\widetilde{K})$ with $|V'| \leq 1$ (possibly V' is empty) such that $\widetilde{K} - V'$ is a clique. For an almost-clique $\widetilde{K}$, we use $ex(\widetilde{K})$ to denote the minimum value of $ex(G)$ for any (oriented, labelled) graph G such that $\widetilde{K} = U(G)$, and we say that $\widetilde{K} \in \Pi$ if and only if $G \in \Pi$ for every (oriented, labelled) graph G with underlying graph $\widetilde{K}$.

▶ **Definition 4.** We use AK^+_Π to denote the class of all graphs $G \in \mathcal{G}$ such that $U(G)$ is an almost-clique and $ex_\Pi(G) > 0$. For any strongly λ-extendible property which diverges on cliques, we use $\inf_{AK}$ to denote the value $\inf_{(G \in AK^+)} ex(G)$.

Note that the class AK^+_Π contains an infinite number of graphs. Hence, it could be the case that $\inf_{AK} = 0$. In the next section, we will show that for any strongly λ-extendible property which diverges on cliques, it holds that $\inf_{AK} > 0$.

3 Preliminary Results

We begin with some preliminary results. The first two lemmas state how, in two special cases, the excess of a graph G can be bounded in terms of the excesses of its subgraphs.

▶ **Lemma 5.** *Let G be a connected (possibly labelled and/or oriented) graph and v a cut vertex of G. Then $\gamma(G) = \sum_{X \in \mathcal{C}(G-\{v\})} \gamma(G[X \cup \{v\}])$ and $\beta(G) = \sum_{X \in \mathcal{C}(G-\{v\})} \beta(G[X \cup \{v\}])$.*

▶ **Lemma 6.** *Let $G \in \mathcal{G}$ be a connected graph and let $V(G) = V_1 \cup V_2$ such that $V_1 \cap V_2 = \emptyset$, $V_1 \neq \emptyset$, $V_2 \neq \emptyset$. Let c_1 be the number of components of $G[V_1]$ and c_2 be the number of components of $G[V_2]$. If $ex(G[V_1]) \geq k_1$ and $ex(G[V_2]) \geq k_2$, then $ex(G) \geq k_1 + k_2 - \frac{1-\lambda}{2}(c_1 + c_2 - 1)$.*

Proof. Observe that the size of a minimum spanning forest for $G[V_i]$ is $|V_i| - c_i$, for $i \in \{1, 2\}$, and the size of a minimum spanning tree for G is $|V_1| + |V_2| - 1$. Therefore $\gamma(G) = \gamma(G[V_1]) + \gamma(G[V_2]) + \lambda|E(V_1, V_2)| + \frac{1-\lambda}{2}(c_1 + c_2 - 1)$. Let H_i be a Π-subgraph of $G[V_i]$, for $i \in \{1, 2\}$. By the strong λ-subgraph extension property, there exists a Π-subgraph H of G such that $H = (V_1 \cup V_2, E(H_1) \cup E(H_2) \cup F)$, where $F \subseteq E(V_1, V_2)$ is such that $|F| \geq \lambda|E(V_1, V_2)|$. Therefore $\beta(G) \geq \beta(G[V_1]) + \beta(G[V_2]) + \lambda|E(V_1, V_2)|$. It follows that $ex(G) = \beta(G) - \gamma(G) \geq \beta(G[V_1]) + \beta(G[V_2]) - \gamma(G[V_1]) - \gamma(G[V_2]) - \frac{1-\lambda}{2}(c_1 + c_2 - 1) = ex(G[V_1]) + ex(G[V_2]) - \frac{1-\lambda}{2}(c_1 + c_2 - 1) \geq k_1 + k_2 - \frac{1-\lambda}{2}(c_1 + c_2 - 1)$. ◀

We now prove some useful facts about strongly λ-extendible properties which diverge on cliques. In particular, we show that for a property Π which diverges on cliques, $ex(K_j)$ increases as j increases; this motivated our choice of the name. We also show that $\inf_{AK}$ is necessarily a constant greater than 0.

▶ **Lemma 7.** *Let $ex(K_j) = a \geq \frac{1-\lambda}{2}$ for some $j \in \mathbb{N}$. Then, for every almost-clique $\widetilde{K}$ with at least $j + 1$ vertices, $ex(\widetilde{K}) \geq a - \frac{1-\lambda}{2}$.*

Proof. Let $G \in \mathcal{G}$ be a graph such that $U(G) = \widetilde{K}$, where $\widetilde{K}$ is an almost-clique with at least $j+1$ vertices. Consider a partition of $V(G)$ into two sets V_1 and V_2 such that $U(G[V_1]) = K_j$ and $G[V_2]$ is connected. Then, by Lemma 6, $ex(G) \geq a - \frac{1-\lambda}{2}$. ◀

▶ **Lemma 8.** *Let Π be a strongly λ-extendible property which diverges on cliques, and let j, a be such that $ex(K_j) = \frac{1-\lambda}{2} + a$, $a > 0$. Then $ex(K_{rj}) \geq \frac{1-\lambda}{2} + ra$ for each $r \in \mathbb{N}$. Furthermore, $\lim_{s \to +\infty} ex(K_s) = +\infty$.*

Proof. We will prove the first claim by induction. The case $r = 1$ being trivial, suppose that the claim holds for r: we will prove it for $r + 1$. Consider a partition of $V(K_{(r+1)j})$ into two sets U, with jr vertices, and W, with j vertices. It holds by hypothesis that $ex(K_{(r+1)j}[W]) \geq \frac{1-\lambda}{2} + a$ and, by induction assumption, $ex(K_{(r+1)j}[U]) \geq \frac{1-\lambda}{2} + ra$. Therefore, by Lemma 6, $ex(K_{(r+1)j}) \geq \frac{1-\lambda}{2} + (r + 1)a$.

This shows that there exists a subsequence $\{s_r\}_{r \in \mathbb{N}}$ of $\{s\}_{s \in \mathbb{N}}$ such that $ex(K_{s_r})$ is a strictly increasing function of r. To conclude the proof, it is enough to apply Lemma 7. ◀

▶ **Lemma 9.** *Let Π be a strongly λ-extendible property which diverges on cliques. Then $\inf_{AK} > 0$.*

Proof. Since Π diverges on cliques, there exists j such that $ex(K_j) \geq \frac{1-\lambda}{2} + a$, $a > 0$. Then, by Lemma 6, for every graph $G \in AK^+$ with at least $j + 1$ vertices, $ex(G) \geq a$. Therefore, $\inf_{AK} \geq \min(a, \min_{\{G \in AK^+ : |V(G)| \leq j\}} ex(G)) > 0$ (note that $\{G \in AK^+ : |V(G)| \leq j\}$ is a finite set, hence the minimum of $ex(G)$ is defined and is positive). ◀

4 Polynomial kernel for divergence

In this section we show that APT(Π) has a polynomial kernel, as long as Π diverges on cliques and all cliques with at least two vertices have positive excess. We also give some technical lemmas which will be required in the next section.

Recall the partition $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_{\geq 3}$ of the blocks of $G - S$. Since $|S| < \frac{6k}{1-\lambda}$, and the number of cut vertices in $G - S$ is bounded by the number of blocks in $G - S$, it is enough to prove upper bounds on $|\mathcal{B}_0|, |\mathcal{B}_1|, |\mathcal{B}_2|, |\mathcal{B}_{\geq 3}|$, and $|B_{\mathsf{int}}|$ for every block B in $G - S$.

▶ **Definition 10.** Let v be a cut vertex of G and let $X \subseteq V(G) \setminus \{v\}$ be such that $G[X]$ is a component of $G - \{v\}$ and the underlying graph of $G[X \cup \{v\}]$ is a 2-connected almost-clique. Then we say that $G[X \cup \{v\}]$ is a *dangling component* with root v.

It is possible that a graph G could have many dangling components, all with excess 0. This makes it possible to have large NO-instances. Therefore we require the following reduction rule, which can be run in polynomial time.

▶ Reduction Rule 1. Let $G \in \mathcal{G}$ be a connected graph with at least two 2-connected components and let G' be a dangling component. Then if $ex(G') = 0$, delete $G' - \{v\}$ (where v is root of G') and leave k the same.

▶ **Lemma 11.** *Let Π be a strongly λ-extendible property which diverges on cliques and let G be a connected graph reduced under Reduction Rule 1. Then the number of dangling components is less than $\frac{k}{\inf_{AK}}$, or the instance is a YES-instance.*

▶ **Theorem 12.** *Let Π be a strongly λ-extendible property which diverges on cliques and let G be a connected graph reduced under Reduction Rule 1. If there exists $s \in S$ such that $\sum_{B \in \mathcal{B}} |N_G(B_{int}) \cap \{s\}|$ is at least $(\frac{16}{1-\lambda} + \frac{2}{\inf_{AK}})k - 2$, then the instance is a YES-instance.*

Proof. Let $U = \{s\}$. For every block B of $G - S$ such that $|N_G(B_{\mathsf{int}}) \cap \{s\}| = 1$, pick a vertex in $N(s) \cap B_{\mathsf{int}}$ and add it to U. Since the vertices are chosen in the interior of different blocks of $G - S$, $G[U]$ is a star and therefore it is in Π by block additivity. By Lemma 5, $ex(G[U]) = \frac{1-\lambda}{2}d$, where d is the degree of s in $G[U]$. Let c be the number of components of $G - U$, and assume that U is constructed such that d is maximal and c is minimal. By Lemma 6, $ex(G) \geq \frac{1-\lambda}{2}(d - c)$.

We will now show that c is bounded. The number of components of $G - U$ which contain a vertex of $S \setminus \{s\}$ is bounded by $(|S| - 1) < \frac{6k}{1-\lambda} - 1$. In addition, the number of components of $G - S$ which contain at least two blocks from which a vertex has been added to U is at most $\frac{d}{2}$.

Now, let T be a component of $G - S$ such that in the graph G, no vertex in $T - U$ has a neighbor in $S \setminus \{s\}$ and $|U \cap V(T)| = 1$. Firstly, suppose that T contains only one block B of $G - S$. Let $\{v\} = U \cap V(T)$. Note that, by the current assumptions, $N(S \setminus \{s\}) \cap V(T) \subseteq \{v\}$. If v is the only neighbor of s in T, then it is a cut vertex in G, hence B is a dangling component of G. If s has another neighbor v' in T and v has no neighbor in S different from s, then s is a cut vertex, therefore $G[V(B) \cup \{s\}]$ is a dangling component. Finally, if v has at least two neighbors in S and s has at least another neighbor v' in T, let U' be the star obtained by replacing v with v' in U, and observe that T is connected to $S \setminus \{s\}$ in $G - U'$, contradicting the minimality of c.

Now, suppose that T contains at least two blocks of $G - S$. In this case, every block except B does not contain neighbors of S. In particular, this holds for at least one leaf block B' in T. Hence, B' is a dangling component.

This ensures that carefully choosing the vertices of U we may assume that any component of $G - U$ still contains a vertex of $S \setminus \{s\}$, or contains at least two blocks from which a vertex of U has been chosen, or contains part of a dangling component. Hence, the number of components of $G - U$ is at most $\frac{6k}{1-\lambda} - 1 + \frac{d}{2} + \frac{k}{\inf_{AK}}$.

Therefore, if $d \geq (\frac{16}{1-\lambda} + \frac{2}{\inf_{AK}})k - 2$, then $ex(G) \geq k$. ◀

The next corollary follows from the fact that any block in $\mathcal{B}_0$ or $\mathcal{B}_1$ which contains no vertices adjacent to S is a dangling component, and the number of these must be bounded since each has a positive excess.

▶ **Corollary 13.** *Let Π be a strongly λ-extendible property which diverges on cliques and let G be a connected graph reduced under Reduction Rule 1. Then $|\mathcal{B}_0| + |\mathcal{B}_1| \leq ((\frac{16}{1-\lambda} + \frac{2}{\inf_{AK}})k - 2)\frac{6k}{1-\lambda} + \frac{k}{\inf_{AK}}$, or the instance is a* YES*-instance.*

The next corollary follows from the fact that every component in $G - S$ must contain a vertex which is adjacent to S in G.

▶ **Corollary 14.** *Let Π be a strongly λ-extendible property which diverges on cliques and let G be a connected graph reduced under Reduction Rule 1. Then either $G - S$ has at most $((\frac{16}{1-\lambda} + \frac{2}{\inf_{AK}})k - 2)\frac{6k}{1-\lambda} + \frac{k}{\inf_{AK}}$ components, or the instance is a* YES*-instance.*

As $\inf_{AK} > 0$, if we have too many blocks in $G - S$ with positive excess, we have a YES-instance.

▶ **Lemma 15.** *Let Π be a strongly λ-extendible property which diverges on cliques. If $G - S$ contains at least $((\frac{16}{1-\lambda} + \frac{2}{\inf_{AK}})k - 1)\frac{6k}{\inf_{AK}(1-\lambda)} + \frac{k}{(\inf_{AK})^2} + \frac{k-1}{\inf_{AK}}$ blocks with positive excess, then the instance is a* YES*-instance.*

The next two lemmas are not used in the proof of Theorem 18, but are needed for the next section.

▶ **Lemma 16.** *Let $G \in \mathcal{G}$ be a connected graph and $S \subseteq V(G)$ be such that $G - S$ is a forest of cliques. Then $|\mathcal{B}_{\geq 3}| \leq 3|\mathcal{B}_1|$.*

▶ **Lemma 17.** *Let Π be a strongly λ-extendible property which diverges on cliques, and let j, a be such that $ex(K_j) = \frac{1-\lambda}{2} + a$, $a > 0$. If $|B_{int}| \geq \lceil \frac{4k}{a} + \frac{1-\lambda}{2a} \rceil j$ for any block B of $G - S$, then the instance is a* YES*-instance.*

▶ **Theorem 18.** *Let Π be a strongly λ-extendible property which diverges on cliques, and suppose $ex(K_i) > 0$ for all $i \geq 2$. Then APT(Π) has a kernel with at most $\mathcal{O}(k^2)$ vertices.*

Proof. Let $j \in \mathbb{N}$ be such that $ex(K_j) = \frac{1-\lambda}{2} + a$, where $a > 0$. Let $V(G-S) = V' \cup V'' \cup V'''$, where V' is the set of vertices contained in blocks with exactly one vertex, V'' is the set of vertices contained in blocks with between 2 and $j-1$ vertices and V''' is the set of vertices contained in blocks with at least j vertices (note that in general $V'' \cap V''' \neq \emptyset$). Observe that the blocks containing V' are isolated blocks, therefore by Corollary 13 there exists a constant c_1 depending only on Π such that $|V'| \leq c_1 k^2$, or the instance is a YES-instance. By Lemma 15, there exists a constant c_2 depending only on Π such that $|V''| \leq c_2(j-1)k^2$, or the instance is a YES-instance.

Now, let $\mathcal{B}''$ be the set of blocks of $G - S$ which contain at least j vertices. For every block $B \in \mathcal{B}''$, let $rj + l$ be the number of its vertices, where $0 \leq l < j$. Note that, by Lemma 8 and Lemma 6, $ex(B) \geq ra$. Let $d = \sum_{B \in \mathcal{B}''} r$ and let G'' be the union of all components of $G - S$ which contain a block in $\mathcal{B}''$. By Corollary 14, we may assume that G'' has at most

$((\frac{16}{1-\lambda}+\frac{2}{\inf_{AK}})k-2)\frac{6k}{1-\lambda}+\frac{k}{\inf_{AK}}$ components. Furthermore, by repeated use of Lemma 5, we get that $ex(G'') \geq da$. Observe that $G - G''$ has at most $|S| \leq \frac{6k}{1-\lambda}$ components: then, by Lemma 6, $ex(G) \geq da - \frac{1-\lambda}{2}(((\frac{16}{1-\lambda}+\frac{2}{\inf_{AK}})k-2)\frac{6k}{1-\lambda}+\frac{k}{\inf_{AK}}+\frac{6k}{1-\lambda}-1)$. Therefore if $d \geq ((\frac{16}{1-\lambda}+\frac{2}{\inf_{AK}})k-1)\frac{6k}{a(1-\lambda)}+\frac{k}{(a\inf_{AK})}+\frac{k-1}{a}$, the instance is a YES-instance. Otherwise, $|V'''| \leq 2dj \leq c_3jk^2$, where c_3 is a constant which depends only on Π, which means that $|V(G)| \leq \frac{6k}{1-\lambda} + (c_1 + c_2(j-1) + c_3j)k^2$. ◀

5 Kernel when $\lambda \neq \frac{1}{2}$, $K_3 \in \Pi$ or Π hereditary

We first show that if $\lambda \neq \frac{1}{2}$ or $K_3 \in \Pi$, then APT(Π) has a kernel with $O(k^2)$ vertices. This follows from Theorem 18 together with the following lemmas.

▶ **Lemma 19.** *Let Π be a strongly λ-extendible property. If $\lambda \neq \frac{1}{2}$ or $K_3 \in \Pi$, then Π diverges on cliques.*

▶ **Lemma 20.** *Let Π be a strongly λ-extendible property. If $\lambda \neq \frac{1}{2}$ or $K_3 \in \Pi$, then $ex(K_i) > 0$ for all $i \geq 2$.*

▶ **Theorem 21.** *Let Π be a strongly λ-extendible property. If $\lambda \neq \frac{1}{2}$ or $K_3 \in \Pi$, then APT(Π) has a kernel with $\mathcal{O}(k^2)$ vertices.*

We now consider the case when Π is a hereditary property. Here we limit ourselves to properties defined on simple or oriented graphs - i.e. we assume that membership in Π is independent of any edge-labelling.

Consider first simple graphs. We know by Theorem 21 that if $K_3 \in \Pi$ then we have a polynomial kernel. So consider the case when $K_3 \notin \Pi$.

▶ **Theorem 22.** *Let Π be a strongly λ-extendible property with $\lambda = \frac{1}{2}$. Suppose Π is hereditary and $G \notin \Pi$ for any $G \in \mathcal{G}$ such that $U(G) = K_3$. Then $\Pi = \{G \in \mathcal{G} : G \text{ is bipartite}\}$.*

Proof. First, assume for the sake of contradiction that Π contains a non-bipartite graph H. Then H contains an odd cycle C_l. By choosing l as small as possible we may assume that C_l is a vertex-induced subgraph of H. Then, since Π is hereditary, C_l is in Π. Note that if $l = 3$, then $U(C_3) = K_3$, so this is not the case. Consider the graph H' obtained from C_l adding a new vertex v and an edge from v to every vertex of C_l. Since both C_l and $K_1 = \{v\}$ are in Π, by the strong λ-subgraph extension property we can find a subgraph of H' which contains C_l, v and at least half of the edges between v and C_l. Since l is odd, for any choice of $\frac{l+1}{2}$ edges there are two of them, say $e_1 = vx$ and $e_2 = vy$, such that the edge xy is in C_l. Therefore, since Π is hereditary, $H'[v, x, y] \in \Pi$, which leads to a contradiction, as $U(H'[v, x, y]) = K_3$.

Now, we will show that all connected bipartite graphs are in Π, independently from any possible labelling and/or orientation. We will proceed by induction. The claim is trivially true for $j = 1, 2$. Assume $j \geq 3$ and that every bipartite graph with $l < j$ vertices is in Π. Consider any connected bipartite graph H with j vertices. Consider a vertex v such that $H' = H - \{v\}$ is connected. By induction hypothesis, $H' \in \Pi$. Consider the graph H'' obtained from H' and G_2, where G_2 is any graph in $\mathcal{G}$ with $U(G_2) = K_2$ (let $V(G_2) = \{v_1, v_2\}$), adding an edge from v_i to $w \in V(H')$ if and only if in H there is an edge from v to w. Colour red the edges from v_1 and blue the edges from v_2.

Since $G_2 \in \Pi$ by inclusiveness and $H' \in \Pi$, by the strong λ-subgraph extension property there must exist a subgraph $\widetilde{H}$ of H'' which contains G_2, H' and at least half of the edges between them. Note that the red edges are exactly half of the edges and that if $\widetilde{H}$ contains

all of them and no blue edges, then we can conclude that H is in Π by block additivity. The same holds if $\widetilde{H}$ contains every blue edge and no red edge.

If, on the contrary, $\widetilde{H}$ contains one red and one blue edge, we will show that it contains a vertex-induced cycle of odd length, which leads to a contradiction according to the first part of the proof. First, suppose that both these edges contain $w \in V(H')$: if this happens, $\widetilde{H}$ contains a cycle of length 3 as a vertex-induced subgraph.

Now, suppose $\widetilde{H}$ contains a red edge v_1w_1 and a blue edge v_2w_2. Note that w_1 and w_2 are in the same partition and, since H' is connected, there is a path from w_1 to w_2 which has even length. Together with v_1w_1, v_2w_2 and v_1v_2, this gives a cycle of odd length. Choosing the shortest path between w_1 and w_2, we may assume that the cycle is vertex-induced.

Thus, we conclude that the only possible choices for $\widetilde{H}$ are either picking the red edges or picking the blue edges, which concludes the proof. ◀

When Π is the property of being bipartite, APT(Π) is exactly the problem Max-Cut ATLB. Crowston et. al. [5] showed that Max-Cut ATLB has a kernel with $\mathcal{O}(k^3)$ vertices. Thus, Theorem 21 together with Theorem 22 give us the following result:

▶ **Theorem 23.** *Let Π be a strongly λ-extendible property on simple graphs, with $\lambda = \frac{1}{2}$, and suppose Π is hereditary. Then APT(Π) has a kernel with $\mathcal{O}(k^2)$ or $\mathcal{O}(k^3)$ vertices.*

We now consider oriented graphs. Observe that, up to isomorphism, there are two possible oriented graphs whose underlying graphs are K_3. Let $\vec{K}_3$ denote the oriented graph G with $U(G) = K_3$ whose edges form an oriented cycle. We will call $\vec{K}_3$ the *oriented* triangle. Let $\overrightarrow{K}_3$ denote the unique acyclic oriented graph G with $U(G) = K_3$. We will call $\overrightarrow{K}_3$ the *non-oriented* triangle.

If neither $\vec{K}_3$ or $\overrightarrow{K}_3$ have property Π then Theorem 22 shows that APT(Π) is equivalent to Max-Cut ATLB and again we have a $O(k^3)$-vertex kernel. If on the other hand both $\vec{K}_3$ and $\overrightarrow{K}_3$ have property Π then we have a $O(k^2)$ kernel by Theorem 21. So it remains to consider the case when exactly one of $\vec{K}_3, \overrightarrow{K}_3$ has property Π. By Lemma 24, this is only possible if $\overrightarrow{K}_3 \in \Pi, \vec{K}_3 \notin \Pi$.

▶ **Lemma 24.** *Let Π be a strongly λ-extendible property on oriented graphs, with $\lambda = \frac{1}{2}$, and suppose Π is hereditary. If $\vec{K}_3 \in \Pi$, then $\overrightarrow{K}_3 \in \Pi$.*

So now suppose $\overrightarrow{K}_3 \in \Pi, \vec{K}_3 \notin \Pi$.

▶ **Lemma 25.** *Let Π be a strongly λ-extendible property on oriented graphs, with $\lambda = \frac{1}{2}$, and suppose Π is hereditary. If $\overrightarrow{K}_3 \in \Pi$, then $ex(K_4) > \frac{1}{4}$. In particular, Π diverges on cliques.*

▶ **Lemma 26.** *Let Π be a strongly λ-extendible property on oriented graphs, with $\lambda = \frac{1}{2}$, and suppose Π is hereditary, $\vec{K}_3 \notin \Pi$ and $\overrightarrow{K}_3 \in \Pi$. Then $ex(K_j) > 0$ for every $j \neq 3$.*

Using Lemma 26 together with Corollary 13 and Lemmas 12, 15, 16 and 17, we have a bound on the size and number of all blocks in $G - S$, except for those block in $\mathcal{B}_2$ which have excess exactly 0 and have no internal vertices adjacent to S. Let $\mathcal{B}_2^0$ be this set of blocks. By Lemma 26 and the fact that $K_2, \overrightarrow{K}_3 \in \Pi$, all blocks in $\mathcal{B}_2^0$ are oriented triangles. In order to bound the number of these triangles, we require the following lemma and reduction rule. Lemma 27 bounds the number of blocks in $\mathcal{B}_2^0$ which can contain a neighbor of S (since by definition such a vertex must be a cut vertex); Reduction Rule 2 allows us to limit the remaining blocks by ensuring that no two of them can be adjacent.

▶ **Lemma 27.** *Let Q_0 denote the set of cut vertices of $G - S$ which only appear in blocks in $\mathcal{B}_2^0$. Let Π be a strongly λ-extendible property on oriented graphs, with $\lambda = \frac{1}{2}$, and suppose Π is hereditary, $\overrightarrow{K}_3 \notin \Pi$ and $\overleftrightarrow{K}_3 \in \Pi$. Let (G, k) be an instance of APT(Π) reduced by Reduction Rule 1. For any $s \in S$, if $|Q_0 \cap N(s)| \geq ((32 + \frac{2}{\inf_{AK}})k - 2)48k - \frac{4k}{\inf_{AK}} + 4k$, then the instance is a* YES*-instance.*

▶ Reduction Rule 2. Let $B_1, B_2 \in \mathcal{B}_2$ be such that $V(B_1) \cap V(B_2) = \{v\}$, $B_1 = \overrightarrow{K}_3 = B_2$, $\{v\} \cap N(S) = \emptyset$ and $(B_i)_{\mathsf{int}} \cap N(S) = \emptyset$ for $i = 1, 2$. Let $\{w_i\} = (B_i)_{\mathsf{int}}$ and $\{u_i\} = V(B_i) \setminus \{v, w_i\}$ for $i = 1, 2$. If $G - \{v\}$ is disconnected, delete v, w_1, w_2, identify u_1 and u_2 and set $k' = k$. Otherwise, delete v, w_1, w_2 and set $k' = k - \frac{1}{4}$.

Putting everything together, we can prove the following theorem.

▶ **Theorem 28.** *Let Π be a strongly λ-extendible property on oriented graphs, with $\lambda = \frac{1}{2}$, and suppose Π is hereditary. Then APT(Π) has a kernel with $\mathcal{O}(k^2)$ or $\mathcal{O}(k^3)$ vertices.*

Putting together Theorem 21, Theorem 23, and Theorem 28, we get our main result, Theorem 1.

6 Conclusion

We have succeeded in showing that APT(Π) has a polynomial kernel for nearly all strongly λ-extendible Π. The only cases in which the polynomial kernel question remains open are those in which $\lambda = \frac{1}{2}$ and either Π is not hereditary, or membership in Π depends on the labellings on edges. For the cases when $\lambda \neq \frac{1}{2}$ or Π contains all triangles, we can improve our kernel from $\mathcal{O}(k^3)$ vertices to $\mathcal{O}(k^2)$ vertices. It would be nice to show a $O(k^2)$ kernel in all cases.

The bound of Poljak and Turzík extends to edge-weighted graphs - for any strongly λ-extendible property Π and any connected graph G with weight function $w : E(G) \to \mathbb{R}^+$, there exists a subgraph H of G such that $H \in \Pi$ and $w(H) \geq \lambda w(G) + \frac{(1-\lambda)w(T)}{2}$, where T is a minimum weight spanning tree of G. The natural question following from our results is whether the weighted version of APT(Π) affords a polynomial kernel.

References

1 N. Alon, G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. Solving max-r-sat above a tight lower bound. In *SODA 2010*, pages 511–517. ACM-SIAM, 2010.

2 H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proceedings of the 4th Workshop on Parameterized and Exact Computation (IWPEC 2009)*, volume 5917 of *Lecture Notes in Computer Science*, pages 17–37, 2009.

3 H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

4 H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (Meta) Kernelization. In *FOCS 2009*, pages 629–638. IEEE, 2009.

5 R. Crowston, G. Gutin, M. Jones, and G. Muciaccia. Maximum balanced subgraph problem parameterized above lower bound. *CoRR*, abs/1212.6848, 2012.

6 R. Crowston, M. Jones, and M. Mnich. Max-cut parameterized above the Edwards-Erdős bound. *CoRR*, abs/1112.3506, 2011. http://arxiv.org/abs/1112.3506.

7 R. Crowston, M. Jones, and M. Mnich. Max-cut parameterized above the Edwards-Erdős bound. In A. Czumaj, K. Mehlhorn, A. M. Pitts, and R. Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick,*

UK, July 9–13, 2012, Proceedings, Part I, volume 7391 of *Lecture Notes in Computer Science*, pages 242–253. Springer, 2012.

8 R. Crowston, M. Jones, G. Muciaccia, G. Philip, A. Rai, and S. Saurabh. Polynomial Kernels for λ-extendible Properties Parameterized Above the Poljak-Turzík Bound. *ArXiv e-prints*, Oct. 2013.

9 H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *STOC 2010*, 2010.

10 R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2010.

11 C. S. Edwards. Some extremal properties of bipartite subgraphs. *Canadian Journal of Mathematics*, 25:475–483, 1973.

12 C. S. Edwards. An improved lower bound for the number of edges in a largest bipartite subgraph. In *Recent Advances in Graph Theory*, pages 167–181. 1975.

13 F. V. Fomin, D. Lokshtanov, N. Misra, and S. Saurabh. Planar F-deletion: Approximation, Kernelization and optimal FPT algorithms. In *FOCS*, pages 470–479, 2012.

14 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In *SODA*, pages 503–510, 2010.

15 L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *STOC 08*, pages 133–142. ACM, 2008.

16 J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT news*, 38(1):31–45, 2007.

17 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer -Verlag, 2006.

18 S. Kratsch and M. Wahlström. Compression via matroids: a randomized polynomial kernel for odd cycle transversal. In *SODA*, pages 94–103, 2012.

19 S. Kratsch and M. Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS 2012)*, pages 450–459. IEEE, 2012.

20 M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999.

21 M. Mnich, G. Philip, S. Saurabh, and O. Suchý. Beyond max-cut: λ-extendible properties parameterized above the Poljak-Turzík bound. In D. D'Souza, T. Kavitha, and J. Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 412–423. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

22 M. Mnich, G. Philip, S. Saurabh, and O. Suchý. Beyond max-cut: λ-extendible properties parameterized above the Poljak-Turzík bound. *CoRR*, abs/1207.5696, 2012. http://arxiv.org/abs/1207.5696.

23 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

24 S. Poljak and D. Turzík. A polynomial time heuristic for certain subgraph optimization problems with guaranteed worst case bound. *Discrete Mathematics*, 58(1):99–104, 1986.

25 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Communications*, pages 85–103, 1972.

On the Parameterised Complexity of String Morphism Problems

Henning Fernau[1], Markus L. Schmid[1], and Yngve Villanger[2]

1 **Fachbereich IV – Abteilung Informatikwissenschaften**
Universität Trier, Trier, Germany
{Fernau,MSchmid}@uni-trier.de
2 **Department of Informatics**
University of Bergen, Bergen, Norway
yngve.villanger@gmail.com

Abstract

Given a source string u and a target string w, to decide whether w can be obtained by applying a string morphism on u (i. e., uniformly replacing the symbols in u by strings) constitutes an NP-complete problem. For example, the target string $w :=$ `baaba` can be obtained from the source string $u :=$ `aba`, by replacing `a` and `b` in u by the strings `ba` and `a`, respectively. In this paper, we contribute to the recently started investigation of the computational complexity of the string morphism problem by studying it in the framework of parameterised complexity.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, F.4.3 Formal Languages

Keywords and phrases String Problems, String Morphisms, Parameterised Complexity, Exponential Time Hypothesis, Pattern Languages

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.55

1 Introduction

Many of the typical string problems are concerned with special kinds of string operations like, e. g., concatenating strings with each other, deleting symbols from or inserting symbols into a string or replacing symbols by other symbols or even by other strings. Among the most prominent of these string problems are *string-to-string correction*, *sequence alignment* as well as the *longest common subsequence* and *shortest common supersequence problem*. The complexity of these problems have been intensely studied, both in the classical sense as well as in the parameterised setting (see, e. g., [1, 9]).

In this work, we investigate string problems that arise from a less well-known operation on strings, i. e., mapping a *source string* u to a *target string* w by uniformly (i. e., by a mapping) replacing the symbols of u by strings. For example, we can turn the source string $u :=$ `abba` into the target string $w :=$ `bbaaaaabba` by replacing `a` and `b` of u by the strings `bba` and `aa`, respectively. On the other hand, $w' :=$ `abaaaaaabb` cannot be obtained from u in a similar way. The *string morphism problem* (denoted by StrMorph) is to decide for two given strings u and w, whether or not w can be obtained from u by this kind of operation. Due to its simple definition, variants of this $\mathcal{NP}$-complete problem can be found in many different areas of theoretical computer science. In fact, many respective results are scattered throughout the literature without pointers to each other and consulting the existing literature suggests that variants of the string morphism problem have emerged and have been investigated in different contexts without knowledge of other related work.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 55–66

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

A Brief History of the String Morphism Problem

The origin of the string morphism problem is usually traced back to the 1979 paper by Angluin [3], in which she introduced the model of *pattern languages* (the membership problem of which is essentially the string morphism problem), but, independently and at the same time, it has also been studied by Ehrenfeucht and Rozenberg in [11]. Garey and Johnson [17], by referring to a private communication with Aho and Ullman from 1977, report the $\mathcal{NP}$-completeness of the problem Regular Expression Substitution, for which, on close inspection, the string morphism problem turns out to be a natural subproblem.

Since their introduction, Angluin's pattern languages have been intensely studied in the context of learning theory and formal language theory. While questions of learnability as well as language theoretical properties were the main focus of research, results regarding the complexity of their membership problem (except of its general $\mathcal{NP}$-completeness) were sparse and only appeared as by-products (see, e. g., [3,18,20,28]). In the pattern matching community, independent of Angluin's work, the string morphism problem has been investigated in terms of a special kind of pattern matching paradigm, called parameterised pattern matching (see [2, 4, 8, 13]). The string morphism problem can also be seen as the solvability problem for word equations where one side does not contain variables (for more details on word equations, see Mateescu and Salomaa [24]). Combinatorial properties of the operation of uniformly replacing the symbols in a string by other strings are investigated in numerous other areas of theoretical computer science and discrete mathematics, such as (un-)avoidable patterns (cf. Jiang et al. [22]), the ambiguity of morphisms (cf. Freydenberger et al. [16]) and equality sets (cf. Harju and Karhumäki [19]). Last but not least, the string morphism problem can also be found in practical applications. More precisely, it constitutes a special case of the matchtest for *regular expressions with backreferences* (see, e. g., Câmpeanu et al. [6]), which nowadays are a standard element of most text editors and programming languages.

The Contribution of this Paper

A systematic study of the computational complexity of StrMorph has been taken on just recently. In [25–27], several possibilities are presented of how to restrict the structure of the source strings, such that StrMorph can be solved in polynomial time, and in [13], the $\mathcal{NP}$-completeness of a large number of strongly restricted versions of StrMorph is shown.

In the literature mentioned above, different *variants* of StrMorph are considered, each tailored to different aspects and research questions. The most common variants arise from whether or not we allow symbols to be *erased* (i. e., replaced by the empty string), whether or not we allow *constants* (also called *terminals*) in the source string, which cannot be replaced and whether or not the replacement function needs to be *injective* (i. e., different symbols cannot be replaced by the same string). While the subtle difference of whether or not symbols can also be erased has a substantial impact on decidability questions for pattern languages, the differentiation of StrMorph in an injective and a non-injective version is motivated by pattern matching tasks. In addition to these different variants of StrMorph, we can observe many natural *parameters* (in the definition of the following parameters, let u be a source string over the alphabet A and let w be a target string over the alphabet B).

- p_1: The cardinality of A.
- p_2: The length of w.
- p_3: The cardinality of B.
- p_4: The maximum length of the strings substituted for the symbols in u.
- p_5: The maximum number of occurrences of any symbol in u.

In [13], for every variant of StrMorph and for every subset of the above parameters, it is shown whether the chosen parameters can be bounded by constants such that the resulting version of StrMorph can be solved in polynomial time or whether it is still $\mathcal{NP}$-complete. For example, if the cardinality of A is bounded by a constant, then all variants of StrMorph can be solved in polynomial time. This trivially follows from the fact that there is a simple brute-force algorithm that runs in time that is exponential only in p_1. Close inspection reveals that StrMorph can also be solved in time that is exponential only in p_2 (for details, the reader is referred to [18] and also [13]). Consequently, in terms of parameterised complexity, StrMorph parameterised by p_1 or p_2 is in XP and the question arises whether these problems are in FPT. Unfortunately, as a main result of this work, we report that both of the above parametrisations are $W[1]$-hard, even if additional parameters are taken into account. More precisely, we show that the following versions of StrMorph are $W[1]$-hard:

1. *all variants* of StrMorph parameterised by (p_1, p_3, p_5),
2. *all variants* of StrMorph (except the nonerasing ones) parameterised by (p_2, p_3, p_4, p_5).

For obtaining the first result, we extend a result by Stephan et al. [29], who showed the $W[1]$-hardness for a special variant of StrMorph parameterised by p_1 and in order to prove the second result, we devise a new reduction from k-Multicoloured-Clique. With respect to the parameters defined above, this leaves only very few parameterised variants of StrMorph that could possibly be in FPT. In this regard, we report the fixed parameter tractability of the following versions of StrMorph:

3. *all variants* of StrMorph parameterised by (p_1, p_2),
4. *all variants* of StrMorph parameterised by (p_1, p_4),
5. *all nonerasing variants* of StrMorph parameterised by p_2,
6. *the nonerasing, injective variant* of StrMorph parameterised by (p_3, p_4).

The above results 1 to 6 (in conjunction with results from [13]) completely settle the fixed parameter tractability of all possible parameterised variants of StrMorph, with respect to the parameters p_1 to p_5. We complement these results by showing for all the $W[1]$-hard cases $W[1]$-membership or $W[P]$-membership.

We conclude the paper by demonstrating the unlikeness of a subexponential algorithm for an important variant of the string morphism problem by applying the *Exponential Time Hypothesis*. Due to space constraints, for most of our results we only provide proof sketches.

2 Preliminaries

Let $\mathbb{N} := \{1, 2, 3, \ldots\}$. For an arbitrary alphabet A, a *string* (*over* A) is a finite sequence of symbols from A, and ε is the *empty string*. The notation A^+ refers to the set of all non-empty strings over A, and $A^* := A^+ \cup \{\varepsilon\}$. For the *concatenation* of two strings w_1, w_2 we write $w_1\, w_2$. We say that a string $v \in A^*$ is a *substring* (or *factor*) of a string $w \in A^*$ if there are $u_1, u_2 \in A^*$ such that $w = u_1\, v\, u_2$. The powers w^i, $i \in \mathbb{N}$, of a string w are inductively defined by $w^1 := w$ and $w^i = w\, w^{i-1}$. The notation $|K|$ stands for the size of a set K or the length of a string K. By $|w|_b$, we denote the number of occurrences of $b \in A$ in w, by $w[i, j]$, we denote the factor from position i to j in w and $w[i] := w[i, i]$. Let A and B be alphabets. A mapping $h : A^* \to B^*$ with $h(u\, w) = h(u)\, h(w)$, for every $u, w \in A$, is a *morphism*. It can be easily verified that a morphism is uniquely defined by the images $h(b)$, $b \in A$. If, for every $b \in A$, $h(b) \neq \varepsilon$, then h is said to be *nonerasing*. If, for all $b, c \in A$ with $b \neq c$, $h(b) \neq \varepsilon$ and $h(c) \neq \varepsilon$ implies $h(b) \neq h(c)$, then h is E-*injective* and h is *injective* if it is E-injective and nonerasing. The *size* of a morphism h is defined by $|h| := \max\{|h(b)| \mid b \in A\}$. Let A and B

be alphabets with $B \subseteq A$. A morphism $h : A^* \to B^*$ that satisfies $h(b) = b$, for every $b \in B$, is a *substitution*.

▶ **Example 1.** Let $u_1 := x\,x\,y\,z\,y$, $u_2 := x\,\mathtt{a}\,y\,\mathtt{b}\,y\,x\,y$, $w_1 := \mathtt{abababababab}$, $w_2 := \mathtt{bacbabbacb}$ and $w_3 := \mathtt{abaabbbababab}$. The nonerasing morphism h_1 defined by $h_1(x) := h_1(y) := h_1(z) := \mathtt{ab}$ satisfies $h_1(u_1) = w_1$. However, u_1 cannot be mapped to $w_1\mathtt{b} =: w_1'$ by a nonerasing morphism. If, on the other hand, we do not restrict ourselves to nonerasing morphisms, then $h_2(u_1) = w_1'$, where h_2 is defined by $h_2(x) := h_2(y) := \varepsilon$ and $h_2(z) := w_1'$. It can be verified that $g_1(u_2) = w_2$, where g_1 is a substitution defined by $g_1(x) := \mathtt{bacb}$, $g_1(y) := \varepsilon$ and $g_2(u_2) = w_3$, where g_2 is a substitution defined by $g_2(x) := g_2(y) := \mathtt{ab}$. Furthermore, u_2 cannot be mapped to w_2 by a nonerasing substitution and u_2 cannot be mapped to w_3 by an E-injective or injective substitution.

Next, we define several variants of string morphism problems. The following is the most general string morphism problem, which shall serve as a base for the definitions of all the further restricted versions.

StrMorph
Instance: Two strings u and w over some alphabets A and B.
Question: Does there exist a morphism $h : A^* \to B^*$ with $h(u) = w$?

By StrSubst, we denote the version of StrMorph, where instead for a morphism we are looking for a substitution. By adding the prefixes NE, Inj and Ne-Inj, we denote the variants of the problems StrMorph and StrSubst, where the morphism (the substitution) needs to be nonerasing, E-injective and nonerasing injective, respectively. Let SMP be the class containing exactly these 8 variants of the string morphism problem, i. e.,

$$\text{SMP} := \{Z\text{-StrMorph}, Z\text{-StrSubst} \mid Z \in \{\varepsilon, \text{Ne}, \text{Inj}, \text{Ne-Inj}\}\}.$$

Next, we fix some notation that shall be used throughout the paper. For an instance (u, w) of one of the above defined string morphism problems, u is called the *source string*, w is called the *target string* and the respective alphabets A and B with $u \in A^*$ and $w \in B^*$ are called the *source* and *target alphabet*, respectively. From now on, the target alphabet is always denoted by Σ and the source alphabet is X for string morphism problems and $(X \cup \Sigma)$ for string substitution problems, where $X \subseteq \{x_1, x_2, x_3, \ldots\}$. The symbols in Σ are called *terminals* and the symbols in X are called *variables*. For any string $u \in (\Sigma \cup X)^*$, by $\text{var}(u)$ we refer to the set of variables occurring in u and $|u|_{\text{var}}$ is the maximum number of occurrences of a variable in u, i. e., $|u|_{\text{var}} := \max\{|u|_x \mid x \in \text{var}(u)\}$.

For the problems in SMP, we consider the following parameters: $|\text{var}(u)|$ (the number of variables in the source string), $|\Sigma|$ (the cardinality of the target alphabet), $|w|$ (the length of the target string), $|u|_{\text{var}}$ (the maximum number of occurrences of a variable) and $|h|$ (the size of the morphism or substitution). A *list of parameters* is a tuple $[\rho_1, \rho_2, \ldots, \rho_k]$, where $1 \leq k \leq 5$, $\{\rho_1, \rho_2, \ldots, \rho_k\} \subseteq \{|\text{var}(u)|, |\Sigma|, |w|, |u|_{\text{var}}, |h|\}$. For example, $[|\text{var}(u)|, |\Sigma|, |h|]$ is a list of parameters. For every $K \in \text{SMP}$ and every list L of parameters, by L-K, we denote the problem K parameterised by the parameters in L, e. g., $[|\Sigma|, |u|_{\text{var}}]$-Ne-StrMorph is the parameterised version of Ne-StrMorph, where the cardinality of the target alphabet and the maximum number of occurrences per variable are parameters. We wish to point out that all parameters but $|h|$ are implicitly given by the source and target string. In particular, for negative instances of the string morphism problems, the parameter $|h|$ is undefined. Hence, as a convention, whenever we consider parameterised problems L-K, $K \in \text{SMP}$, where L

contains $|h|$, we assume that the parameter $|h|$ is explicitly given as input along with the source and target string.

In the following, we briefly recall some of the main concepts of parameterised complexity theory (for an overview, the reader is referred to Flum and Grohe [14]). The class of fixed parameter tractable problems is denoted by FPT and in order to show fixed parameter *intractability*, we use the class $W[1]$. The Clique problem parameterised by the size of the clique is denoted by k-Clique and k-Multicoloured-Clique is the problem to decide for a graph $\mathcal{G} := (V, E)$ and a partition $V_1, V_2, \ldots, V_k$ of V, such that every V_i is an independent set, whether or not $\mathcal{G}$ has a clique of size k. It is a well-known fact that both k-Clique and k-Multicoloured-Clique are complete for $W[1]$ (with respect to parameterised reductions) [12]. Another $W[1]$-complete problem that we use is Short-NTM-Comp, i. e., to decide for a nondeterministic Turing machine M, a string w over the input alphabet of M and a parameter $k \in \mathbb{N}$ whether or not M has an accepting computation for w with at most k steps (see Cai et al. [5], Downey et al. [10], Cesati [7]). The classes of polynomial and nondeterministically polynomial time solvable problems are denoted by $\mathcal{P}$ and $\mathcal{NP}$, respectively.

In Section 4, in order to argue for the unlikeness of a subexponential algorithm, we shall apply the *Exponential Time Hypothesis* (ETH) by Impagliazzo, Paturi, and Zane [21], which, informally speaking, is the conjecture that 3SAT cannot be solved in time $2^{o(n)}$. For an introduction to ETH, the reader is referred to [15, 23].

3 The Parameterised Complexity of String Morphism Problems

In this section, we show for *every* list of parameters L and for *every* $K \in$ SMP, whether or not L-K is fixed parameter tractable or $W[1]$-hard. We start with the hardness results and then present fpt-algorithms for all the other cases. This section is concluded by showing $W[1]$-membership and $W[P]$-membership for the $W[1]$-hard cases.

3.1 $W[1]$-Hardness

As explained in Section 1, it can be easily seen that all variants of StrMorph can be solved in polynomial time if $|\mathrm{var}(u)|$ or $|w|$ is bounded by a constant. Furthermore, as shall be explained in Section 3.2, it also follows trivially that all variants of StrMorph are in FPT if parameterised by $|\mathrm{var}(u)|$ *and* $|w|$ at the same time. Hence, the most interesting question is whether this also holds if either $|\mathrm{var}(u)|$ or $|w|$ is a parameter. We shall show that this is very unlikely, since the corresponding parameterised versions of the string morphism problems are $W[1]$-hard. First, we consider the case that $|\mathrm{var}(u)|$ is a parameter and $|w|$ is not a parameter, for which we can show $W[1]$-hardness, even if $|u|_{\mathrm{var}}$ and $|\Sigma|$ are parameters, too.

▶ **Theorem 2.** *For every $K \in$ SMP, $[|\mathrm{var}(u)|, |\Sigma|, |u|_{\mathrm{var}}]$-$K$ is $W[1]$-hard.*

In [29], Stephan et al. give a reduction from k-Clique to $[|\mathrm{var}(u)|, |\Sigma|, |u|_{\mathrm{var}}]$-NE-StrSubst, which proves its $W[1]$-hardness. This reduction can be extended to a parameterised reduction for all problems $[|\mathrm{var}(u)|, |\Sigma|, |u|_{\mathrm{var}}]$-$K$, $K \in$ SMP, which implies Theorem 2.

Next, we consider the case where $|w|$ is a parameter instead of $|\mathrm{var}(u)|$. In this regard, we can state a rather strong result, i. e., the $W[1]$-hardness for all (but the nonerasing) variants of string morphism problems parameterised by all the considered parameters except $|\mathrm{var}(u)|$.

▶ **Theorem 3.** *For every $Z \in \{\mathrm{Inj}, \varepsilon\}$ and $K \in \{\mathrm{StrMorph}, \mathrm{StrSubst}\}$, the problem $[|w|, |\Sigma|, |u|_{\mathrm{var}}, |h|]$-$Z$-$K$ is $W[1]$-hard.*

It shall be explained later in Section 3.2 that the nonerasing variants of the string morphism problem are trivially fixed parameter tractable if parameterised by $|w|$.

The reductions that have been used in order to prove Theorem 2 are of no use for proving Theorem 3, since they produce target strings whose lengths depend on the size of the graph and not on the size of the clique. For the proof of Theorem 3, we utilise the problem k-Multicoloured-Clique, i. e., we define a mapping Φ that maps a given graph $\mathcal{G} := (V, E)$ and a partition $V_1, V_2, \ldots, V_k$ of V to a source string $u \in (\Sigma \cup X)^+$ and a target string $w \in \Sigma^+$, where $\Sigma := \{\mathtt{a}_{\{i,j\}} \mid 1 \leq i \leq j \leq k, i \neq j\} \cup \{\$\}$ and $X := \{x_e \mid e \in E\}$. For the sake of concreteness, we define, for every i, $1 \leq i \leq k$, $V_i := \{v_{i,1}, v_{i,2}, \ldots, v_{i,t_i}\}$. Note that, for every i, j, $1 \leq i \leq j \leq k$, $i \neq j$, the symbols $\mathtt{a}_{\{i,j\}}$ and $\mathtt{a}_{\{j,i\}}$ are considered identical. For every i, j, $1 \leq i < j \leq k$, we define

$$\overline{u}_{i,j} := \$\, x_{e_{i,j,1}}\, x_{e_{i,j,2}} \ldots x_{e_{i,j,l_{i,j}}}\, \$ \text{ and } \overline{w}_{i,j} := \$\, \mathtt{a}_{\{i,j\}}\, \$\,,$$

where $e_{i,j,1}, e_{i,j,2}, \ldots, e_{i,j,l_{i,j}}$ is an enumeration of exactly the edges between V_i and V_j. Furthermore,

$$\overline{u} := \overline{u}_{1,2}\, \overline{u}_{1,3} \ldots \overline{u}_{1,k}\, \overline{u}_{2,3}\, \overline{u}_{2,4} \ldots \overline{u}_{2,k}\, \ldots\, \overline{u}_{k-1,k}\,,$$
$$\overline{w} := \overline{w}_{1,2}\, \overline{w}_{1,3} \ldots \overline{w}_{1,k}\, \overline{w}_{2,3}\, \overline{w}_{2,4} \ldots \overline{w}_{2,k}\, \ldots\, \overline{w}_{k-1,k}\,.$$

Next, for every i, $1 \leq i \leq k$, we define a gadget $(\widetilde{u}_i, \widetilde{w}_i)$ in the following way. For every j, p, $1 \leq p \leq t_i$, $1 \leq j \leq k$, $i \neq j$, we define $\widehat{u}_{i,p,j} := x_{e_{p,j,1}}\, x_{e_{p,j,2}} \ldots x_{e_{p,j,s_{p,j}}}$, where $e_{p,j,1}, e_{p,j,2}, \ldots, e_{p,j,s_{p,j}}$ is an enumeration of exactly the edges between vertex $v_{i,p}$ and some vertex of V_j. Next, for every p, $1 \leq p \leq t_i$, we define

$$\widehat{u}_{i,p} := \widehat{u}_{i,p,1}\, \widehat{u}_{i,p,2} \ldots \widehat{u}_{i,p,i-1}\, \widehat{u}_{i,p,i+1}\, \widehat{u}_{i,p,i+2} \ldots \widehat{u}_{i,p,k}\,.$$

We are now ready to define the gadget $(\widetilde{u}_i, \widetilde{w}_i)$:

$$\widetilde{u}_i := \widehat{u}_{i,1}^2\, \widehat{u}_{i,2}^2 \ldots \widehat{u}_{i,t_i}^2,$$
$$\widetilde{w}_i := (\mathtt{a}_{\{i,1\}}\, \mathtt{a}_{\{i,2\}} \ldots \mathtt{a}_{\{i,i-1\}}\, \mathtt{a}_{\{i,i+1\}}\, \mathtt{a}_{\{i,i+2\}} \ldots \mathtt{a}_{\{i,k\}})^2\,.$$

Furthermore, we define $\widetilde{u} := \$\, \widetilde{u}_1\, \$\, \widetilde{u}_2\, \$ \ldots \$\, \widetilde{u}_k\, \$$, $\widetilde{w} := \$\, \widetilde{w}_1\, \$\, \widetilde{w}_2\, \$ \ldots \$\, \widetilde{w}_k\, \$$ and, finally, $u := \overline{u}\, \widetilde{u}$ and $w := \overline{w}\, \widetilde{w}$.

▶ **Lemma 4.** *Let $\mathcal{G} := (V, E)$ be a graph, let $V_1, V_2, \ldots, V_k$ be a partition of V, such that every V_i is an independent set and let $(u, w) := \Phi(\mathcal{G}, V_1, V_2, \ldots, V_k)$. There exists a clique of size k in $\mathcal{G}$ if and only if there exists an* E*-injective substitution h of size* 1 *with $h(u) = w$.*

Proof Sketch. Every variable $x_e \in X$ corresponds to the edge $e \in E$. Due to the gadgets $(\overline{u}_{i,j}, \overline{w}_{i,j})$, $1 \leq i < j \leq k$, for every edge e between V_i and V_j the corresponding variable can either be substituted by symbol $\mathtt{a}_{\{i,j\}}$ or by the empty string ε, but, for every i, j, $1 \leq i \leq j \leq k$, $i \neq j$, exactly one variable corresponding to an edge between V_i and V_j must be mapped to $\mathtt{a}_{\{i,j\}}$. So mapping $\overline{u}$ to $\overline{w}$ corresponds to choosing exactly one edge between each two sets V_i and V_j, $1 \leq i \leq j \leq k$, $i \neq j$. In order to conclude that these edges are the edges of a clique, we somehow have to impose the condition that, for every i, $1 \leq i \leq k$, all the chosen edges connecting a vertex from V_i to vertices from each V_j, $1 \leq j \leq k$, $i \neq j$, are adjacent to exactly the same vertex of V_i. This condition is guaranteed by the gadgets $(\widetilde{u}_i, \widetilde{w}_i)$, $1 \leq i \leq k$, which can be seen in the following way. For a fixed i, $1 \leq i \leq k$, $\widetilde{u}_i$ contains all vertices between $v_{i,1}$ and vertices in V_1, then all vertices between $v_{i,1}$ and vertices in V_2 and so on until all vertices between $v_{i,1}$ and V_k are listed (this is

represented by $\widehat{u}_{i,1}$). Then, in the same way, all vertices between vertex $v_{i,2}$ and vertices in $V_1, V_2, \ldots, V_k$ are listed, and so on. The string $\widetilde{w}_i$, i. e., the counterpart to $\widetilde{u}_i$, contains a listing of edges between V_i and the other sets V_j, $1 \leq j \leq k$, $i \neq j$. The fact that all the $\widehat{u}_{i,p}$, $1 \leq p \leq t_i$, in $\widetilde{u}_i$ are squared and $\widetilde{w}_i$ is a square, too, allows us to conclude that the whole string $\widetilde{w}_i$ must be completely generated by one $\widehat{u}_{i,p}$, for some p, $1 \leq p \leq t_i$. This means that there is actually one distinct p, $1 \leq p \leq t_i$, such that *all* the edges between V_i and the other sets V_j, $1 \leq j \leq k$, $i \neq j$, are adjacent to this vertex $v_{i,p} \in V_i$. ◀

We note that Φ is an fpt-reduction with respect to the parameters $|w|$, $|\Sigma|$, $|u|_{\text{var}}$ and $|h|$.

▶ Proposition 5. Let $\mathcal{G}$ be a graph, let $V_1, V_2, \ldots, V_k$ be a partition of V and let $(u, w) := \Phi(\mathcal{G}, V_1, V_2, \ldots, V_k)$. Then $|w|$ and $|\Sigma|$ are bounded by a function of k and $|u|_{\text{var}} = 3$.

However, the reduction Φ only works for the problems Z-StrSubst, $Z \in \{\text{Inj}, \varepsilon\}$, but it can be extended to the problems Z-StrMorph, $Z \in \{\text{Inj}, \varepsilon\}$, as well, i. e., to a mapping Φ' that maps a k-Multicoloured-Clique instance to a source string $u' \in X^+$ and a target string $w' \in \Sigma^+$. To this end let $\mathcal{G} := (V, E)$ be a graph and let $V_1, V_2, \ldots, V_k$ be a partition of V, such that every V_i is an independent set. Since Φ' is very similar to Φ, we shall only point out in which regards they differ. The main difference is that for Φ', instead of using occurrences of the symbol \$ in u, we use an occurrence of a new variable per each occurrence of \$. Furthermore, in order to maintain the E-injectivity, each of these new variables has to match its own individual symbol in w. More formally, for every i, j, $1 \leq i < j \leq k$, we define $\overline{u}_{i,j} := z_{\mathfrak{c}_{i,j}}\, x_{e_{i,j,1}}\, x_{e_{i,j,2}} \ldots x_{e_{i,j,l_{i,j}}}\, z_{\mathfrak{c}_{i,j}}$, $\overline{w}_{i,j} := \mathfrak{c}_{i,j}\, \mathtt{a}_{\{i,j\}}\, \mathfrak{c}_{i,j}$ and $\widetilde{u} := z_{\$_1}\, \widetilde{u}_1\, z_{\$_2}\, \widetilde{u}_2\, z_{\$_3} \ldots z_{\$_k}\, \widetilde{u}_k\, z_{\$_{k+1}}$, $\widetilde{w} := \$_1\, \widetilde{w}_1\, \$_2\, \widetilde{w}_2\, \$_3 \ldots \$_k\, \widetilde{w}_k\, \$_{k+1}$, where the factors $\widetilde{u}_i$ and $\widetilde{w}_i$, $1 \leq i \leq k$, are defined as in the definition of Φ. Furthermore, analogously to the definition of Φ, we define $\overline{u}$ and $\overline{w}$ to be the concatenations of the factors $\overline{u}_{i,j}$ and $\overline{w}_{i,j}$, respectively. Finally, we define $u' := z_\% \, z_{\widehat{\%}}\, z_\% \, s \, z_{\widehat{\%}}\, \overline{u}\, z_{\widehat{\%}}\, \widetilde{u}$ and $w' := \% \, \widehat{\%} \, \% \, r \, \widehat{\%} \, \overline{u} \, \widehat{\%} \, \widetilde{w}$, where s is a concatenation of all the new variables of form $z_{\mathfrak{c}_{i,j}}$ and $z_{\$_i}$ and r is the corresponding concatenation of the symbols $\mathfrak{c}_{i,j}$ and $\$_i$. We note that Lemma 4 as well as Proposition 5 still hold with respect to Φ', which concludes the proof of Theorem 3.

▶ **Lemma 6.** *Let $\mathcal{G} := (V, E)$ be a graph, let $V_1, V_2, \ldots, V_k$ be a partition of V, such that every V_i is an independent set, let $(u', w') := \Phi'(\mathcal{G}, V_1, V_2, \ldots, V_k)$ and let $(u, w) := \Phi(\mathcal{G}, V_1, V_2, \ldots, V_k)$. There exists an* E*-injective morphism h with $h(u') = w'$ if and only if there exists an* E*-injective morphism g with $g(u) = w$.*

▶ Proposition 7. Let $\mathcal{G}$ be a graph, let $V_1, V_2, \ldots, V_k$ be a partition of V and let $(u', w') := \Phi'(\mathcal{G}, V_1, V_2, \ldots, V_k)$. Then $|w'|$ and $|\Sigma|$ are bounded by a function of k and $|u'|_{\text{var}} = 3$.

3.2 Fixed Parameter Tractability

We now present two brute-force algorithms for the string morphism problems. Let u be a source string and let w be a target string. The algorithm BF-1$_{\text{StrMorph}}$ on input (u, w) works as follows. All tuples $(w_1, w_2, \ldots, w_{|\text{var}(u)|})$ of factors of w are enumerated and if for such a tuple $h(u) = w$ holds, where $h(x_i) := w_i$, $1 \leq i \leq |\text{var}(u)|$, then the output is `YES` and `NO` otherwise. Obviously, BF-1$_{\text{StrMorph}}(u, w) =$ `YES` if and only if there exists a morphism h with $h(u) = w$. In an analogous way, for every $K \in$ SMP, we can define an algorithm BF-1$_K$, which solves the problem K. We only have to make sure that, depending on the problem K, we only enumerate m-tuples of factors of w that induce an injective, a nonerasing or an injective nonerasing morphism (or substitution).

▶ Proposition 8. Let $K \in$ SMP. The runtime of BF-1$_K(u, w)$ is $\mathrm{O}\left(|u| \times |w| \times \left(|w|^2\right)^{|\text{var}(u)|}\right)$.

We now slightly change the brute-force algorithm BF-1_K from above. To this end, let u be a source string, let w be a target string over some target alphabet Σ and let $k \in \mathbb{N}$. The algorithm BF-2_{StrMorph} on input (u, w, Σ, k) works just as BF-1_{StrMorph}, but instead of enumerating all tuples $(w_1, w_2, \ldots, w_{|\text{var}(u)|})$ of factors of w, it enumerates all tuples $(w_1, w_2, \ldots, w_{|\text{var}(u)|})$ of strings over Σ with $|w_i| \leq k$, $1 \leq i \leq |\text{var}(u)|$, and checks whether one of them induces a morphism h with $h(u) = w$. It can be easily verified that BF-$2_{\text{StrMorph}}(u, w, \Sigma, k) =$ `YES` if and only if there exists a morphism h of size k with $h(u) = w$. In a similar way as done for algorithm BF-1_{StrMorph}, for every $K \in$ SMP, we can extend BF-2_{StrMorph} to BF-2_K, which solves the problem K.

▶ Proposition 9. Let $K \in$ SMP. The runtime of the algorithm BF-$2_K(u, w, \Sigma, k)$ is $\mathrm{O}\left(|u| \times k \times \left(k \times |\Sigma|^k\right)^{|\text{var}(u)|}\right)$.

By applying the above brute-force algorithms for solving the string morphism problems, we can conclude the following fpt-results:

▶ **Theorem 10.**

A. *For every $K \in$ SMP, $[|\text{var}(u)|, |w|]$-K is in FPT.*
B. *For every $Z \in \{\text{Ne}, \text{Ne-Inj}\}$ and $K \in \{\text{StrMorph}, \text{StrSubst}\}$, $[|w|]$-Z-K is in FPT.*
C. *For every $K \in \{\text{StrMorph}, \text{StrSubst}\}$, $[|h|, |\Sigma|]$-Ne-Inj-K is in FPT.*
D. *For every $K \in$ SMP, $[|\text{var}(u)|, |h|]$-K is in FPT.*

Proof Sketch. Obviously, BF-1_K is an fpt-algorithm for $[|\text{var}(u)|, |w|]$-K, $K \in$ SMP, which proves A. For the Ne variants of the string morphism problems, we can assume $|w| \geq |u| \geq |\text{var}(u)|$; thus, for every $Z \in \{\text{Ne}, \text{Ne-Inj}\}$ and $K \in \{\text{StrMorph}, \text{StrSubst}\}$, BF-$1_{Z\text{-}K}(u, w)$ has a runtime of $\mathrm{O}(|u| \times |w| \times (|w|^2)^{|w|}))$, which proves B.

For every $k \in \mathbb{N}$, there are $l := \sum_{i=1}^{k} |\Sigma|^i$ different non-empty strings over Σ with length at most k. Thus, if $|\text{var}(u)| > l$, then every morphism h with $|h| \leq k$ is necessarily non-injective and if, on the other hand, $|\text{var}(u)| < l$, then BF-$2_{\text{Ne-Inj-}K}(u, w, \Sigma, k)$, $K \in \{\text{StrMorph}, \text{StrSubst}\}$, has an fpt-runtime since $|\text{var}(u)| \leq l$. This proves C.

Finally, in order to prove D, we note that, for every $k \in \mathbb{N}$, a morphism or a substitution h of size k can introduce at most $|\text{var}(u)| \times k$ new symbols. Hence, for every $K \in$ SMP, BF-$2_K(u, w, \Gamma, k)$ solves $[|\text{var}(u)|, |h|]$-K in fpt-time, where Γ (with $|\Gamma| \leq |\text{var}(u)| \times k$) is the alphabet over which the images with respect to h are defined. ◀

We conclude this section by pointing out that the results presented in Section 3.1 and 3.2 completely settle the fixed parameter tractability of all possible parameterised variants of string morphism problems, with respect to the parameters considered in the context of this work. In order to verify this claim, it is helpful to recall these results in form of a table.

Problems	$\lvert\text{var}(u)\rvert$	$\lvert w\rvert$	$\lvert u\rvert_{\text{var}}$	$\lvert h\rvert$	$\lvert\Sigma\rvert$	Complexity	Reference
SMP	p	p	–	–	–	FPT	Thm. 10.A
{Ne, Ne-Inj}-{StrMorph, StrSubst}	–	p	–	–	–	FPT	Thm. 10.B
Ne-Inj-{StrMorph, StrSubst}	–	–	–	p	p	FPT	Thm. 10.C
SMP	p	–	–	p	–	FPT	Thm. 10.D
SMP	p	–	p	–	6	$W[1]$-hard	Thm. 2
$\{\varepsilon$, Inj$\}$-{StrMorph, StrSubst}	–	p	3	1	p	$W[1]$-hard	Thm. 3

In the above table, an entry p means that the problems denoted in the row are parameterised by the parameter in the column and an integer entry constitutes a constant bound for this parameter. We note that all the cases parameterised by both $|\text{var}(u)|$ and $|w|$ are settled

by row 1. Furthermore, all the cases parameterised by $|w|$, but not by $|\mathrm{var}(u)|$ are settled by rows 2 and 6, and all the cases parameterised by $|\mathrm{var}(u)|$, but not by $|w|$ are settled by rows 4 and 5. In order to see that all the cases parameterised neither by $|\mathrm{var}(u)|$ nor by $|w|$ are settled as well, we need to take a closer look.

From row 5, we can only conclude that as long as $|h|$ is not a parameter, then all variants are $W[1]$-hard. However, for the cases where $|h|$ is a parameter, we can only conclude from row 6 the $W[1]$-hardness for all but the NE and NE-INJ variants, and, in addition to that, from row 3 we can conclude the FPT-membership for the NE-INJ variant, where $|\Sigma|$ is a parameter, too. Consequently, for every $K \in \{\text{STRMORPH}, \text{STRSUBST}\}$ and $Z \in \{\text{NE}, \text{NE-INJ}\}$, the following cases are open: (1) $[|h|, |u|_{\mathrm{var}}, |\Sigma|]$-NE-$K$, (2) $[|h|, |\Sigma|]$-NE-K, (3) $[|h|, |u|_{\mathrm{var}}]$-$Z$-$K$ and (4) $[|h|]$-Z-K. In [13] it has been shown that the problems NE-K are $\mathcal{NP}$-complete even if the parameters $|h|$, $|u|_{\mathrm{var}}$ and $|\Sigma|$ are bounded by constants, which implies that, unless $\mathcal{P} = \mathcal{NP}$, the problems of cases (1) and (2) are not in XP; thus, they are not in FPT. The same holds for the problems Z-K with respect to parameters $|h|$ and $|u|_{\mathrm{var}}$, which, in a similar way, implies that the problems of cases (3) and (4) are not in FPT.

3.3 $W[1]$-Membership and $W[P]$-Membership

In this section, we investigate the $W[1]$-membership and $W[P]$-membership for the $W[1]$-hard variants of string morphism problems. In this regard, we first explain why the problems $[|\mathrm{var}(u)|, |u|_{\mathrm{var}}]$-$K$ and $[|w|]$-K, $K \in \mathrm{SMP}$, are in $W[1]$.

▶ **Theorem 11.** *Let $K \in \mathrm{SMP}$. The problems $[|\mathrm{var}(u)|, |u|_{\mathrm{var}}]$-$K$ and $[|w|]$-K are in $W[1]$.*

Proof Sketch. We sketch a reduction from the problem $[|\mathrm{var}(u)|, |u|_{\mathrm{var}}]$-STRSUBST to the problem SHORT-NTM-COMP, which can be extended to the other problems $[|\mathrm{var}(u)|, |u|_{\mathrm{var}}]$-$K$, $K \in \mathrm{SMP}$. To this end, let $u := u_0\, y_1\, u_1\, y_2\, u_2\, y_3 \ldots y_n\, u_n$, $y_i \in X$, $1 \leq i \leq n$, $u_j \in \Sigma^*$, $0 \leq j \leq n$, be the source string and let $w \in \Sigma^*$ be the target string. The Turing machine starts with the string $u' := y_1 y_2 \ldots y_n$ on the tape. It then guesses a subset $S \subseteq \mathrm{var}(u)$ and replaces every occurrence of a $y_i \in S$ by T_ε (which means that this variable is erased). Every occurrence of a variable $y_i \notin S$ is replaced by $T^{(i)}_{j,k}$, for some j, k, $1 \leq j \leq k \leq |w|$ (which means that the i^{th} occurrence of a variable is allocated to factor $w[j,k]$ of the target string). It now only remains to check, for every factor $T^{(i_1)}_{j_1,k_1}\, T^{l}_{\varepsilon}\, T^{(i_2)}_{j_2,k_2}$, $0 \leq l \leq n-2$, whether $u_{i_1}\, u_{i_1+1} \ldots u_{i_2-1} = w[k_1+1, j_2-1]$, and to check, for every i_1, i_2, $1 \leq i_1 < i_2 \leq n$, with $u'[i_1] = T^{(i_1)}_{j_1,k_1}$, $u'[i_2] = T^{(i_2)}_{j_2,k_2}$ and $y_{i_1} = y_{i_2}$, whether $w[j_1,k_1] = w[j_2,k_2]$. All the data that the Turing machine requires to perform these checking procedures can be computed beforehand in polynomial time by the transformation machine; thus, the number of steps of the Turing machine is bounded by $g(|u'|)$, for some function g. Since $|u'| \leq |u|_{\mathrm{var}} \times |\mathrm{var}(u)|$, the number of steps is bounded by a function of the parameters. This reduction can be easily adapted to all the other problems $[|\mathrm{var}(u)|, |u|_{\mathrm{var}}]$-$K$, $K \in \mathrm{SMP}$.

In order to reduce the problems $[|w|]$-STRSUBST to SHORT-NTM-COMP, we need a slightly different approach: the Turing machine performs its computation on the target string instead on the source string. More precisely, the Turing machine starts with the target string w on the tape, nondeterministically initialises a counter c, $1 \leq c \leq |u|$, and then moves over w from left to right. In every step, a terminal symbol is replaced by the number c and then c is nondeterministically set to a value j, $c \leq j \leq |u|$. After this procedure, the input tape contains an increasing sequence of $|w|$ numbers between 1 and $|u|$. Every consecutive sequence of occurrences of the same number i, $1 \leq i \leq |u|$, means that the corresponding factor of w is "produced" by $u[i]$ (note that this can be a variable or a terminal symbol). Now it only needs to be checked whether this allocation of factors from w to the symbols of

u induces a substitution that maps u to w. All the data required for this checking procedure can be computed beforehand in polynomial time by the transformation machine; thus, the number of steps of the Turing machine is bounded by $g(|w|)$, for some function g. It is straightforward to adapt this reduction to the other problems $[|\mathrm{var}(w)|]$-K, $K \in \mathrm{SMP}$. ◀

We wish to point out that if a problem L-K, $K \in \mathrm{SMP}$, has shown to be in $W[1]$, then this $W[1]$-membership is preserved if we add other parameters to L. Hence, Theorem 11 implies $W[1]$-completeness for all the $W[1]$-hard versions of string morphism problems, except $[|\mathrm{var}(u)|, |\Sigma|]$-$K$ and $[|\mathrm{var}(u)|]$-K, $K \in \mathrm{SMP}$, for which $W[1]$-membership does not follow, since in the above reduction, we need $|u|_{\mathrm{var}}$ as a parameter, too. However, for the problems $[|\mathrm{var}(u)|]$-K, $K \in \mathrm{SMP}$, we can show a weaker result, i. e., their $W[P]$-membership (which then also carries over to the problems $[|\mathrm{var}(u)|, |\Sigma|]$-$K$, $K \in \mathrm{SMP}$).

▶ **Theorem 12.** *For every* $K \in \mathrm{SMP}$, $[|\mathrm{var}(u)|]$-$K \in W[P]$.

Proof Sketch. A problem Π parameterised by k is in $W[P]$ if and only if there is a computable function h, a polynomial p and a nondeterministic Turing machine M deciding Π such that on every run with input w the machine M performs at most $p(|w|)$ steps, at most $h(k) \times \log(|w|)$ of them being nondeterministic (see Flum and Grohe [14]).

We only show how $[|\mathrm{var}(u)|]$-Ne-StrMorph can be solved by a Turing machine with the above properties (this procedure can be easily adapted to all other cases). Let $u \# w$ be the input, where $u = y_1\, y_2 \ldots y_m$, $y_i \in \mathrm{var}(u)$, $1 \leq i \leq m$, is the source string and w is the target string. The Turing machine nondeterministically guesses numbers $l_1, l_2, \ldots, l_{|\mathrm{var}(u)|} \in \mathbb{N}$, which induce a factorisation of w that fits to u, i. e., $w = w_1 w_2 \ldots w_m$, where, for every i, $1 \leq i \leq m$, j, $1 \leq j \leq |\mathrm{var}(u)|$, $|w_i| = l_j$ if $y_i = x_j$. The Turing machine then accepts if, for every p, q, $1 \leq p < q \leq |u|$, $y_p = y_q$ implies $w_p = w_q$, and rejects otherwise. The correctness of this procedure is obvious, it only remains to show that it satisfies the above mentioned conditions. All the tasks that need to be performed by the Turing machine can be carried out in time polynomial in $|w|$ and $|u|$. Furthermore, the only nondeterministic steps are the ones that are performed in order to guess the factorisation of w and these are $\mathrm{O}(|\mathrm{var}(u)| \times \log(|w|))$ many, since the factorisation is completely determined by the lengths of the $|\mathrm{var}(u)|$ factors and each of these lengths is bounded by $|w|$. ◀

4 A Lower Bound

For most string problems, it is a natural assumption that the alphabet Σ is fixed (in fact, it often has very small cardinality as, e. g., 2 if we are dealing with binary numbers or 4 in the case of DNA sequences). Furthermore, if we use strings with variables (i. e., source strings) for specifying a class of similar string objects (which is a typical application of strings with variables), then, for many applications, there are only finitely many string objects that can replace the variables. Hence, the problems $[|h| \leq k_1, |\Sigma| \leq k_2]$-$K$, $K \in \mathrm{SMP}$, $k_1, k_2 \in \mathbb{N}$ (i. e., the parameters $|h|$ and $|\Sigma|$ are bounded by k_1 and k_2, respectively), are of special interest. We recall that Proposition 9 demonstrates that, for every constants $k_1, k_2 \in \mathbb{N}$ and for every $K \in \mathrm{SMP}$, $[|h| \leq k_1, |\Sigma| \leq k_2]$-$K$ can be solved in time $\mathrm{O}(|u| \times k_1 \times (k_1 \times k_2^{k_1})^{|\mathrm{var}(u)|}) = |u| \times 2^{\mathrm{O}(|\mathrm{var}(u)|)}$. Next, we show that if $k_2 \geq 2$, then, for every $K \in \{\mathrm{StrMorph}, \mathrm{StrSubst}\}$, there does not exist an algorithm that solves $[|h| \leq k_1, |\Sigma| \leq k_2]$-$K$ in time $(|u||w|)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(|\mathrm{var}(u)|)}$, unless ETH fails.

To this end, we define a reduction Φ from 3SAT to StrSubst. Let $C := \{c_1, c_2, \ldots, c_m\}$ be a set of three-literal-clauses with variables $v_1, v_2, \ldots, v_n$ ($\neg v_i$ denotes the negation of v_i). We define $\overline{u} := \mathtt{¢}\, x_1\, \overline{x}_1\, \mathtt{¢}\, x_2\, \overline{x}_2\, \mathtt{¢} \ldots \mathtt{¢}\, x_n\, \overline{x}_n\, \mathtt{¢}$, $\overline{w} := \mathtt{¢}\, (\mathtt{a}\, \mathtt{¢})^n$ and, for every clause $c_i :=$

$\{p_{i_1}, p_{i_2}, p_{i_3}\}$, $1 \leq i \leq m$, we define $\widehat{u}_i := \mathtt{c}\, y_{i_1}\, y_{i_2}\, y_{i_3}\, z_{i,1}\, z_{i,2}\, \mathtt{c}\, z_{i,1}\, z_{i,2}\, z'_{i,1}\, z'_{i,2}\, \mathtt{c}$ and $\widehat{w}_i := \mathtt{c}\,\mathtt{a}\,\mathtt{a}\,\mathtt{a}\,\mathtt{c}\,\mathtt{a}\,\mathtt{a}\,\mathtt{c}$, where $y_{i_j} := x_{i_j}$ if $p_{i_j} = v_{i_j}$ and $y_{i_j} := \overline{x}_{i_j}$ if $p_{i_j} = \neg v_{i_j}$, $1 \leq j \leq 3$. Finally, $u := \overline{u}\,\widehat{u}_1\,\widehat{u}_2 \ldots \widehat{u}_m$, $w := \overline{w}\,\widehat{w}_1\,\widehat{w}_2 \ldots \widehat{w}_m$ are the source and target strings constructed by Φ.

▶ **Lemma 13.** *Let C be a 3CNF formula and let $(u, w) := \Phi(C)$. The formula C is satisfiable if and only if there exists a substitution h of size 1 with $h(u) = w$.*

We note that $\Phi(C)$ produces a source string u with $|u| = 3n + 1 + 12m$ and a target string $w \in \{\mathtt{a}, \mathtt{c}\}^*$ with $|w| = 2n + 1 + 8m$, where n is the number of Boolean variables and m is the number of clauses of C. This implies that, for every $k_1, k_2 \in \mathbb{N}$, $k_2 \geq 2$, if $[|h| \leq k_1, |\Sigma| \leq k_2]$-StrSubst can be solved in time $(|u||w|)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(|\mathrm{var}(u)|)}$, then 3SAT can be solved in time $(m+n)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(m+n)}$.

Furthermore, the reduction Φ can be extended to morphisms, i. e., to a reduction Φ' that maps a Boolean formula C with n variables and m clauses to a source string u that only contains variables, i. e., $u \in X^*$, with $|u| = \mathrm{O}(n) + \mathrm{O}(m)$. To this end, let C be a set of m three-literal-clauses with n variables and let $(u, w) := \Phi(C)$. First, we obtain a source string $u' \in X^*$ from u by substituting every occurrence of $\mathtt{c}$ by an occurrence of the new variable $y_{\mathtt{c}}$. Next, we define $u'' := y_{\mathtt{c}}\, y_{\mathtt{c}}\, (u')^2$ and $w'' := \mathtt{c}\,\mathtt{c}\,(w)^2$. Obviously, $|u''| = 2|u| + 2 = \mathrm{O}(n) + \mathrm{O}(m)$. In order to prove that Φ' is a valid reduction, it is sufficient to show that there exists a substitution h of size 1 with $h(u) = w$ if and only if there exists a morphism g of size 1 with $g(u'') = w''$. The *only if* direction is obvious, since if $h(u) = w$, then $g(u'') = w''$, where $g(x) := h(x)$, $x \in \mathrm{var}(u)$, and $g(y_{\mathtt{c}}) := \mathtt{c}$. For the *if* direction, we observe that if $g(u'') = w''$ and $g(y_{\mathtt{c}}) = \mathtt{c}$, then $g(u) = w$ holds as well. If, on the other hand, $g(y_{\mathtt{c}}) = \varepsilon$, then $g(u^2) = w''$, which is a contradiction, since w'' is not a square.

Hence, for every $k_1, k_2 \in \mathbb{N}$, $k_2 \geq 2$, if we can solve $[|h| \leq k_1, |\Sigma| \leq k_2]$-$K$, $K \in \{\text{StrMorph}, \text{StrSubst}\}$, in time $(|u||w|)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(|\mathrm{var}(u)|)}$, then 3SAT can be solved in time $(m+n)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(m+n)}$. Furthermore, by applying the *Sparsification Lemma* (see [21]), we can obtain the following result.

▶ **Theorem 14.** *For every $K \in \{\text{StrMorph}, \text{StrSubst}\}$ and $k_1, k_2 \in \mathbb{N}$, $k_2 \geq 2$, $[|h| \leq k_1, |\Sigma| \leq k_2]$-$K$ cannot be solved in time $(|u||w|)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(|\mathrm{var}(u)|)}$, unless ETH fails.*

References

1. F. N. Abu-Khzam, H. Fernau, M. A. Langston, S. Lee-Cultura, and U. Stege. A fixed-parameter algorithm for string-to-string correction. *Discrete Optimization*, 8:41–49, 2011.
2. A. Amir and I. Nor. Generalized function matching. *Journal of Discrete Algorithms*, 5:514–523, 2007.
3. D. Angluin. Finding patterns common to a set of strings. In *Proc. 11th Annual ACM Symposium on Theory of Computing, STOC 1979*, pages 130–141, 1979.
4. B. S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52:28–42, 1996.
5. L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. On the parameterized complexity of short computation and factorization. *Archive for Mathematical Logic*, 36:321–337, 1997.
6. C. Câmpeanu, K. Salomaa, and S. Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14:1007–1018, 2003.
7. M. Cesati. The Turing way to parameterized complexity. *Journal of Computer and System Sciences*, 67:654–685, 2003.
8. R. Clifford, A. W. Harrow, A. Popa, and B. Sach. Generalised matching. In *Proc. 16th International Symposium on String Processing and Information Retrieval, SPIRE 2009*, volume 5721 of *LNCS*, pages 295–301, 2009.

9 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

10 R.G. Downey, M.R. Fellows, B. Kapron, M.T. Hallett, and H.T. Wareham. Parameterized complexity of some problems in logic and linguistics (extended abstract). In *Proc. 2nd Workshop on Structural Complexity and Recursion-theoretic Methods in Logic Programming*, volume 813 of *LNCS*, pages 89–101, 1994.

11 A. Ehrenfeucht and G. Rozenberg. Finding a homomorphism between two words is NP-complete. *Information Processing Letters*, 9:86–88, 1979.

12 M. R. Fellows, D. Hermelin, F. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 401:53–61, 2009.

13 H. Fernau and M. L. Schmid. Pattern matching with variables: A multivariate complexity analysis. In *Proc. 24th Annual Symposium on Combinatorial Pattern Matching, CPM 2013*, volume 7922 of *LNCS*, pages 83–94, 2013.

14 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

15 F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. Springer, 2010.

16 D. D. Freydenberger, D. Reidenbach, and J. C. Schneider. Unambiguous morphic images of strings. *International Journal of Foundations of Computer Science*, 17:601–628, 2006.

17 M. R. Garey and D. S. Johnson. *Computers And Intractability*. W. H. Freeman and Company, 1979.

18 M. Geilke and S. Zilles. Learning relational patterns. In *Proc. 22nd International Conference on Algorithmic Learning Theory, ALT 2011*, volume 6925 of *LNCS*, pages 84–98, 2011.

19 T. Harju and J. Karhumäki. Morphisms. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 7, pages 439–510. Springer, 1997.

20 O. Ibarra, T.-C. Pong, and S. Sohn. A note on parsing pattern languages. *Pattern Recognition Letters*, 16:179–182, 1995.

21 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

22 T. Jiang, E. Kinber, A. Salomaa, K. Salomaa, and S. Yu. Pattern languages with and without erasing. *International Journal of Computer Mathematics*, 50:147–163, 1994.

23 D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *EATCS Bulletin*, 105:41–72, 2011.

24 A. Mateescu and A. Salomaa. Finite degrees of ambiguity in pattern languages. *RAIRO Informatique théoretique et Applications*, 28:233–253, 1994.

25 D. Reidenbach and M. L. Schmid. A polynomial time match test for large classes of extended regular expressions. In *Proc. 15th International Conference on Implementation and Application of Automata, CIAA 2010*, volume 6482 of *LNCS*, pages 241–250, 2011.

26 D. Reidenbach and M. L. Schmid. Patterns with bounded treewidth. In *Proc. 6th International Conference on Language and Automata Theory and Applications, LATA 2012*, volume 7183 of *LNCS*, pages 468–479, 2012.

27 M. L. Schmid. *On the Membership Problem for Pattern Languages and Related Topics*. PhD thesis, Dept. of Computer Science, Loughborough University, 2012.

28 T. Shinohara. Polynomial time inference of pattern languages and its application. In *Proc. 7th IBM Symposium on Mathematical Foundations of Computer Science*, pages 191–209, 1982.

29 F. Stephan, R. Yoshinaka, and T. Zeugmann. On the parameterised complexity of learning patterns. In *Proc. 26th International Symposium on Computer and Information Sciences, ISCIS 2011*, pages 277–281.

Partially Polynomial Kernels for Set Cover and Test Cover*

Manu Basavaraju[1], Mathew C. Francis[2], M. S. Ramanujan[3], and Saket Saurabh[1,3]

1 **University of Bergen, Norway**
manu.basavaraju@ii.uib.no
2 **Indian Statistical Institute (Chennai Centre), India**
mathew@isichennai.res.in
3 **Institute of Mathematical Sciences, India**
{msramanujan|saket}@imsc.res.in

Abstract

In a typical covering problem we are given a universe $\mathcal{U}$ of size n, a family $\mathcal{S}$ ($\mathcal{S}$ could be given implicitly) of size m and an integer k and the objective is to check whether there exists a subfamily $\mathcal{S}' \subseteq \mathcal{S}$ of size at most k satisfying some desired properties. If $\mathcal{S}'$ is required to contain all the elements of $\mathcal{U}$ then it corresponds to the classical Set Cover problem. On the other hand if we require $\mathcal{S}'$ to satisfy the property that for every pair of elements $x, y \in \mathcal{U}$ there exists a set $S \in \mathcal{S}'$ such that $|S \cap \{x, y\}| = 1$ then it corresponds to the Test Cover problem. In this paper we consider a natural parameterization of Set Cover and Test Cover. More precisely, we study the $(n-k)$-Set Cover and $(n-k)$-Test Cover problems, where the objective is to find a subfamily $\mathcal{S}'$ of size at most $n-k$ satisfying the respective properties, from the kernelization perspective. It is known in the literature that both $(n-k)$-Set Cover and $(n-k)$-Test Cover do not admit polynomial kernels (under some well known complexity theoretic assumptions). However, in this paper we show that they do admit "partially polynomial kernels". More precisely, we give polynomial time algorithms that take as input an instance $(\mathcal{U}, \mathcal{S}, k)$ of $(n-k)$-Set Cover ($(n-k)$-Test Cover) and return an equivalent instance $(\tilde{\mathcal{U}}, \tilde{\mathcal{S}}, \tilde{k})$ of $(n-k)$-Set Cover (respectively $(n-k)$-Test Cover) with $\tilde{k} \leq k$ and $|\tilde{\mathcal{U}}| = \mathcal{O}(k^2)$ ($|\tilde{\mathcal{U}}| = \mathcal{O}(k^7)$). These results allow us to generalize, improve and unify several results known in the literature. For example, these immediately imply traditional kernels when input instances satisfy certain "sparsity properties". Using a part of our kernelization algorithm for $(n-k)$-Set Cover, we also get an improved FPT algorithm for this problem which runs in time $\mathcal{O}(4^k k^{\mathcal{O}(1)}(m+n))$ improving over the previous best of $\mathcal{O}(8^{k+o(k)}(m+n)^{\mathcal{O}(1)})$. On the other hand the partially polynomial kernel for $(n-k)$-Test Cover implies the first single exponential FPT algorithm, an algorithm with running time $\mathcal{O}(2^{\mathcal{O}(k^2)}(m+n)^{\mathcal{O}(1)})$. We believe such an approach will also be useful for other covering problems as well.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Set Cover, Test Cover, Kernelization, Parameterized Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.67

* The work done by Manu Basavaraju has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 267959. Mathew C. Francis was partially supported by the INSPIRE Faculty Award of the Department of Science and Technology of the Government of India (DST) and the Institute of Mathematical Sciences. Saket Saurabh was supported by Parameterized Approximation, ERC Starting Grant 306992.

1 Introduction

The input to a covering problem consists of a universe $\mathcal{U}$ of size n, a family $\mathcal{S}$ ($\mathcal{S}$ could be given implicitly) of size m and an integer k and the objective is to check whether there exists a subfamily $\mathcal{S}' \subseteq \mathcal{S}$ of size at most k satisfying some desired properties. If $\mathcal{S}'$ is required to contain all the elements of U, then it corresponds to the classical Set Cover problem. On the other hand if we require $\mathcal{S}'$ to satisfy the property that for every pair of elements $x, y \in U$ there exists a set $S \in \mathcal{S}'$ such that $|S \cap \{x, y\}| = 1$, then it corresponds to the Test Cover problem. We study both these problems from the point of view of kernelization – a subfield of parameterized complexity.

The goal of parameterized complexity is to find ways of solving NP-hard problems more efficiently than brute force: here the aim is to restrict the combinatorial explosion to a parameter that is hopefully much smaller than the input size. Formally, a *parameterization* of a problem is assigning an integer k to each input instance and we say that a parameterized problem is *fixed-parameter tractable (FPT)* if there is an algorithm that solves the problem in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the size of the input and f is an arbitrary computable function depending on the parameter k only. For more background, the reader is referred to the following monograph [10].

In this paper, we consider a natural parameterization of Set Cover and Test Cover. More precisely, we study the $(n-k)$-Set Cover and the $(n-k)$-Test Cover problems, where the objective is to find a subfamily $\mathcal{S}'$ of size at most $n-k$ with k as the parameter. Both these questions are motivated by the fact that there is a trivial solution of size n and the fact that Set Cover is W[2]-hard when parameterized by the solution size [10]. Both these parameterizations have been studied before in the literature. Gutin et al. [8] showed that the $(n-k)$-Set Cover (or equivalently, the $(m-k)$-Hitting Set) problem has an algorithm with running time $2^{\mathcal{O}(k^2)}(m+n)^{\mathcal{O}(1)}$. This was further improved by Crowston et al. [4], who gave a deterministic algorithm for this problem running in time $(2e)^{2k+\mathcal{O}(\log k)}(m+n)^{\mathcal{O}(1)}$ and a randomized algorithm running in expected time $8^{k+\mathcal{O}(\sqrt{k})}(m+n)^{\mathcal{O}(1)}$. Crowston et al. [5] studied, among other parameterizations, the $(n-k)$-Test Cover problem with the parameter k and gave an algorithm with running time $f(k) \cdot (m+n)^{\mathcal{O}(1)}$ for this problem. The function $f(k)$ has a tower of 2's in its exponent. In another paper, Crowston et al. [3] studied this problem on set families where the size of each set is bounded by some constant r. However, in this paper our focus would be on obtaining *partially polynomial kernels* for the $(n-k)$-Set Cover and $(n-k)$-Test Cover problems.

Preprocessing (data reduction or kernelization) as a strategy of coping with hard problems is universally used in almost every implementation. The history of preprocessing, such as applying reduction rules to simplify truth functions, can be traced back to the 1950's. Today, combining tools from parameterized complexity and classical complexity it has become possible to derive upper and lower bounds on the sizes of reduced instances, or so called *kernels*. A parameterized problem is said to admit a *kernel* if there is a polynomial time algorithm (the degree of polynomial is independent of k), called a *kernelization* algorithm, that reduces the input instance to an instance with size bounded by some function $h(k)$ of k only, while preserving the answer. If the function $h(k)$ is polynomial in k, then we say that the problem admits a polynomial kernel. While positive kernelization results have appeared regularly over the last two decades, the first results establishing infeasibility of polynomial kernels for specific problems have appeared only recently. In particular, Bodlaender et al. [2] and Fortnow and Santhanam [7] have developed a framework for showing that a problem does not admit a polynomial kernel unless coNP $\subseteq$ NP/Poly, which is deemed unlikely. In

this paper, the parameterizations we consider for Set Cover and Test Cover are known to exclude polynomial kernels unless an unlikely collapse happens in complexity theory. In particular, Gutin et al. [8] showed that the $(n-k)$-Set Cover problem does not have a polynomial kernel unless coNP $\subseteq$ NP/Poly while Gutin et al. [9] showed that the $(n-k)$-Test Cover problem does not have a polynomial kernel unless coNP $\subseteq$ NP/Poly.

However, recently, a weaker notion of kernelization was introduced by Betzler et al. [1] who called it "partial" kernelization, where, instead of reducing the entire instance to one bounded by the size of the parameter as is the case with traditional kernelization algorithms, we try to bound "part" of the instance. For example, in the case of covering problems, instead of bounding the entire instance, the objective could be to give a bound on the either the universe or the family of sets as a polynomial function of k. For most covering problems, a bound on either the universe or the family of sets as a function of k immediately implies an FPT algorithm and in that respect, partially polynomial kernels are just as useful as traditional kernels.

Our contribution. In this paper, we show the existence of partially polynomial kernels for both the $(n-k)$-Set Cover and $(n-k)$-Test Cover problems by giving polynomial time algorithms which produce an equivalent instance where the size of the universe is bounded by a polynomial in the parameter. In particular, we give polynomial time algorithms that, given instances of $(n-k)$-Set Cover and $(n-k)$-Test Cover, return equivalent instances where the size of the universe is $\mathcal{O}(k^2)$ and $\mathcal{O}(k^7)$ respectively. Both our algorithms are based around computing a greedy mini-set (test) cover and using that as the starting point, we design "marking" rules with which we identify irrelevant elements and sets. The marking rules and proofs for $(n-k)$-Set Cover are relatively simpler when compared to the more complex marking rules and more involved proofs required for $(n-k)$-Test Cover. These results allow us to generalize, improve and unify several results known in the literature [8, 4, 5, 3].

Our results imply traditional kernels for "sparse instances" of these problems, that is, instances where each element appears in a constant number of sets or the bipartite element-sets incidence graph excludes some fixed graphs as an induced subgraph or every set has bounded size. We also show how we can use a part of our kernelization algorithm for $(n-k)$-Set Cover to obtain an algorithm with running time $\mathcal{O}(4^k k^{\mathcal{O}(1)}(m+n))$ which gives a significant improvement over the previously known best deterministic algorithm running in time $(2e)^{2k+\mathcal{O}(\log k)}(m+n)^{\mathcal{O}(1)}$ and the best randomized algorithm running in expected time $8^{k+\mathcal{O}(\sqrt{k})}(m+n)^{\mathcal{O}(1)}$. Finally, the partially polynomial kernel for $(n-k)$-Test Cover implies the first single exponential FPT algorithm, an algorithm with running time $\mathcal{O}(2^{\mathcal{O}(k^2)}(m+n)^{\mathcal{O}(1)})$, for the same. This significantly improves over the running time of the earlier algorithm for the problem [5]. We believe such an approach will be useful for other covering problems as well.

2 Bounded Universe kernel for $(n-k)$-Set Cover

In this section we give results for the $(n-k)$-Set Cover problem. We denote an instance of the problem by $(\mathcal{U}, \mathcal{S}, k)$ where $\mathcal{U}$ is the universe to be covered, $\mathcal{S}$ is the family of sets available and k is the parameter. Let $\mathcal{Q}(\mathcal{T})$ be the set of elements covered by the subcollection $\mathcal{T} \subseteq \mathcal{S}$, that is $\mathcal{Q}(\mathcal{T}) = \bigcup_{T \in \mathcal{T}} T$. A collection of sets $\mathcal{T}$ is said to be a *k-mini set cover* if $|\mathcal{Q}(\mathcal{T})| \geq |\mathcal{T}| + k$ and $|\mathcal{T}| \leq k$. If a sub collection $\mathcal{T} \subseteq \mathcal{S}$ is a k-mini set cover, then we say that $\mathcal{T}$ is a k-mini set cover *of* $\mathcal{S}$. The definition of mini set covers is motivated by Lemma 2 below. Note that the notion of a k-mini set cover is actually equivalent to the k-mini-hitting

set defined in [8] and a lemma equivalent to Lemma 2 is already proved in that paper. But for the sake of completeness, we restate the definition and proofs related to mini-hitting sets using the set cover terminology so that they appear natural in the context of mini-set covers. We first describe the **Greedy-mini-set**$(\mathcal{S}, k)$ algorithm which constructs a collection $\mathcal{T} \subseteq \mathcal{S}$ with certain properties, which we show imply the presence of a k-mini set cover.

Greedy-mini-set$(\mathcal{S}, k)$
Start with $\mathcal{T} = \emptyset$.
Repeat until there are no more sets in $\mathcal{S} \setminus \mathcal{T}$ that can be added to $\mathcal{T}$ or until $|\mathcal{T}| \geq k$:
If there exists $T \in \mathcal{S} \setminus \mathcal{T}$ such that $|\mathcal{Q}(\mathcal{T} \cup \{T\})| \geq |\mathcal{Q}(\mathcal{T})| + 2$, then $\mathcal{T} = \mathcal{T} \cup \{T\}$.

▶ **Lemma 1.** *If a collection of sets $\mathcal{S}$ is such that $|\mathcal{Q}(\mathcal{S})| \geq |\mathcal{S}| + k$, then $\mathcal{S}$ contains a k-mini set cover.*

Proof. Let $\mathcal{T}$ be the set returned by the algorithm **Greedy-mini-set**$(\mathcal{S}, k)$. Clearly, $|\mathcal{T}| \leq k$. We claim that $\mathcal{T}$ is a k-mini set cover. Suppose that it is not. Then, it must be the case that $|\mathcal{Q}(\mathcal{T})| - |\mathcal{T}| < k$. Then, for every set $S \in \mathcal{S} \setminus \mathcal{T}$, $|\mathcal{Q}(\{S\}) \setminus \mathcal{Q}(\mathcal{T})| \leq 1$. This implies that $|\mathcal{Q}(\mathcal{S} \setminus \mathcal{T}) \setminus \mathcal{Q}(\mathcal{T})| \leq |\mathcal{S} \setminus \mathcal{T}|$. Therefore, we have that $|\mathcal{T}| + |\mathcal{S} \setminus \mathcal{T}| > |\mathcal{Q}(\mathcal{T})| - k + |\mathcal{Q}(\mathcal{S} \setminus \mathcal{T}) \setminus \mathcal{Q}(\mathcal{T})|$, which implies that $|\mathcal{S}| > |\mathcal{Q}(\mathcal{S})| - k$, which is a contradiction. This completes the proof of the lemma. ◀

▶ **Lemma 2.** *An instance $(\mathcal{U}, \mathcal{S}, k)$ of $(n-k)$-Set Cover is a Yes instance if and only if there is a k-mini set cover $\mathcal{T} \subseteq \mathcal{S}$.*

Proof. We can assume that $\mathcal{Q}(\mathcal{S}) = \mathcal{U}$ as otherwise, there is no solution. Suppose that $\mathcal{S}$ contains a k-mini set cover $\mathcal{T}$. By definition, $|\mathcal{Q}(\mathcal{T})| \geq |\mathcal{T}| + k$. For every element $u \in \mathcal{U} \setminus \mathcal{Q}(\mathcal{T})$, add a set from $\mathcal{S}$ containing u to $\mathcal{T}$. Let $\mathcal{T}'$ denote the collection of sets so obtained. Clearly, $\mathcal{T}'$ is a set cover. Note that we add $n - |\mathcal{Q}(\mathcal{T})|$ sets to $\mathcal{T}$ to obtain $\mathcal{T}'$. Therefore, $|\mathcal{T}'| = n - |\mathcal{Q}(\mathcal{T})| + |\mathcal{T}| \leq n - k$ and hence $\mathcal{T}'$ is a set cover of size at most $n - k$ for $\mathcal{U}$.

If $\mathcal{S}$ contains a set cover $\mathcal{T}$ of size at most $n - k$, then we have $|\mathcal{T}| \leq n - k$ and $\mathcal{Q}(\mathcal{T}) = n$, and therefore by Lemma 1, $\mathcal{T}$ contains a k-mini set cover. This completes the proof of the lemma. ◀

Now we are ready to prove the two main results of this section.

▶ **Theorem 3.** *There is an algorithm that, given an instance $(\mathcal{U}, \mathcal{S}, k)$ of $(n-k)$-Set Cover, runs in polynomial time and returns an equivalent instance $(\mathcal{U}', \mathcal{S}', k)$ such that $|\mathcal{U}'| \leq 2k^2 - 2$.*

Proof. We first run the algorithm **Greedy-mini-set**$(\mathcal{S}, k)$ to construct a collection of sets $\mathcal{T}$. If $\mathcal{T}$ is a k-mini set cover, then we are done. Suppose that $\mathcal{T}$ is not a k-mini set cover. Note that if $|\mathcal{T}| \geq k$, then $\mathcal{T}$ is a k-mini set cover. So we can assume that $|\mathcal{T}| < k$. Since $\mathcal{T}$ is not a k-mini set cover, this implies that $|\mathcal{Q}(\mathcal{T})| \leq 2k - 2$.

Now any set in $\mathcal{S} \setminus \mathcal{T}$ covers at most one element from $\mathcal{U} \setminus \mathcal{Q}(\mathcal{T})$. Given $u \in \mathcal{Q}(\mathcal{T})$ and $v \in \mathcal{U} \setminus \mathcal{Q}(\mathcal{T})$, we shall denote by $\mathcal{S}_{u,v}$ all the sets in $\mathcal{S}$ that contain both u and v. We now carry out a marking procedure that marks at most $k(2k-2)$ elements from $\mathcal{U} \setminus \mathcal{Q}(\mathcal{T})$. For each element $u \in \mathcal{Q}(\mathcal{T})$, we mark k as yet unmarked elements from $\mathcal{U} \setminus \mathcal{Q}(\mathcal{T})$ such that $\mathcal{S}_{u,v} \neq \emptyset$. If there are no such k elements, we mark as many as we can. Note that during the marking procedure, no element is ever marked more than once. For an element $u \in \mathcal{Q}(\mathcal{T})$, let M_u denote the elements that are marked for u. Let $\mathcal{U}' = \mathcal{Q}(\mathcal{T}) \cup \bigcup_{u \in \mathcal{Q}(\mathcal{T})} M_u$. We shall define $\mathcal{S}' \subseteq \mathcal{S}$ to be all those sets which contain only elements from $\mathcal{U}'$ and we return the instance $(\mathcal{U}', \mathcal{S}', k)$. Note that $|\mathcal{U}'| \leq 2k^2 - 2$ and that the marking procedure can be carried

out in polynomial time. Therefore, it only remains for us to show that the instance $(\mathcal{U}', \mathcal{S}', k)$ of $(n-k)$-Set Cover is equivalent to the instance $(\mathcal{U}, \mathcal{S}, k)$ of the same problem.

Any k-mini set cover for $(\mathcal{U}', \mathcal{S}', k)$, is also clearly a k-mini set cover for $(\mathcal{U}, \mathcal{S}, k)$. To prove the other direction, we show that if there exists a k-mini set cover for $(\mathcal{U}, \mathcal{S}, k)$, then there exists a k-mini set cover for $(\mathcal{U}', \mathcal{S}', k)$.

Let $\mathcal{T}'$ be a k-mini set cover for $(\mathcal{U}, \mathcal{S}, k)$ which has the least number of sets from $\mathcal{S} \setminus \mathcal{S}'$. We shall show that $\mathcal{T}' \subseteq \mathcal{S}'$ and therefore it is also a k-mini set cover for $(\mathcal{U}', \mathcal{S}', k)$. Suppose that $\mathcal{T}' \not\subseteq \mathcal{S}'$. Pick a set $T \in \mathcal{T}'$ such that $T \notin \mathcal{S}'$. Let $W = T \cap \mathcal{Q}(\mathcal{T})$ and let z be the only element in $T \setminus \mathcal{Q}(\mathcal{T})$. Note that z exists, as otherwise $T \subseteq \mathcal{Q}(\mathcal{T}) \subseteq \mathcal{U}'$ in which case T would have been chosen in $\mathcal{S}'$. Also, $W \neq \emptyset$ because $|T| \geq 2$ (if $|T| \leq 1$, then $\mathcal{T}' \setminus \{T\}$ is a k-mini set cover with smaller number of sets from $\mathcal{S} \setminus \mathcal{S}'$).

Consider an element $u \in W$. The fact that $T \notin \mathcal{S}'$ means that $z \notin M_u$. This implies that the marking procedure marked k elements from $\mathcal{U} \setminus \mathcal{Q}(\mathcal{T})$ other than z for u. Therefore, $|M_u| = k$. Since $T \setminus \mathcal{Q}(\mathcal{T}) = \{z\}$ and $M_u \subset \mathcal{U} \setminus \mathcal{Q}(\mathcal{T})$, we have $T \cap M_u = \emptyset$. Recall that every set in $\mathcal{S}$ contains at most one element from $\mathcal{U} \setminus \mathcal{Q}(\mathcal{T})$. This means that every set in $\mathcal{S}$ contains at most one element from M_u. As $|\mathcal{T}' \setminus \{T\}| \leq k-1$, it must be the case that at least one of the elements in M_u is not covered by $\mathcal{T}'$. For every element $u \in W$, we define $r(u)$ to be an element from $M_u \setminus \mathcal{Q}(\mathcal{T}')$. Recall that for distinct $u, v \in W$, the sets M_u and M_v are disjoint by definition. This means that for distinct $u, v \in W$, $r(u) \neq r(v)$. We denote by S_u any set from the collection $\mathcal{S}_{u,r(u)}$ (recall that $\mathcal{S}_{u,r(u)}$ is the collection of sets in $\mathcal{S}$ which contain both u and $r(u)$). Note that $\mathcal{S}_{u,r(u)} \neq \emptyset$ since $r(u) \in M_u$. We now claim that the collection $\mathcal{T}'' = (\mathcal{T}' \setminus \{T\}) \cup \bigcup_{u \in W} S_u$ is also a k-mini set cover.

Note that the collection $\mathcal{T}''$ covers every element covered by the collection $\mathcal{T}'$ except z since $W \subseteq \bigcup_{u \in W} S_u \subseteq \mathcal{Q}(\mathcal{T}'')$. However, for each $u \in W$, $\mathcal{T}''$ also covers at least one element that was not covered by $\mathcal{T}'$ (recall that the element $r(u)$ is not covered by $\mathcal{T}'$). Also, since for distinct $u, v \in W$, $r(u) \neq r(v)$, we have $\left|\left(\bigcup_{u \in W} S_u\right) \setminus \mathcal{Q}(\mathcal{T}')\right| \geq |W|$. Thus, $|\mathcal{Q}(\mathcal{T}'')| \geq |\mathcal{Q}(\mathcal{T}')| - 1 + |W|$. Clearly, we have $|\mathcal{T}''| = |\mathcal{T}'| + |W| - 1$. Therefore, since $\mathcal{T}'$ was a k-mini set cover, $|\mathcal{Q}(\mathcal{T}'')| \geq |\mathcal{T}'| + k + |W| - 1 = |T''| + k$. Now, by Lemma 1, $\mathcal{T}''$ contains a k-mini set cover. Since for every $u \in W$, $S_u \in \mathcal{S}'$, this k-mini set cover uses at least one less set from $\mathcal{S} \setminus \mathcal{S}'$ than $\mathcal{T}'$, which contradicts our choice of $\mathcal{T}'$. Therefore, we conclude that $\mathcal{T}'$ is a k-mini set cover for $(\mathcal{U}', \mathcal{S}', k)$. This completes the proof of the theorem. ◀

For our next result, we need the notion of *path decomposition*. Let $G = (V, E)$ be a graph. A *path decomposition* of G is a pair $(T, \mathcal{X} = \{X_t\}_{t \in V(T)})$ where T is a path and $\mathcal{X}$ is a collection of subsets of V such that:

- $\forall e = \{u, v\} \in E,\ \exists t \in V(T) : \{u, v\} \subseteq X_t$ and
- $\forall v \in V,\ T[\{t \mid v \in X_t\}]$ is non-empty and connected.

The *width* of $(T, \mathcal{X})$ is equal to $\max\{|X_t| - 1 \mid t \in V(T)\}$ and the *pathwidth* of $G = (V, E)$, denoted by $\mathbf{pw}(G)$, is the minimum width over all path decompositions of G.

▶ **Theorem 4.** *There is an algorithm that, given an instance $(\mathcal{U}, \mathcal{S}, k)$ of $(n-k)$-Set Cover, runs in time $\mathcal{O}(4^k k^{\mathcal{O}(1)}(m+n))$ and either returns a set cover of size at most $n-k$ or correctly concludes that no such set cover exists.*

Proof. Let $\mathcal{T}$ be the set returned by the algorithm **Greedy-mini-set**$(\mathcal{S}, k)$. If $\mathcal{T}$ is a k-mini set cover, then using Lemma 2, we can construct in polynomial time and return an $(n-k)$-set cover. So let us assume that $\mathcal{T}$ is not a k-mini set cover. Note that if $|\mathcal{T}| \geq k$, then $\mathcal{T}$ is a k-mini set cover. So we can assume that $|\mathcal{T}| < k$. Since $\mathcal{T}$ is not a k-mini set cover, this implies that $|\mathcal{Q}(\mathcal{T})| \leq 2k-2$. Furthermore, we have that every set in $\mathcal{S} \setminus \mathcal{T}$ covers at most 1 element from $\mathcal{U} \setminus \mathcal{Q}(\mathcal{T})$.

We now consider the incidence bipartite graph $B = (X, Y)$ of the family $(\mathcal{U}, \mathcal{S})$ where the vertices in X correspond to $\mathcal{S}$, those in Y correspond to $\mathcal{U}$. Let Y' be those vertices in Y which correspond to $\mathcal{U} \setminus \mathcal{Q}(\mathcal{T})$. Since every set in $\mathcal{S} \setminus \mathcal{T}$ covers at most 1 element from $\mathcal{U} \setminus \mathcal{Q}(\mathcal{T})$, it is clear that in B, a vertex in X has at most one neighbour in Y'. Therefore, the induced subgraph $B' = B[X \cup Y']$ is a disjoint union of isolated vertices and stars. This implies that $\mathbf{pw}(B') \leq 1$. Therefore, we have that $\mathbf{pw}(B) \leq 2k$ since $|Y \setminus Y'| < 2k$.

Now, it only remains for us to solve the following problem. Given a set family $(\mathcal{U}, \mathcal{S})$ and an integer k such that the corresponding incidence bipartite graph B has pathwidth at most $2k$, the objective is to find the minimum set cover of this family. Observe that since X and Y are the bipartition's of B which correspond to $\mathcal{U}$ and $\mathcal{S}$ respectively, the objective is to find the smallest subset of Y which *dominates* the set X. Therefore, this is an instance of the Red-Blue Dominating Set problem on graphs of bounded pathwidth, which is known to have a $2^p(n+m)^{\mathcal{O}(1)}$ time algorithm due to [6, Lemma 6], where p is the pathwidth of the graph. In fact, the algorithm mentioned in [6, Lemma 6] can be made to work in time $\mathcal{O}(2^p p^{\mathcal{O}(1)}(m+n))$. Using the fact that $\mathcal{Q}(\mathcal{T}) < 2k$ and by making sure that we do only one pass through $\mathcal{S}$, we can obtain $\mathcal{T}$ and $\mathcal{Q}(\mathcal{T})$ in time $\mathcal{O}(k(m+n))$ and thus the path decomposition of B of width at most $2k$ can be obtained in time $\mathcal{O}(k(m+n))$. Therefore, using the algorithm mentioned in [6], we get an algorithm for $(n-k)$-Set Cover running in time $\mathcal{O}(4^k k^{\mathcal{O}(1)}(m+n))$. This completes the proof of the theorem. ◀

Since $(n-k)$-Set Cover is parameterized equivalent to $(m-k)$-Hitting Set we get the following result as a corollary to Theorem 4.

▶ **Corollary 5.** $(m-k)$-Hitting Set *can be solved in time* $\mathcal{O}(4^k k^{\mathcal{O}(1)}(m+n))$.

The algorithm mentioned in Corollary 5 improves the known (deterministic as well as randomized) algorithms for $(m-k)$-Hitting Set obtained in [4]. This algorithm was used as a subroutine in [4] to obtain a FPT algorithm for $(\nu(F)+k)$-SAT. In $(\nu(F)+k)$-SAT we are a given a CNF formula Φ and a positive integer k and the objective is to test whether there exists an assignment to its variables that satisfy at least $(\nu(F)+k)$ clauses of Φ. Here, $\nu(F)$ denotes the size of maximum matching in the variable clause incidence graph of Φ. Since it is the algorithm for $(m-k)$-Hitting Set that dominates the running time of the algorithm for $(\nu(F)+k)$-SAT in [4], we get the following result for $(\nu(F)+k)$-SAT using the Corollary 5.

▶ **Corollary 6.** $(\nu(F)+k)$-SAT *can be solved in time* $\mathcal{O}(4^k(m+n)^{\mathcal{O}(1)})$.

3 Bounded Universe Kernel for $(n-k)$-Test Cover

We start by recalling the problem definition. Given a universe $\mathcal{U}$, we shall call a set $T \subseteq \mathcal{U}$ a *test*. A test T is said to *separate* $u, v \in \mathcal{U}$ if $|T \cap \{u, v\}| = 1$. A collection of tests $\mathcal{T}$ is a *test cover* for $\mathcal{U}$, if for any $u, v \in \mathcal{U}$, there exists a test $T \in \mathcal{T}$ that separates u and v. Given a universe of size n and a collection of tests $\mathcal{S}$, the $(n-k)$-Test Cover problem is the problem of deciding whether $\mathcal{S}$ contains a test cover of size at most $n-k$. We denote an instance of the problem by $(\mathcal{U}, \mathcal{S}, k)$.

3.1 Starting with the Greedy Algorithm

Given a collection of tests $\mathcal{T}$, define the equivalence relation $R_{\mathcal{T}}$ on $\mathcal{U}$ as follows: $uR_{\mathcal{T}}v$ if there is no test in $\mathcal{T}$ that separates u and v. The equivalence classes of $R_{\mathcal{T}}$ are said to be the *classes induced by* $\mathcal{T}$ in $\mathcal{U}$. It is easy to see that the classes induced by $\mathcal{T}$ form a partition

of $\mathcal{U}$. Note that if $\mathcal{T}$ is a test cover, then $\mathcal{T}$ induces exactly n classes in $\mathcal{U}$. We denote by $\mathcal{C}_{\mathcal{U}}(\mathcal{T})$ the set of classes induced by $\mathcal{T}$ in $\mathcal{U}$. When the universe $\mathcal{U}$ is clear from the context, we abbreviate this to just $\mathcal{C}(\mathcal{T})$.

For any test $T \subseteq \mathcal{U}$, we shall define $\overline{T}$ to be $\mathcal{U} \setminus T$. The following observation follows directly from the fact for any test T, both T and $\overline{T}$ separate exactly the same pairs of elements in $\mathcal{U}$.

▶ **Observation 1.** *For any collection of tests $\mathcal{T}$ and $T \in \mathcal{T}$, $\mathcal{C}(\mathcal{T}) = \mathcal{C}((\mathcal{T} \setminus \{T\}) \cup \overline{T})$.*

A collection of tests $\mathcal{T}$ is said to be a *k-mini test cover* if $|\mathcal{C}(\mathcal{T})| \geq |\mathcal{T}| + k$ and $|\mathcal{T}| \leq 2k$. We first describe the **Greedy-mini-test$(\mathcal{S}, k)$** algorithm from [5]. Like before, this algorithm constructs a collection $\mathcal{T}$ of tests from $\mathcal{S}$ which has some special properties.

Greedy-mini-test$(\mathcal{S}, k)$
Start with $\mathcal{T} = \emptyset$.
Repeat until there are no more tests in $\mathcal{S} \setminus \mathcal{T}$ that can be added to $\mathcal{T}$ or until $|\mathcal{T}| \geq 2k - 2$:
If there exists $T \in \mathcal{S} \setminus \mathcal{T}$ such that $|\mathcal{C}(\mathcal{T} \cup \{T\})| \geq |\mathcal{C}(\mathcal{T})| + 2$, then $\mathcal{T} = \mathcal{T} \cup \{T\}$.
Else if there exist $T_1, T_2 \in \mathcal{S} \setminus \mathcal{T}$, such that $|\mathcal{C}(\mathcal{T} \cup \{T_1, T_2\})| \geq |\mathcal{C}(\mathcal{T})| + 3$, then $\mathcal{T} = \mathcal{T} \cup \{T_1, T_2\}$.

Given a subcollection $\mathcal{T} \subseteq \mathcal{S}$, we say that $(\mathcal{S}, \mathcal{T})$ is *saturated* if for any test $T \in \mathcal{S} \setminus \mathcal{T}$, $|\mathcal{C}(\mathcal{T} \cup \{T\})| \leq |\mathcal{C}(\mathcal{T})| + 1$ and for any pair of tests $T_1, T_2 \in \mathcal{S} \setminus \mathcal{T}$, $|\mathcal{C}(\mathcal{T} \cup \{T_1, T_2\}| \leq |\mathcal{C}(\mathcal{T})| + 2$. Note that if $\mathcal{T}$ was a collection of tests chosen by **Greedy-mini-test(S, k)**, then $(\mathcal{S}, \mathcal{T})$ is saturated.

▶ **Observation 2.** *Let $(\mathcal{S}, \mathcal{T})$ be saturated. Let $T \in \mathcal{S} \setminus \mathcal{T}$ and $\tilde{S} = (\mathcal{S} \cup \{\mathcal{U} \setminus T\}) \setminus \{T\}$. Then, $(\tilde{S}, \mathcal{T})$ is also saturated.*

Observation 2 follows directly from Observation 1. The following lemmas, that appeared in [5], can be seen to have proofs similar to those of Lemmas 1 and 2.

▶ **Lemma 7** ([5])**.** *If a collection of tests $\mathcal{S}$ is such that $|\mathcal{C}(\mathcal{S})| \geq |\mathcal{S}| + k$, then $\mathcal{S}$ contains a k-mini test cover.*

▶ **Lemma 8** ([5])**.** *A collection $\mathcal{S}$ of tests contains a test cover of size at most $n - k$ if and only if it contains a k-mini test cover.*

Let $\mathcal{T} \subseteq \mathcal{S}$ be a collection of tests. Now we say that any test $T \in \mathcal{S} \setminus \mathcal{T}$ intersects a class R in $\mathcal{C}(\mathcal{T})$ *trivially* if $T \cap R = R$ or $T \cap R = \emptyset$. Otherwise, we say that the intersection of T with R is *non-trivial*. If T has a non-trivial intersection with R, then consider the two sets $A = R \cap T$ and $B = R \setminus T$. It is easy to see that no two elements u, v such that $u, v \in A$ or $u, v \in B$ are separated by any test in $\mathcal{T} \cup \{T\}$ and that any two elements u, v such that $u \in A$ and $v \in B$ are separated by $T \in \mathcal{T} \cup \{T\}$, implying that A and B are two classes in $\mathcal{C}(\mathcal{T} \cup \{T\})$. Note that $R = A \cup B$. We therefore say that the class R of $\mathcal{C}(\mathcal{T})$ is *split* into the classes A and B of $\mathcal{C}(\mathcal{T} \cup \{T\})$.

▶ **Observation 3.** *For any collection of tests $\mathcal{T}$, and a test $T \in \mathcal{S} \setminus \mathcal{T}$, any class R of $\mathcal{C}(\mathcal{T})$ is split into at most two classes of $\mathcal{C}(\mathcal{T} \cup \{T\})$. In particular, the classes of $\mathcal{C}(\mathcal{T})$ which have trivial intersections with $\mathcal{T}$ are also classes of $\mathcal{C}(\mathcal{T} \cup \{T\})$, whereas the classes of $\mathcal{C}(\mathcal{T})$ with non-trivial intersections with T are split into two classes of $\mathcal{C}(\mathcal{T} \cup \{T\})$.*

3.2 Outer Phase of the Kernelization Algorithm

Run **Greedy-mini-test$(\mathcal{S}, k)$**. Let $\mathcal{T}$ be the tests that were chosen by the greedy algorithm. Clearly, $(\mathcal{S}, \mathcal{T})$ is saturated. If $\mathcal{T}$ is a k-mini test cover, we are done. Suppose that $\mathcal{T}$ is

not a k-mini test cover. Observe that in the first iteration of the algorithm, two tests get added together, creating four classes. After that each iteration adds at least $\frac{3}{2}x$ classes where x is the number of tests added to $\mathcal{T}$ in that iteration. Therefore, we can conclude that $|\mathcal{C}(\mathcal{T})| \geq \frac{3}{2}(|\mathcal{T}|-2)+4 = |\mathcal{T}| + \frac{|\mathcal{T}|+2}{2}$. This shows that when $|\mathcal{T}| \geq 2k-2$, $\mathcal{T}$ is a k-mini test cover. Therefore, as $\mathcal{T}$ is not a k-mini test cover, we have $|\mathcal{T}| \leq 2k-3$ and $|\mathcal{C}(\mathcal{T})| \leq 3k-4$.

We shall call the classes in $\mathcal{C}(\mathcal{T})$ ***regions***.

▶ **Observation 4.** $(*)$ [1] *Any test T can have non-trivial intersection with at most one region.*

Given a region R, a strict subset X of R is said to be a ***chunk*** in R if there exists a test $T \in \mathcal{S}$ such that $R \cap T = X$. Let $T \in \mathcal{S} \setminus \mathcal{T}$ and $R \in \mathcal{C}(\mathcal{T})$. Let $X = R \cap T$ be a chunk in a region R such that $|X| > |R|/2$. We construct a new collection $\tilde{S}$ of tests by replacing T with $\mathcal{U} \setminus T$ in $\mathcal{S}$. By Observation 2, $(\tilde{S}, \mathcal{T})$ is saturated. Now, we set $\mathcal{S} = \tilde{S}$. We repeat this procedure for every such $T \in \mathcal{S} \setminus \mathcal{T}$. After this is done, $(\mathcal{S}, \mathcal{T})$ is still saturated and therefore satisfies Observation 4 and moreover, $\mathcal{S}$ has the property that for any test $T \in \mathcal{S} \setminus \mathcal{T}$ and for any region $R \in \mathcal{C}(\mathcal{T})$, $|T \cap R| \leq |R|/2$.

▶ **Observation 5.** *Given any two chunks $X_1 = R \cap T_1$ and $X_2 = R \cap T_2$ where $T_1, T_2 \in \mathcal{S} \setminus \mathcal{T}$, then either one of them is a subset of the other or the two are disjoint.*

Proof. Suppose for the sake of contradiction that $X_1 \cap X_2 \neq \emptyset$ and $X_1 \setminus X_2 \neq \emptyset$ and $X_2 \setminus X_1 \neq \emptyset$. Note that, by the argument given above, we can assume that $|X_1| \leq |R|/2$ and $|X_2| \leq |R|/2$. Since $X_1 \cap X_2 \neq \emptyset$, this means that $X_1 \cup X_2 \neq R$, or in other words, $R \setminus (X_1 \cup X_2) \neq \emptyset$. This, together with the fact that $X_1 \cap X_2 \neq \emptyset$, $X_1 \setminus X_2 \neq \emptyset$ and $X_2 \setminus X_1 \neq \emptyset$ implies that $|\mathcal{C}(\mathcal{T} \cup \{T_1, T_2\})| = |\mathcal{C}(\mathcal{T})| + 3$. This contradicts the fact that $(\mathcal{S}, \mathcal{T})$ was saturated. ◀

3.3 A step of the kernelization algorithm applied to a region

Let $R_1, R_2, \ldots, R_t$ be the regions in $\mathcal{C}(\mathcal{T})$. We now restrict our attention to a region R and define the set of indices $I = \{i \colon 1 \leq i \leq t \text{ and } R_i \neq R\}$. We shall denote by $P' = (I \times I) \setminus \{(i,i) \colon i \in I\}$ the set of pairs (i,j) from I with $i \neq j$. We define the set P as $P = P' \cup \{(0,0), (-1,-1)\}$.

We say that a chunk X in R is an (i,j)***-chunk***, where $(i,j) \notin \{(0,0), (-1,-1)\}$, if there exists a test T with $R \cap T = X$ and $T \cap R_i = R_i$ and $T \cap R_j = \emptyset$. We call a chunk X in R a $(0,0)$***-chunk*** if there exists a test T with $R \cap T = X$ and $T \cap R_i = R_i$, for all $R_i \neq R$. And we call a chunk X in R a $(-1,-1)$***-chunk*** if there exists a test T with $R \cap T = X$ and $T \cap R_i = \emptyset$, for all $R_i \neq R$. We now start a marking procedure in which we mark some of the chunks in R with pairs from P. The marking is done in two phases.

Phase I. In this phase, we use only the labels from $P \setminus \{(-1,-1)\}$. Each chunk is considered for marking at most once. A chunk that has already been considered for marking is called ***checked***. We choose for checking a chunk such that every chunk contained in it is both checked and unmarked. We mark this chunk with a label "(i,j)" such that this chunk is an (i,j)-chunk and there are not already $6k+3$ chunks marked with the label "(i,j)". If such a label does not exist, we leave this chunk unmarked. In either case, this chunk has now become a checked chunk. This procedure goes on until there are no more chunks to check or until each label in $P \setminus \{(-1,-1)\}$ has been marked on $6k+3$ chunks.

[1] Due to lack of space, proofs of results marked $(*)$ are omitted.

Note that a chunk will be marked with at most one pair. Also note that no chunk marked with some pair "(i, j)" will strictly contain a marked chunk. The chunks that have been marked after this phase of the marking procedure shall be called **solid chunks**. In the rest of the paper, we use the term ***irrelevant chunk*** to refer to a chunk that does not contain any solid chunk. Note that by definition, irrelevant chunks are unmarked and it shall shortly be clear that they remain unmarked even after the second phase of marking.

▶ **Observation 6.** $(*)$ *If X is an irrelevant (i, j)-chunk, then there exist $6k + 3$ solid chunks that are marked "(i, j)" each of which is disjoint from X.*

Phase II. A ***wrapper*** is a chunk that strictly contains some solid chunk. We say that two wrappers belong to the same ***level*** if they contain the same solid chunks. Note that by Observation 5, all wrappers of the same level form a chain under inclusion. We now start the second phase of marking chunks. In this phase, we mark wrappers in each level separately. In each level, for each pair (i, j) from P, we mark with "(i, j)" the $2k + 1$ smallest and $2k + 1$ largest unmarked (i, j)-wrappers in the chain formed by (i, j)-wrappers in that level. If for some pair (i, j), there are no $4k + 2$ unmarked (i, j)-wrappers in that level, we mark all the unmarked (i, j)-wrappers in that level. Note that there can be at most one pair marked on any wrapper.

3.3.1 Pruning the Universe and the Family with respect to a region

From $\mathcal{S}$, delete every test T such that $T \cap R$ is an unmarked chunk. Let $\mathcal{S}' \subseteq \mathcal{S}$ denote the resulting collection of tests. Observe that $\mathcal{T} \subseteq \mathcal{S}'$.

Note that the chunks that remain in R are all marked chunks. Now, from $\mathcal{U}$, we delete elements as follows. For every solid chunk, we retain one element in it and delete all other elements. For every deleted element v in a solid chunk, we define $rep(v)$ to be the vertex that is retained from that chunk.

▶ **Claim 1.** *If v is an element that was deleted from a solid chunk X, then there is no test in $\mathcal{S}'$ that separates v and $rep(v)$.*

Proof. Let x and y be any two elements in X. Suppose there is a test $T \in \mathcal{S}'$ such that $x \in T$ and $y \notin T$, then it means that $T \cap R$ is a chunk that contains x and not y. Therefore, $T \cap R$ is a strict subset of X (because of Observation 5). But observe that $T \cap R$ is a marked chunk. This contradicts the fact that X was a solid chunk. Therefore there is no test $T \in \mathcal{S}'$ that separates any two elements of X. Thus, we can infer that there is no test in $\mathcal{S}'$ which separates v and $rep(v)$. ◀

For a wrapper W that is not the smallest in its level, we shall define $rep(W)$ to be an element (chosen arbitrarily) in $W \setminus W'$, where W' is the largest wrapper of the same level that is strictly contained in it.

For every marked wrapper M that is not the smallest marked wrapper in its level, let M' be the largest marked wrapper that is strictly contained in it. We delete every element in $M \setminus M'$ except $rep(M)$. Again, for every element $v \in M \setminus M'$ that gets deleted, we let $rep(v) = rep(M)$. (Note that for $v \in M \setminus M'$, we have $rep(v) \in M \setminus M'$).

▶ **Claim 2.** $(*)$ *If v is an element that was deleted from $M \setminus M'$, then there is no test $T \in \mathcal{S}'$ that separates v and $rep(v)$.*

For a marked wrapper M that is the smallest marked wrapper in its level, let M' be the set of all elements in M that is not contained in any marked subchunk of M. If M' is not

empty, we delete every element of M' except one. As before, for every deleted element v in M', we let $rep(v)$ be the element in M' that is not deleted.

▶ **Claim 3.** $(*)$ *If v is an element that was deleted from M', then there is no test $T \in \mathcal{S}'$ that separates v and $rep(v)$.*

Let R' be the set of elements in R that are not inside any marked chunk. If R' is not empty, delete every element from R' except one. Here again, for every deleted element v, we denote by $rep(v)$ the element in R' that is retained.

▶ **Claim 4.** $(*)$ *If v is an element that was deleted from the R', then there is no test $T \in \mathcal{S}'$ that separates v and $rep(v)$.*

We denote by $\mathcal{U}' \subseteq \mathcal{U}$ the set of elements that do not get deleted.

▶ **Lemma 9.** *If $v \in \mathcal{U} \setminus \mathcal{U}'$, then there is no test in $\mathcal{S}'$ that separates v and $rep(v)$.*

Proof. The proof is immediate from the above claims. ◀

▶ **Corollary 10.** *There is no class in $\mathcal{C}(\mathcal{S}')$ that consists only of elements from $\mathcal{U} \setminus \mathcal{U}'$.*

Proof. Suppose there is a class $D \in \mathcal{C}(\mathcal{S}')$ such that $D \subseteq \mathcal{U} \setminus \mathcal{U}'$. Consider any element $v \in D$. By Lemma 9, there is no test in $\mathcal{S}'$ that separates v and $rep(v)$. This implies that $rep(v) \in D$. As $rep(v) \in \mathcal{U}'$, this contradicts the assumption that $D \subseteq \mathcal{U} \setminus \mathcal{U}'$. ◀

Note that if $\mathcal{Z} \subseteq \mathcal{S}'$, then every class in $\mathcal{C}(\mathcal{Z})$ is a union of some classes from $\mathcal{C}(\mathcal{S}')$. This immediately gives us the following corollary.

▶ **Corollary 11.** *If $\mathcal{Z} \subseteq \mathcal{S}'$, then there is no class in $\mathcal{C}(\mathcal{Z})$ that consists only of elements from $\mathcal{U} \setminus \mathcal{U}'$.*

3.3.2 Proof of correctness and the size bound on $\mathcal{U}' \cap R$

Now, we shall show that our kernelization algorithm, given an instance $(\mathcal{U}, \mathcal{S}, k)$ of $(n-k)$-Test Cover produces an equivalent instance $(\mathcal{U}', \mathcal{S}', k)$ of $(n-k)$-Test Cover. In particular, we shall show that there is a k-mini test cover for $(\mathcal{U}, \mathcal{S}, k)$ if and only if there is a k-mini test cover for $(\mathcal{U}', \mathcal{S}', k)$. Note that any k-mini test cover for $(\mathcal{U}', \mathcal{S}', k)$ is also a k-mini test cover for $(\mathcal{U}, \mathcal{S}, k)$ and therefore the "if" direction is clear. To show the other direction, we shall show that whenever there is a k-mini test cover for $(\mathcal{U}, \mathcal{S}, k)$, there is also a k-mini test cover for $(\mathcal{U}', \mathcal{S}', k)$. This is proved in Lemma 14.

First, we show how, when given a k-mini test cover of $(\mathcal{U}, \mathcal{S}, k)$ that contains some deleted tests (i.e., some tests from $\mathcal{S} \setminus \mathcal{S}'$), we can obtain a new k-mini test cover of $(\mathcal{U}, \mathcal{S}, k)$ that contains a lesser number of deleted tests. Lemma 12 shows how this can be done when $\mathcal{Z}$ contains at least one deleted test whose chunk in R was an irrelevant chunk. Then, Lemma 13 shows how this can be done when $\mathcal{Z}$ contains no deleted test whose chunk in R is irrelevant (i.e., for every deleted test in $\mathcal{Z}$, its chunk in R is an unmarked wrapper).

▶ **Lemma 12.** $(*)$ *Let $\mathcal{Z}$ be a k-mini test cover for $(\mathcal{U}, \mathcal{S}, k)$. Let $T \in \mathcal{Z} \setminus \mathcal{S}'$ such that there is no other test $T' \in \mathcal{Z} \setminus \mathcal{S}'$ having $T' \cap R \subset T \cap R$. Also let $T \cap R$ be an irrelevant chunk (that is, $T \cap R$ is a chunk that was not marked and also did not contain any marked chunk). Then there is a collection $\mathcal{F} \subseteq \mathcal{S}'$ such that $(\mathcal{Z} \setminus \{T\}) \cup \mathcal{F}$ contains a k-mini test cover for $(\mathcal{U}, \mathcal{S}, k)$.*

▶ **Lemma 13.** $(*)$ *Let $\mathcal{Z}$ be a k-mini test cover of $(\mathcal{U}, \mathcal{S}, k)$ such that for each $T' \in \mathcal{Z} \setminus \mathcal{S}'$, $T' \cap R$ is a wrapper. Let $T \in \mathcal{Z} \setminus \mathcal{S}'$. Then there is a collection $\mathcal{F} \subseteq \mathcal{S}'$ such that $(\mathcal{Z} \setminus \{T\}) \cup \mathcal{F}$ contains a k-mini test cover for $(\mathcal{U}, \mathcal{S}', k)$.*

▶ **Lemma 14.** *If there is a k-mini test cover $\mathcal{Z} \subseteq \mathcal{S}$ of $\mathcal{U}$, then there is a k-mini test cover $\mathcal{Z}' \subseteq \mathcal{S}'$ of $\mathcal{U}'$.*

Proof. Since there is a k-mini test cover $\mathcal{Z}$ of $\mathcal{U}$, there must be a k-mini test cover $\mathcal{Z}'$ of $\mathcal{U}$ which contains the least number of tests from $\mathcal{S} \setminus \mathcal{S}'$. We claim that $\mathcal{Z}'$ is the required k-mini test cover of $\mathcal{U}'$ that contains no test from $\mathcal{S} \setminus \mathcal{S}'$. First, let us suppose for the sake of contradiction that $\mathcal{Z}' \cap (\mathcal{S} \setminus \mathcal{S}') \neq \emptyset$. Consider the case when there exists a test $T \in \mathcal{Z}' \cap (\mathcal{S} \setminus \mathcal{S}')$ such that $T \cap R$ is an irrelevant chunk. Then there exists a $T' \in \mathcal{Z}' \cap (\mathcal{S} \setminus \mathcal{S}')$ such that $T' \cap R$ is an irrelevant chunk and there is no $Z \in \mathcal{Z}' \cap (\mathcal{S} \setminus \mathcal{S}')$ with $Z \cap R \subset T' \cap R$ (recall that any chunk contained in an irrelevant chunk is also irrelevant). Then by Lemma 12, we have a k-mini test cover of $\mathcal{U}$ with lesser number of tests from $\mathcal{S} \setminus \mathcal{S}'$ than $\mathcal{Z}'$, which is a contradiction. Therefore, we can assume that there is no test in $T \in \mathcal{Z}' \cap (\mathcal{S} \setminus \mathcal{S}')$ such that $T \cap R$ is an irrelevant chunk. Now consider any $T \in \mathcal{Z}' \cap (\mathcal{S} \setminus \mathcal{S}')$. We know that $T \cap R$ is a wrapper. Then by Lemma 13, we can find a k-mini test cover of $\mathcal{U}$ that has lesser number of tests from $\mathcal{S} \setminus \mathcal{S}'$ than $\mathcal{Z}'$, which is again a contradiction. Therefore, we have $\mathcal{Z}' \cap (\mathcal{S} \setminus \mathcal{S}') = \emptyset$ or in other words, $\mathcal{Z}' \subseteq \mathcal{S}'$. Now, from Corollary 11, we know that $|\mathcal{C}_{\mathcal{U}'}(\mathcal{Z}')| = |\mathcal{C}_{\mathcal{U}}(\mathcal{Z}')|$. Thus, $\mathcal{Z}'$ is a k-mini test cover of $\mathcal{U}'$ as well. ◀

▶ **Lemma 15.** $|\mathcal{U}' \cap R| = \mathcal{O}(k^6)$.

Proof. In Phase I, we mark at most $(6k+3)|P| = \mathcal{O}(k^3)$ chunks. In Phase II, there at most $\mathcal{O}(k^3)$ different levels and in each level, we mark at most $2(2k+1)|P| = \mathcal{O}(k^3)$ wrappers. Thus there are totally $\mathcal{O}(k^6)$ marked wrappers. Therefore, the total number of marked chunks in R is $\mathcal{O}(k^6)$. Since each element of $\mathcal{U}' \cap R$, except the one representative element that is not contained in any chunk in R, is associated with a chunk (a solid chunk or a marked wrapper), and since this association is one-to-one, $|\mathcal{U}' \cap R| = \mathcal{O}(k^6)$. ◀

3.4 Final Result and Algorithms for $(n-k)$-Test Cover

We summarize the kernelization algorithm in the following theorem.

▶ **Theorem 16.** *There is an algorithm that, given an instance $(\mathcal{U}, \mathcal{S}, k)$ of $(n-k)$-Test Cover, runs in polynomial time and returns an equivalent instance $(\mathcal{U}', \mathcal{S}', k)$ such that $|\mathcal{U}'| = \mathcal{O}(k^7)$ and $|\mathcal{S}'| = \mathcal{O}(2^{3k} \cdot k^7)$.*

Proof. We run the outer phase of the algorithm only once. Let $R_1, R_2, \ldots, R_t$ be the regions in $\mathcal{C}(\mathcal{T})$ after the outer phase is done. Now we do the marking and pruning procedure iteratively on every region in $R_1, R_2, \ldots, R_t$. In each iteration, the universe size decreases. In particular, after the marking and pruning procedure is done on a region R_i, by Lemma 15, the number of elements in R_i becomes $\mathcal{O}(k^6)$. Let $\mathcal{U}'$ denote the set of elements that remain undeleted from $\mathcal{U}$ after the marking and pruning has been done on all the regions. Since there are at most $3k$ regions, $|\mathcal{U}'| = \mathcal{O}(k^7)$.

We shall now count the number of tests $\mathcal{S}'$. There can be at most 2^{3k} tests that have non-trivial intersection with no region. Given a particular region, the number of tests in $\mathcal{S}'$ that have a non-trivial intersection with that region can be bounded as follows. Each such test is associated with exactly one chunk in that region. Since, in $(\mathcal{U}', \mathcal{S}', k)$, there are at most $\mathcal{O}(k^6)$ chunks in each region, and because there are at most $3k$ regions in total, there can be at most 2^{3k} tests in $\mathcal{S}'$ which have the same chunk in this region. So there are at most $\mathcal{O}(2^{3k} \cdot k^6)$ tests that have non-trivial intersection with one particular region. Since there are at most $3k$ regions, the total number of tests in $\mathcal{S}'$ is at most $\mathcal{O}(2^{3k} \cdot k^7)$.

It is easy to see that the marking and pruning procedure can be done in polynomial time and hence the algorithm runs in polynomial time. ◀

▶ **Theorem 17.** *There is an algorithm that, given an instance $(\mathcal{U}, \mathcal{S}, k)$ of $(n-k)$-Test Cover, runs in time $\mathcal{O}(2^{\mathcal{O}(k^2)}(m+n)^{\mathcal{O}(1)})$ and either returns a test cover of size at most $n-k$ or correctly concludes that no such test cover exists.*

Proof. After generating an equivalent instance of size $\mathcal{O}(2^{3k} \cdot k^7)$ using the kernelization algorithm which runs in time $\mathcal{O}(k^{\mathcal{O}(1)}(m+n)^{\mathcal{O}(1)})$, we can check whether this is a Yes instance by checking for the existence of a k-mini test cover. This can be determined by considering all possible collections of tests containing at most $2k$ tests. There are $\mathcal{O}((2^{3k} \cdot k^7)^{2k})$ such collections and each of them can be checked in time $\mathcal{O}((m+n)^{\mathcal{O}(1)})$, which completes the proof. ◀

4 Conclusion

In this paper we studied the $(n-k)$-Set Cover and $(n-k)$-Test Cover problems and obtained partially polynomial kernels for both these problems by designing reduction rules that bound the universe size. We showed the utility of these rules and kernels by obtaining a linear time FPT algorithm with an improved dependence on the parameter for $(n-k)$-Set Cover and $(m-k)$-Hitting Set. Furthermore, we also obtained an FPT algorithm for $(\nu(F)+k)$-SAT which improves over the previous best algorithm with respect to the dependence on the parameter, as well as the first single exponential FPT algorithm for $(n-k)$-Test Cover. We believe that in general, the existing upper bounds for other problems can also be improved by using partially polynomial kernels, and in particular, we believe that our reduction rules could be applicable to other covering problems as well.

References

1 Nadja Betzler, Jiong Guo, Christian Komusiewicz, and Rolf Niedermeier. Average parameterization and partial kernelization for computing medians. *J. Comput. Syst. Sci.*, 77(4):774–789, 2011.

2 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

3 Robert Crowston, Gregory Gutin, Mark Jones, Gabriele Muciaccia, and Anders Yeo. Parameterizations of test cover with bounded test sizes. *CoRR*, abs/1209.6528, 2012.

4 Robert Crowston, Gregory Gutin, Mark Jones, Venkatesh Raman, Saket Saurabh, and Anders Yeo. Fixed-parameter tractability of satisfying beyond the number of variables. *Algorithmica*, pages , to appear, 2013.

5 Robert Crowston, Gregory Gutin, Mark Jones, Saket Saurabh, and Anders Yeo. Parameterized study of the test cover problem. In *MFCS*, volume 7464, pages 283–295, 2012.

6 Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54(2):181–207, 2009.

7 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct pcps for np. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

8 Gregory Gutin, Mark Jones, and Anders Yeo. Kernels for below-upper-bound parameterizations of the hitting set and directed dominating set problems. *Theor. Comput. Sci.*, 412(41):5744–5751, 2011.

9 Gregory Gutin, Gabriele Muciaccia, and Anders Yeo. (non-)existence of polynomial kernels for the test cover problem. *Inf. Process. Lett.*, 113(4):123–126, 2013.

10 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Springer-Verlag, 2006.

Parameterized Complexity of the Anchored k-Core Problem for Directed Graphs*

Rajesh Chitnis[1], Fedor V. Fomin[2], and Petr A. Golovach[2]

1 **Department of Computer Science, University of Maryland at College Park, USA**
rchitnis@cs.umd.edu

2 **Department of Informatics, University of Bergen, Bergen, Norway**
{fedor.fomin,petr.golovach}@ii.uib.no

Abstract

We consider the Directed Anchored k-Core problem, where the task is for a given directed graph G and integers b, k and p, to find an induced subgraph H with at least p vertices (the core) such that all but at most b vertices (the anchors) of H have in-degree at least k. For undirected graphs, this problem was introduced by Bhawalkar, Kleinberg, Lewi, Roughgarden, and Sharma [ICALP 2012]. We undertake a systematic analysis of the computational complexity of Directed Anchored k-Core and show that:

- The decision version of the problem is NP-complete for every $k \geq 1$ even if the input graph is restricted to be a planar directed acyclic graph of maximum degree at most $k + 2$.
- The problem is fixed parameter tractable (FPT) parameterized by the size of the core p for $k = 1$, and W[1]-hard for $k \geq 2$.
- When the maximum degree of the graph is at most Δ, the problem is FPT parameterized by $p + \Delta$ if $k \geq \frac{\Delta}{2}$.

1998 ACM Subject Classification F.2.2, G.2.1, G.2.2

Keywords and phrases Parameterized complexity, directed graphs, anchored k-core

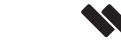
Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.79

1 Introduction

Degree-constrained subgraph problems have been extensively studied in theoretical computer science. One can describe degree constrained subgraph problems in the following general setting: given a (un)directed graph G, find a maximum/minimum sized (induced, connected) subgraph H subject to some condition $\mathcal{C}$ imposed on the degrees of vertices. For example, Independent Set or (Induced) Matching can be seen as problems within this framework. In this paper, we study an interesting variant of the degree-constrained subgraph problem where we have to find a large subgraph with all vertices satisfying constrains except a small set of anchors vertices. While such type of problems arise naturally in different settings, in particular in social sciences, adding of anchors can bring to non-trivial computational challenges.

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement n. 267959 and from NSF CAREER award 1053605, NSF grant CCF-1161626, ONR YIP award N000141110662, DARPA/AFOSR grant FA9550-12-1-0423, a University of Maryland Research and Scholarship Award (RASA) and a Summer International Research Fellowship from the University of Maryland.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 79–90



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

More precisely, the k-core of a directed graph G is defined as the largest subgraph H such that $\deg^-_H(v) \geq k$ for every $v \in V(H)$. This notion was introduced by Seidman [17] and is a well-known concept in the theory of social networks. It has also been studied in various social sciences literature [8, 9]. It is easy to see that we can find the k-core of a given directed graph in polynomial time by the following procedure: iteratively remove any vertex that has in-degree less than k. However, one might not want to strictly enforce the condition of in-degree being at least k for every vertex. In particular, we allow for a small number of special vertices (called anchors) which can have arbitrary in-degrees, but their purpose in the k-core is to augment the in-degrees of the non-anchored vertices. Bhawalkar et al. [2] introduced the Anchored k-Core problem for (undirected) graphs. In the Anchored k-Core problem the input is an undirected graph $G = (V, E)$ and integers b, k, and the task is to find an induced subgraph H of maximum size with all vertices but at most b (which are anchored) to be of degree at least k. In this work we extend the notion of anchored k-core to directed graphs and define the parameterized version of the problem formally:

Directed Anchored k-Core (Dir-AKC)
Input: A directed graph $G = (V, E)$ and integers b, k, p.
Parameter 1: b.
Parameter 2: k.
Parameter 3: p.
Question: Do there exist sets of vertices $A \subseteq U \subseteq V(G)$ such that $\lvert A\rvert \leq b$, $\lvert U\rvert \geq p$, and every $v \in U \setminus A$ satisfies $d^-_{G[U]}(v) \geq k$?

We will refer to the set A as the set of *anchors* and to the graph $H = G[U]$ as the *anchored k-core*. Note that the undirected version of Anchored k-Core problem can be modeled by the directed version: simply replace each edge $\{u, v\}$ by arcs (u, v) and (v, u). Keeping the parameters b, k, p unchanged it is now easy to see that the two instances are equivalent.

Connection to Preventing Unraveling in Social Networks: Social networks are generally represented by making use of undirected or directed graphs, where the edge set represents the relationship between individuals in the network. The undirected graph model works fine for some networks, say Facebook, but the nature of interaction on some social networks such as Twitter is asymmetrical: the fact that user A follows user B does not imply that user B also follows A. In this case, it is more appropriate to model interactions in the network by *directed* graphs. We add a directed edge (u, v) if v follows u. We can consider a model of *user management* where there is a threshold value k, such that each individual with less than k people to follow (or equivalently whose in-degree is less than k) drops out of the network. This process can be contagious, and may affect even those individuals who initially were linked to more than k people. An extreme example of this was given by Schelling (see page 17 of [15]): consider a directed path on n vertices and let $k = 1$. The left-endpoint has in-degree zero, it drops out and now the in-degree of its only out-neighbor in the path becomes zero and it drops out as well. It is not hard to see that this way the whole network eventually drops out as the result of a *cascade of iterated withdrawals*, i.e., the 1-core of this graph is the empty set. The unraveling process described above in Schelling's example of a directed path can be highly undesirable in many scenarios. One can attempt to prevent this unraveling by introducing a few special vertices (called anchors) by "buying" them with extra incentives.

Parameterized Complexity: We are mainly interested in the parameterized complexity of Anchored k-Core. For the general background, we refer to the books by Downey and Fellows [10], Flum and Grohe [12] and Niedermeier [14]. Parameterized complexity

is basically a two dimensional framework for studying the computational complexity of a problem. One dimension is the input size n and another one is a parameter k. A problem is said to be *fixed parameter tractable* (or FPT) if it can be solved in time $f(k) \cdot n^{O(1)}$ for some function f. A problem is said to be in XP, if it can be solved in time $O(n^{f(k)})$ for some function f. The W-hierarchy is a collection of computational complexity classes: we omit the technical definitions here. The following relation is known amongst the classes in the W-hierarchy: FPT = W[0] $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \ldots$. It is widely believed that FPT $\neq$ W[1], and hence if a problem is hard for the class W[i] (for any $i \geq 1$) then it is considered to be fixed-parameter intractable.

Previous Results: Bhawalkar et al. [2] initiated the algorithmic study of Anchored k-Core on undirected graphs. In particular, they obtained the following dichotomy result: the decision version of the problem is solvable in polynomial time for $k \leq 2$ and is NP-complete for all $k \geq 3$. Moreover, for $k \geq 3$ the problem remains NP-complete even on planar graphs [6]. This motivates the study of the problem for $k \geq 3$ from the viewpoint of parameterized complexity. Unfortunately, the problem is W[2]-hard parameterized by b [2] and W[1]-hard parameterized by p even for $k = 3$ [6].

Our Results: In this paper we initiate the study of Anchored k-Core on directed graph and provide a new insight into the computational complexity of the problem. We obtain the following results.

- The decision version of Dir-AKC is NP-complete for every $k \geq 1$ even if the input graph is restricted to be a planar directed acyclic graph (DAG) of maximum degree at most $k + 2$. Thus the directed version is in some sense strictly harder than the undirected version which is known be in P if $k \leq 2$, and NP-complete if $k \geq 3$ [2]. These results are proven in Section 2.
- The NP-hardness result for Dir-AKC motivates us to make a more refined analysis of the Dir-AKC problem via the paradigm of parameterized complexity. We obtain (Section 3) the following dichotomy result: Dir-AKC is FPT parameterized by p if $k = 1$, and W[1]-hard if $k \geq 2$.

This fixed-parameter intractability result parameterized by p forces us to consider the complexity on special classes of graphs such as bounded-degree directed graphs or directed acyclic graphs.

- In Section 4, for graphs of degree upper bounded by Δ, we show that the Dir-AKC problem is FPT parameterized by $p + \Delta$ if $k \geq \frac{\Delta}{2}$. In particular, it implies that Dir-AKC is FPT parameterized by p for directed graphs of maximum degree at most four.
- We complement tractability results by showing in Section 5 that if $k < \frac{\Delta}{2}$ and $\Delta \geq 3$, then Dir-AKC is W[2]-hard when parameterized by the number of anchors b even for DAGs. On the other hand, the problem is FPT when parameterized by $\Delta + p$ for DAGs of maximum degree at most Δ. Note that we can always assume that $b \leq p$, and hence any FPT result with parameter b implies FPT result with parameter p as well. On the other side, any hardness result with respect to p implies the same hardness with respect to b.

Due to space limitations, some proofs are omitted here. They can be found in [5].

2 Preliminaries

We consider finite directed and undirected graphs without loops or multiple arcs. The vertex set of a (directed) graph G is denoted by $V(G)$ and its edge set (arc set for a directed graph)

by $E(G)$. The subgraph of G induced by a subset $U \subseteq V(G)$ is denoted by $G[U]$. For $U \subset V(G)$ by $G - U$ we denote the graph $G[V(G) \setminus U]$. For a directed graph G, we denote by G^* the undirected graph with the same set of vertices such that $\{u, v\} \in E(G^*)$ if and only if $(u, v) \in E(G)$. We say that G^* is the *underlying* graph of G.

Let G be a directed graph. For a vertex $v \in V(G)$, we say that u is an *in-neighbor* of v if $(u, v) \in E(G)$. The set of all in-neighbors of v is denoted by $N_G^-(v)$. The *in-degree* $d_G^-(v) = |N_G^-(v)|$. Respectively, u is an *out-neighbor* of v if $(v, u) \in E(G)$, the set of all out-neighbors of v is denoted by $N_G^+(v)$, and the *out-degree* $d_G^+(v) = |N_G^+(v)|$. The *degree* $d_G(v)$ of a vertex v is the sum $d_G^-(v) + d_G^+$, and the *maximum degree* of G is $\Delta(G) = \max_{v \in V(G)} d_G(v)$. A vertex v of $d_G^-(v) = 0$ is called a *source*, and if $d_G^+(v) = 0$, then v is a *sink*. Observe that isolated vertices are sources and sinks simultaneously.

Let G be a directed graph. For $u, v \in V(G)$, it is said that v can be *reached* (or *is reachable*) from u if there is a directed $u \to v$ path in G. Respectively, a vertex v can be reached from a set $U \subseteq V(G)$ if v can be reached from some vertex $u \in U$. Notice that each vertex is reachable from itself. We denote by $R_G^+(u)$ ($R_G^+(U)$ respectively) the set of vertices that can be reached from a vertex u (a set $U \subseteq V(G)$ respectively). Let $R_G^-(u)$ denote the set of all vertices v such that u can be reached from v.

For two non-adjacent vertices s, t of a directed graph G, a set $S \subseteq V(G) \setminus \{s, t\}$ is said to be an $s - t$ *separator* if $t \notin R_{G-S}^+(s)$. An $s - t$ separator S is *minimal* if no proper subset $S' \subset S$ is an $s - t$ separator.

The notion of important separators was introduced by Marx [13] and generalized for directed graphs in [7]. We need a special variant of this notion. Let G be a directed graph, and let s, t be non-adjacent vertices of G. A minimal $s - t$ separator is an *important* $s - t$ *separator* if there is no $s - t$ separator S' with $|S'| \leq |S|$ and $R_{G-S}^-(t) \subset R_{G-S'}^-(t)$. The following lemma is a variant of Lemma 4.1 of [7]. Notice that to obtain it, we should replace the directed graph in Lemma 4.1 of [7] by the graph obtained from it by reversing the direction of all arcs.

▶ **Lemma 1** ([7]). *Let G be a directed graph with n vertices, and let s, t be non-adjacent vertices of G. Then for every $h \geq 0$, there are at most 4^h important $s - t$ separators of size at most h. Furthermore, all these separators can be enumerated in time $O(4^h \cdot n^{O(1)})$.*

As further we are interested in the parameterized complexity of Dir-AKC, we show first NP-hardness of the problem (the proof is given in [5]).

▶ **Theorem 2.** *For any $k \geq 1$,* Dir-AKC *is* NP*-complete, even for planar DAGs of maximum degree at most $k + 2$.*

We conclude this section by the simple observation that Dir-AKC is in XP when parameterized by the number of anchors b. For a directed graph G with n vertices, we can consider all the at most n^b possibilities to choose the anchors, and then recursively delete non-anchor vertices that have the in-degree at most $k - 1$. Trivially, if we obtain a directed graph with at least p vertices for some selection of the anchors, we have a solution and otherwise we can answer NO.

3 Dir-AKC parameterized by the size of the core

In this section we consider the Dir-AKC problem for fixed k when p is the parameter, and obtain the following dichotomy: If $k = 1$ then the Dir-AKC problem is FPT parameterized by p, otherwise for $k \geq 2$ it is W[1]-hard parameterized by p.

▶ **Theorem 3.** *For $k = 1$, the* Dir-AKC *problem is solvable in time $2^{O(p)} \cdot n^2 \log n$ on digraphs with n vertices.*

Proof. The proof is constructive, and we describe an FPT algorithm for the problem. Without loss of generality, we assume that $b < p \leq n$.

We apply the following preprocessing rule reducing the instance to an acyclic graph. Let $C_1, \ldots, C_r$ be the non-trivial strongly connected components of G, i.e., $|V(C_i)| \geq 2$ for $i \in \{1, \ldots, r\}$. Note that for each $i \in \{1, \ldots, r\}$ and any $v \in V(C_i)$, $d^-_{C_i}(v) \geq 1$. By making use of Tarjan's algorithm [18], $C_1, \ldots, C_r$ can be found in linear time. Let $R = R^+_G\Big(\bigcup_{i=1}^r V(C_i)\Big)$ be the set of vertices reachable from these strongly connected components. Then every $v \in R$ satisfies $d^-_{G[R]}(v) \geq 1$. If $|R| \geq p$, then $H = G[R]$ is an anchored 1-core of size at least p for the empty set of anchors. If $b \geq p - |R| > 0$, then we select in $V(G) \setminus R$ any arbitrary $b' = p - |R|$ vertices $a_1, \ldots, a_{b'}$. In this case we output the set of anchors $A = \{a_1, \ldots, a_{b'}\}$ and the graph $H = G[A \cup R]$. Otherwise, if $b < p - |R|$, we set $G' = G - R$ and $p' = p - |R|$ and consider a new instance of Dir-AKC with the graph G' and the parameter p'.

To see that the rule is safe, it is sufficient to observe that a set of anchors A and a subgraph H' of size at least p' is a solution of the obtained instance if and only if $(A, H = G[V(H') \cup R])$ is a solution for the original problem. Let us remark that the preprocessing rule can be easily performed in time $O(n^2)$.

From now we can assume that G has no non-trivial strongly connected components, i.e., G is a directed acyclic graph. Denote by $S = \{s_1, \ldots, s_h\}$ the set of sources of G. If $|S| \leq b$, then set $A = S$. In this case, we output the pair $(A, H = G)$. The pair (A, H) is a solution because every vertex $v \in V(G) \setminus S$ satisfies $d^-_G(v) \geq 1$. It remains to consider the case when $|S| > b$. For $i \in \{1, \ldots, h\}$, let $R_i = R^+_G(s_i)$. Then $V(G) = R^+_G(S) = \bigcup_{i=1}^h R_i$. Without loss of generality, we can assume that every anchored vertex is from S. Indeed, if s_i is an anchor, then each vertex of R_i can be included in a solution. Hence for every anchor $a \in R_j \setminus \{s_j\}$, we can delete this anchor from A and replace it by s_j. Since we can choose anchors only from S, we are able to reduce the problem to Partial Set Cover.

Partial Set Cover
Input : A collection $X = \{X_1, \ldots, X_r\}$ of subsets of a finite n-element set U and positive integers p, b.
Parameter: p.
Question: Are there at most b subsets $X_{i_1}, \ldots, X_{i_b}$, $1 \leq i_1 < \ldots < i_b \leq r$, covering at least p elements of U, i.e., $\lvert\bigcup_{j=1}^b X_{i_j}\rvert \geq p$?

Bläser [3] showed that Partial Set Cover is FPT parameterized by p and can be solved in time $O(2^{O(p)} \cdot rn \log n)$. For Dir-AKC, we consider the collection of subsets $\{R_1, \ldots, R_r\}$ of $V(G)$. If we can select at most b subsets $R_{i_1}, \ldots, R_{i_b}$ such that $|\cup_{j=1}^b R_{i_j}| \geq p$, we return the solution with anchors $A = \{s_{i_1}, \ldots, s_{i_b}\}$ and $H = G[\bigcup_{j=1}^b R_{i_j}]$. Otherwise, we return a NO-answer.

Because our preprocessing can be done in time $O(n^2)$ and Partial Set Cover is solvable in time $2^{O(p)} \cdot n^2 \log n$, we conclude that the total running time is $2^{O(p)} \cdot n^2 \log n$. ◀

Now we complement Theorem 3 by showing that for $k \geq 2$, Dir-AKC becomes hard parameterized by the core size (the proof is in [5]).

▶ **Theorem 4.** *For any fixed $k \geq 2$, the* Dir-AKC *problem is* W[1]*-hard parameterized by p, even for DAGs.*

4 Dir-AKC on graphs of bounded degree

In this section we show that Dir-AKC problem is FPT parameterized by $\Delta + p$ if $k \geq \frac{\Delta}{2}$.

In our algorithms we need to check the existence of solutions for Dir-AKC that have bounded size. It can be observed that if we are interested in solutions (A, H) such that $p \leq |V(H)| \leq q$, then for every positive q, we can express this problem in First Order Logic. It was proved by Seese [16] that any graph problem expressible in First Order Logic can be solved in linear time on (directed) graphs of bounded degree. Later this result was extended for much more rich graph classes (see [11]). These meta theorems are very general, but do not provide good upper bounds on the running time for particular problems. Hence, we give the following lemma. Our algorithms use the random separation technique due to Cai et al. [4] (which is a variant of the color coding method introduced by Alon et al. [1]) .

▶ **Lemma 5.** *There is a randomized algorithm with running time $2^{O(\Delta q)} \cdot n$ that for an instance of* Dir-AKC *with an n-vertex directed graph of maximum degree at most Δ and a positive integer $q \geq p$, either returns a solution (A, H) with $V(H) \geq p$ or gives the answer that there is no solution with $|V(H)| \leq q$. Furthermore, the algorithm can be derandomized, and the deterministic variant runs in time $2^{O(\Delta q)} \cdot n \log n$.*

Proof. Consider an instance of Dir-AKC with an n-vertex directed graph G of maximum degree at most Δ. We assume that $b \leq p \leq n$. For given $q \geq p$, to decide if G contains a solution of size at most q, we do the following.

We color each vertex of G uniformly at random with probability $\frac{1}{2}$ by one of two colors, say red or blue. Let R be the set of vertices colored red. Observe that if there is a solution (A, H) with $|V(H)| \leq q$, then with probability at least $\frac{1}{2^q}$ all vertices of H are colored red and with probability at least $\frac{1}{2^{\Delta q}}$ all in- and out-neighbors of the vertices of H that are outside of H are colored blue. Using this observation, we assume that H is the union of some weakly connected components of the graph $G[R]$ induced by red vertices.

In time $O(\Delta n)$ we find all weakly connected components of $G[R]$. If there is a component C with at least $b+1$ vertices of in-degree at most $k-1$ (in C), then we discard this component as it cannot be a part of any solution. Denote by $C_1, \ldots, C_r$ the remaining components. For $i \in \{1, \ldots, r\}$, let $A_i = \{v \in V(C_i) | d^-_{C_i}(v) < k\}$, $b_i = |A_i|$ and $p_i = |V(C_i)|$.

Thus everything boils down to the problem of finding a set $I \subseteq \{1, \ldots, r\}$ such that $\sum_{i \in I} b_i \leq b$ and $\sum_{i \in I} p_i \geq p$. But this is the well known Knapsack problem, which is solvable in time $O(bn)$ by dynamic programming. If we obtain a solution I, then we output (A, H), where $A = \cup_{i \in I} A_i$ and $H = G[\cup_{i \in I} V(C_i)]$. Otherwise, we return a NO-answer. Notice that this algorithm can also find a solution (A, H) with $|V(H)| > q \geq p$.

It remains to observe that for any positive number $\alpha < 1$, there is a constant c_α such that after running our randomized algorithm $c_\alpha \cdot 2^{\Delta q}$ times, we either find a solution (A, H) or can claim that with probability α that it does not exist.

This algorithm can be derandomized by the technique proposed by Alon et al. [1]: replace the random colorings by a family of at most $2^{O(\Delta q)} \cdot \log n$ hash functions which are known to be constructible in time $2^{O(\Delta q)} \cdot n \log n$. ◀

Our next aim is to prove that for $k > \Delta/2$ the Dir-AKC problem is FPT when parameterized by $\Delta + b$.

▶ **Lemma 6.** *Let Δ be a positive integer. If $k > \Delta/2$, then the* Dir-AKC *problem can be solved in time $2^{O(\Delta^2 b)} \cdot n \log n$ for n-vertex directed graphs of maximum degree at most Δ.*

Proof. Suppose (A, H) is a solution for the Dir-AKC problem. Let us observe that because $k > \Delta/2$, for every vertex $v \in V(H)\setminus A$, we have $d^-_H(v) > d^+_H(v)$. Recall that for any directed graph, the sum of in-degrees equals the sum of out-degrees. Then

$$\sum_{v\in V(H)\setminus A}(d^-_H(v) - d^+_H(v)) = \sum_{v\in A}(d^+_H(v) - d^-_H(v)).$$

Since for every vertex $v \in V(H) \setminus A$, $d^-_H(v) - d^+_H(v) \geq 1$, we have that

$$|V(H) \setminus A| \leq \sum_{v\in V(H)\setminus A}(d^-_H(v) - d^+_H(v)).$$

On the other hand, $d^+_H(v) - d^-_H(v) \leq \Delta$, and we arrive at

$$|V(H) \setminus A| \leq \sum_{v\in V(H)\setminus A}(d^-_H(v) - d^+_H(v)) = \sum_{v\in A}(d^+_H(v) - d^-_H(v)) \leq \Delta|A|.$$

Hence, $|V(H)| \leq (\Delta+1)|A| \leq (\Delta+1)b$. Using this observation, we can solve the Dir-AKC problem as follows. If $p > (\Delta+1)b$, then we return a NO-answer. If $p \leq (\Delta+1)b$, we apply Lemma 5 for $q = (\Delta+1)b$, and solve that problem in time $2^{O(\Delta^2 b)} \cdot n \log n$. ◀

Now we show that if $k = \frac{\Delta}{2}$ then the Dir-AKC problem is FPT parameterized by $\Delta + p$.

▶ **Lemma 7.** *Let Δ be a positive integer. If $k = \Delta/2$, then the* Dir-AKC *problem can be solved in time $2^{O(\Delta^3 b+\Delta^2 bp)} \cdot n^{O(1)}$ for n-vertex directed graphs of maximum degree at most Δ.*

Proof. We describe an FPT algorithm. Consider an instance of the Dir-AKC problem. Without loss of generality we assume that $b < p \leq n$.

We apply the following preprocessing rule. Suppose that G has a (weakly) connected component C such that for any $v \in V(C)$, $d^-_C(v) = d^+_C(v) = k$. If $b \geq p - |V(C)|$, then we choose a set A of $b' = p - |V(C)|$ vertices arbitrary in $V(G) \setminus V(C)$. Then we return a YES-answer, as the anchors A and $H = G[A \cup V(C)]$ is a solution. Otherwise, if $b < p - |V(C)|$, we let $G' = G - V(C)$ and $p' = p - |V(C)|$. Now we consider a new instance of the problem with the graph G' and the parameter p'. To see that the rule is safe, it is sufficient to observe that a set of anchors A and a subgraph H' of size at least p' is a solution of the obtained instance if and only if A and $H = G[V(H') \cup V(C)]$ is a solution for the original problem. From now we assume that G has no such components.

We need the following claim.

Claim A. *If an instance of the* Dir-AKC *problem has a core with at least $(\Delta p + 1)b + 1$ vertices, then it has a solution (A, H) with the following property: there is a vertex $t \in V(H) \setminus A$ reachable in H from any vertex of H. Moreover, for each vertex v of H, there is a path from v to t with all vertices except v in $V(H) \setminus A$.*

Proof of Claim A. Let (A, H') be a solution with the set of anchors A and such that $V(H') > (\Delta p + 1)b$.

We show that $V(H') = R^+_{H'}(A)$, i.e., all vertices of H' are reachable from the anchors. To obtain a contradiction, suppose that there is a vertex $u \in V(H')$ such that $u \notin R^+_{H'}(A)$. Let $U = R^-_{H'}(u)$, i.e., U is the set of vertices from which we can reach u. Clearly, $A \cap U = \emptyset$. Therefore, $d^-_{H'}(v) \geq k = \Delta/2$ for $v \in U$. Notice that for a vertex $v \in U$, $N^-_{H'}(v) \subseteq U$ by the

definition. Hence, $d^-_{G[U]}(v) \geq k = \Delta/2$ for $v \in U$. Because the sum of in-degrees equals the sum of out-degrees, for every vertex $v \in U$, we have that $d^-_{G[U]}(v) = d^+_{G[U]}(v) = k = \Delta/2$. Then $C = G[U]$ is a component of G such that for every $v \in V(C)$, $d^-_C(v) = d^+_C(v) = k$, but such components are excluded by the preprocessing; a contradiction.

Observe now that if $d^-_{H'}(v) < d^+_{H'}(v)$, then $d^-_{H'}(v) < k$ and thus $v \in A$. Hence, by adding at most Δb (maybe multiple) arcs from $V(H') \setminus A$ to A, joining the vertices $v \in V(H')$ of degrees $d^-_{H'}(v) > d^+_{H'}(v)$ with vertices of degrees $d^-_{H'}(v) < d^+_{H'}(v)$, we can transform H' into a disjoint union of directed Eulerian graphs. Since $V(H') = R^+_{H'}(A)$, each of these directed Eulerian graphs contains at least one vertex of A. Thus the set of arcs of H' can be covered by at most Δb arc-disjoint directed walks, each walk starting from a vertex of A and never coming back to A. Because $d^-_{H'}(v) \geq k$ for $v \in V(H') \setminus A$, we have that $|E(G')| \geq k(|V(H')| - b) > \Delta kbp$. Then there is a walk W with at least $kp + 1$ arcs. Let $a \in A$ be the first vertex of W and let t be the last vertex of the walk. The walk W visits a only once, t and all other vertices of W are visited at most k times. We conclude that W has at least p vertices.

Let $R = R^-_{H'-A}(t)$ and let $A' = \{a \in A \mid N^+_{H'}(a) \cap R \neq \emptyset\}$. Consider $H = G[R \cup A']$. Since $V(W) \subseteq V(H)$, $|V(H)| \geq p$. For any $v \in V(H) \setminus A$, the in-neighbors of v in H' are in H by the construction and, therefore, $d^-_H(v) \geq k$. It remains to observe that to select at most b anchors, we take $A' \subseteq V(H)$. ◀

Using Claim A, we proceed with our algorithm. We try to find a solution such that H has at most $q = (\Delta p+1)b$ vertices by applying Lemma 5. It takes time $O(2^{O(\Delta^2 bp)} \cdot n \log n)$. If we obtain a solution, then we return it and stop. Otherwise, we conclude that every core contains at least $(\Delta p + 1)b + 1$ vertices. By Claim A, we can search for a solution H with a non-anchor vertex t which is reachable from all other vertices of H by directed paths avoiding A. Notice that since t is a non-anchor vertex, we have that $d^-_G(t) \geq k$. We try at most n possibilities for all possible choices of t, and solve our problem for each choice. Clearly, if we get a YES-answer for one of the choices, we return it and stop. Otherwise, if we fail, we return a NO-answer.

From now we assume that we already selected t. We denote by G' the graph obtained from G by adding an artificial source vertex s joined by arcs with all the vertices $v \in V(G)$ with $d^-_G(v) < k$. Observe that $(s, t) \notin E(G')$.

Suppose that (A, H) is a solution with the set of anchors A such that $t \in V(H) \setminus A$ is reachable in H from any vertex of H by a path with all inner vertices in $V(H) \setminus A$. Denote by $\delta_{G'}(H)$ the set $\{v \in V(H) \mid N^-_{G'}(v) \setminus V(H) \neq \emptyset\}$, i.e., $\delta_{G'}(H)$ contains vertices that have in-neighbors outside H. We need a chain of claims about the structure of H in G'.

Claim B. $|\delta_{G'}(H) \setminus A| \leq \Delta b$.

Proof of Claim B. Let $X = \{v \in V(H) \mid d^-_H(v) \geq k \text{ and } d^+_H(v) < k\}$, $Y = \{v \in V(H) \mid d^-_H(v) = d^+_H(v) = k\}$ and $Z = \{v \in V(H) \mid d^-_H(v) < k\}$. Clearly,

$$\sum_{v \in X} (d^-_H(v) - d^+_H(v)) + \sum_{v \in Y} (d^-_H(v) - d^+_H(v)) = \sum_{v \in Z} (d^+_H(v) - d^-_H(v))$$

Observe that $d^-_H(v) - d^+_H(v) \geq 1$ for $v \in X$, $d^-_H(v) - d^+_H(v) = 0$ for $v \in Y$ and $d^+_H(v) - d^-_H(v) \leq \Delta$ for $v \in Z$. Hence, $|X| \leq \Delta |Z|$. If $d^-_H(v) < k$ for $v \in V(H)$, then $v \in A$. It follows that $Z \subseteq A$ and $|Z| \leq b$. We have $|X| \leq \Delta b$. Consider a vertex $v \in \delta_{G'}(H) \setminus A$. It has at least one in-neighbor outside H in G and $d^-_H(v) \geq k$. Then $d^+_H(v) < k$ and $v \in X$. We conclude that $\delta_{G'}(H) \setminus A \subseteq X$ and $|\delta_{G'}(H) \setminus A| \leq \Delta b$. ◀

Claim C. *There is an $s-t$ separator S in G' of size at most $(\Delta(k-1)+1)b$ such that $V(H)\setminus A \subseteq R^-_{G'-S}(t)$.*

Proof of Claim C. Let $S = \left(\delta_{G'}(H)\cap A\right) \cup \left(\bigcup_{v\in\delta_{G'}(H)\setminus A}(N^-_G(v)\setminus V(H)\right)$, i.e., the set containing all anchors that are in $\delta_{G'}$, and for each non-anchor vertex of $\delta_{G'}$ containing all its in-neighbors outside of H. Consider a directed (s,t)-path P in G'. Let v be the first vertex in P that is in $V(H)$ and let u be its predecessor in P. If $v\in A$, then $v\in S$. If $v\notin A$, then $u\neq s$ as H has no non-anchor vertices with in-degree at most $k-1$ in G. Then $u\in S$. We conclude that each (s,t)-path contains a vertex of S, i.e., this set is an $s-t$ separator.

Observe that $V(H)\setminus A \subseteq R^-_{G'-S}(t)$ by the definition of S and the fact that t can be reached from any vertex of H in this graph by a path with all inner vertices in $V(H)\setminus A$.

It remains to show that $|S|\leq(\Delta(k-1)+1)b$. By Claim B, $|\delta_{G'}(H)\setminus A|\leq\Delta b$. A vertex $v\in\delta_{G'}(H)\setminus A$ has at least one out-neighbor in H because t is reachable from v. Then v has at most $k-1$ in-neighbors outside H. Hence $|S|\leq|A|+(k-1)(\delta_{G'}(H)\setminus A)\leq(\Delta(k-1)+1)b$. ◀

Now we can prove the following claim about important $s-t$ separators in G'.

Claim D. *There is an important $s-t$ separator S^* of size at most $(\Delta(k-1)+1)b$ in G' such that $V(H)\subseteq R^-_{G'-S^*}(t)\cup S^*$.*

Proof of Claim D. By Claim C, there is an $s-t$ separator S' in G' of size at most $(\Delta(k-1)+1)b$ such that $V(H)\setminus A\subseteq R^-_{G'-S'}(t)$. Notice that S' not necessary a minimal separator, but there is a minimal $s-t$ separator $S\subseteq S'$. Clearly, $|S|\leq(\Delta(k-1)+1)b$.

We show that $V(H)\subseteq R^-_{G'-S}(t)\cup S$. Because $R^-_{G'-S'}(t)\subseteq R^-_{G'-S}(t)$, we have that $V(H)\setminus A\subseteq R^-_{G'-S}(t)$. Also if an anchor a is in $R^-_{G'-S'}(t)$, then $a\in R^-_{G'-S}(t)$. Let $a\in A\cap S'$. If $a\in A\cap S$, then $a\in R^-_{G'-S}(t)\cup S$. If $a\notin S$, then by Claim C, a has an out-neighbor $v\in R^-_{G'-S'}(t)$ and in this case we have $a\in R^-_{G'-S}(t)$.

It remains to observe that there is an important $s-t$ separator S^* such that $|S^*|\leq|S|\leq(\Delta(k-1)+1)b$ and $R^-_{G'-S}(t)\subseteq R^-_{G'-S^*}(t)$. Therefore, $V(H)\subseteq R^-_{G'-S}(t)\cup S\subseteq R^-_{G'-S^*}(t)\cup S^*$. ◀

The next step of our algorithm is to check all important $s-t$ separators in G' of size at most $(\Delta(k-1)+1)$. By Lemma 1, there are at most $4^{(\Delta(k-1)+1)b}$ important $s-t$ separators and they can be listed in time $2^{O(\Delta^2 b)}\cdot n^c$. For each important $s-t$ separator S^*, we consider the set of vertices $U=R^-_{G'-S^*}(t)\cup S^*$ and decide whether there is a solution such that $V(H)\subseteq U$. If we have a solution for some S^*, then we return a YES-answer and stop. Otherwise, if we fail to find such a solution for all important separators, we use Claim D to deduce that there is no solution.

From now on, we assume that an important $s-t$ separator S^* is given and that $U=R^-_{G'-S^*}(t)\cup S^*$. In what follows, we describe a procedure of finding a solution with $V(H)\subseteq U$.

Denote by D the set $\{v\in U\mid d^-_G(v)>0\}$. We need the following observation.

Claim E. *Set D contains at most $(\Delta+1)(\Delta(k-1)+1)b$ vertices.*

Proof of Claim E. Let $Q=G[U]$. Let $X=\{v\in V(Q)\mid d^-_Q(v)\geq k \text{ and } d^+_Q(v)<k\}$, $Y=\{v\in V(Q)\mid d^-_Q(v)=d^+_Q(v)=k\}$ and $Z=\{v\in V(Q)\mid d^-_Q(v)<k\}$. Clearly,

$$\sum_{v\in X}(d^-_Q(v)-d^+_Q(v))+\sum_{v\in Y}(d^-_Q(v)-d^+_Q(v))=\sum_{v\in Z}(d^+_Q(v)-d^-_Q(v))$$

Observe that $d^-_Q(v)-d^+_Q(v) \geq 1$ for $v \in X$, $d^-_Q(v)-d^+_Q(v) = 0$ for $v \in Y$ and $d^+_Q(v)-d^-_Q(v) \leq \Delta$ for $v \in Z$. Hence, $|X| \leq \Delta|Z|$.

Recall that G' is obtained from G by joining s with all vertices of in-degree at most $k-1$. Since S^* is an $s-t$ separator, if for $v \in U$, $d^-_Q(v) < k$, then $v \in S^*$. Hence, $Z \subseteq S^*$ and $|Z| \leq |S^*| \leq (\Delta(k-1)+1)b$. If for for $v \in U$, $d^-_G(v) > k$, then $v \in X \cup Z$. We conclude that $|D| \leq |X| + |Z| \leq (\Delta+1)|Z| \leq (\Delta+1)(\Delta(k-1)+1)b$. ◀

Recall that set $\delta_{G'}(H)$ contains vertices of H that have in-neighbors outside of H. If $v \in \delta_{G'}(H) \setminus A$, then it has at least k in-neighbors in H and at least one in-neighbor outside H. Notice that $s \notin N^-_{G'}(v)$ because $d^-_G(v) \geq d^-_H(v) \geq k$. Hence, $d^-_G(v) > k$. Because $V(H) \subseteq U$, $\delta_{G'}(H) \setminus A \subseteq D$. By Claim C, $|\delta_{G'}(H) \setminus A| \leq \Delta b$, and by Claim E, $|D| \leq (\Delta+1)(\Delta(k-1)+1)b$. We consider all at most $2^{(\Delta+1)(\Delta(k-1)+1)b}$ possibilities to select $\delta_{G'}(H) \setminus A$. For each choice of $\delta_{G'}(H) \setminus A$, we guess the arcs that join the vertices that are outside H with the vertices of $\delta_{G'}(H) \setminus A$ and delete them. Denote the graph obtained from G by F. Recall that from each vertex v of $\delta_{G'}(H) \setminus A$, there is a directed path to t that avoids A. Hence, v has at least one out-neighbor in H and at most $\Delta-1$ in-neighbors in G. Also v has at least k in-neighbors in H, and we delete at most $d^-_G(v)-k$ arcs. Therefore, for v we choose at most $k-1$ arcs out of at most $\Delta-1$ arcs. We can upper bound the number of possibilities for v by $2^{\Delta-1}$, and the total number of possibilities for $\delta_{G'}(H) \setminus A$ by $2^{(\Delta-1)\Delta b}$.

Observe that (A, H) is a solution for the new instance of Dir-AKC, where G is replaced by F for a correct guess of the deleted arcs. Also each solution for the new instance provides a solution for the graph G, because if we put deleted arcs back, then we can only increase the in-degrees. Hence, we can check for each possible choice of the set of deleted arcs, whether the new instance has a solution. If for some choice we obtain a solution, then we return a YES-answer. Otherwise, if we fail for all choices, then we return a NO-answer. Further we assume that F is given.

Denote by F' the graph obtained from F by the addition of a vertex s joined by arcs with all the vertices $N^+_{G'}(s)$. Now $\delta_{F'}(H) = \{v \in V(H) \mid N^-_{F'}(v) \setminus V(H) \neq \emptyset\}$. By the choice of F, $\delta_{F'}(H) = \delta_{G'}(H) \cap A$ and, therefore, $|\delta_{F'}(H)| \leq b$. Also $\delta_{F'}(H)$ is an $s-t$ separator in F' by Claim C.

Now we can prove the following.

Claim F. *There is an important $s-t$ separator $\hat{S}$ of size at most b in F' such that $(\hat{S}, G[R^-_{F'-\hat{S}}(t) \cup \hat{S}])$ is a solution for the instance of the* Dir-AKC *problem for the graph G.*

Proof of Claim F. Let $U = R^-_{F'-\hat{S}}(t) \cup \hat{S}$. It was already observed that $\delta^*_{G'}(H)$ is an $s-t$ separator in F' of size at most b. Then there is a minimal $s-t$ separator $S \subseteq \delta^*_{G'}(H)$. Clearly, $|S| \leq b$.

As before in the proof of Claim D, we show that $V(H) \subseteq R^-_{F'-S}(t) \cup S$. Because for any vertex v of H, there is a directed (v,t) path with all inner vertices in $V(H) \setminus A$, $V(H) \setminus A \subseteq R^-_{F'-\delta_{F'}(H)}(t)$. Because $R^-_{F'-\delta_{F'}(H)}(t) \subseteq R^-_{F'-S}(t)$ we have $V(H) \setminus A \subseteq R^-_{F'-S}(t)$. Also if $a \in A$ is in $R^-_{F'-\delta_{F'}(H)}(t)$, then $a \in R^-_{F'-S}(t)$. Let $a \in A \cap \delta_{F'}(H)$. Trivially, if $a \in A \cap S$, then $a \in R^-_{F'-S}(t) \cup S$. If $a \notin S$, then a has an out-neighbor $v \in R^-_{F'-\delta_{F'}(H)}(t)$ and $a \in R^-_{F'-S}(t)$. Then there is an important $s-t$ separator $\hat{S}$ such that $|\hat{S}| \leq |S| \leq b$ and $R^-_{F'-S}(t) \subseteq R^-_{F'-\hat{S}}(t)$. Therefore, $V(H) \subseteq R^-_{F'-S}(t) \cup S \subseteq R^-_{F'-S^*}(t) \cup S^*$, and $|U| \geq p$.

It remains to observe that s is adjacent to all vertices of G with in-degrees at most $k-1$ and S^* is an $s-t$ separator. It immediately follows that for any vertex $v \in R^-_{F'-S^*}(t)$, $d^-_{F(U)}(v) \geq k$. Then $(\hat{S}, G[R^-_{F'-\hat{S}}(t) \cup \hat{S}])$ is a solution. ◀

The final step of our algorithm is to enumerate all important $s-t$ separators $\hat{S}$ of size at most b in F', which number by Lemma 1 is at most 4^b, and for each $\hat{S}$, check whether $(\hat{S}, G[R^-_{F'-\hat{S}}(t) \cup \hat{S}])$ is a solution. Recall that all these separators can be listed in time $2^{O(b)} \cdot n^c$. We return a YES-answer if we obtain a solution for some important separator, and a NO-answer otherwise.

To complete the proof, let us observe that each step of the algorithm runs either in polynomial or FPT time. Particularly, the preprocessing is done in time $O(\Delta n)$. Then we check the existence of a solution of a bounded size in time $2^{O(\Delta^2 bp)} \cdot n \log n$. Further we consider at most n possibilities to choose t. For each t, we consider at most $4^{(\Delta(k-1)+1)b}$ important $s-t$ separators S^*. Recall, that they can be listed in time $2^{O(\Delta^2 b)} \cdot n^c$ for some constant c. Then for each S^*, we have at most $2^{(\Delta+1)(\Delta(k-1)+1)b+(\Delta-1)}$ possibilities to construct F, and it can be done in time $2^{O(\Delta^3 b)} + O(\Delta n)$. Finally, there are at most 4^b important $s-t$ separators $\hat{S}$ and they can be listed in time $2^{O(b)} \cdot n$ for some c. We conclude that the total running time is $2^{O(\Delta^3 b+\Delta^2 bp)} \cdot n^c$ for some constant c. ◀

Combining Lemmas 6 and 7, we obtain the following theorem.

▶ **Theorem 8.** *Let Δ be a positive integer. If $k \geq \frac{\Delta}{2}$, then the* Dir-AKC *problem can be solved in time $2^{O(\Delta^3 b+\Delta^2 bp)} \cdot n^{O(1)}$ for n-vertex directed graphs of maximum degree at most Δ.*

Theorems 3 and 8 give the next corollary.

▶ **Corollary 9.** *The* Dir-AKC *problem can be solved in time $2^{O(bp)} \cdot n^{O(1)}$ for n-vertex directed graphs of maximum degree at most 4.*

5 Conclusions

We proved that Dir-AKC is NP-complete even for planar DAGs of maximum degree at most $k+2$. It was also shown that Dir-AKC is FPT when parameterized by $p+\Delta$ for directed graphs of maximum degree at most Δ whenever $k \geq \Delta/2$. It is natural to ask whether the problem is FPT for other values k. This question is interesting even for the special case $\Delta = 5$ and $k = 2$.

For the special case of directed acyclic graphs (DAGs) we understand the complexity of the problem much better. Theorem 4 showed that Dir-AKC on DAGs is W[1]-hard parameterized by p for every fixed $k \geq 2$, when the degree of the graph is not bounded. We now show the following theorem (the proof is in [5]) that gives W[2]-hardness of Dir-AKC when parameterized by the number of anchors b (recall that we can always assume that $b \leq p$).

▶ **Theorem 10.** *For any $\Delta \geq 3$ and any positive $k < \frac{\Delta}{2}$,* Dir-AKC *is* W[2]*-hard (even on DAGs) when parameterized by the number of anchors b on graphs of maximum degree at most Δ.*

The complexity of Dir-AKC parameterized by b on DAGs for the case of $k \geq \frac{\Delta}{2}$ is left open. However we can show that Dir-AKC is FPT on DAGs of maximum degree Δ, when parameterized by $\Delta + p$ (the proof is in [5].)

▶ **Theorem 11.** *For any positive integers p and Δ,* Dir-AKC *can be solved in time $2^{O(\Delta p)} \cdot n^2 \log n$ for n-vertex DAGs of maximum degree at most Δ.*

Let us remark that this result can be easily extended for any class of directed acyclic graphs $\mathcal{G}$ such that the corresponding class of underlaying graphs $\{G^* | G \in \mathcal{G}\}$ has (locally) bounded expansion by making use of the results by Dvorak et al. [11]. Finally, what happens when the input graph is planar? We know that the problem is NP-complete on planar graphs for fixed $k \geq 1$ and maximum degree $k + 2$. Is the problem FPT on planar directed graphs when parameterized by the size of the core p?

References

1 N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
2 K. Bhawalkar, J. M. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: The anchored k-core problem. In *ICALP '12*, vol. 7392 of *Lecture Notes in Computer Science*, pages 440–451, 2012.
3 M. Bläser. Computing small partial coverings. *IPL.*, 85(6):327–331, 2003.
4 L. Cai, S. M. Chan, and S. O. Chan. Random separation: A new method for solving fixed-cardinality optimization problems. In *IWPEC'06*, vol. 4169 of *Lecture Notes in Computer Science*, pages 239–250, 2006.
5 Ra. H. Chitnis, F. V. Fomin, and P. A. Golovach. Parameterized complexity of the anchored k-core problem for directed graphs. *CoRR*, abs/1304.5870, 2013.
6 R. H. Chitnis, F. V. Fomin, and P. A. Golovach. Preventing unraveling in social networks gets harder. In *AAAI '13*. AAAI Press, 2013.
7 R. H. Chitnis, M. T. Hajiaghayi, and D. Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. In *SODA '12*, pages 1713–1725. SIAM, 2012.
8 M. Chwe. Structure and Strategy in Collective Action 1. *American Journal of Sociology*, 105(1):128–156, 1999.
9 M. Chwe. Communication and Coordination in Social Networks. *The Review of Economic Studies*, 67(1):1–16, 2000.
10 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
11 Z. Dvorak, D. Král, and R. Thomas. Deciding first-order properties for sparse graphs. In *FOCS*, pages 133–142. IEEE Computer Society, 2010.
12 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.
13 D. Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006.
14 R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.
15 T.C. Schelling. *Micromotives and Macrobehavior*. WW Norton, 2006.
16 D. Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.
17 S. B. Seidman. Network Structure and Minimum Degree. *Social networks*, 5(3):269–287, 1983.
18 R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

Böhm Trees as Higher-Order Recursive Schemes

Pierre Clairambault[1] and Andrzej S. Murawski[2]

1 **CNRS, ENS de Lyon, Inria, UCBL, Université de Lyon, Laboratoire LIP**
2 **DIMAP and Department of Computer Science, University of Warwick**

Abstract

Higher-order recursive schemes (HORS) are schematic representations of functional programs. They generate possibly infinite ranked labelled trees and, in that respect, are known to be equivalent to a restricted fragment of the λY-calculus consisting of ground-type terms whose free variables have types of the form $o \to \cdots \to o$ (with o being a special case).

In this paper, we show that *any* λY-term (with no restrictions on term type or the types of free variables) can actually be represented by a HORS. More precisely, for any λY-term M, there exists a HORS generating a tree that faithfully represents M's (η-long) Böhm tree. In particular, the HORS captures higher-order binding information contained in the Böhm tree. An analogous result holds for finitary PCF.

As a consequence, we can reduce a variety of problems related to the λY-calculus or finitary PCF to problems concerning higher-order recursive schemes. For instance, Böhm tree equivalence can be reduced to the equivalence problem for HORS. Our results also enable MSO model-checking of Böhm trees, despite the general undecidability of the problem.

1998 ACM Subject Classification F.3.3 Studies of Program Constructs, F.4.1 Mathematical Logic

Keywords and phrases Lambda calculus, Böhm trees, Recursion Schemes

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.91

1 Introduction

Higher-order recursive schemes (HORS) are a class of programming schemes introduced to account for recursive procedures with higher-order parameters [7]. They can be viewed as grammars describing a potentially infinite tree. As tree generating devices, HORS can be identified with a limited fragment of the λY-calculus [19] consisting of ground-type terms whose free variables have types of order at most 1. The free variables play the role of tree constructors. HORS have recently been intensively investigated in connection with program verification. Notably, Ong [16] showed that monadic second-order logic (MSO) is decidable over trees generated by HORS and Kobayashi [12] took advantage of the result to propose a novel approach to the verification of higher-order functional programs.

As already mentioned, HORS are naturally viewed as a fairly small fragment of the λY-calculus: they generate potentially infinite trees, whereas normal forms of arbitrary λY-terms additionally contain variable bindings. This binding information cannot be represented in HORS explicitly. In fact, it turns out that MSO becomes undecidable over Böhm trees of λY-terms, if the binding relation is included in the signature. Nevertheless, as we show in the paper, HORS still allow one to generate faithful representations of (η-long) Böhm trees of λY-terms, where binding is encoded indirectly via *De Bruijn levels* [8]. We also prove an analogous representation theorem for the finitary (finite datatypes) variant PCF_f of PCF [17].

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 91–102

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

Our results make it possible to recast a variety of problems related to the λY-calculus or finitary PCF as problems concerning higher-order recursive schemes. For example, we obtain a reduction of Böhm tree equivalence in the λY-calculus or PCF_f to the (tree) equivalence problem for HORS. Unfortunately, the latter is currently not known to be decidable and is closely related to the equivalence problem for *deterministic* collapsible pushdown automata [10]. The Böhm tree equivalence problem for PCF_f also has semantic significance, because PCF Böhm trees [2] are concrete representations of the strategies representing the terms in game semantics, and therefore characterize contextual equivalence in PCF with respect to contexts featuring state and control effects.

Other consequences, in the form of decidability results, can be derived by applying Ong's decidability result to representations of Böhm trees obtained through our theorems. In this way, one can show that numerous problems for λY or PCF_f are decidable. Examples include normalizability, finiteness, solvability or having a Böhm tree prefixed by a given finite term. Some of the results are already known, while others appear new.

Thus far, higher-order verification based on HORS focussed on model-checking trees extracted from programs. Our contribution opens the perspective of applying model-checking to terms with binding and arbitrary free variables, such as higher-order components of closed programs.

2 λY-calculus

2.1 Böhm trees of λY-terms

We work with simple types built from a single atom o using the arrow type constructor. They are defined by the grammar given below.

$$\theta ::= o \mid \theta \to \theta$$

The order of a type is defined as follows.

$$ord(o) = 0 \qquad ord(\theta_1 \to \theta_2) = \max(ord(\theta_1) + 1, ord(\theta_2))$$

Terms are considered up to α-equivalence, and equipped with β-reduction and η-expansion (respectively written $\to_\beta$ and $\to_\eta$). We write $\simeq_{\beta\eta}$ for the symmetric reflexive and transitive closure of $\to_\beta$ and $\to_\eta$. We shall consider several extensions of the simply-typed λ-calculus over the types introduced above.

- The $\lambda\bot$-calculus additionally contains a constant $\bot_o : o$. More generally, we shall write $\bot_\theta$ for terms specified as follows.

$$\bot_\theta = \begin{cases} \bot_o & \theta \equiv o \\ \lambda x^{\theta_1}.\bot_{\theta_2} & \theta \equiv \theta_1 \to \theta_2 \end{cases}$$

  For $\lambda\bot$-terms, let us define a partial order $\sqsubseteq$ (relating only terms with equal types) by

$$\frac{}{M \sqsubseteq M} \qquad \frac{}{\bot_o \sqsubseteq M} \qquad \frac{M_1 \sqsubseteq M_2}{\lambda x.M_1 \sqsubseteq \lambda x.M_2} \qquad \frac{M_1 \sqsubseteq M_1' \qquad M_2 \sqsubseteq M_2'}{M_1\ M_2 \sqsubseteq M_2'\ M_2'}.$$

  We will also consider the extension of the $\lambda\bot$-calculus to infinite terms, which we refer to as the $\lambda\bot^\infty$-calculus. The partial order $\sqsubseteq$ can be extended to $\lambda\bot^\infty$-terms to yield an ω-cpo.
- The λY-calculus contains a family of fixed-point operators $Y_\theta : (\theta \to \theta) \to \theta$, where θ ranges over arbitrary types, equipped with the reduction rule $Y_\theta\ M \to_Y M\ (Y_\theta\ M)$.

▶ **Definition 1.** Given a λY-term M and $n \in \mathbb{N}$, the nth approximant of M, written $M \upharpoonright n$, is a $\lambda\perp$-term defined by

$$\begin{array}{rclcrcl} x \upharpoonright n & = & x & & \lambda x.M \upharpoonright n & = & \lambda x.(M \upharpoonright n) \\ M\ N \upharpoonright n & = & (M \upharpoonright n)\ (N \upharpoonright n) & & Y_\theta \upharpoonright n & = & \lambda f^{\theta\to\theta}.f^n(\perp_\theta). \end{array}$$

▶ **Definition 2.** The η-long Böhm tree of a $\lambda\perp$-term $\Gamma \vdash M : \theta$, written $\mathsf{BT}(M)$, is its β-normal η-long form. For a λY-term $\Gamma \vdash M$, the η-long Böhm tree, also denoted by $\mathsf{BT}(M)$, is defined to be $\bigsqcup_n \mathsf{BT}(M \upharpoonright n)$.

Being η-long, these infinite normal forms might be more adequately called Nakajima trees [15]. However, their PCF counterparts are generally called PCF *Böhm trees* [6]. So, for consistency, we call them *η-long Böhm trees*, and for conciseness we will often omit *η-long*.

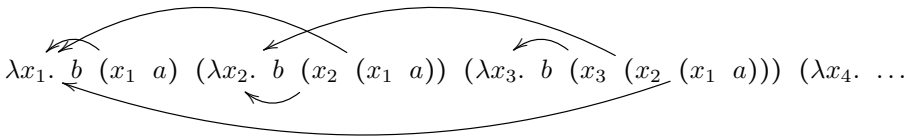
▶ **Definition 3.** λY-terms satisfying $\Gamma \vdash M : o$ are called *ground*. Given $n \geq 0$, we shall say that a ground term $\Gamma \vdash M : o$ is of *level* n if, for all $(x : \theta) \in \Gamma$, we have $ord(\theta) < n$.

Note that the free variables in a ground term of level 2 can have types of order 0 or 1, i.e. they are of the form $o \to \cdots \to o$, where o has at least one occurrence. Thus, Böhm trees of such terms are (possibly infinite) trees.

The restrictions on types of free variables in ground terms of level 2 are analogous to the restriction concerning types of terminal symbols in higher-order recursion schemes [7]. In fact, it can be shown that the Böhm trees of level-2 ground terms coincide with trees generated by higher-order recursive schemes [18], and therefore have a decidable MSO theory.

In this paper, we are interested in the study of the Böhm trees of *arbitrary* λY-terms. Unlike in the case of level-2 ground terms, Böhm trees of arbitrary λY-terms involve *binders*, as illustrated by the following example.

▶ **Example 4.** Consider $G = Y_{o\to(o\to o)\to o}\ (\lambda f^{o\to(o\to o)\to o}.\lambda y^o.\lambda x^{o\to o}.\ b\ (x\ y)\ (f\ (x\ y)))\ a$, which has type $(o \to o) \to o$ in context $a : o, b : o \to ((o \to o) \to o) \to o$. Its Böhm tree is



where the arrows indicate *binding information*: for each variable occurrence, the arrow indicates the location of the associated binder. Note that because all bound variables are used infinitely often, countably many variable names are required to represent binding.

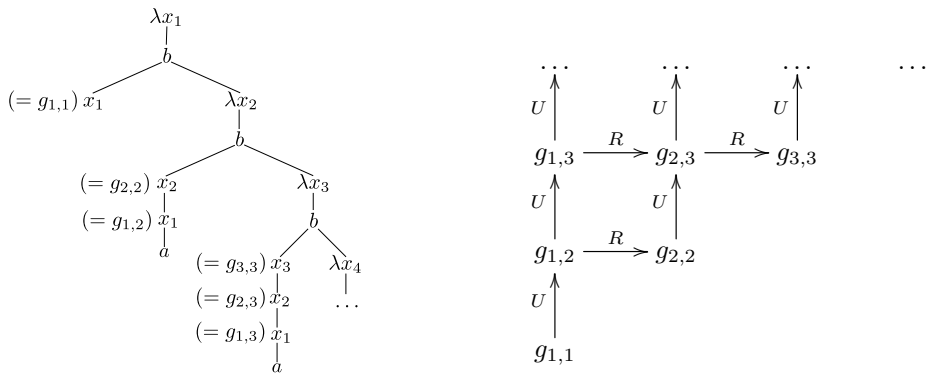
This binding information is far from innocent: in fact, we prove in the next section that it makes MSO undecidable. In particular, the term G above has an undecidable MSO theory.

2.2 Undecidability of MSO with binders

Let us first make formal what we mean by MSO on Böhm trees with binders.

▶ **Definition 5.** A **binding structure** $\mathcal{B} = (\mathcal{S}, \to)$ is a labelled transistion system, namely a set of *states* $\mathcal{S}$ and a relation $\to \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$, where $\mathcal{L}$ is the set of *labels* $\mathcal{L} = \mathbb{N} \cup \{\lambda\}$. For $l \in \mathcal{L}$, we write $a \xrightarrow{l} b$ for $(a, l, b) \in \to$.

Every $\lambda\perp^\infty$-term M defines a binding structure $\mathcal{B}(M)$ in which the states are nodes in the syntax tree of M, $a \xrightarrow{n} b$ holds if b is the n-th child of a and $a \xrightarrow{\lambda} b$ expresses the fact that



(a) The Böhm tree of G **(b)** The half-grid in $\mathsf{BT}(G)$

Figure 1 Construction of the half-grid.

b is a binder-node λx and a an occurrence of x. The MSO formulas over binding structures are given as follows.

$$\phi \quad ::= \quad X\ \mathrm{x} \mid \mathrm{x} \xrightarrow{\lambda} \mathrm{y} \mid \mathrm{x} \xrightarrow{n} \mathrm{y} \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \exists \mathrm{x}.\ \phi \mid \forall \mathrm{x}.\ \phi \mid \exists X.\ \phi \mid \forall X.\ \phi$$

We write the first-order variables x, y in a different font to distinguish them from the variable names of the λ-calculus. The validity of an MSO formula ϕ on a binding structure $\mathcal{B}$, written $\mathcal{B} \vDash \phi$, is defined as usual.

▶ **Theorem 6.** *MSO is undecidable on binding structures corresponding to Böhm trees of λY-terms.*

Proof. We reduce the well-known undecidable problem of deciding MSO on the infinite half-grid to MSO over binding structures generated by λY-terms. In particular, we will show that the binding structure $\mathcal{B}(\mathsf{BT}(G))$ contains an MSO-definable half-grid. The nodes of the half-grid will be occurrences of bound variables in $\mathsf{BT}(G)$, which are definable by the formula $\mathrm{grid}(\mathrm{x}) \triangleq \exists \mathrm{y}.\ \mathrm{x} \xrightarrow{\lambda} \mathrm{y}$.

We say that a bound variable occurrence x_i is at *layer* j if it occurs after j applications of b. Then for any $1 \le i \le j$, there is a unique occurrence of the bound variable x_i at layer j, we denote it by $g_{i,j}$. The $g_{i,j}$'s will be the nodes of the half-grid; in Figure 1a we describe the intended correspondence between occurrences of bound variables in $\mathsf{BT}(G)$ and the half-grid. The following two relations correspond respectively to moving right and up inside the grid.

$$\mathrm{x} \xrightarrow{R} \mathrm{y} \quad \triangleq \quad \mathrm{y} \xrightarrow{0} \mathrm{x}$$
$$\mathrm{x} \xrightarrow{U} \mathrm{y} \quad \triangleq \quad \exists \mathrm{z}.\ \mathrm{z} \xrightarrow{0^*} \mathrm{x} \wedge \mathrm{z} \xrightarrow{10^*} \mathrm{y} \wedge \exists \mathrm{x}'.\ \mathrm{x} \xrightarrow{\lambda} \mathrm{x}' \wedge \mathrm{y} \xrightarrow{\lambda} \mathrm{x}'$$

where $\xrightarrow{e_1 e_2}$ and $\xrightarrow{e^*}$ are respectively the relational composition of $\xrightarrow{e_1}$ and $\xrightarrow{e_2}$ and the transitive closure of $\xrightarrow{e}$, both MSO-definable. The relations define the half-grid in the sense that for any $1 \le i \le j$ and $1 \le i' \le j'$ we have $g_{i,j} \xrightarrow{R} g_{i',j'}$ iff $i' = i + 1$ and $j' = j$, and $g_{i,j} \xrightarrow{U} g_{i',j'}$ iff $i' = i$ and $j' = j + 1$. Consequently, the half-grid is MSO-definable within $\mathsf{BT}(G)$ and, thus, the latter must have an undecidable MSO theory. ◀

2.3 De Bruijn representations of binders

Theorem 6 exhibits a difference in expressivity between HORS and the λY-calculus. Nonetheless, in the following, we shall show how to construct HORS that faithfully capture

Böhm trees generated by λY-terms. To that end, we shall use binder-free representations. As a consequence, we will be able to recast problems concerning the λY-calculus in the setting of HORS. In particular, we will retain the decidability of MSO model-checking such representations, which will provide us with a technique to verify a variety of properties of λY-terms in spite of Theorem 6.

Our binder-free representation scheme will rely on *De Bruijn levels* [8]: each bound variable x is given an index i, where i is the sum of a starting index and the number of lambdas enclosing the binding location (De Bruijn levels should not be confused with De Bruijn *indices*, which associate numbers to variable uses and not their points of introduction, and can associate different numbers to different occurrences of the same variable).

▶ **Example 7.** Consider the term $M = \lambda f^{(o\to o)\to o}.Y_o\ (\lambda y^o.f\ (\lambda x^o.b\ x\ y))$, of type $((o \to o) \to o) \to o$ in context $b : o \to o \to o$. The Böhm tree of M, annotated with De Bruijn levels, has the shape $\lambda x_1.x_1\ (\lambda x_2.x_0\ x_2\ (x_1\ (\lambda x_3.x_0\ x_3\ (x_1\ (\lambda x_4 \ldots$, where $b : o \to o \to o$ is given an index of 0. Note that each variable introduction and use are now labelled with a natural number. This representation is unique, in the sense that, for a fixed choice of indices for free variables and a choice of a starting index, two α-equivalent terms must have the same representation.

With De Bruijn levels, $\lambda\bot$-terms can be represented as level-2 ground terms, typable in a special context defined below.

▶ **Definition 8.** Let Γ_{rep} be the following context.

$$\{\, z : o,\ \ succ : o \to o,\ \ var : o \to o,\ \ app : o \to o \to o,\ \ lam : o \to o \to o \,\}$$

Given $n \in \mathbb{N}$, we write $\overline{n}$ for the term $succ^n(z)$.

In particular, the terms $\overline{n}$ will be used to represent binding information as De Bruijn levels.

▶ **Definition 9.** Let $\Gamma \vdash M : \theta$ be a $\lambda\bot$-term and let $\nu : \Gamma \to \mathbb{N}$ be an injection. We define another $\lambda\bot$-term $\Gamma_{\mathsf{rep}} \vdash \mathsf{rep}_\nu(M, n) : o$ by

$$\begin{array}{lcl@{\qquad}lcl} \mathsf{rep}_\nu(\bot, n) & = & \bot & \mathsf{rep}_\nu(\lambda x.M, n) & = & lam\ \overline{n}\ \mathsf{rep}_{\nu\oplus\{x\mapsto n\}}(M, n+1) \\ \mathsf{rep}_\nu(x, n) & = & var\ \overline{\nu(x)} & \mathsf{rep}_\nu(MN, n) & = & app\ \mathsf{rep}_\nu(M, n)\ \mathsf{rep}_\nu(N, n). \end{array}$$

Note that, as long as $n > \max_{(x:\theta)\in\Gamma} \nu(x)$, $\mathsf{rep}_\nu(M, n)$ faithfully represents the syntactic structure of M. Given $\Gamma \vdash M : \theta$, where $\Gamma = \{x_1 : \theta_1, \cdots, x_k : \theta_k\}$, let $\nu : \Gamma \to \mathbb{N}$ be defined by $\nu(x_i) = i - 1$. We define $\mathsf{rep}(M)$ to be $\mathsf{rep}_\nu(M, k)$.

For any finite $\lambda\bot$-term M, $\mathsf{rep}(M)$ gives a binder-free representation of M as a ground $\lambda\bot$-term in context Γ_{rep}. Being monotonic, the function rep preserves ω-chains, and therefore can be used to represent the Böhm tree of any λY-term as a ground infinite $\lambda\bot^\infty$-term in context Γ_{rep}. This invites the question whether this infinite $\lambda\bot^\infty$-term can always be obtained as the Böhm tree of a level-2 ground λY-term.

▶ **Example 10.** Consider again the term M of Example 7. The binder-free representation of M's Böhm tree can be captured as an open (infinite) term in context Γ_{rep}, namely, $M_\infty = lam\ \overline{1}\ T(2)$, where

$$T(n) = app\ (var\ \overline{1})\ (lam\ \overline{n}\ (app\ (app\ (var\ \overline{0})\ (var\ \overline{n})\ T(n+1))))$$

More precisely, we have $\bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n)) = M_\infty$. In this case, it is easy to extract a λY-term $\Gamma_{\mathsf{rep}} \vdash M_{rep} : o$ such that $\mathsf{BT}(M_{rep}) = M_\infty$ from the definition of M_∞. This

can be done by expressing the coinductive definition of M_∞ inside the λY-calculus as $M_{rep} = lam\ \overline{1}\ (Y_{o\to o}\ M_{step}\ \overline{2})$, where

$$M_{step} = \lambda T^{o\to o}.\ \lambda n^{o}.app\ (var\ \overline{1})\ (lam\ n\ (app\ (app\ (var\ \overline{0})\ (var\ n))\ (T\ (succ\ n)))).$$

Consequently, the Böhm tree of M_{rep} is a binder-free representation of the Böhm tree of M.

In our paper, we show how to obtain such representations in a systematic way.

▶ **Theorem 11.** *Let $\Gamma \vdash M : \theta$ be a λY-term. There exists a level-2 λY-term $\Gamma_{\mathsf{rep}} \vdash M_{rep} : o$ such that $\mathsf{BT}(M_{rep}) = \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$.*

Next we describe the construction underlying our proof of Theorem 11. In this section we only provide the necessary definitions, delegating the proof and a methodological discussion to Section 3. The construction uses *normalization by evaluation* [9] and, in particular, the observation that the technique can be internalized within the λY-calculus in our case.

Consider $\Gamma \vdash M : \theta$ with $\Gamma = \{x_1 : \theta_1, \cdots, x_k : \theta_k\}$. First we apply a simple transformation on type annotations inside M and substitute $o \to o$ for o: let M^* be defined as $M[o \to o/o]$. The substitution is meant to be applied to types θ occurring in λ-abstractions $(\lambda x^{\theta}.M)$ and fixed-point combinators (Y_θ). Observe that $x_1 : \theta_1^*, \cdots, x_k : \theta_k^* \vdash M^* : \theta^*$, where

$$o^* = o \to o, \qquad (\theta' \to \theta'')^* = (\theta')^* \to (\theta'')^*.$$

Next we define by mutual recursion two families of λ-terms $\{\downarrow_\theta\}$, $\{\uparrow_\theta\}$, subject to the following conditions: $\Gamma_{\mathsf{rep}} \vdash \downarrow_\theta : \theta^* \to o \to o$ and $\Gamma_{\mathsf{rep}} \vdash \uparrow_\theta : (o \to o) \to \theta^*$. We write $\lambda.M$ for $\lambda z^o.M$, where z does *not* occur in M.

$$\begin{aligned}
\downarrow_o &= \lambda x^{o\to o}.x \\
\downarrow_{\theta'\to\theta''} &= \lambda x^{(\theta'\to\theta'')^*}.\ \lambda v^o.\ lam\ v\ (\downarrow_{\theta''}\ (\ x\ (\uparrow_{\theta'}\ \lambda.(var\ v))\ )\ (succ\, v)\ ) \\
\uparrow_o &= \lambda x^{o\to o}.x \\
\uparrow_{\theta'\to\theta''} &= \lambda e^{o\to o}.\ \lambda a^{(\theta')^*}.\ \uparrow_{\theta''}\ (\ \lambda v^o.\ app\ (e\ v)\ (\downarrow_{\theta'}\ a\ v)\ )
\end{aligned}$$

Now, with all the definitions in place, it turns out that one can take M_{rep} to be

$$\downarrow_\theta\ \ M^*[\ \uparrow_{\theta_1} \lambda.(var\ \overline{0})/x_1\ ,\ \cdots\ ,\ \uparrow_{\theta_k} \lambda.(var\ \overline{k-1})/x_k\ ]\ \ \overline{k}.$$

In the next section we shall prove that M_{rep} indeed represents the Böhm tree of M in the sense of Theorem 11. Before that, we illustrate the outcome of the construction on an example.

▶ **Example 12.** Let us revisit the term M from Example 7. We have

$$M^* = \lambda f^{((o\to o)\to(o\to o))\to(o\to o)}.Y_{o\to o}\ (\lambda y^{o\to o}.f\ (\lambda x^{o\to o}.b\ x\ y))$$

and, thus, $M_{rep} = \downarrow_{((o\to o)\to o)\to o}\ \ M^*[\uparrow_{o\to o\to o} \lambda.(var\ \overline{0})/b]\ \ \overline{1}$. By unfolding the definitions of $\downarrow_\theta, \uparrow_\theta$ and applying β-reduction the term can be rewritten. In fact, taking advantage of the β-equivalences

$$\begin{aligned}
\uparrow_{o\to o\to o} &\simeq_\beta \lambda e_1^{o\to o}.\lambda a_1^{o\to o}.\lambda a_2^{o\to o}.\lambda v_2^o.\ app\ (app\ (e_1\ v_2)\ (a_1\ v_2))\ (a_2\ v_2) \\
\downarrow_{((o\to o)\to o)\to o} &\simeq_\beta \lambda x^{(((o\to o)\to o)\to o)^*}.\lambda v_1^o.\ lam\ v_1\ (x\ M_{aux}\ (succ\ v_1)),
\end{aligned}$$

where $M_{aux} = \lambda a^{(o\to o)^*}.\lambda v_2^o.\ app\ (var\ v_1)\ (lam\ v_2\ (a\ (\lambda.var\ v_2)\ (succ\ v_2)))$, one can show that M_{rep} is β-equivalent to

$$lam\ \overline{1}\ (Y\ (\lambda y.\ \lambda v.app\ (var\ \overline{1})\ (lam\ v\ (app\ (app\ (var\ \overline{0})\ (var\ v))\ (y\ (succ\ v))))))\ \overline{2}),$$

the manually constructed term from Example 10.

3 Internalized normalization by evaluation

In this section, we present the mathematical development that led to the term transformation of the previous section. As already mentioned, the main idea comes from *normalization by evaluation* (NBE). NBE is a general method used to construct normal forms for terms through their denotational semantics; the reader is referred to [9] for an introduction. The basic idea is to interpret terms in a suitable semantic universe. By soundness, the interpretation will be invariant under syntactic reduction. Moreover, if one uses a class of domains expressive enough to encode representations of normal forms, then one can attempt to extract a representation of the normal form of a term from the semantics. Notably, the extraction can be performed by evaluating the interpretation map on well-chosen elements of the domain, which yields an entirely semantic normalization procedure. NBE has been described for a variety of languages, including the simply-typed λ-calculus [5] as well as richer languages with coproducts [4], or type theory [1]. Our presentation of NBE follows along the lines of [9]. The main novelty is the internalization of the process within the λY-calculus itself, which ultimately allows us to present NBE as a term transformation.

3.1 Semantic NBE for the λY-calculus

We first describe a domain semantics for the λY-calculus that will be exploited to obtain an NBE procedure. We assume the basic vocabulary of domain theory, e.g. [20]. Let us recall that an ω-cpo is a partial order where each ω-chain (of the form $x_1 \leq x_2 \leq \dots$) has a supremum. We will interpret types as *pointed* ω-cpos, i.e., ω-cpos with a bottom element written $\bot$. A function $f : A \to B$ is *continuous* if it preserves suprema of ω-chains. The set of continuous functions from A to B, ordered pointwise, is itself a pointed ω-cpo, denoted by $A \to B$.

Our NBE procedure will generate representations of Böhm trees of λY-terms as elements of the pointed ω-cpo of $\lambda\bot^\infty$-terms of ground type in context Γ_{rep}, henceforth referred to as E. Representations based on De Bruijn levels lack compositionality: the indices present in a subterm depend on the number of variables abstracted in the context in which the subterm occurs. In order to overcome the difficulty, instead of constructing directly De Bruijn terms in E, we will construct *term families* [9], intuitively functions $\mathbb{N} \to E$, which - unlike De Bruijn terms - can be manipulated compositionally.

We now describe our semantic NBE for λY. In our case, E will also be used to represent natural numbers, so term families will be interpreted as elements of the pointed ω-cpo $\widehat{E} = E \to E$, where the first E plays the role of $\mathbb{N}$. For the sake of clarity, we will sometimes write N instead of E to highlight subterms representing natural numbers. This should not create any confusion.

Our NBE relies on the standard interpretation of the λY-calculus with $[\![o]\!] = \widehat{E}$. Note that we write $\boldsymbol{\lambda}$, rather than λ, to stress that the semantic operation of function abstraction (on continuous functions between pointed ω-cpos) is meant.

$$[\![o]\!] = \widehat{E} \qquad [\![x]\!]_\rho = \rho(x) \qquad [\![\bot]\!]_\rho = \bot \qquad [\![\lambda x^\theta.M]\!]_\rho = \boldsymbol{\lambda} a^{[\![\theta]\!]}.[\![M]\!]_{\rho \oplus \{x \mapsto a\}}$$

$$[\![\theta_1 \to \theta_2]\!] = [\![\theta_1]\!] \to [\![\theta_2]\!] \qquad [\![M\ N]\!]_\rho = [\![M]\!]_\rho([\![N]\!]_\rho) \qquad [\![Y_\theta]\!]_\rho = \bigsqcup_n \boldsymbol{\lambda} f^{[\![\theta]\!] \to [\![\theta]\!]}.f^n(\bot)$$

We claimed above that unlike raw De Bruijn terms, term families can be constructed compositionally. This is done with the help of the following continuous functions, which

generalize term constructors of E to $\widehat{E}$.

$$\begin{array}{lcl}\widehat{var} & = & \boldsymbol{\lambda} v^N.\boldsymbol{\lambda} n^N.var\ v : N \to \widehat{E}\\ \widehat{app} & = & \boldsymbol{\lambda} e_1^{\widehat{E}}.\boldsymbol{\lambda} e_2^{\widehat{E}}.\boldsymbol{\lambda} n^N.app\ (e_1(n))\ (e_2(n)) : \widehat{E} \to \widehat{E} \to \widehat{E}\\ \widehat{lam} & = & \boldsymbol{\lambda} f^{N\to\widehat{E}}.\boldsymbol{\lambda} n^N.lam\ n\ (f(n)(succ\ n)) : (N \to \widehat{E}) \to \widehat{E}\end{array}$$

Using these extended constructors, one can define families of continuous functions, traditionally called *reification* ($\Downarrow_\theta \colon [\![\theta]\!] \to \widehat{E}$) and *reflection* ($\Uparrow_\theta \colon \widehat{E} \to [\![\theta]\!]$).

$$\begin{array}{lcllcl}\Downarrow_o x & = & x & \Downarrow_{\theta_1\to\theta_2} x & = & \widehat{lam}(\boldsymbol{\lambda} n^N.\ \Downarrow_{\theta_2} (x(\Uparrow_{\theta_1} (\widehat{var}(n)))))\\ \Uparrow_o e & = & e & \Uparrow_{\theta_1\to\theta_2} e & = & \boldsymbol{\lambda} x^{[\![\theta_1]\!]}.\ \Uparrow_{\theta_2} (\widehat{app}(e)(\Downarrow_{\theta_2} (x)))\end{array}$$

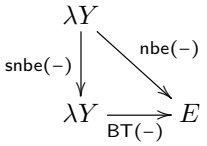
For closed terms $\vdash M : \theta$, the normalization-by-evaluation function can now be defined by setting $\mathsf{nbe}(M) = \Downarrow_\theta ([\![M]\!])(\overline{0})$. In order to apply the same method to open terms $\Gamma \vdash M : \theta$, where $\Gamma = \{x_1 : \theta_1, \ldots, x_k : \theta_k\}$, we need to build a canonical inhabitant of $[\![\Gamma]\!]$ by defining the *canonical* semantic environment $\rho_\Gamma : \Pi_{x_i \in \Gamma} [\![\theta_i]\!]$, associating $\Uparrow_{\theta_i} (\boldsymbol{\lambda}\_^N.var\ (\overline{i-1})) \in [\![\theta_i]\!]$ to any x_i. This leads to a generalized variant of the normalization function, defined by: $\mathsf{nbe}(M) = \Downarrow_\theta ([\![M]\!]_{\rho_\Gamma})(\overline{k})$.

▶ Proposition 13. For any $\lambda\bot$-term $\Gamma \vdash M$ in β-normal η-long form, $\mathsf{nbe}(M) = \mathsf{rep}(M)$. For any λY-term $\Gamma \vdash M : \theta$, $\mathsf{nbe}(M) = \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$.

Proof. The first part is proved by induction on M. For the second part, we reason equationally as follows: $\mathsf{nbe}(M) \overset{(1)}{=} \Downarrow_\theta ([\![M]\!]_{\rho_\Gamma})(\overline{k}) \overset{(2)}{=} \Downarrow_\theta (\bigsqcup_n [\![M \upharpoonright n]\!]_{\rho_\Gamma})(\overline{k}) \overset{(3)}{=} \bigsqcup_n \Downarrow_\theta ([\![M \upharpoonright n]\!]_{\rho_\Gamma})(\overline{k}) \overset{(4)}{=} \bigsqcup_n \Downarrow_\theta ([\![\mathsf{BT}(M \upharpoonright n)]\!]_{\rho_\Gamma})(\overline{k}) \overset{(5)}{=} \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$ where (1) is by definition of NBE, (2) is by induction on M from the definition of interpretation, (3) is by continuity of $\Downarrow_\theta$ and application. Since $M \upharpoonright n$ is a simply-typed $\lambda\bot$-term, it has a normal form $\mathsf{BT}(M \upharpoonright n)$ such that $M \upharpoonright n \simeq_{\beta\eta} \mathsf{BT}(M \upharpoonright n)$. By soundness of the domain semantics, this implies that $[\![M \upharpoonright n]\!]_\rho = [\![\mathsf{BT}(M \upharpoonright n)]\!]_\rho$ and (4) is true. Finally, (5) follows from the first part. ◀

3.2 Internalization

Following Section 2, for any λY-term $\Gamma \vdash M : \theta$, we define the syntactic NBE by $\mathsf{snbe}(M) = \downarrow_\theta M^*[\uparrow_{\theta_i} \lambda.(var\ \overline{i-1})/x_i]\ \overline{k}$ so that $\Gamma_{\mathsf{rep}} \vdash \mathsf{snbe}(M) : o$ ($\downarrow_\theta$ and $\uparrow_\theta$ are the terms defined in Section 2). In the remainder of this section we show the correctness of snbe, namely $\mathsf{BT}(\mathsf{snbe}(M)) = \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$, by noting that snbe amounts to an internalization of the NBE procedure inside the λY-calculus, i.e., that the following triangle commutes.



To carry out the argument, below we introduce another interpretation of terms $\Gamma_{\mathsf{rep}}, \Gamma \vdash \theta$, written $\langle\!-\!\rangle$, which will treat variables from Γ_{rep} as constants. Accordingly, the sequents will be interpreted by a continuous function from $\langle\Gamma\rangle$ to $\langle\theta\rangle$, with Γ_{rep} omitted. Similarly, the semantic environment for a context $\Gamma_{\mathsf{rep}}, x_1 : \theta_1, \ldots, x_k : \theta_k$ is simply $\rho : \Pi_{x_i \in \Gamma} \langle\theta_i\rangle$. In the same vein, whenever we write $\Gamma \vdash M : \theta$ from now on, we will assume that none of the variables from Γ_{rep} occurs in Γ (this does not affect the generality of our results, as free variables of a λY-term can be renamed to avoid clashes with Γ_{rep}). Below we let z range over the variables from Γ_{rep}.

$$\begin{array}{rclcrcl}
\langle o\rangle & = & E & & \langle x\rangle_\rho & = & \rho(x) & & \langle var\rangle_\rho & = & \boldsymbol{\lambda} n^E.var\ n \\
\langle \theta\to\theta'\rangle & = & \langle\theta\rangle\to\langle\theta'\rangle & & \langle M\ N\rangle_\rho & = & \langle M\rangle_\rho(\langle N\rangle_\rho) & & \langle app\rangle_\rho & = & \boldsymbol{\lambda} e_1^E.\boldsymbol{\lambda} e_2^E.app\ e_1\ e_2 \\
 & & & & \langle \lambda x^\theta.M\rangle_\rho & = & \boldsymbol{\lambda} a^{\langle\theta\rangle}.\langle M\rangle_{\rho\oplus\{x\mapsto a\}} & & \langle lam\rangle_\rho & = & \boldsymbol{\lambda} n^E.\boldsymbol{\lambda} e^E.lam\ n\ e \\
\langle z\rangle_\rho & = & z & & \langle Y_\theta\rangle_\rho & = & \bigsqcup_n \boldsymbol{\lambda} f^{\langle\theta\rangle\to\langle\theta\rangle}.f^n(\bot) & & \langle succ\rangle_\rho & = & \boldsymbol{\lambda} n^E.succ\ n
\end{array}$$

The interpretations $[\![-]\!]$ and $\langle - \rangle$ are related by the following lemma, which is easily proved by induction. We apply $\langle - \rangle$ to $M^\star$ on the understanding that $\Gamma \vdash M : \theta$ implies $\Gamma_{\mathsf{rep}}, \Gamma^\star \vdash M^\star : \theta^\star$ and that $[\![\Gamma]\!] = \langle \Gamma_{\mathsf{rep}}, \Gamma^\star\rangle$.

▶ **Lemma 14.** *For any type θ, $[\![\theta]\!] = \langle\theta^\star\rangle$. For any λY-term $\Gamma \vdash M : \theta$ and any semantic environment $\rho \in [\![\Gamma]\!]$, we have $[\![M]\!]_\rho = \langle M^\star\rangle_\rho$.*

On the other hand, $\langle - \rangle$ is related to Böhm trees by the following lemma.

▶ **Lemma 15.** *For any λY-term $\Gamma_{\mathsf{rep}} \vdash M : o$, we have $\langle M\rangle = \mathsf{BT}(M)$.*

Proof. Suppose first that $\Gamma_{\mathsf{rep}} \vdash M : o$ is a $\lambda\bot$-term in β-normal η-long form, i.e. $M = \mathsf{BT}(M)$. Then M must be an applicative term built from $\bot, z, succ, var, lam$ and app. By induction on M it follows that $\langle M\rangle = M$ and, thus, $\langle M\rangle = \mathsf{BT}(M)$.

Now consider arbitrary $\Gamma_{\mathsf{rep}} \vdash M : o$, and $n \in \mathbb{N}$. By construction, $M \upharpoonright n$ is a $\lambda\bot$-term. By normalization of the simply-typed λ-calculus, $M \upharpoonright n$ has a Böhm tree $\mathsf{BT}(M \upharpoonright n)$ such that $M \upharpoonright n \simeq_{\beta\eta} \mathsf{BT}(M \upharpoonright n)$. By soundness of the denotational model, it follows that $\langle M \upharpoonright n\rangle = \langle \mathsf{BT}(M \upharpoonright n)\rangle$. By the first part above, we can deduce $\langle M \upharpoonright n\rangle = \mathsf{BT}(M \upharpoonright n)$. The lemma then follows by continuity. ◀

Putting all the lemmas together, we arrive at our main result.

▶ **Theorem 16.** *For any λY-term $\Gamma \vdash M : \theta$, $\mathsf{BT}(\mathsf{snbe}(M)) = \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$.*

Proof. From the following calculations. $\mathsf{BT}(\mathsf{snbe}(M)) \overset{(1)}{=} \langle \mathsf{snbe}(M)\rangle \overset{(2)}{=} \langle \downarrow_\theta\ M^*[\ \uparrow_{\theta_i} \lambda.(var\ \overline{i-1})/x_i]\ \overline{k}\rangle \overset{(3)}{=} \langle\downarrow_\theta\rangle(\langle M^*\rangle_{x_i\mapsto\langle\uparrow_{\theta_i}\lambda.(var\ \overline{i-1})\rangle})(\langle\overline{k}\rangle) \overset{(4)}{=} \Downarrow_\theta\ ([\![M]\!]_{x_i\mapsto\Uparrow_{\theta_i}(\boldsymbol{\lambda}.(var\ \overline{i-1}))})(\overline{k}) \overset{(5)}{=} \mathsf{nbe}(M) \overset{(6)}{=} \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$. (1) follows from Lemma 15, (2) and (3) from the definitions of snbe and $\langle - \rangle$ respectively, (4) is a consequence of $\langle\downarrow_\theta\rangle = \Downarrow_\theta$, $\langle\uparrow_\theta\rangle = \Uparrow_\theta$ and Lemma 14. Finally, (5) follows from the definition of nbe and (6) from Proposition 13. ◀

We conclude this section with a discussion on the effect of the $M \mapsto \mathsf{snbe}(M)$ transformation on the order of terms (the *order* of a λY-term, written $ord(M)$, is the maximal order of the types of its subterms). We note that, given $\Gamma \vdash M : \theta$, the term $\mathsf{snbe}(M)$ is of order at most $ord(M) + 2$, because it contains $\downarrow_\theta$, which has type $\theta^* \to (o \to o)$ of order $ord(\theta) + 2$. However, as in Example 12, $\mathsf{snbe}(M)$ can be simplified by one size-decreasing β-reduction step to M' of order $ord(M) + 1$. Furthermore, if one is interested in obtaining an equivalent HORS then M' can be transformed into a HORS of order $ord(M)$ [18].

4 Generalization to PCF

Here we show that the methodology of the previous section can be applied to PCF [17]. This brings us closer to realistic programs, which can branch on data values. PCF extends the λY-calculus in that the ground type o is replaced with the type of natural numbers, basic arithmetic and zero testing. It was designed to be Turing-complete, so in order to avoid obvious undecidability we focus on its finitary variants. In what follows we consider boolean PCF_2 in which o is the type of booleans, but it should be clear that the techniques extend to any finite ground type.

4.1 PCF_2 and PCF_2 Böhm trees

PCF_2 types are defined by the grammar $\theta ::= \mathtt{B} \mid \theta \to \theta$ and terms are given by $M, N ::= tt \mid ff \mid \text{if } M \text{ then } N_1 \text{ else } N_2 \mid x \mid \lambda x^\theta.M \mid M\ N \mid Y_\theta\ M$. The following typing rules apply.

$$\frac{}{\Gamma \vdash tt : \mathtt{B}} \qquad \frac{}{\Gamma \vdash ff : \mathtt{B}} \qquad \frac{\Gamma \vdash M : \mathtt{B} \qquad \Gamma \vdash N_1 : \theta \qquad \Gamma \vdash N_2 : \theta}{\Gamma \vdash \text{if } M \text{ then } N_1 \text{ else } N_2 : \theta}$$

For PCF_2, the canonical forms that we wish to generate are PCF Böhm trees: they are (potentially infinite) trees generated by the rules displayed below.

$$\frac{}{\Gamma \vdash \bot, tt, ff : \mathtt{B}} \qquad \frac{\Gamma, \overrightarrow{\ldots, x_i : A_i, \ldots} \vdash M : \mathtt{B}}{\Gamma \vdash \lambda\overrightarrow{x}.M : \overrightarrow{A} \to \mathtt{B}} \qquad \frac{\overrightarrow{\Gamma \vdash M_i : \theta_i} \quad \Gamma \vdash N_1, N_2 : \mathtt{B} \quad (x : \overrightarrow{\theta} \to \mathtt{B}) \in \Gamma}{\Gamma \vdash \text{if } x\ \overrightarrow{M} \text{ then } N_1 \text{ else } N_2 : \mathtt{B}}$$

Such trees were first introduced along with the game semantics for PCF [11, 2], serving as a syntactic counterpart to *strategies*. Later developments in game semantics [3, 13] showed that two terms of PCF have the same strategy, i.e. the same PCF Böhm tree, if and only if they are indistinguishable by evaluation contexts having access to computational effects such as state and control operators. This makes PCF Böhm trees a natural candidate for a representation of higher-order programs to be used for verification purposes, since purely functional programs are eventually executed in runtime environments that have access to such effects.

We also consider the variant $\mathrm{PCF}_{2,\bot}$ consisting of Y-free terms of PCF_2, extended with a constant $\bot : \mathtt{B}$. They are partially ordered by the obvious adaptation of the partial order $\sqsubseteq$ on $\lambda\bot$-terms, still written $\sqsubseteq$. As before, this partial order extends to an ω-cpo on infinite terms of $\mathrm{PCF}_{2,\bot}$, referred to as $\mathrm{PCF}^\infty_{2,\bot}$.

The operational semantics for PCF_2 for which PCF Böhm trees can be taken to be canonical representatives is obtained by adding the rules listed below to β, η and Y. For brevity, if M then N_1 else N_2 is written as $\mathrm{if}(M, N_1, N_2)$. The rules are restricted to the cases where both sides typecheck.

$$\mathrm{if}(tt, N_1, N_2) \to_{\beta'_1} N_1 \qquad M \to_{\eta'} \mathrm{if}(M, tt, ff) \qquad \mathrm{if}(M, N_1, N_2)\ N' \to_{\gamma_1} \mathrm{if}(M, N_1\ N', N_2\ N')$$
$$M \to_{\eta'} \mathrm{if}(M, tt, ff) \qquad \mathrm{if}(\mathrm{if}(M, N_1, N_2), N_3, N_4) \to_{\gamma_2} \mathrm{if}(M, \mathrm{if}(N_1, N_3, N_4), \mathrm{if}(N_2, N_3, N_4))$$

We write $\simeq_{\mathrm{PCF}}$ for the corresponding equational theory. For $\mathrm{PCF}_{2,\bot}$ we also add the rule $\mathrm{if}(\bot, N_1, N_2) \to_\bot \bot$. The following property can then be established by standard means.

▶ Proposition 17. For any term $\Gamma \vdash M : \theta$ of $\mathrm{PCF}_{2,\bot}$ there is a PCF Böhm tree $\Gamma \vdash \mathsf{BT}(M) : \theta$ such that $M \simeq_{\mathrm{PCF}} \mathsf{BT}(M)$.

To define Böhm trees of arbitrary PCF_2-terms $\Gamma \vdash M : \theta$, we define the nth approximant of M, written $M \upharpoonright n$, by replacing fixpoint combinators Y_θ with $\lambda f^\theta.f^n(\bot_\theta)$, yielding a term of $\mathrm{PCF}_{2,\bot}$. In the general case, the PCF Böhm tree is then defined by $\mathsf{BT}(M) = \bigsqcup_n \mathsf{BT}(M \upharpoonright n)$.

4.2 Representability

We would like to represent PCF Böhm trees of PCF_2-terms as level-2 ground λY-terms. To that end, we shall use a new context Γ_{pcf}, defined as Γ_{rep} augmented with $\{\, tt : o, ff : o, \mathit{if} : o \to o \to o \to o \,\}$ to take the shape of PCF Böhm trees into account. Just as before, the set of ground $\lambda\bot^\infty$-terms typed in context Γ_{pcf} forms a pointed ω-cpo. It will be the target of our transformation and, again, we will denote it by E. The representation function $\mathsf{rep}(-)$ of Section 2 extends easily to a function mapping any $\mathrm{PCF}_{2,\bot}$-term $\Gamma \vdash M : \theta$ to a $\lambda\bot$-term $\Gamma_{\mathsf{pcf}} \vdash \mathsf{rep}(M) : o$ using De Bruijn levels. Our representation result for PCF_2 can then be stated as follows.

▶ **Theorem 18.** *For any* PCF_2 *term* $\Gamma \vdash M : \theta$*, there is a level-2* λY*-term* $\Gamma_{\mathsf{pcf}} \vdash M_{rep} : o$ *such that* $\mathsf{BT}(M_{rep}) = \bigsqcup_n \mathsf{rep}(\mathsf{BT}(M \upharpoonright n))$*.*

To construct M_{rep} from M, we first apply the following syntactic transformation translating types and terms of PCF_2 to λY-terms, combining the operation $(-)^*$ on λY-terms defined in Section 2 with an elimination of booleans based on Church encodings.

$$\begin{array}{rcl@{\qquad}rcl} \mathtt{B}^* & = & (o \to o) \to (o \to o) \to (o \to o) & tt^* & = & \lambda a^{o\to o}.\lambda b^{o\to o}.a \\ (\theta' \to \theta'')^* & = & (\theta')^* \to (\theta'')^* & ff^* & = & \lambda a^{o\to o}.\lambda b^{o\to o}.b \end{array}$$

$$(\text{if } M \text{ then } N_1 \text{ else } N_2)^* = \lambda \overrightarrow{x}^{\overrightarrow{\theta^*}}.\lambda a^{o\to o}.\lambda b^{o\to o}.M^*\ (N_1^*\ \overrightarrow{x}\ a\ b)\ (N_2^*\ \overrightarrow{x}\ a\ b)$$

where, in the last equation, N_1 and N_2 have type $\overrightarrow{\theta} \to \mathtt{B}$. The translation $(-)^*$ can be extended to all the other term constructors in the obvious way. Note that, since this translation gives us a λY-term, we can already apply the transformation of Section 2 and obtain a λY-term generating the Böhm tree of M^*. To get the PCF *Böhm tree of* M instead one can modify the $\downarrow_\theta, \uparrow_\theta$ terms as follows.

$$\downarrow_{\mathtt{B}} = \lambda x^{\mathtt{B}^*}.x\ (\lambda.tt)\ (\lambda.ff) \qquad \uparrow_{\mathtt{B}} = \lambda e^{o\to o}.\lambda a^{o\to o}.\lambda b^{o\to o}.\lambda n^o.if\ (e\ n)\ (a\ n)\ (b\ n)$$

$$\begin{array}{rcl} \downarrow_{\theta'\to\theta''} & = & \lambda x^{(\theta'\to\theta'')^*}.\ \lambda v^o.\ lam\ v\ (\downarrow_{\theta''} (\ x\ (\uparrow_{\theta'} \lambda.(var\ v))\ )\ (succ\, v)\ ) \\ \uparrow_{\theta'\to\theta''} & = & \lambda e^{o\to o}.\ \lambda a^{(\theta')^*}.\ \uparrow_{\theta''} (\ \lambda v^o.\ app\ (e\ v)\ (\downarrow_{\theta'} a\ v)\ ) \end{array}$$

Assuming $\Gamma = x_1 : \theta_1, \ldots, x_k : \theta_k$, we can then set $M_{rep} = \downarrow_\theta M^*[\uparrow_{\theta_i} \lambda.var\ \overline{i-1}/x_i]\ \overline{k}$.

The notion of *order* on types and terms is generalised to PCF_2 by setting $ord(\mathtt{B}) = 0$. As for λY, M_{rep} can be simplified by one β-reduction step to M' of order $ord(M) + 2$. M' can then be transformed into a HORS of order $ord(M) + 1$ [18].

5 Conclusion

We have shown that faithful representations of η-long Böhm trees of arbitrary λY-terms and PCF Böhm trees of PCF_2-terms can be generated by higher-order recursion schemes. We can use the result to reduce various decision problems related to the λY-calculus or PCF_2 to problems for higher-order recursive schemes.

The first class of problems to which the reduction technique can be applied are equivalence problems.

▶ **Corollary 19.** *The following problems are recursively equivalent: HORS equivalence, Böhm tree equivalence in the* λY*-calculus,* PCF *Böhm tree equivalence in* PCF_2*.*

The decidability status of these problems is currently unknown, but they are known to be related to two other interesting problems: the equivalence problem for deterministic collapsible pushdown automata [10] and contextual equivalence for PCF_2-terms with respect to contexts with state and control effects (contextual equivalence for PCF_2-terms with respect to purely functional contexts is undecidable, even without Y [14]).

In a different direction, we can exploit the decidability result for MSO on trees generated by HORS [16] to decide a variety of properties of Böhm trees. Here the undecidability result for binders places a limit on the kind of binding-related information that can be captured by MSO on our representations. However, many interesting problems on Böhm trees still fall within the range of our representation.

▶ **Corollary 20.** *The following problems are decidable for* λY*- and* PCF_2*-terms: normalizability (is a given term equivalent to a* Y*-free term?), finiteness (is the Böhm tree/*PCF

*Böhm tree finite, i.e. a $\lambda\bot$/$\mathrm{PCF}_{2,\bot}$-term?), solvability (is the Böhm tree/*PCF *Böhm tree non-empty after leading abstractions?), prefix (does the Böhm tree/*PCF *Böhm tree have a given finite prefix?).*

The results follow from the fact that all the relevant properties are expressible in MSO. Normalizability and solvability for the λY-calculus were already known to be decidable [19]. To the best of our knowledge, the other consequences are new. In future work, we would like to understand the exact complexity of these problems and see whether our results yield optimal bounds.

Acknowledgment. We would like to thank Sylvain Salvati for pointing out to us the undecidability argument for MSO on trees with binding. This research was conducted while the first author was in Cambridge, supported by the Advanced Grant ECSYM. The second author gratefully acknowledges the support of the EPSRC (EP/J019577/1).

References

1 A. Abel, T. Coquand, and P. Dybjer. Normalization by evaluation for Martin-Löf type theory with typed equality judgements. In *LICS*, pp 3–12, 2007.
2 S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Inf. and Comput.*, 163:409–470, 2000.
3 S. Abramsky and G. McCusker. Linearity, sharing and state. In *Algol-like languages*, pp 297–329, Birkhaüser, 1997.
4 T. Altenkirch, P. Dybjer, M. Hofmann, and P. J. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *LICS*, pp 303–312, 2001.
5 U. Berger and H. Schwichtenberg. An inverse of the evaluation functional for typed lambda-calculus. In *LICS*, pp 203–212, 1991.
6 P.-L. Curien. Abstract Böhm trees. *Math. Struct. in Comput. Sci.*, 8(6):559–591, 1998.
7 W. Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982.
8 N. G. De Bruijn. Lambda calculus notation with nameless dummies. In *Indagationes Mathematicae (Proceedings)*, volume 75, pages 381–392. Elsevier, 1972.
9 P. Dybjer and A. Filinski. Normalization and partial evaluation. In *APPSEM*, LNCS 2395, pp 137–192, 2000.
10 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In LICS, pp 452–461, 2008.
11 J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF. *Inf. and Comput.*, 163(2):285–408, 2000.
12 N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pp 416–428, 2009.
13 J. Laird. Full abstraction for functional languages with control. In *LICS*, pp 58–67, 1997.
14 R. Loader. Finitary PCF is not decidable. *Theor. Comput. Sci.*, 266(1-2):341–364, 2001.
15 R. Nakajima. Infinite normal forms for the λ-calculus. In *Proc. Symp. λ-calculus and Computer Science*, pages 62–82. Springer-Verlag, 1975.
16 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pp 81–90, 2006.
17 G. D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5:223–255, 1977.
18 S. Salvati and I. Walukiewicz. Recursive schemes, Krivine machines, and collapsible pushdown automata. In *RP*, LNCS 7550, pp 6–20, 2012.
19 R. Statman. On the lambda Y-calculus. *Ann. Pure Appl. Logic*, 130(1-3):325–337, 2004.
20 G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993. Foundations of Computing Series.

Evaluation is MSOL-compatible*

Sylvain Salvati[1] and Igor Walukiewicz[2]

1 **LaBRI, INRIA / Université Bordeaux, France**
2 **LaBRI, CNRS / Université Bordeaux, France**

Abstract

We consider simply-typed lambda calculus with fixpoint operators. Evaluation of a term gives as a result the Böhm tree of the term. We show that evaluation is compatible with monadic second-order logic (MSOL). This means that for a fixed finite vocabulary of terms, the MSOL properties of Böhm trees of terms are effectively MSOL properties of terms themselves. Theorems of this kind have been known for some graph operations: unfolding, and Muchnik iteration. Similarly to those results, our main theorem has diverse applications. It can be used to show decidability results, to construct classes of graphs with decidable MSOL theory, or to obtain MSOL formulas expressing behavioral properties of terms. Another application is decidability of a control-flow synthesis problem.

1998 ACM Subject Classification F.3 Logics and Meanings of Programs

Keywords and phrases Simply typed λY-calculus, monadic second order logic, Transfer Theorem, infinitary systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.103

1 Introduction

Rice's theorem tells us that no non-trivial property of the behavior of a Turing machine can be decided by looking at the machine itself. In this paper we consider a much simpler abstract computing system: simply-typed lambda-calculus with fixpoint operators. We denote it λY. A behavior of a λY-term is its Böhm tree. Since not all λY-terms have normal forms, Böhm trees are a standard choice for the result of a computation of a term. To express properties of results we use monadic second-order logic (MSOL) because it is a fundamental logic over trees. The *Transfer Theorem* we prove says that if we fix a finite set of term variables then every MSOL property of Böhm trees is effectively an MSOL property of terms. In other words, we show that MSOL is compatible with evaluation.

The λY-calculus is a standard formalism in the functional language community. It can be seen as a simplification of the programming language PCF introduced by Plotkin [37]. By now, it is usual [3] to approach the semantics of the lambda-calculus through infinite Böhm trees. Such a tree is just the normal form of the term, if the term has one. Otherwise it is a potentially infinite tree representing the visible part of the infinite computation of the term. We assume that all constants in the vocabulary have a type of order at most 2. This assumption is made in all the works considering language theoretic properties of Böhm trees, as it guarantees that a Böhm tree is just a labeled ranked tree. In this paper we consider also infinite λY-terms. This is less standard but introduces relatively few complications while substantially strengthening of the main theorem.

* Supported by ANR 2010 BLAN 0202 01 FREC

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 103–114

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

Our Transfer Theorem says that for a fixed finite vocabulary of terms, an MSOL formula φ can be effectively transformed into an MSOL formula $\widehat{\varphi}$ such that for every term M over the fixed vocabulary: M satisfies $\widehat{\varphi}$ iff the Böhm tree of M satisfies φ. In terminology of Courcelle [12], our result says that evaluation of λY-terms is MSOL-compatible. A flagship example of a result of this kind is MSOL-compatibility of the unfolding operation [43, 14]. A finite automaton, i.e. a graph, can be unfolded into a tree of all its possible executions: this tree can be seen as the evaluation of the automaton. The MSOL-compatibility of the unfolding means that the MSOL theory of this tree can be effectively reduced to the MSOL theory of the automaton itself. The more powerful Muchnik iteration [43, 45] allows to get a similar result for pushdown automata and higher-order pushdown automata. This way we obtain the pushdown hierarchy of trees with a decidable MSOL theory. Here we consider λY-terms instead of automata as a computational model, and show the analogous result for evaluation instead of unfolding or Muchnik iteration. The operation of evaluation cannot be directly compared to the other two since it works on different objects (trees with back edges instead of graphs). Yet in a well-known context where these operations can be compared, the evaluation operation is strictly stronger. Indeed, every tree in the pushdown hierarchy [11] is a Böhm tree of some term, but not vice versa [36].

Related work: Under a different syntax λY-calculus has been intensively studied by the language theoretic community. One can cite the PhD thesis of Fischer [18] on macro languages, the work of Engelfriet and Schmidt on IO and OI [16, 17], or the work of Damm on (safe) recursive schemes [15]. More recently Knapik, Niwinski and Urzyczyn [21] considered recursive schemes as generators of infinite trees, and studied the model-checking problem for such trees. After a series of intermediate results [2, 22, 1]; Ong [31] has shown that the model-checking problem of MSOL properties for such trees is decidable. It has been already clear to Engelfriet and Schmidt as well as to Damm that the grammars, or recursive schemes they study are a different representation of λY-terms (and their subclasses). Indeed, trees generated by recursive schemes are just Böhm trees of the corresponding terms of λY-calculus.

The Transfer Theorem for evaluation presented here is stronger than Ong's theorem in at least two aspects. First, it holds also for (possibly irregular) infinite λY-terms. Second, and more importantly, the theorem gives an effective way of expressing properties of results of evaluation (Böhm trees) in terms of properties of terms: the construction of the formula $\widehat{\varphi}$ depends on φ and a fixed vocabulary, but does not depend on a term M. For example, since finiteness of a tree is definable in MSOL, we immediately obtain that if we restrict to λY-terms over a fixed finite vocabulary then the set of terms having a (finite) normal form is MSOL definable. We show that removing the restriction to a fixed finite vocabulary would imply the collapse of the polynomial hierarchy. In the last section of the paper we give several other applications of additional power provided by the Transfer Theorem.

This work sheds some light on unfolding and Muchnik's theorems. It shows that they are in some sense special cases of the Transfer Theorem for evaluation. Knapik and Courcelle [13] have used the unfolding theorem to prove a special case of our theorem for infinite terms with variables only of type 0. They also mention that the method can be extended to all safe terms thus avoiding the use of Muchnik's theorem. Due to a result of Parys [36] we know that the approach based on unfoldings or Muchnik's theorem cannot work for all (unsafe) terms. Our proof proposes a different approach to substitution offered by the Krivine machine and that overcomes this difficulty (cf. Section 4).

MSOL properties of higher-order systems is an active area of research. After the seminal paper of Knapik, Niwinski and Urzyczyn[21]; Caucal has introduced the pushdown hier-

archy [11] that since has been an object of intensive study from the logical point of view [10]. Many interesting properties of higher-order programs can be analyzed with recursive schemes and automata [24, 23, 25, 27, 33]. The decidability result of Ong has been revisited in a number of ways [19, 26, 39, 8].

The study of MSOL-compatible operations has a long history. Directly relevant here is an iteration operator proposed by Stupp; he showed that it is MSOL-compatible [44]. Rabin's theorem [38] on decidability of MSOL on infinite binary trees is an immediate corollary of this result. Much later Muchnik has proposed a strengthening of Stupp's iteration, and sketched the proof of MSOL-compatibility of this operation [43]. Courcelle and Walukiewicz have shown MSOL-compatibility of the unfolding operation [14] not using Muchnik iteration. Later Walukiewicz has given a detailed proof of MSOL-compatibility of Muchnik iteration [45]. This result has then been extended and adapted to other logics [6, 30, 29]. For a survey of compatible operations and their applications we refer the reader to [5].

Plan of the paper: In the next section we introduce infinitary λY-calculus: a simply typed λ-calculus of infinite terms with fixpoint operators. We define Böhm trees and show that, as for finite terms, they still define a standard notion of values.

Section 3 presents the main theorem. For this it describes how terms are represented as logical structures so that we can talk about their MSOL properties. Our representation requires that we have a fixed finite set of λ-variables. At the same time we do not need to restrict the number of Y-variables. We show that the theorem is not likely to hold if the number of λ-variables is not fixed. We also compare our theorem to that of Ong [31]. An overview of proof methods is presented in Section 4.

Section 5 gives three applications of the Transfer Theorem. In the conclusion section we give more relations between the Transfer Theorem and other results in the literature. The extended version of the paper [40], presents all the proofs as well as some more comments and discussions.

2 Infinitary λY-calculus

In this paper, we work with the infinitary simply typed λ-calculus with fixpoints. We start with the notions of tree signatures, and of infinitary terms together with the operational semantics of the infinitary λY-calculus. We adopt a slightly modified syntax where Y, the fixpoint operator, is used as a variable binder rather than as a combinator. This allows us to distinguish between the variables that may be bound by Y from those that may be bound by λ. Such a distinction is of little importance for the calculus itself, but it will allow us to get a more general theorem later when we put restrictions on the number of λ-variables. Next, we define the notion of *Böhm trees*. As it can be expected, Böhm trees are actual normal forms of infinitary terms, and every infinitary term has a normal form. Moreover, a Böhm tree of a term of type 0 over a tree signature is just a ranked tree.

Syntax and operational semantics. The *set of types* is constructed from a unique *basic type* 0 using a binary operation $\to$. Thus 0 is a type and if α, β are types, so is $(\alpha \to \beta)$. As usual, so as to use fewer parentheses, we consider that $\to$ associates to the right. For example, $0 \to 0 \to 0$ stands for $(0 \to (0 \to 0))$. We will write $0^i \to 0$ as short notation for $0 \to 0 \to \cdots \to 0 \to 0$, where there are $i+1$ occurrences of 0. The order of a type is defined by: $order(0) = 1$, and $order(\alpha \to \beta) = max(1 + order(\alpha), order(\beta))$.

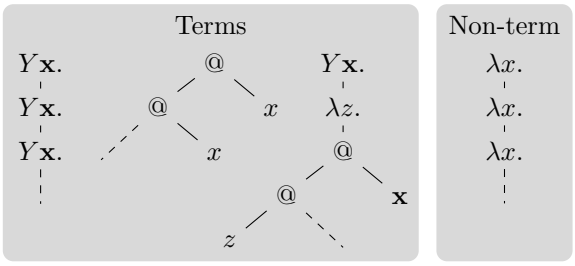
A *signature*, denoted Σ, is a set of typed constants (in general noted $c^\alpha, \dots$), that is symbols with associated types. Of special interest to us will be *tree signatures* where all constants other than the special constant Ω have order at most 2. Observe that types of

order 2 have the form $0^i \to 0$ for some i. For simplicity of notation we will always assume that $i = 2$, but our results do not depend on this convention.

Terms will be built over two disjoint countable sets of typed variables: λ-variables and Y-variables: written x^α and $\mathbf{x}^\alpha$ respectively. We assume that for every type α we have a constant Ω^α to denote the undefined term of type α. We will also have typed application symbols $@^\alpha$, and typed binders Y^α as well as $\lambda^{\alpha\to\beta}$. For all types α we define simultaneously the sets of infinite terms of type α as trees satisfying the following conditions.

- A node labeled by Ω^α, x^α, $\mathbf{x}^\alpha$, or c^α is a term of type α.
- A tree with the root labeled $@^\beta$ having as the left subtree a term of type $\alpha \to \beta$ and as a right subtree a term of type α, is a term of type β.
- A tree with the root labeled $\lambda^{\alpha\to\beta}x^\alpha$ with the unique immediate subtree being a term of type β, is a term of type $\alpha \to \beta$.
- A tree with a root labeled $Y^\alpha\mathbf{x}^\alpha$ with the unique immediate subtree being a term of type α, is a term of type α.

Some examples of infinitary λY-terms, as well as trees that are not terms, are presented in the Figure to the right. Notice that all variables and constructors carry type labelings that make typings of terms unique. We shall often omit those labels when they are unnecessary for the understanding or when they can be inferred from the context. We will also use standard conventions and write (MN) for $@\,M\,N$, and $N_0N_1\dots N_p$ for $(\dots(N_0N_1)\dots N_p)$.



The reason why we need to distinguish between λ-variables and Y-variables is that, the main theorem we prove is about terms which use finitely many λ-variables but possibly infinitely many Y-variables. As a small remark, if the main theorem did not need to make this assumption, we could simply get rid of Y-binders. Indeed the term $Y\mathbf{x}.N$ has the same Böhm tree (see section 2) as the term $rec(\lambda x.N[x/\mathbf{x}])$ where $rec = \lambda f.f(f(f(f\dots)))$. This shows that λ-abstraction, or parameter instantiation, is more powerful than recursive definition in the context of the infinitary λ-calculus.

The notion of capture-avoiding substitution can be easily extended to infinitary λ-calculus. Given two terms M and N we shall write $M[N/x]$ (*resp.* $M[N/\mathbf{x}]$) for the result of the capture-avoiding substitution of N for the free occurrences of x (*resp.* $\mathbf{x}$) in M. It is important to notice that x or $\mathbf{x}$ may have infinitely many free occurrences in M, and that computing the result of $M[N/x]$ or of $M[N/\mathbf{x}]$ may require an infinite amount of work. This is why, following a suggestion made in Terese [20], our technique is based on the Krivine machine which computes with explicit substitutions and avoids this infinite overhead.

We may now define β-contraction on infinitary terms as the direct extension of β-contraction on finite terms. Concerning δ-contraction, we need to adapt its definition to our slightly modified syntax. The relation of δ-contraction $\to_\delta$ is the smallest relation that is compatible with the syntax of infinitary λY-calculus and so that $Y\mathbf{x}.M$ δ-contracts to $M[Y\mathbf{x}.M/\mathbf{x}]$. We let $\beta\delta$-contraction, $\to_{\beta\delta}$ to be the union of the relations $\to_\beta$ and $\to_\delta$. Of course, the subject reduction property for simple types transfers from finite terms to infinite ones so that the reduction preserves typing.

We recall the notion of weak head normal form and of weak head reduction: the reduction strategy of the Krivine machine. An infinitary term M is in *weak head normal* form, if it is

of the form $\lambda x^{\alpha}.N$ for some term N, or if it is of the form $hN_1 \dots N_n$, with h being either a variable or a constant different from Ω.

When M is an infinitary term of the form $(\lambda x.\ P)P_1 \dots P_n$ or $(Y\mathbf{x}.\ P)P_1 \dots P_n$, then M has a *head-redex*. The operation of reducing M to $P[P_1/x]P_2 \dots P_n$ or to $P[Y\mathbf{x}.P/\mathbf{x}]P_1 \dots P_n$, respectively, is called *head-contracting* M. We write $M \to_h N$ when M head-contracts to N; we write $M \to_h^* N$ for head-reduction in some finite number of head-contraction steps.

Böhm trees. We now define a notion of *normal form* for infinitary terms. There is no particular difficulty to do it for all infinite λY-terms. Yet, since we consider infinitary λY-terms as generators of infinite trees, we are really interested only in closed terms of type 0. In order to further simplify the notation we will from the start consider only terms over a tree signature, and assume that all the constants are binary.

▶ **Definition 1** (Böhm trees). Given a closed term M of type 0 over a tree signature, we define its Böhm tree, $BT(M)$, as follows:

1. if $M \to_h^* bN_1N_2$ where b is a constant different from Ω, then $BT(M)$ is a tree with root labeled b and with $BT(N_1)$ and $BT(N_2)$ as its subtrees.
2. otherwise $BT(M) = \Omega^{\alpha}$.

Observe that in our case Böhm trees are just labeled infinite binary trees. In a sense that can be made precise, Böhm trees are normal forms of terms. As such they are terms too, but due to their special shape we do not need to use application nodes to represent them.

The reader may be surprised that we talk about Böhm trees while in general weak head reduction computes Lévy-Longo trees. It turns out that the two notions coincide when working with tree signatures and terms of type 0. This is why we have preferred to use the better known notion.

3 Transfer Theorem for evaluation

In this section we present the main theorem of the paper. The theorem relates logical theories of terms and Böhm trees. We will consider monadic-second order logic (MSOL) on such objects. For this, it will be essential to restrict to some finite set of λ-variables: both free and bound. On the other hand, we will be able to handle infinitely many Y-variables. Once we make it clear how to represent terms and Böhm trees as logical structures, we will also state our main theorem. We will justify our representation of terms by showing two facts: (i) the set of all terms is definable in MSOL, (ii) unless the polynomial-time hierarchy collapses, the main theorem is false when we do not fix the number of bound λ-variables.

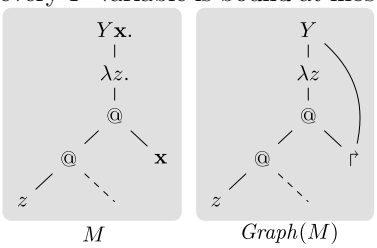
Terms will be represented as labeled, potentially infinite, graphs. For us here a labeled graph is a structure of the form $\mathcal{M} = \langle V, E_1, E_2, \{P_i\}_{i=1,\dots,n}\rangle$, where V is the set of vertices, E_1 and E_2 are binary relations on vertices, and every P_i is a subset of vertices. We will have two edge relations representing left and right successors. In our case predicates P_i will form a partition of V. So every vertex will have a unique label given by the predicate it belongs too.

Monadic second-order logic (MSOL) is an extension of first-order logic with quantification over sets of elements and the membership predicate $x \in Z$, where x is a variable ranging over elements, and Z is a variable ranging over sets of elements. The definition of satisfiability of a formula φ in a structure $\mathcal{M}$, denoted $\mathcal{M} \vDash \varphi$, is standard.

Let us fix a tree signature Σ with finitely many constants other than Ω. As postulated at the beginning of Section 2, for simplicity of the notation all the constants are binary. We would like to consider terms as models of formulas of monadic second-order logic. We will work with terms over some arbitrary but finite vocabulary. We take a finite set of typed λ-variables $\mathcal{X} = \{x_1^{\alpha_1}, \dots, x_k^{\alpha_k}\}$, and a finite set of types $\mathcal{T}$. We denote by $Terms(\Sigma, \mathcal{T}, \mathcal{X})$

the set of *infinite closed concrete terms* M over the signature Σ such that M uses only λ-variables from $\mathcal{X}$, and every subterm of M has a type in $\mathcal{T}$. We call *concrete terms*, terms that are not considered up to α-conversion, so it makes sense to say that bound λ-variables in M should come from $\mathcal{X}$. Observe that we do not put restrictions on the use Y-variables. It will be convenient to assume that every Y-variable in M is bound at most once: for every Y-variable $\mathbf{x}$ there is at most one occurrence of $Y\mathbf{x}$ in M.

A term from $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ is a labeled tree where the labels come from a finite alphabet, but for the Y-variables and Y-binders. We will now eliminate the possible source of infiniteness of labels related to Y-variables and Y-binders. Take a closed term M considered as a tree. For every node of this tree labeled by a Y-variable $\mathbf{x}^\alpha$ we put an E_1-edge from the node to the node labeled $Y^\alpha\mathbf{x}^\alpha$. Since M is closed, such a node exists and is an ancestor of the node labeled by $\mathbf{x}$; such a node is also unique since we assume that every Y-variable is bound at most once. In the next step we introduce a new symbol $\Rsh^\alpha$ and for every node labeled with a Y-variable of type α, we change its label to $\Rsh^\alpha$. Finally, we replace all labels of the form $Y^\alpha\mathbf{x}^\alpha$ by just Y^α. This way we have eliminated all occurrences of Y-variables from labels, but now a term is represented not as a labeled tree but as a labeled graph. Let us denote it by $\mathit{Graph}(M)$. Observe that the nodes of this graph have labels from a finite set, call it $\mathit{Talph}(\Sigma, \mathcal{T}, \mathcal{X})$. There are two edge relations, E_1 and E_2, in $\mathit{Graph}(M)$ since nodes labeled by application symbol have both left and right successors. The nodes with other labels have either one successor, given by E_1, or no successor (nodes labeled with labels from $\Sigma \cup \mathcal{X}$).



Since $\mathit{Graph}(M)$ is a labeled graph over a finite alphabet, it makes sense to talk about satisfiability of an MSOL formula in this graph. We will just write $M \vDash \varphi$ instead of $\mathit{Graph}(M) \vDash \varphi$. The first, easy but important, observation is that for fixed Σ, $\mathcal{T}$, $\mathcal{X}$, there is an MSOL formula determining if a graph is of the form $\mathit{Graph}(M)$ for some $M \in \mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$. Indeed, in this case we deal with models over the signature consisting of two binary relations E_1, E_2 and a unary relation P_b for every $b \in \mathit{Talph}(\Sigma, \mathcal{T}, \mathcal{X})$. The formula should say that predicates P_b form the partition of the set of vertices and then express conditions from the definition of infinite λY-terms on page 106. These conditions are clearly expressible in MSOL as they talk about dependencies between labels of a node and its successors.

For a closed term $M \in \mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ of type 0, its Böhm tree is a tree with nodes labeled by symbols from Σ. Hence one can talk about satisfiability of MSOL formulas in $BT(M)$. The Transfer Theorem says that evaluation, that is the function assigning to a term its Böhm tree, is MSOL compatible.

▶ **Theorem 2** (Transfer Theorem). *Let Σ be a finite tree signature, $\mathcal{X}$ a finite set of typed variables, and $\mathcal{T}$ a finite set of types. For every MSOL formula φ one can effectively construct an MSOL formula $\widehat{\varphi}$ such that for every λY-term $M \in \mathit{Terms}(\Sigma, \mathcal{T}, X)$ of type 0:*

$$BT(M) \vDash \varphi \qquad \textit{iff} \qquad M \vDash \widehat{\varphi}.$$

Notice that the formula $\widehat{\varphi}$ is independent from M; if it were dependent on M then we would simply obtain Ong's theorem since we could take $\widehat{\varphi}$ to be either *true* or *false*. At first sight the proof presented in [32] resembles our Transfer Theorem: for a recursive scheme G it constructs a tree $\lambda(G)$ and a property automaton that works on trees of the form $\lambda(G)$. Looking closer (definition on page 11 of [32]) the alphabet of $\lambda(G)$ depends on the size of G, since terms are η-expanded, and all bound variables are supposed to be different. So

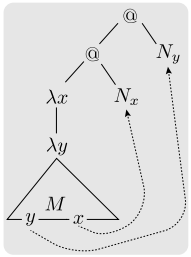
when fixing the alphabet of the property automaton, one fixes the number of rules in the recursive scheme (except maybe for rules for terminals of type 0). In our case the formula $\widehat{\varphi}$ is constructed for an infinite family of terms provided they use only the lambda-variables from $\mathcal{X}$. For completeness of this comparison we should add that Ong states that his approach can be straightforwardly adapted to give the proof of the Transfer Theorem [35]. Ong also observes that this adaptation would lower the complexity of his algorithm to match the one from [26]. In Section 5 we show that, a variant of global model-checking [9, 7] which was so far considered as a genuine extension of Ong's Theorem follows directly from the Transfer Theorem.

A natural question to ask is whether our restrictions on terms can be further weakened. Observe that the restriction on the fixed finite set of types is essential to be able to define in MSOL the set of representations of well typed terms. Concerning the restriction on the λ-variables, in principle it is possible to represent terms with an unbounded number of λ-variables by using the same trick for λ-binders as the one we have used for the Y-binders. However, we conjecture that the Transfer Theorem does not hold when we allow infinitely many λ-variables. Below we give a simple argument under the hypothesis that the polynomial hierarchy is strict.

Using a fixed set of types, it is possible to represent a Quantified Boolean Formula (QBF) θ by a λ-term M_θ whose Böhm tree is just a constant *true* iff θ is true. Moreover M_θ can be obtained in linear time from θ. Suppose that the Transfer Theorem would hold without restricting the number of λ-variables. There would then be a $\widehat{\varphi}$ so that for every QBF θ, $M_\theta \vDash \widehat{\varphi}$ iff θ is true. But verifying a fixed MSOL formula against some finite structure is in the polynomial hierarchy, while the validity of QBF formulas is PSPACE-complete. Thus, this more general formulation of the Transfer Theorem would imply the unlikely conclusion that the polynomial hierarchy collapses.

4 Overview of the proof

Our objective is to reduce the monadic theory of $BT(M)$ to the monadic theory of M, and moreover to do this in a smooth fashion depending only on the vocabulary of M. An instructive sub-case of terms of order ≤ 2 has been considered by Courcelle and Knapik [13]. They consider infinite closed terms of type 0 over a finite set of variables of type 0. Moreover, although it is somehow implicit in their formulation, they assume that every subterm has a type from some fixed finite set of types. In this case they are able to show the Transfer Theorem using the compatibility of MSOL with respect to the unfolding operation.



To explain the method proposed by Courcelle and Knapik let us look at the term $(\lambda x.\lambda y.M)N_xN_y$. This term is represented as a tree the same way as described in the previous section. In the figure to the right we have singled out occurrences of x and y in M. It turns out that it is possible in MSOL to define links between bound variables and the terms that would be substituted for these variables if β-reduction were performed (dashed arrows in the picture). For this the two assumptions on the number of variables and the number of types stated above are essential. Then, intuitively, the normal form of the term can be obtained by following the links and jumping over lambda-binders. So the normal form of the term can be defined in the unfolding of the term with links. Thanks to the compatibility of the unfolding, every MSOL formula over the unfolding can be translated into a formula over a term itself.

This method does not work when there are bound variables of type other than 0. The

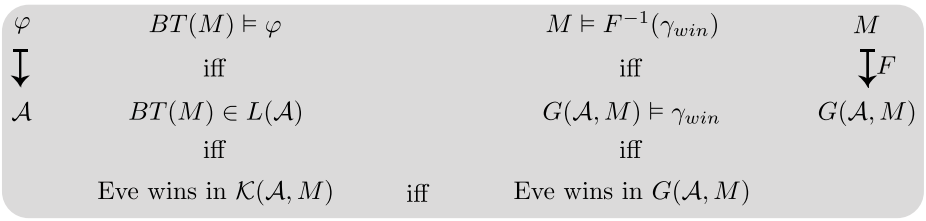


Figure 1 Schema of the main part of the proof.

reason is that when reducing a redex of a variable, say of a type $0 \to 0$, a new redex can be created. Moreover, in order to avoid variable capture, variable renaming may be needed. In consequence, after doing the operation described in the previous paragraph we can get a term with much more, or even infinitely many variables. This phenomenon would not happen if we started from an infinite safe term [21, 4] since no variable renaming would be needed when reducing such a term. So one can use the method above to reduce simultaneously all the redexes of the highest order in a safe term. This would give a β-equivalent safe term over the same set of variables but with the smaller maximal order of a redex. Hence one can repeat the argument until all the redexes are eliminated. In consequence, this gives yet another method to show the decidability of MSOL theory of Böhm trees generated by safe terms, that is all the trees in the pushdown hierarchy. As noted in the introduction this method cannot be extended to all infinite simply-typed terms. The Krivine machine [28] provides a solution by deferring substitutions till "the last moment". The issues discussed here are related to β-reduction, the fixpoints do not enter into the picture. Indeed, as remarked on page 106, if we admit infinite terms then fixpoints can be simply eliminated. So the core of the Transfer Theorem is essentially about β-reduction in infinite terms.

The proof of Transfer Theorem is presented in the full version of the paper. Its schema is presented in Figure 1. It simplifies the exposition to think that all constants are binary, so that $BT(M)$ is a binary tree. There is a non-deterministic parity automaton $\mathcal{A}$ on infinite binary trees that recognizes trees that are models of φ. For a term M we define a game $\mathcal{K}(\mathcal{A}, M)$. This game is presented in terms of configurations of a Krivine machine. Eve wins in $\mathcal{K}(\mathcal{A}, M)$ iff $BT(M)$ is accepted by $\mathcal{A}$. An important step is to construct a reduced game $G(\mathcal{A}, M)$ with the property that Eve wins in the later game iff she wins in $\mathcal{K}(\mathcal{A}, M)$. For this we use Krivine machines in a similar way as in [39] the difference being that now we need to handle infinite terms. According to [35] the reduction from "$BT(M) \in L(\mathcal{A})$" to some equivalent of $\mathcal{G}(\mathcal{A}, M)$ can be also done by adapting the methods from [31] and [34]. Finally, and this is another important step, we show that $G(\mathcal{A}, M)$ is MSOL definable inside M (considered as the graph $Graph(M)$). Since there is an MSOL formula γ_{win} describing games where Eve wins, this gives the desired formula $\widehat{\varphi}$. Actually this schema works only for terms some special form that we call canonical. To complete the proof we show that the translation of M into a canonical form is an MSOL transduction [12].

5 Consequences of the Transfer Theorem for evaluation

The objective of this section is to present several corollaries of the Transfer Theorem. The first concerns the so-called global model-checking problem. For a given finite term M and a formula φ we want to find a finite representation of the set of nodes in the Böhm tree of M where the formula holds. For this we need to have a way of representing subtrees of the Böhm tree. One simple method is to just use the term obtained by the reduction sequence

producing that subtree. This method gives an effective way of representing every node v of $BT(M)$ by a term N_v from $Terms(\Sigma, \mathcal{T}, \mathcal{X})$ (where $\mathcal{T}$ is the set of types of subterms of M, and $\mathcal{X}$ is the set variables appearing in M) so that $BT(N_v)$ is the subtree of $BT(M)$ rooted at node v. The Transfer Theorem gives immediately the following corollary:

▶ **Corollary 3.** *Let M be a finite term of λY-calculus and φ an MSOL formula. Nodes of $BT(M)$ can be represented by terms of $Terms(\Sigma, \mathcal{T}, \mathcal{X})$ so that $N \vDash \widehat{\varphi}$ iff the tree rooted in the node of $BT(M)$ represented by N satisfies φ.*

Let us explain in more detail how nodes of $BT(M)$ can be represented by terms over the same vocabulary as M. One can observe that a term obtained by a reduction sequence from M can be written as a term with explicit substitutions $(N_0[\sigma_0])(N_1[\sigma_1])\dots(N_k[\sigma_k])$ where $N_0 \dots N_k$ are terms and $\sigma_0, \dots, \sigma_k$ are substitutions ranging over terms with explicit substitutions. The important point of this representation is that the terms $N_0, \dots, N_k$ as well as all the terms appearing in substitutions are subterms of M. One can easily prove this observation directly by induction on the number of reductions. It also follows immediately from the formalization of the reduction with the Krivine machine. Explicit substitution notation can then be eliminated by introducing λ-abstractions. An expansion of a term with substitutions, $E(N[\sigma])$, is defined by induction as $E(N[\sigma]) = (\lambda x_0 \dots x_n.N)E(\sigma(x_0)) \dots E(\sigma(x_n))$; where $x_0, \dots, x_n$ are the variables free in N. Hence $E(N_0[\sigma_0])E(N_1[\sigma_1]) \dots E(N_k[\sigma_k])$ belongs to $Terms(S, \mathcal{T}, \mathcal{X})$ which allows us to use the Transfer Theorem. Recall that, but for [39], other papers on global model-checking make a detour to collapsible pushdown automata [9, 7].

Decidability of higher-order matching restricted to $Terms(\Sigma, \mathcal{T}, \mathcal{X})$ without fixpoint operators follows also by the same type of reasoning. The problem is stated as follows. Given a finite term M, and a closed finite term K: can one substitute terms for the free variables of M so that the result is β-equal to K?

▶ **Corollary 4.** *Fix $\Sigma, \mathcal{T}, \mathcal{X}$. For every pair of finite terms $M, K \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$ such that K is closed and does not have fixpoint operators, it is decidable if there is a substitution σ in $Terms(\Sigma, \mathcal{T}, \mathcal{X})$ such that $M\sigma =_\beta K$.*

Let us briefly sketch the argument. For a term M we will write $shape(M)$ for an MSOL formula defining the set of terms that can be obtained from M by substitutions. This formula just states that the top of the tree is term M, but in places where M has free variables the tree is arbitrary. Since K is a finite term without Y binder, it has a normal form K' that is a finite term. Since K' does not have free variables, $shape(K)$ defines exactly one term. Suppose that $shape(K)$ is our formula φ and take $\widehat{\varphi}$ given by the theorem. Consider the formula $shape(M) \wedge \widehat{\varphi}$. A term satisfying this formula is obtained from M by a substitution, and that its normal form is K'. So the higher-order matching problem for M and K reduces to satisfiability of this formula. Observe that in principle the substitution can use infinite terms. Nevertheless, recall that if there is a tree satisfying an MSOL formula then there is a regular one. So if there is a solution to a matching problem then there is one in finite terms. Here we allowed both M and σ to use fixpoint operators, we can disallow this by adding one more conjunct to the formula above. The decidability of the case without fixpoints has been shown by Schmidt-Schauß [42].

Next, we would like to present a kind of synthesis result. The task is to construct a program satisfying a given specification from a given finite set of modules. The specification is given by an MSOL formula φ. Every module is a finite λY-term.

▶ **Corollary 5.** *Given a formula φ and finite set of closed λY-terms $M_1 \dots, M_l$, it is decidable if there is a closed term K constructed from $M_1 \dots, M_l$ by means of application, Y-variables and Y-binders, such that $BT(K) \vDash \varphi$. If there is such a term then there exists a finite one.*

The requirement that K does not use constants from the signature is essential, since otherwise we would get a trivial solution ignoring the modules. For the proof of the corollary consider terms of the form $(\lambda x_1 \dots x_l.N)M_1 \dots M_l$ where N is constructed only with applications, the variables $\{x_1, \dots, x_l\}$, Y-variables and Y-binders. Indeed, after reducing this term l-times we get a term as required in the corollary. So the restriction on the shape of K can be expressed by a formula $\mathit{shape}((\lambda x_1 \dots x_l.z)M_1 \dots M_l) \wedge \gamma$, where γ is a formula saying that the only labels appearing in the subtree starting in the node corresponding to z are variables $x_1, \dots, x_l$, application, Y-variables, and fixpoint binders. Summing-up, the solution to the problem stated in the corollary is to decide the satisfiability of the formula

$$\widehat{\varphi} \wedge \mathit{shape}((\lambda x_1 \dots x_l.z)M_1 \dots M_l) \wedge \gamma \ .$$

This formula asks for a term having a particular shape and whose Böhm tree satisfies $\widehat{\varphi}$. Once again, if there is a solution then there is a regular solution, that is a finite term.

6 Conclusions

We have shown that every MSOL property of Böhm trees is effectively an MSOL property of terms. The possibility that such a Transfer Theorem may hold has been indicated by a number of results in the literature: Ong's Theorem [31] stating that the MSOL properties of Böhm trees of λY-terms is decidable; Courcelle and Knapik's [13] result showing a similar theorem for a variant of evaluation of first-order terms; and finally, a result of Salvati [41] demonstrating that in the context of finite λ-calculus, recognizability is preserved under inverse homomorphism.

The translation we propose for going from MSOL properties of Böhm trees to MSOL properties of λY-terms not only works for infinite terms but also depends on very few properties of terms themselves. We just need to fix a finite set of λ-variables that a term can use as well as a finite set of types that the subterms of a term may have. Apart from this, the translation does not depend on any other structural properties of terms. This is rather surprising as the set of terms that use a fixed number of λ-variables does not form a set that is closed under $\beta\delta$-reduction. For this reason the method of Courcelle and Knapik could not be extended to higher-order types, since due to α-conversion intermediate results of evaluation could need an unbounded number of λ-variables. The invariants on reduction that we express with the Krivine machine overcome this difficulty.

The obvious question is the relation of our Transfer Theorem concerning $\beta\delta$-reduction to two other transfer results concerning unfolding operation [14] and Muchnik iteration [43, 45], respectively. Recall that while Muchnik iteration is strictly stronger than unfolding, the two operations can be used to define the classes of trees forming so called pushdown hierarchy [11, 10].

We cannot compare directly the three results since the Transfer Theorems for unfolding and Muchnik iteration work on graphs, while our theorem works on λY-terms that, seen as graphs, are trees with back edges. For this reason we cannot say that our result implies the other two. Yet, if restricted to trees with back edges, the result on unfolding is a special case of our result: the unfolding can be simulated by δ-reduction. We do not know if it is possible to simulate Muchnik iteration directly using $\beta\delta$-reduction. Nevertheless, it is well-known that, up to simple MSOL-interpretations, every tree in the pushdown hierarchy is a Böhm tree of a finite term. Hence, similarly to Muchnik's iteration, $\beta\delta$-reduction allows to obtain all trees in the pushdown hierarchy. Moreover, recent result of Parys [36] shows that there is a term whose Böhm tree does not belong to the pushdown hierarchy. This proves that

our Transfer Theorem is not a consequence of Muchnik's theorem. The study of relations between Muchnik iteration and $\beta\delta$-reduction, as well as constructing new hierarchies using the new transfer theorem, are interesting directions for further research.

References

1. Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(1):1–23, 2007.
2. Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In *TLCA'05*, volume 3461 of *LNCS*, pages 39–54, 2005.
3. H. Barendregt. *The Lambda Calculus*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.
4. William Blum and C.-H. Luke Ong. The safe lambda calculus. *Logical Methods in Computer Science*, 5(1), 2009.
5. Achim Blumensath, Thomas Colcombet, and Christof Löding. Logical theories and compatible operations. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 73–106. AUP, 2008.
6. Achim Blumensath and Stephan Kreutzer. An extension to Muchnik's theorem. *Journal of Logic and Computation*, 13:59–74, 2005.
7. C. Broadbent, A. Carayol, L. Ong, and O. Serre. Recursion schemes and logical reflection. In *LICS*, pages 120–129, 2010.
8. Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. A saturation method for collapsible pushdown systems. In *ICALP (2)*, volume 7392 of *LNCS*, pages 165–176, 2012.
9. Christopher H. Broadbent and C.-H. Luke Ong. On global model checking trees generated by higher-order recursion schemes. In *FOSSACS*, volume 5504 of *LNCS*, pages 107–121, 2009.
10. A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS'03*, volume 2914 of *LNCS*, pages 112–124, 2003.
11. Didier Caucal. On infinite terms having a decidable monadic theory. In *MFCS'02*, volume 2420 of *LNCS*, pages 165–176, 2002.
12. Bruno Courcelle. Monadic second-order graph transductions: A survey. *Theoretical Computer Science*, 126:53–75, 1994.
13. Bruno Courcelle and Teodor Knapik. The evaluation of first-order substitution is monadic second-order compatible. *TCS*, 281:177–206, 2002.
14. Bruno Courcelle and Igor Walukiewicz. Monadic second-order logic, graphs and unfoldings of transition systems. *Ann. of Pure and Appl. Log.*, 92:35–62, 1998.
15. W. Damm. The IO- and OI-hierarchies. *Theor. Comp. Sci.*, 20:95–207, 1982.
16. J. Engelfriet and E. M. Schmidt. IO and OI. I. *Journal of computer and system sciences*, 15:328–353, 1977.
17. J. Engelfriet and E. M. Schnidt. IO and OI. II. *Journal of computer and system sciences*, 16:67–99, 1978.
18. M. J. Fischer. *Grammars with macro-like productions*. PhD thesis, Harvard University, 1968.
19. M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461, 2008.
20. Richard Kennaway and Fer-Jan de Vries. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*, chapter 12, pages 668–711. Cambridge University Press, 2003.

21 T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, volume 2303 of *LNCS*, pages 205–222, 2002.

22 T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *ICALP*, volume 3580 of *LNCS*, pages 1450–1461, 2005.

23 N. Kobayashi. Higher-order program verification and language-based security. In *ASIAN*, volume 5913 of *LNCS*, pages 17–23, 2009.

24 N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pages 416–428. ACM, 2009.

25 N. Kobayashi. Types and recursion schemes for higher-order program verification. In *APLAS*, volume 5904 of *LNCS*, pages 2–3, 2009.

26 N. Kobayashi and L. Ong. A type system equivalent to modal mu-calculus model checking of recursion schemes. In *LICS*, pages 179–188, 2009.

27 Naoki Kobayashi and C.-H. Luke Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011.

28 J-L. Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

29 Dietrich Kuske. Compatibility of Shelah and Stupp's and Muchnik's iteration with fragments of monadic second order logic. In *STACS*, volume 1 of *LIPIcs*, pages 467–478, 2008.

30 Dietrich Kuske and Markus Lohrey. Monadic chain logic over iterations and applications to pushdown systems. In *LICS*, pages 91–100, 2006.

31 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90, 2006.

32 C.-H. L. Ong. On model-checking trees generating by higher-order recursion schemes. Long version available from the author's web page, 2006.

33 C.-H. Luke Ong and Steven James Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. In *POPL*, pages 587–598, 2011.

34 Luke Ong. Local computation of beta-reduction by game semantics. Version of February 2013, received by email form the author.

35 Luke Ong. Private communication, February 2013.

36 Pawel Parys. On the significance of the collapse operation. In *LICS'12*, pages 521–530, 2012.

37 G. D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977.

38 M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–23, 1969.

39 S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *ICALP (2)*, volume 6756 of *LNCS*, pages 162–173, 2011.

40 S. Salvati and I. Walukiewicz. Evaluation is MSOL compatible. Technical report, INRIA-CNRS, 2013. http://hal.inria.fr/hal-00773126.

41 Sylvain Salvati. Recognizability in the simply typed lambda-calculus. In *WoLLIC*, volume 5514 of *LNCS*, pages 48–60, 2009.

42 Manfred Schmidt-Schauß. Decidability of arity-bounded higher-order matching. In *CADE*, volume 2741 of *LNCS*, pages 488–502, 2003.

43 A.L. Semenov. Decidability of monadic theories. In *MFCS'84*, volume 176 of *LNCS*, pages 162–175, 1984.

44 Jonathan Stupp. The lattice-model is recursive in the original model. Institute of Mathematics, The Hebrew University, Jerusalem, January 1975.

45 Igor Walukiewicz. Monadic second-order logic on tree-like structures. *Theor. Comput. Sci.*, 275(1-2):311–346, 2002.

Model Checking and Functional Program Transformations*

Axel Haddad

LIAFA (Université Paris Diderot / CNRS)
LIGM (Université Paris Est / CNRS)

Abstract

We study a model for recursive functional programs called higher order recursion schemes (HORS). We give new proofs of two verification related problems: reflection and selection for HORS. The previous proofs are based on the equivalence between HORS and collapsible pushdown automata and they lose the structure of the initial program. The constructions presented here are based on shape preserving transformations, and can be applied on actual programs without losing the structure of the program.

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

Keywords and phrases Higher-order recursion schemes, Model checking, Tree automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.115

1 Introduction

Recursion schemes were introduced in the early 70s as a model of computation, describing the syntactical part of a functional program [18, 6, 7, 8]. Originally they only handled order-0 (constants) or order-1 (functions on constants) expressions, but not higher-order types. Higher-order versions of recursion schemes were later introduced to deal with functions taking functions as arguments [13, 9, 10].

Recently, the focus came back on higher-order recursion schemes when considering them as generators of possibly infinite trees [14, 19, 12]. Indeed, roughly speaking a recursion scheme is a deterministic rewriting system on typed terms that generates ad infinitum a unique infinite tree. As the trees they generate are very general and as they can capture the computation tree of (higher-order) functional programs, studying their logical properties leads to very natural and challenging problems.

The most popular one is *(local) model-checking*: for a given recursion scheme and a formula *e.g.* from monadic second order logic (MSO) or μ-calculus, decide whether the tree generated by the scheme satisfies the formula. Following partial decidability results for the subclass of *safe* recursion schemes [14, 5], Ong proved, using the notion of traversals, the decidability of MSO model-checking for the whole class of trees generated by recursion schemes [19]. Since then, other proofs of this result have been obtained using different approaches: Hague, Murawski, Ong and Serre established the equivalence of schemes and higher-order collapsible pushdown automata (CPDA), and then showed the MSO decidability by reduction to parity games on collapsible pushdown automata [12]; following ideas from [1], Kobayashi and Ong [17] developed the type system of [15] to obtain a new proof. Finally, Salvati and Walukiewicz used Krivine machines to establish the MSO decidability of λ-Y-calculus, which is a typed lambda calculus with recursion, equivalent to higher order recursion schemes [21].

* This work was supported by the project AMIS (ANR 2010 JCJC 0203 01 AMIS).

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 115–126

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

Another important problem is *global model-checking*: for a given recursion scheme and a formula, provide a finite representation of the set of nodes in the tree generated by the scheme where the formula holds. Broadbent, Carayol, Ong and Serre answered this question using a so-called endogenous approach to represent the set of nodes: they showed how to construct another scheme generating the same tree as the original scheme except that now those nodes where the formula holds are marked [3]. They refer to this property as the *logical reflection*. The technique they used relies on the equivalence between schemes and CPDA.

Going further with the idea of marking a set of nodes, Carayol and Serre considered in [4] the following problem called *logical selection* and generalising global model-checking: for a given recursion scheme and an MSO formula $\varphi[X]$ with one free set variable, provide (if it exists) a finite representation of a set S of nodes in the tree generated by the scheme such that $\varphi[S]$ holds. They show that one can construct another scheme generating the same tree as the original scheme except that now those nodes are marked and describe a set S with the previous property. Again, they relied on the equivalence with CPDA.

One interest for logical reflection and logical selection is that they can be used to modify a scheme in a useful way. Indeed, assume some (syntax tree of a) program is described by a scheme and that it contains bad behaviours: using logical reflection, one can get a new scheme where those bad behaviours are marked and this latter scheme can then easily be modified to remove these behaviours. Using logical selection, one can *e.g.* select branches in the syntax tree so that a given property is satisfied in the resulting subtree. A drawback of the approach in [3, 4] that goes back and forth between schemes and CPDA is that the scheme that is finally obtained is structurally very far from the original one (the names of the non terminals as well as the shape of the rewriting rules have been lost): hence this is a serious problem if one is interested in doing automated correction of programs or even synthesis.

Our main contribution in this paper is to provide proofs of both logical reflection and selection without appealing to CPDA as we only reason on schemes. Our constructions avoid the loss of the structure, *i.e.* the solution scheme is obtained only by duplicating and annotating some parts of the initial scheme and the transformation is easily reversible. Our constructions are based on the type system and the game used by Kobayashi and Ong in their proof of the model-checking decidability [17]. There is no known correspondence between these proofs and the former ones. In order to prove the logical reflection, we introduce the notion of morphism inspired by denotational semantics, and we give a morphism for MSO. In order to prove the logical selection, we embed carefully a winning strategy of the model-checking game into the scheme.

In Section 2 we give the basic definitions and in Section 3 we introduce the two problems we are looking at in the paper. In Section 4 we introduce morphisms, explain how to "embed" a morphism into a scheme, and give some possible applications. In Section 6, we show how to use morphisms to obtain a new proof of logical reflection. In Section 7, we deal with logical selection. In Section 5, we present a simple example that describes how we can use a morphism describing a property to transform a scheme. An extended version of this work is available online: `http://hal.archives-ouvertes.fr/hal-00865682`.

2 Preliminaries

In this section we give the definition of a higher order recursion scheme, which is a deterministic rewriting system that produces an infinite tree. It handles terms of typed symbols, i.e. each symbol is a constant or a function that can have functions as arguments. Terms may represent expressions of higher order programming languages, for example the term **`Apply Copy File`**

means "apply the function `Copy` to the file". In this example an intuitive rewrite rule for the function `Apply` would be the term where the first argument is applied on the second argument, written: `Apply` $\varphi\ x \to \varphi\ x$.

Types. *Types* are defined by the grammar $\tau ::= o \mid \tau \to \tau$ and o is called the *ground type*. Considering that $\to$ is associative to the right (i.e. $\tau_1 \to (\tau_2 \to \tau_3)$ can be written $\tau_1 \to \tau_2 \to \tau_3$), any type τ can be written uniquely as $\tau_1 \to \cdots \to \tau_k \to o$. The integer k is called the *arity* of τ. We define the *order of a type* by $\text{order}(o) = 0$ and $\text{order}(\tau_1 \to \tau_2) = \max(\text{order}(\tau_1) + 1, \text{order}(\tau_2))$. For instance $o \to o \to o \to o$ is a type of order 1 and arity 3, $(o \to o) \to (o \to o)$, that can also be written $(o \to o) \to o \to o$ is a type of order 2.We let $\tau^\ell \to \tau'$ be a shorthand for $\underbrace{\tau \to \cdots \to \tau}_{\ell \text{ times}} \to \tau'$.

Terms. Let Γ be a finite set of typed symbols, and Let Γ^τ denote the set of symbols of type τ. For all type τ, we define the set of *terms* of type τ, $\mathcal{T}^\tau(\Gamma)$ as the smallest set containing the symbols of type τ and the application of a term of type $\tau' \to \tau$ to a term of type τ' for all τ'; formally: $\Gamma^\tau \subseteq \mathcal{T}^\tau(\Gamma)$ and $\bigcup_{\tau'} \{t\ s \mid t \in \mathcal{T}^{\tau' \to \tau}(\Gamma), s \in \mathcal{T}^{\tau'}(\Gamma)\} \subseteq \mathcal{T}^\tau(\Gamma)$. We shall write $\mathcal{T}(\Gamma)$ for the set of terms of any type, and $t : \tau$ if t has type τ. The arity of a term t, $arity(t)$, is the arity of its type. Remark that any term t can be uniquely written as $t = \alpha\ t_1 \dots t_k$ with $\alpha \in \Gamma$ and $0 \leq k \leq \text{arity}(\alpha)$. We say that α is the *head* of the term t. For instance, let $\Gamma = \{F : (o \to o) \to o \to o\ ,\ G : o \to o \to o\ ,\ H : (o \to o)\ ,\ \mathtt{a} : o\}$. Then $F\ H$ and $G\ \mathtt{a}$ are terms of type $o \to o$; $F(G\ \mathtt{a})\ (H\ (H\ \mathtt{a}))$ is a term of type o; $F\ \mathtt{a}$ is not a term since F is expecting a first argument of type $o \to o$ while $\mathtt{a}$ has type o.

A set of symbols of order at most 1 (i.e. each symbol has type o or $o \to o \to \cdots \to o$) is called a *signature*. In the following, we use the letters t, r, s to denote terms, and given a tuple of term $\vec{t} = (t_1, \dots, t_k)$ we may use the shorthand $s\ \vec{t}$ to denote $s\ t_1 \dots t_k$.

Contexts. Let $t : \tau$, $t' : \tau'$ be two terms, $x : \tau'$ be a symbol of type τ', then we write $t_{[x \mapsto t']} : \tau$ for the term obtained by substituting all occurrences of x by t' in the term t. A *τ-context* is a term $C[\bullet^\tau] \in \mathcal{T}(\Gamma \uplus \{\bullet^\tau : \tau\})$ containing exactly one occurrence of $\bullet^\tau$; it can be seen as an application turning a term into another, such that for all $t : \tau$, $C[t] = C[\bullet^\tau]_{[\bullet^\tau \mapsto t]}$. In general we will only talk about ground type contexts where $\tau = o$ and we will omit to specify the type when it is clear. For instance, if $C[\bullet] = F \bullet\ (H\ (H\ a))$ and $t' = G\ \mathtt{a}$ then $C[t'] = F\ (G\ \mathtt{a})\ (H\ (H\ \mathtt{a}))$.

Rewrite Rules. Given two disjoint sets of symbols $\Gamma, \mathcal{V}$ where $\mathcal{V}$ is called a set of *variables*, we define a (fully applied) *rewrite rule* on Γ and $\mathcal{V}$ as a pair of terms of $\mathcal{T}(\Gamma \uplus \mathcal{V})$ of type o written $F\ x_1 \dots x_k \to e$ with $F \in \Gamma$, $x_1, \dots, x_k \in \mathcal{V}$ such that for all $i \neq j$ $x_i \neq x_j$, and $e \in \mathcal{T}(\Gamma \uplus \{x_1, \dots, x_k\})$. Given a set of rewrite rules $\mathcal{R}$, we define the *rewriting relation* $\to \in\ \mathcal{T}(\Gamma)^2$ as $t \to t'$ iff there exists a context $C[\bullet]$, a rewrite rule $F\ x_1 \dots x_k \to e$, and a term $F\ t_1\ \dots\ t_k : o$ such that $t = C[F\ t_1 \dots t_k]$ and $t' = C[e_{[x_1 \mapsto t_1] \dots [x_k \mapsto t_k]}]$. We call $F\ t_1\ \dots\ t_k : o$ a *redex*. We define $\to^*$ as the reflexive and transitive closure of $\to$. Finally we say that a set of rewrite rules is *deterministic*[1] if for all $F \in \Gamma$ there exists at most one rule of the form $F\ x_1 \dots x_k \to e$.

Trees. Let Σ be a finite signature, m be the maximum arity in Σ and $\perp : o$ be a fresh symbol. A (ranked) *tree* t over $\Sigma \uplus \perp$ is a mapping $t : dom^t \to \Sigma \uplus \perp$, where dom^t is a prefix-closed subset of $\{1, \dots, m\}^*$ such that if $u \in dom^t$ and $t(u) = a$ then $\{j \mid u \cdot j \in dom^t\} = \{1, \dots, arity(a)\}$.

Given a node $u \in dom^t$, we define the subtree of t rooted at u as the tree t_u such that $dom^{t_u} = \{j \mid u \cdot j \in dom^t\}$ and for all $v \in dom^{t_u}$, $t_u(v) = t(u \cdot v)$. Given $a : o^k \to o$ and

[1] Although the rules are deterministics, there may be several possible rules to apply in a term.

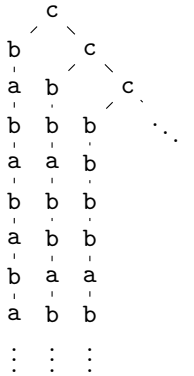
some trees $t_1, \ldots, t_k$ we use the notation $a\ t_1 \ldots t_k$ to denote the tree t' whose domain is $dom^{t'} = \bigcup_i i \cdot dom^{t_i}$, $t'(\varepsilon) = a$ and $t'(i \cdot u) = t_i(u)$. Note that there is a direct bijection between ground terms of $\mathcal{T}^o(\Sigma \uplus \bot)$ and finite trees. Hence we will freely allow ourselves to treat ground terms over $\Sigma \uplus \bot$ as trees. We define the partial order $\sqsubseteq$ over trees as $t \sqsubseteq t'$ if $dom^t \subseteq dom^{t'}$ and for all $u \in dom^t$, $t(u) = t'(u)$ or $t(u) = \bot$. Given a (possibly infinite) sequence of trees $t_0, t_1, t_2, \ldots$ such that $t_i \sqsubseteq t_{i+1}$ for all i, the set of all t_i has a supremum that is called the *limit tree* of the sequence.

Higher Order Recursion Schemes. A *higher order recursion scheme (HORS)* $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ is a tuple such that: $\mathcal{V}$ is a finite set of typed symbols called *variables*; Σ is a finite signature, called the *set of terminals*; $\mathcal{N}$ is a finite set of typed symbols called *non-terminals*; $\mathcal{R}$ is a deterministic set of rewrite rules on $\Sigma \uplus \mathcal{N}$ and $\mathcal{V}$, such that there is exactly one rule per non terminal and no rule for terminals; $S \in \mathcal{N}$ is the *initial non-terminal*.

We define inductively the $\bot$*-transformation* $(\cdot)^\bot : \mathcal{T}^o(\mathcal{N} \uplus \Sigma) \to \mathcal{T}^o(\Sigma \uplus \{\bot : o\})$ turning a ground term into a finite tree as: $(F\ t_1\ \ldots\ t_k)^\bot = \bot$ for all $F \in \mathcal{N}$ and $(a\ t_1\ \ldots\ t_k)^\bot = a\ t_1^\bot \ldots t_k^\bot$ for all $a \in \Sigma$. We define a *derivation*, as a possibly infinite sequence of terms linked by the rewrite relation, and we will mainly look at derivations where the first term of the sequence is equal to the initial non terminal. Let $t_0 = S \to t_1 \to t_2 \to \cdots$ be a derivation, then one can check that $(t_0)^\bot \sqsubseteq (t_1)^\bot \sqsubseteq (t_2)^\bot \sqsubseteq \ldots$, hence it admits a limit. One can prove[2] that the set of all such limit trees has a greatest element that we denote $\|\mathcal{G}\|$ and refer to as the *value tree* of $\mathcal{G}$. Note that $\|\mathcal{G}\|$ is the supremum of $\{t^\bot \mid S \to^* t\}$. Given a term $t : o$, we denote by $\mathcal{G}_t$ the scheme obtained by transforming G such that it starts derivations with the term t, formally, $\mathcal{G}_t = \langle \mathcal{V}, \Sigma, \mathcal{N} \uplus \{S'\}, \mathcal{R} \uplus \{S' \to t\}, S' \rangle$. One can prove that if $t \to t'$ then $\|\mathcal{G}_t\| = \|\mathcal{G}_{t'}\|$. We call $\|\mathcal{G}_t\|$ the value tree of t in $\mathcal{G}$.

Parallel derivation. Intuitively, the *parallel rewriting* from a term t is obtained by rewriting all the redexes in t simultaneously. Formally, given a term $t = \alpha\ t_1 \ldots t_k$ with $\alpha \in \Sigma \uplus \mathcal{N}$, we define inductively the parallel rewriting $t^\star$ of t as if $\alpha \in \Sigma$ or $k < \text{arity}(\alpha)$, then $t^\star = \alpha\ t_1^\star \ldots t_k^\star$, if $\alpha \in \mathcal{N}$ and $k = \text{arity}(\alpha)$, let $\alpha\ x_1 \ldots x_k \to e$ be the rewrite rule associated to α, we have $t^\star = e_{[\forall i\ x_i \mapsto t_i^\star]}$. Given t, t', we write $t \Rightarrow t'$ for the relation "$t' = t^\star$". Notice that if $t \Rightarrow t'$ and $t' \to^* t''$ then $t \to^* t''$ (in particular $t \to^* t'$). Furthermore the derivation $S \Rightarrow t_1 \Rightarrow t_2 \Rightarrow \ldots$ leads to $\|\mathcal{G}\|$.

▶ **Example 1.** Let $\mathcal{G} = \langle \Sigma, \mathcal{V}, \mathcal{N}, S, \mathcal{R} \rangle$ with $\Sigma = \{\mathtt{a}, \mathtt{b} : o \to o, c : o \to o \to o\}$, $\mathcal{V} = \{x : o, \varphi : o \to o\}$, $\mathcal{N} = \{S : o, I, F : (o \to o) \to o, D, A : (o \to o) \to o \to o\}$ and $\mathcal{R} = \{S \to F\ \mathtt{b},\ \ D\ \varphi\ x \to \varphi\ (\varphi\ x),\ \ A\ \varphi\ x \to \varphi\ (\mathtt{a}\ x),\ \ I\ \varphi \to \varphi\ (I\ \varphi),\ \ F\ \varphi\ \to\ \mathtt{c}\ (I\ (A\ \varphi))\ (F\ (D\ \varphi))\}$.
Remark the rewrite rule associated to D: it means that for any function t, $D\ t$ is simply the function obtained by composing t with itself. The rule associated to I is also interesting: for any function t, $I\ t$ leads to the infinite iteration of t. For example the term $I\ \mathtt{a}$ can be derived to obtain $\mathtt{a}\ (\mathtt{a}\ (\mathtt{a}\ (\ \ldots\ ))))$. The tree $\|\mathcal{G}\|$ is depicted on the left, its branches are labelled by $\mathtt{c}^\omega$ or $\mathtt{c}^n \cdot (\mathtt{b}^{2^{(n-1)}} \cdot \mathtt{a})^\omega$ for all $n \geq 1$.



Parity tree automaton. A *non-deterministic max parity automaton* (we will just refer to them as automata in the following) is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$ with Σ a finite signature, Q a finite set called the *set of states*, $\delta \subseteq \{q \xrightarrow{a} (q_1, \ldots, q_{\text{arity}(a)}) \mid a \in \Sigma, q, q_1, \ldots, q_{\text{arity}(a)} \in Q\}$

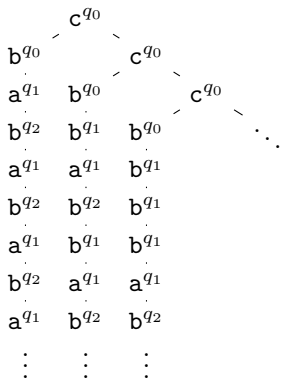
[2] The existence of a value tree is a consequence the confluence property of HORS.

is called the *transition relation*, $q_0 \in Q$ the *initial state*, $\Omega : Q \to \{1, \dots, m_{\max}\}$ for some $m_{\max} \in \mathbb{N}$ the *colouring function*.

Given an infinite tree t on Σ we define a *run* r of $\mathcal{A}$ on t as a tree on $\Sigma \times Q = \{a^q : o^k \to o \mid a : o^k \to o, q \in Q\}$ such that $dom^r = dom^t$, for all $u \in dom^t$, if $r(u) = a^q$ then $t(u) = a$ and there exists $q \xrightarrow{a} q_1, \dots, q_k \in \delta$ such that for all i, $r(u \cdot i) = t(u \cdot i)^{q_i}$, and $r(\varepsilon) = t(\varepsilon)^{q_0}$.

We say that the automaton $\mathcal{A}$ accepts the tree t, written $t \models \mathcal{A}$, if there exists a run r on t such that for every infinite branch $b = (a_0, q_0) \cdot j_0 \cdot (a_1, q_1) \cdot j_1 \cdot \dots$ in r, the greatest colour seen infinitely often in $\Omega(q_0), \Omega(q_1), \Omega(q_2), \dots$ is even. We define $\mathcal{A}_q$ as the automaton obtained by changing in $\mathcal{A}$ the initial state to q: $\mathcal{A}_q = \langle \Sigma, Q, \delta, q, \Omega \rangle$, and we say that $\mathcal{A}$ accepts the tree t from state q, if $t \models \mathcal{A}_q$.

▶ **Example 2.** Let $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \Omega \rangle$ be an automaton with $\Sigma = \{\mathtt{a}, \mathtt{b} : o \to o, c : o \to o \to o\}$, $Q = \{q_0, q_1, q_2\}$, $\Omega = \{q_0 \mapsto 0, q_1 \mapsto 1, q_2 \mapsto 2\}$ and $\delta = \{q_0 \xrightarrow{\mathtt{c}} (q_0, q_0), q_0 \xrightarrow{\mathtt{b}} q_1, q_1 \xrightarrow{\mathtt{a}} q_2, q_1 \xrightarrow{\mathtt{b}} q_1, q_2 \xrightarrow{\mathtt{a}} q_2, q_2 \xrightarrow{\mathtt{b}} q_1\}$.
The automaton $\mathcal{A}$ accepts the trees whose branches are labelled by c^ω or by $c^* \cdot b \cdot (b^* \cdot \mathtt{a})^\omega$. An accepting run of $\mathcal{A}$ on the tree $\|\mathcal{G}\|$ of Example 1 is depicted on the right.



3 Logical reflection and logical selection

In this section we formalise the notion of annotated tree and we define the problems of logical reflection and logical selection we are interested in.

Given a signature Σ, a set X and a tree t on the signature Σ, we define the signature $\Sigma \times X = \{(a, x) : o^k \to o \mid a : o^k \to o \in \Sigma, x \in X\}$ and we say that the tree t' on the signature $\Sigma \times X$ is an *X-annotation* of t, if $dom^t = dom^{t'}$ and if for all $u \in dom^t$, $t'(u) = (t(u), x)$ for some $x \in X$. For example a run of an automaton over a tree t is a Q-annotation of t, with Q being the set of states of the automaton. Furthermore, for any tree t' on the signature $\Sigma \times X$, we define $Unlab(t')$ as the tree on Σ obtained by turning all nodes (a, x) of t' into a *i.e.* the tree obtained by removing the annotated part of the tree.

Given a set of nodes S in a tree t, we define an *S-marking of t* as a $\{\mathtt{0}, \mathtt{1}\}$-annotation t' of t such that for all nodes u, $u \in S$ if and only if $t'(u) = (t(u), \mathtt{1})$. Given a μ-calculus formula φ, we say that t' is a *φ-reflection* of t if it is a marking of the set of nodes u such that the subtree of t rooted at u satisfies the formula φ. Given a monadic second-order logic (MSO) formula $\varphi[x]$ with a first order free variable, we say that t' is a *$\varphi[x]$-reflection* of t if it is a marking of the set of nodes u satisfying $\varphi[u]$. Given an MSO formula $\varphi[X]$ with a second order free variable, we say that t' is a *$\varphi[X]$-selection* of t if it is a marking of a set of nodes S satisfying $\varphi[S]$.

Finally, we define the *μ-calculus reflection*, MSO *reflection*, and MSO *selection* as follow. Given a class R of generators of trees (*i.e.* a set of finitely described elements such that to each $g \in R$ we associate a unique tree $\|g\|$), we say that R is (effectively) *reflective with respect to the μ-calculus* (*resp.* MSO) if for any μ-calculus formula φ (*resp.* any MSO formula $\varphi[x]$) and any tree generator $g \in R$, one can construct another generator g' such that the $\|g'\|$ is a φ-reflection (resp. $\varphi[x]$-reflection) of $\|g\|$. We say that R is (effectively) *selective with respect to* MSO if for any MSO formula $\varphi[X]$ and any generator $g \in R$ such that there

exists a subset S of nodes of $\|g\|$ satisfying $\varphi[S]$, one can construct another generator g' such that $\|g'\|$ is a $\varphi[X]$-selection.

The main contribution of this paper is to give new proofs of the fact that schemes have these properties. In order to do so we use the equivalence results between logic and automata, and we rather prove automata reflexivity and automata selectivity defined as follows.

Given a tree t on the signature Σ and an automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \Omega \rangle$, we define an *$\mathcal{A}$-reflection* of $\mathcal{A}$ on t as a 2^Q-annotation t' of t, such that for all nodes u, $t'(u) = (t(u), Q')$ with $Q' \subseteq Q$ being the set of states q such that $\mathcal{A}_q$ accepts the subtree of t rooted on u; we define an *$\mathcal{A}$-selection* of $\mathcal{A}$ on t as an accepting run of $\mathcal{A}$ on t. We say that a class R of generators of trees is *reflective with respect to automata* if for any automaton $\mathcal{A}$ and any generator $g \in R$, one can construct another generator g' such that $\|g'\|$ is an $\mathcal{A}$-reflection of $\|g\|$. We say that R is *selective with respect to automata* if for any automaton $\mathcal{A}$ and any generator $g \in R$ such that $\|g\| \models \mathcal{A}$, one can construct another generator g' such that $\|g'\|$ is an $\mathcal{A}$-selection of $\|g\|$.

From the equivalence between logics and automata [20] we have that schemes are reflective with respect to the μ-calculus iff they are reflective with respect to automata, and they are selective with respect to MSO iff they are reflective with respect to automata. Furthermore, it is shown in [3] that μ-calculus reflection for schemes implies MSO reflection.

4 Morphisms

4.1 Definitions

In the following we fix a scheme $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{V}, S, \mathcal{R} \rangle$. We define a *typed domain* $\mathcal{D}$ as a set such that each element is typed, and to each element $d : \tau_1 \to \tau_2 \in \mathcal{D}$ is associated a *partial* mapping f_d from $\mathcal{D}^{\tau_1}$ to $\mathcal{D}^{\tau_2}$, where $\mathcal{D}^\tau = \{d \in \mathcal{D} \mid d : \tau\}$. We write $d\ d'$ the element $f_d(d')$.

We define a *morphism* $[\![\cdot]\!] : \mathcal{T}(\Sigma \uplus \mathcal{N}) \to \mathcal{D}$ from terms on $\Sigma \uplus \mathcal{N}$ to the domain $\mathcal{D}$ as a mapping such that (1) if $t : \tau$ then $[\![t]\!] : \tau$, (2) if $t_0 : \tau_1 \to \tau_2$ and $t_1 : \tau_1$, then $[\![t]\!]\ [\![t']\!]$ is defined and equal to $[\![t\ t']\!]$. In the following, we refer to $[\![t]\!]$ as the $\mathcal{D}$-value of the term t.

▶ **Example 3.** Let $\mathcal{D} = \bigcup_\tau \{\bot^\tau : \tau, \top^\tau : \tau\}$ such that for all τ_1, τ_2: $\top^{\tau_1 \to \tau_2}\ \top^{\tau_1} = \top^{\tau_2}$; $\top^{\tau_1 \to \tau_2}\ \bot^{\tau_1} = \top^{\tau_2}$; $\bot^{\tau_1 \to \tau_2}\ \top^{\tau_1} = \top^{\tau_2}$; If $\tau_2 = o$ then $\bot^{\tau_1 \to \tau_2}\ \bot^{\tau_1} = \top^{\tau_2}$, otherwise $\bot^{\tau_1 \to \tau_2}\ \bot^{\tau_1} = \bot^{\tau_2}$. For $t : \tau$, we define $[\![t]\!]$ as $[\![t]\!] = \top^\tau$ if t contains a ground subterm and $[\![t]\!] = \bot^\tau$ otherwise. Then $[\![\cdot]\!]$ is a morphism since $t_1\ t_2$ contains a ground subterm iff t_1 contains a ground subterm, or t_2 contains a ground subterm, or $t_1\ t_2$ is ground.

Note that a morphism is entirely defined by its value on $\Sigma \uplus \mathcal{N}$, i.e. from those values one can compute $[\![t]\!]$ for any term t. Also remark that given a context $C[\bullet]$ and two terms t and t', if $[\![t]\!] = [\![t']\!]$ then $[\![C[t]]\!] = [\![C[t']]\!]$. We say that a morphism $[\![\cdot]\!]$ is *stable by rewriting* if for $t, t' \in \mathcal{T}$ such that $t \to t'$, $[\![t]\!] = [\![t']\!]$.

Finally, given a set of terms $\mathcal{T}' \subseteq \mathcal{T}(\Sigma \uplus \mathcal{N})$, we say that the morphism $[\![\cdot]\!]$ *recognises* $\mathcal{T}'$ if there exists a subset $\mathcal{D}'$ of $\mathcal{D}$ such that $t \in \mathcal{T}'$ if and only if $[\![t]\!] \in \mathcal{D}'$. In Example 3, the morphism recognises the set of terms containing a ground term as a subterm.

4.2 Embedding a morphism into a scheme

We fix a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ and a morphism $[\![\cdot]\!] : \mathcal{T}(\Sigma \uplus \mathcal{N}) \to \mathcal{D}$ on $\mathcal{G}$, stable by rewriting, such that for all type τ, $\mathcal{D}^\tau$ is finite. We transform $\mathcal{G}$ into $\mathcal{G}' = \langle \mathcal{V}', \Sigma', \mathcal{N}', \mathcal{R}', S \rangle$ which, while it is producing a derivation, evaluates $[\![t']\!]$ for any subterm t' of the current term and annotates the term with all these $\mathcal{D}$-values. The new symbols of $\mathcal{G}'$ are symbols

of $\mathcal{G}$ annotated with elements of the domain $\mathcal{D}$, and we define a transformation $(\cdot)^+$ from terms of $\mathcal{G}$ to terms of $\mathcal{G}'$ such that the transformation t^+ of t will be annotated with the $\mathcal{D}$-values of the subterms of t. The tree generated by $\mathcal{G}'$ will be annotated and when one removes these annotations, one retreives back the tree generated by $\mathcal{G}$. More precisely we show the following.

▶ **Theorem 4** (Embedding a morphism). *Given two terms $t, t' : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, if $t \Rightarrow_{\mathcal{G}} t'$, then $t^+ \Rightarrow_{\mathcal{G}'} t'^+$. In particular $Unlab(\|\mathcal{G}'_{t^+}\|) = \|\mathcal{G}_t\|$ (where Unlab is the function that removes the annotations).*

▶ Remark. This transformation keeps the structure of the original scheme *i.e.* the new symbols are simple labelings of the original ones, new rewrite rules are obtained by duplicating some subterms and labeling the symbols, a very simple transformation allows to get back to the original scheme, and there is a direct correspondence between derivations of the two schemes.

4.3 Applications

Embedding properties of subterms during a derivation, or properties of subtrees of the value tree, can be useful for program analysis: instead of saying "There will be a forbidden behaviour in the program" reflection allows to say during the execution of a program "Here, the forbidden behaviour will appear in this subexpression, but the rest of the program is valid". Furthermore, once a property is embedded into a scheme, one can add some new rewrite rules that deal explicitly with wether the property is valid or not. For example one could replace all forbidden subtrees of the value tree by a special symbol FORBIDDEN, as illustrated in Section 5.

Some morphisms generated by type systems are used in [15] to model check a subclass of trivial acceptance condition automata (i.e. automata where there is no colouring function, and the acceptance of a tree simply asks if there exists a run of the automaton on the tree). Then the construction allows one to reflect the accepting states of the automaton (as defined in Section 3). This result has been improved in [22] to deal with the whole class of trivial acceptance condition automata. In Section 6 we extend this result to show that for any parity tree automaton one can create a morphism that reflects the acceptance of the automaton on the value tree.

One can create a model (for example in [2]) to decide whether or not a ground term t would be productive or not (i.e. $\|t\| \neq \bot$). Reflecting the productivity of terms into a scheme $\|\mathcal{G}\|$ allows one to create a scheme $\mathcal{G}'$ on the signature $\Sigma \uplus \{\bot\}$ such that $\|\mathcal{G}'\| = \|\mathcal{G}\|$, but such that no derivation will create some non productive terms. In [11] we developed this idea to compare evaluation policies.

5 An example of scheme transformation

In this section, we present a simple example that describes how one can use the embedding procedure to transform a program.

Let $\mathrm{Map}(\{0,1\})$ be the typed domain inductively defined by $\mathrm{Map}(\{0,1\})^o = \{0,1\}$, $\mathrm{Map}(\{0,1\})^{\tau \to \tau'}$ is the set of total functions from $\mathrm{Map}(\{0,1\})^\tau$ to $\mathrm{Map}(\{0,1\})^{\tau'}$. And given $f : \tau \to \tau'$ and $h : \tau$ in $\mathrm{Map}(\{0,1\})$, $f \cdot h = f(h)$.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be defined by $\mathcal{V} = \{y : o \to o, x : o\}$, $\Sigma = \{\mathtt{a} : o^2 \to o, \mathtt{b} : o, \mathtt{c} : o\}$,

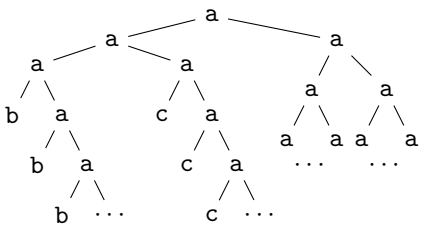


Figure 1 The value tree of the scheme of Section 5.

$\mathcal{N} = \{S : o, H : o \to o, J : o \to o, F : (o \to o) \to o\}$, $\mathcal{R}$ contains the following rewrite rules:

$$\begin{array}{lcllcl} S & \to & \mathtt{a}\ (F\ H)\ (F\ J) & H\ x & \to & \mathtt{a}\ x\ (H\ x) \\ J\ x & \to & \mathtt{a}\ (J\ x)\ (J\ x) & F\ y & \to & \mathtt{a}\ (y\ \mathtt{b})\ (y\ \mathtt{c}). \end{array}$$

The value tree of $\mathcal{G}$ is (partially) depicted in Figure 1. We write $[u, v]$ the mapping $f : \{0, 1\} \to \{0, 1\}$ such that $f(0) = u$ and $f(1) = v$ for all u, v. We define the morphism φ as follows:

$$\begin{array}{llll} \varphi(\mathtt{b}) = 0 & \varphi(\mathtt{c}) = 1 & \varphi(S) = 1 & \varphi(\mathtt{a})\ u\ v = u \vee v \\ \varphi(H)\ u = u & \varphi(J)\ u = 0 & \varphi(F)\ [u, v] = u \vee v. & \end{array}$$

One can check that the morphism φ is stable by rewrite. Furthermore it recognises the property "t has type o, and its value tree contains a $\mathtt{c}$", with the subset $A' = \{1\}$.

We construct a scheme $\mathcal{G}' = \langle \mathcal{V}', \Sigma', \mathcal{N}', S, \mathcal{R} \rangle$ that consists of an embedding of the morphism φ inside the scheme $\mathcal{G}$. $\mathcal{V}' = \{x : o,\ y^0, y^1 : o \to o\}$, $\Sigma' = \{\mathtt{a}^{0,0}, \mathtt{a}^{0,1}, \mathtt{a}^{1,0}, \mathtt{a}^{1,1} : o^2 \to o,\ \mathtt{b}, \mathtt{c} : o\}$, $\mathcal{N}' = \{S : o,\ H^0, H^1, J^0, J^1 : o,\ F^{[0,0]}, F^{[0,1]}, F^{[1,0]}, F^{[1,1]} : (o \to o)^2 \to o\}$, $\mathcal{R}'$ contains the following rewrite rules:

$$\begin{array}{lcl} S & \to & \mathtt{a}^{1,0}\ (F^{[0,1]}\ H^0\ H^1)\ (F^{[0,0]}\ J^0\ J^1) \\ H^0\ x & \to & \mathtt{a}^{0,0}\ x\ (H^0\ x) \\ H^1\ x & \to & \mathtt{a}^{1,1}\ x\ (H^0\ x) \\ J^0\ x & \to & \mathtt{a}^{0,0}\ (J^0\ x)\ (J^0\ x) \\ J^1\ x & \to & \mathtt{a}^{0,0}\ (J^1\ x)\ (J^1\ x) \\ F^{[0,0]}\ y^0\ y^1 & \to & \mathtt{a}^{0,0}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}) \\ F^{[0,1]}\ y^0\ y^1 & \to & \mathtt{a}^{0,1}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}) \\ F^{[1,0]}\ y^0\ y^1 & \to & \mathtt{a}^{1,0}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}) \\ F^{[1,1]}\ y^0\ y^1 & \to & \mathtt{a}^{1,1}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}). \end{array}$$

Let us explain how the rewrite rule related to $F^{[0,1]}$ has been produced. Recall the original rule: $F\ y \to \mathtt{a}\ (y\ \mathtt{b})\ (y\ \mathtt{c})$. The first occurrence of y is applied to $\mathtt{b}$, therefore it should be annotated with $\varphi(\mathtt{b}) = 0$. Similarly, the second occurrence of y should be annotated with $\varphi(\mathtt{c}) = 1$. This justifies the occurrence of y^0 and y^1 on the left hand part of the rule.

The annotation $[0, 1]$ means that this rule will be applied to an argument whose evaluation is the mapping $[0, 1]$, thus the evaluation of $y\ \mathtt{b}$ is equal to $[0, 1]\ \varphi(\mathtt{b}) = [0, 1]\ 0 = 0$ and by a similar reasoning the evaluation of $y\ \mathtt{c}$ is equal to 1. Therefore the occurrence of $\mathtt{a}$ should be annotated with $(0, 1)$.

We want to transform the scheme in order to forbid the derivation of a subterm when its associated value tree will not include any $\mathtt{c}$. For instance, the occurrence of a $\mathtt{c}$ would correspond to a completed service, and thus such a situation witnesses a useless derivation. In the embedded scheme, this can be detected by applying the evaluation of the head over

its annotation. For instance $F^{[0,0]}t_1\ t_2$ is the annotation of a term $F\ t$ whose value is $\varphi(F)\ [0,0] = 0$. Therefore we might turn the rule associated to $F^{[0,0]}$ into $F^{[0,0]}y_1\ y_2 \to$ FORBIDDEN, where FORBIDDEN : o is a new terminal added to the scheme. Here is the whole set of rewrite rules transformed this way.

$$
\begin{array}{lcl}
S & \to & \mathtt{a}^{1,0}\ (F^{[0,1]}\ H^0\ H^1)\ (F^{[0,0]}\ J^0\ J^1) \\
H^0\ x & \to & \text{FORBIDDEN} \\
H^1\ x & \to & \mathtt{a}^{1,1}\ x\ (H^0\ x) \\
J^0\ x & \to & \text{FORBIDDEN} \\
J^1\ x & \to & \text{FORBIDDEN} \\
F^{[0,0]}\ y^0\ y^1 & \to & \text{FORBIDDEN} \\
F^{[0,1]}\ y^0\ y^1 & \to & \mathtt{a}^{0,1}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}) \\
F^{[1,0]}\ y^0\ y^1 & \to & \mathtt{a}^{1,0}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}) \\
F^{[1,1]}\ y^0\ y^1 & \to & \mathtt{a}^{1,1}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}).
\end{array}
$$

6 Logical reflection

In the following we present a morphism based on [17] that recognises the acceptance of a parity tree automaton. Using the construction introduced in Section 4.2, one can construct a scheme that reflects the accepting states of the automaton, which is equivalent to reflect the subtrees accepted by a formula of the μ-calculus. In [3], μ-calculus reflection (and MSO reflection) on schemes is already proven, but this construction uses the equivalence between schemes and collapsible pushdown automata, and the successive transformations (scheme $\to$ pushdown automaton $\to$ reflective pushdown automaton $\to$ reflective scheme) lose the structure of the scheme. In our construction, the structure of the scheme is the preserved, in the sense of Remark 4.2. Since our proof of the logical reflection, as well as the one of logical selection in Section 7, is build on top of the MSO model checking proof of Kobayashi and Ong [17], we first recall their construction and then we explain how to use this result to obtain the logical reflection.

6.1 Kobayashi-Ong result

We fix a non deterministic parity tree automaton $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, \Omega \rangle$ and a scheme $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{V}, S, \mathcal{R} \rangle$. We let $\text{arity}_{\max}$, $\text{order}_{\max}$, and $m_{\max}$ be the maximum arity in $\Sigma \uplus \mathcal{N}$, order in $\Sigma \uplus \mathcal{N}$, colour in $\Omega(Q)$. The idea of the result is to define a type system, and to use this type system in the construction of a two player parity game, such that Eve wins the game if and only if the automaton accepts the value tree of the scheme.

The type system. Kobayashi and Ong introduced a set of *judgement rules* that allow to type a term by an element of the typed set Map, called the set of *mappings*. Mappings of type o are the states Q, and mappings of type $\tau \to \tau'$ are of the form $(\theta_1, m_1) \wedge \ldots \wedge (\theta_k, m_k) \to \theta$ with for all i θ_i is a mapping of type τ, m_i is a color, and θ is a mapping of type τ'. The judgements are of the form $\Gamma \vdash t \triangleright \theta$ meaning that under the environment Γ, one can judge t with the mapping θ. The environment Γ associates some mapping and colors to non terminal and variables, and gives some restriction on the judgements one can make. Terminals are judged according to the transition of the automaton, *i.e.* $a : o^k \to o \in \Sigma$ may be judged as $\emptyset \vdash a \triangleright (q_1, m_1) \to \ldots \to (q_k, m_k) \to q$ with for all i, $m_i = \max(\Omega(q_i), \Omega(q))$ and $q \xrightarrow{a} q_1, \ldots, q_k \in \delta$. This type system keeps track of the colours in order to know exactly what colour has been seen along the term. It is given formally in the extended version.

The game. Now we define a game $\mathbb{G}_{\mathcal{A}}$ in which Eve's states will be triples made of a non terminal, mapping and a colour, and Adam's states will be environments. Eve chooses an environment that can judge the rewrite rule of the current nonterminal with the current atomic mapping, while Adam picks one binding in the current environment. Intuitively, Eve tries to show a well-typing of the terms with respect to the rewrite rules, that would induce a well-coloured run of the automaton, and Adam tries to show that she is wrong. From the state (F, θ, m), Eve has to find an environment Γ such that she can prove $\Gamma \vdash r_F : \theta$, then Adam picks a F and θ' in Γ and asks Eve to prove that θ' is chosen correctly according to the rewrite rule of F. If at some point of a play, Eve cannot find a correct environment, she loses the play; if she can choose the empty environment, Adam would have nothing to choose then she wins the play; if the play is infinite, Eve wins if and only if the greatest colour seen infinitely often is even. The game is also given formally in the extended version.

▶ **Theorem 5** (Kobayashi, Ong 09). *The tree $\|\mathcal{G}\|$ is accepted by $\mathcal{A}$ from the state q if and only if Eve has a winning strategy from the vertex $(S, q, \Omega(q))$ in the game $\mathbb{G}_{\mathcal{A}}$.*

6.2 A morphism for automata reflection

From the Eve's winning strategy, we define a morphism $[\![\cdot]\!] : \mathcal{T}(\Sigma \uplus \mathcal{N}) \to \mathcal{D}$: the domain $\mathcal{D}$ contains the sets of mappings of the same type: for all τ, $\mathcal{D}^\tau = 2^{\mathrm{Map}^\tau}$. Given a nonterminal F, $[\![F]\!] = \{\theta \mid \exists m \text{ } \textit{Eve wins from } (F, \theta, m)\}$, given $a \in \Sigma$, $[\![a]\!] = \{\theta \mid \emptyset \vdash a : \theta\}$, and given $d : \tau_1 \to \tau_2 \in \mathcal{D}$ and $d' : \tau_1 \in \mathcal{D}$ $d\ d' = \{\theta \mid \exists (\theta_1, m_1) \wedge \ldots \wedge (\theta_k, m_k) \to \theta \in d\ \forall i\ \theta_i \in d'\}$.

▶ **Theorem 6.** *The morphism $[\![\cdot]\!]$ recognises the states of the automaton, i.e. for each state $q \in Q$ of the automaton, it recognises the set $\mathcal{T}_q = \{t : o \mid \|\mathcal{G}_t\| \models \mathcal{A}_q\}$ which is the set of ground terms whose associated value tree is recognised by the automaton from state q. Furthermore, it is stable by rewriting.*

Using the construction of Section 4.2, we have the following result.

▶ **Corollary 7** (Automata Reflection). *Higher order recursion schemes are reflective with respect to automata (hence they are reflective with respect to μ-calculus, and to* MSO*).*

7 Selection

Given a signature Σ, we recall the selection problem: given an MSO formula $\varphi[X]$ having one free monadic second order variable X, and a scheme $\mathcal{G}$ such that $\|\mathcal{G}\|$ satisfies the formula $\exists X\ \varphi[X]$, produce another scheme $\mathcal{G}'$ on the signature $\Sigma \times \{0, 1\}$ such that their exists a set S satisfying $\varphi[S]$ and $\|\mathcal{G}'\|$ is a S-marking of $\|\mathcal{G}\|$. Note that MSO selection implies MSO reflection. Indeed, being able to mark the nodes u satisfying the formula $\varphi[x]$ is equivalent to being able to mark a (unique) set satisfying $\psi[X] = \forall x\ x \in X \Leftrightarrow \varphi[x]$.

As mentioned in Section 3, MSO selection is equivalent to automata selection; we show a construction of a scheme annotating itself with an accepting run of the automaton. Since we cannot embed this problem into a morphism, we define another construction, similar to the one of Section 4.2. The construction is also based on the Kobayashi-Ong result presented in Section 5. The main difference between reflexivity and selectivity is that to construct a reflection of an automaton we embedded the winning region of the game into the scheme, while here we will embed the winning strategy of the game to prove the selection.

▶ **Theorem 8** (Automata Selection). *Higher order recursion schemes are selective with respect to automata (hence to MSO).*

Proof sketch. In the following, we give an informal glimpse on the proof, the full (technical) proof can be found in the extended version. Take a scheme $\mathcal{G}$ and an automaton $\mathcal{A}$ such that $\|\mathcal{G}\| \models \mathcal{A}$. We want to construct a scheme $\mathcal{G}'$ such that $\|\mathcal{G}'\|$ is an accepting run of $\mathcal{A}$ on $\|\mathcal{G}\|$. The first observation is that there are some great similarities between the structure of the proof trees in the type system system and the definition of a term. Indeed, one can type proofs and one can apply proofs to one another to get new proofs. For example take two terms $t_0 : o \to o$ and $t_1 : o$ and assume that we have a proof $\mathcal{P}_0$ of $t_0 \triangleright (\theta_1, m_1) \to \theta$ and a proof $\mathcal{P}_1$ of $t_1 \triangleright \theta_1$ under some environments. Then one can put together $\mathcal{P}_0$ and $\mathcal{P}_1$ to obtain a proof of $\vdash t_0\ t_1 \triangleright \theta$. We can see this proof as $\mathcal{P}_0\ \mathcal{P}_1$: the application of $\mathcal{P}_1$ to $\mathcal{P}_0$.

In the actual construction, the transformed scheme will not deal with such proofs, but we use this similarity between proofs and terms to create annotations of terms. Given a term t and a proof $\mathcal{P}$ of a judgement $\Gamma \vdash t \triangleright \theta$, we will define the annotated term $t^{\mathcal{P}}$ where each symbol is annotated by an atomic mapping and a color, verifying that if a non terminal F is annotated by (θ, m) then Γ assiciate F to (θ, m). The term $t^{\mathcal{P}}$ is somehow a trace of the proof $\mathcal{P}$. Now given a rewrite rule $F\ x_1 ... x_k \to e$ in the original scheme, we want to define for all annotated versions $F^{(\theta,m)}$ an associated rewrite rule in the transformed scheme. If (F, θ, m) is a winning vertex of the game, then Eve can choose an environment Γ such that $\Gamma \vdash e \triangleright \theta$. We take a proof $\mathcal{P}$ of this judgement and we define $F^{(\theta,m)} x_1 \dots x_k \to e^{\mathcal{P}}$. Since Eve has chosen the environment Γ with respect to her winning strategy, then for all annotated non terminals $H^{(\theta',m')}$ appearing in $e^{\mathcal{P}}$, (H, θ', m') is winning in the game. In particular, if the initial non terminal of the transformed scheme is $S^{(\theta,m)}$ with (S, θ, m) winning in the game (as it will be), any non terminal in any term of any derivation will be annotated in order to represent a winning vertex in the game. Therefore we do not need to care about which rewrite rule is chosen for $F^{(\theta,m)}$ when (F, θ, m) is not winning.

As we said, the initial non terminal is $(S, q_0, \Omega(q_0))$ which is winning in the game since $\mathcal{A}$ accepts $\|\mathcal{G}\|$ from state q_0. Any terminal a will be annotated by some $(q_1 \to \dots \to q_k \to q, m)$. Then to obtain elements of $\Sigma \times Q$, we just turn any $a^{(q_1 \to \dots \to q_k \to q, m)}$ into a non terminal and we add a rewrite rule transforming it into a^q.

The intuition of why this construction works is the following, based on the proof of the soundness of Kobayashi and Ong construction. To a derivation in the transformed scheme we associate a tree of plays in the game. The terms will be labeled by some $F^{(\theta,m)}$ that Eve has chosen in the environments she picked, and each time Adam choose one such $F^{(\theta,m)}$, it is rewritten according to Eve's strategy. Due to the colour constraints in the type systems, we can show that from the point where $F^{(\theta,m)}$ is created to the point where it is on the head of a redex, the maximum colour that has been seen is m. Furthermore, we have that along an infinite branch, there is an infinite sequence of nonterminals that are rewritten such that each non terminal is created when the previous one is rewritten. This means that we can map an infinite branch to an infinite play in the game. Furthermore the greatest colour seen infinitely often along this branch is equal to the greatest colour seen infinitely often in the sequence of maximum colours appearing between the non terminals of the sequence. And this is equal to the greatest colour seen infinitely often in the play. Since Eve wins in the game, this colour is even, then for any branch of the value tree of $\mathcal{G}'$ the greatest colour seen infinitely often is even, hence it is an accepting run. ◀

8 Conclusion

We have given new shape preserving constructions for logical reflection and logical selection using a scheme-only approach, which can be useful for correction or synthesis of programs.

The complexity is the same as in the solutions proposed so far, *i.e.* the problem is n-EXPTIME complete, and the size of the new scheme is n-EXP the size of the original one n being the order of the scheme. As possible continuation of these work, we may be interesting to see if these results scale for actual program verification, and if they can be included in tools like T-RecS [16], a model-checker for HORS.

References

1 Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. In *CSL'06*, volume 4207 of *LNCS*, pages 104–118, 2006.
2 R. M. Amadio and P.-L. Curien. *Domains and Lambda-Calculi.* CTTCS, 1998.
3 Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *LICS'10*, pages 120–129, 2010.
4 Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes. In *LICS'12*, pages 165–174, 2012.
5 Didier Caucal. On infinite terms having a decidable monadic theory. In *MFCS'02*, volume 2420 of *LNCS*, pages 165–176, 2002.
6 Bruno Courcelle. A representation of trees by languages I. *TCS*, 6:255–279, 1978.
7 Bruno Courcelle. A representation of trees by languages II. *TCS*, 7:25–55, 1978.
8 Bruno Courcelle and Maurice Nivat. The algebraic semantics of recursive program schemes. In *MFCS'78*, volume 64 of *LNCS*, pages 16–30, 1978.
9 Werner Damm. Higher type program schemes and their tree languages. In *Theoretical Computer Science, 3rd GI-Conference*, volume 48 of *LNCS*, pages 51–72, 1977.
10 Werner Damm. Languages defined by higher type program schemes. In *ICALP'77*, volume 52 of *LNCS*, pages 164–179, 1977.
11 Axel Haddad. IO vs OI in higher-order recursion schemes. In *FICS'12*, pages 23–30, 2012.
12 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *LICS'08*, pages 452–461, 2008.
13 Klaus Indermark. Schemes with recursion on higher types. In *MFCS'76*, volume 45 of *LNCS*, pages 352–358, 1976.
14 Teodor Knapik, Damian Niwiński, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *FOSSACS'02*, volume 2303 of *LNCS*, pages 205–222, 2002.
15 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL'09*, pages 416–428, 2009.
16 Naoki Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In *FOSSACS'11*, pages 260–274, 2011.
17 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *LICS'09*, pages 179–188, 2009.
18 M. Nivat. On the interpretation of recursive program schemes. In *Symp. Mat.*, 1972.
19 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS'06*, pages 81–90, 2006.
20 M.O. Rabin. Decidability of second-order theories and automata on infinite trees. In *Trans. Amer. Math. Soc.*, 1969.
21 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. In *ICALP'11*, pages 162–173, 2011.
22 Sylvain Salvati and Igor Walukiewicz. Using models to model-check recursive schemes. In *Typed Lambda Calculi and Applications*, pages 189–204. Springer, 2013.

A Theory of Partitioned Global Address Spaces*

Georgel Calin[1], Egor Derevenetc[2], Rupak Majumdar[3], and Roland Meyer[1]

1 **University of Kaiserslautern, Germany,** {calin,meyer}@cs.uni-kl.de
2 **Fraunhofer ITWM, Germany,** egor.derevenetc@itwm.fraunhofer.de
3 **MPI-SWS, Germany,** rupak@mpi-sws.org

Abstract

Partitioned global address space (PGAS) is a parallel programming model for the development of high-performance applications on clusters. It provides a global address space partitioned among the cluster nodes, and is supported in programming languages like C, C++, and Fortran by means of APIs. Our first contribution is a formal model for the semantics of single program, multiple data programs that use PGAS APIs. Our model reflects the main features of popular real-world APIs such as SHMEM, ARMCI, GASNet, GPI, and GASPI.

A key feature of PGAS is the support for one-sided communication: a node may directly read and write the memory located at a remote node, without explicit synchronization with the processes running on the remote side. One-sided communication increases performance by decoupling process synchronization from data transfer, but requires the programmer to reason about appropriate synchronizations between reads and writes. As a second contribution, we propose and investigate *robustness*, a criterion for correct synchronization of PGAS programs. Robustness corresponds to acyclicity of a suitable happens-before relation defined on PGAS computations. The requirement is finer than classical data race freedom and rules out most false error reports.

Our main technical result is an algorithm for checking robustness of PGAS programs. The algorithm makes use of two insights. We first show that, if a PGAS program is not robust, then there are computations in a certain normal form that violate happens-before acyclicity. Intuitively, normal-form computations delay remote accesses in an ordered way. We then devise an algorithm that checks for cyclic normal-form computations. Essentially, the algorithm is an emptiness check for a novel automaton model that accepts normal-form computations in streaming fashion. Altogether, we prove that the robustness problem is PSPACE-complete.

1998 ACM Subject Classification D.2.4 Software/Program Verification, D.1.3 Concurrent Programming, F.4.3 Formal Languages

Keywords and phrases PGAS, SC preservation, Robustness, Semantics, Formal languages

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.127

1 Introduction

Partitioned global address space (PGAS) is a parallel programming model for the development of high-performance software on clusters. The PGAS model provides a global address space to the programmer that is partitioned among the cluster nodes (see Figure 1(b)). Nodes can read and write their local memories, but additionally access the remote address

* The second author was granted by the Competence Center High Performance Computing and Visualization (CC-HPC) of the Fraunhofer Institute for Industrial Mathematics (ITWM). The work was partially supported by the PROCOPE project ROIS: Robustness under Realistic Instruction Sets.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 127–139

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

```
int x = 1, y = 0;
write(x, rightNeighborRank,
      y, myWriteQ);
barrier();
assert(y == 1);
```

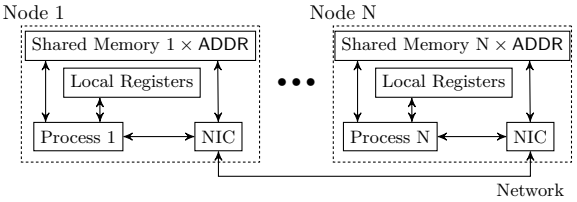


Figure 1 (a) Program **1to1** is the *compute and exchange results* idiom often found in PGAS applications. Each process copies an integer value to its neighbor. `write` asks the hardware to copy the value of address x to y on the right neighboring node. `barrier` blocks until all processes reach the barrier. The assertion can fail, as the `barrier` may return before the `write` completes. (b) PGAS architecture — NIC stands for network interface controller.

space through (synchronous or asynchronous) API calls. PGAS is a popular programming model, and supported by many PGAS APIs, such as SHMEM [10], ARMCI [21], GASNET [4], GPI [19], and GASPI [15], as well as by languages for high-performance computing, such as UPC [11], Titanium [16], and Co-Array Fortran [23].

A key ingredient of PGAS APIs is their support for one-sided communication: a node may directly read and write the memory located at a remote node without explicit synchronization with the remote side, unlike in traditional message passing interfaces. One-sided communication can be efficiently implemented on top of networking hardware featuring remote direct memory access (RDMA), and increases performance of PGAS programs by avoiding unnecessary synchronization between the sender and the receiver [19, 14].

However, the use of one-sided communication introduces additional non-determinism in the ordering of memory reads and writes, and makes reasoning about program correctness harder. Figure 1(a) demonstrates a subtle bug arising out of improper synchronizations: while the barriers ensure all processes are at the same control location, the remote writes may or may not have completed when address `y` is accessed after the barrier.

We make two contributions in this paper.

First, we provide a core calculus of PGAS APIs that models concurrent processes sharing a global address space and accessing remote memory through one-sided reads and writes. Despite the popularity of PGAS APIs in the high-performance computing community, to the best of our knowledge, there were no formal models for common PGAS APIs.

Second, we define and study a correctness criterion called *robustness* for PGAS programs. To understand robustness, we begin with a classical and intuitive correctness condition, *sequential consistency* [18]. A computation is sequentially consistent if its memory accesses happen atomically and in the order in which they were issued. Sequential consistency is too strong a criterion for PGAS programs, where time is required to access remote memory and accesses themselves can be reordered. Robustness is the weaker notion that all computations of the program have the same happens-before (data and control) dependencies [26] as some sequentially consistent computation. Our notion of robustness captures common programming error patterns [13, 20], and is derived from a similar notion in shared memory multiprocessing [26]. Related correctness criteria have been proposed for weak memory models [2, 3, 5, 6, 7, 8, 24].

A simpler correctness property would be *data race freedom* (DRF), in which no two processes access the same address at the same time, with at least one access being a write [1]. Indeed, programs free of data races are sequentially consistent. Unfortunately, DRF is too strong a requirement in practice [25], and leads to numerous false alarms. Many common synchronization idioms for PGAS programs, such as producer-consumer synchronization, and many concurrent data structure implementations, contain benign data races. Instead,

the notion of robustness captures the intuitive requirement that, even when events are reordered in a computation, there are no causality cycles. Our notion of causality is the standard *happens-before* relation from [26].

We study the algorithmic verification of robustness. Our main result is that robustness is decidable (actually PSPACE-complete) for PGAS programs, assuming a finite data domain and finite memory. Note that our model of PGAS programs is infinite-state even when the data domain is finite: one-sided communication allows unboundedly many requests to be in flight simultaneously (a feature modeled in our formalism using unbounded queues).

Our decidability result uses two technical ingredients. First, we show that among all computations violating robustness, there is always one in a certain normal form. The normal form partitions the violating computation into phases: the first phase initiates memory reads and writes, and the latter phases complete the reads and writes in the same order in which they were initiated.

Second, we provide an algorithm to detect violating computations in this normal form. We take a language-theoretic view, and introduce a multiheaded automaton model which can accept violating computations in normal form. Then the problem of checking robustness reduces to checking emptiness for multiheaded automata. Interestingly, since the normal form maintains orderings of accesses, the multiple heads can be exploited to accept violating computations without explicitly modeling unbounded queues of memory access requests. The resulting class of languages contains non-context-free ones (such as $a^n b^n c^n$), but retains sufficient decidability properties. Altogether this yields a PSPACE decision procedure for checking robustness of programs using PGAS APIs.

For lack of space, full constructions and proofs are given in [9].

Related Work. Although PGAS APIs are popular in the high-performance computing community [4, 10, 15, 19, 21], no previous work provides a unifying formal semantics that incorporates one-sided asynchronous communication. As for synchronization correctness, Park et al. proposed a testing framework for data race detection and implemented it for the UPC language [25]. However, these authors note that many data races are actually not harmful, and support the statement by the analysis of the NAS Parallel Benchmarks [22]. For this reason, in contrast to data race freedom [1], we consider robustness as a more precise notion of appropriate synchronization. Several examples from [25] show that harmful data races (like in the `knapsack` example) lead to non-robustness, while benign data races (like in the examples `NPB 3.3 BT` and `SP`) do not.

The robustness problem was posed by Shasha and Snir [26] for shared memory multiprocessing. They showed that non-sequentially consistent computations have a happens-before cycle. Alglave and Maranget [2, 3] extended this result. They developed a general theory for reasoning about robustness problems, even among different architectures. Owens [24] proposed a notion of appropriate synchronization that is based on triangular data races. Compared to robustness, triangular race freedom requires heavier synchronization, which is undesirable for performance reasons.

We consider here the algorithmic problem of checking robustness. For programs running on weak memory models the problem has been addressed in [3, 7, 8], but none of these works provides a (sound and complete) decision procedure. The first complete algorithm for checking robustness of programs running on Total Store Ordering (TSO) architectures was given in [6]. It is based on the following locality property. If a TSO program is not robust, then there is a violating computation where only one process delays commands. This insight leads to a reduction of robustness to reachability in the sequential consistency model [5]. PGAS programs allow more reorderings than TSO ones and, as a consequence, locality

does not hold. Instead, our decision procedure relies on a more complex normal form for computations and on an automata-theoretic algorithm to look for normal-form violations.

2 PGAS Programs

2.1 Features of PGAS Programs

PGAS programs are *single program, multiple data* programs running on a cluster (see Figure 1(b)). At run time, a PGAS program consists of multiple processes executing the same code on different nodes. Each process has a *rank*, which is the index of the node it runs on. The processes can access a global address space partitioned into local address spaces for each process. Local addresses can be accessed directly. Remote addresses (addresses belonging to different processes) are accessed using API calls, which come in different flavors.

SHMEM [10] provides synchronous remote reads where the invoking process waits for completion of the command. Remote write commands are asynchronous, and no ordering is guaranteed between writes, even to the same remote node. The ordering can, however, be enforced by a special fence command.

ARMCI [21] features synchronous as well as asynchronous read and write commands. The asynchronous variants of the commands return a handle that can be waited upon. When the wait on a read handle is over, the data being read has arrived and is accessible. When the wait on a write handle is over, the data being written has been sent to the network but might not have reached its destination. Unlike operations to different nodes, operations to the same remote node are executed in their issuing order.

GASNet [4], like ARMCI, provides both synchronous and asynchronous versions of reads and writes. Commands return a handle that can be waited upon, and a return from a wait implies full completion of the operation. The order in which asynchronous operations complete is intentionally left unspecified.

GPI [19] and GASPI [15] only support asynchronous read and write commands. Each read or write operation is assigned a queue identifier. In GPI, operations with the same queue id and to the same remote node are executed in the order in which they were issued; in GASPI this guarantee does not hold. One can wait on a queue id, and the wait returns when all commands in the queue are fully completed, on both the local and the remote side.

Summing up, in a uniform PGAS programming model it should be possible to

- perform synchronous and asynchronous data transfers,
- assign an asynchronous operation a handle or a queue id,
- wait for completion of an individual command or of all commands in a given queue,
- enforce ordering between operations.

We define a core model for PGAS that supports all these features. Our model only uses asynchronous remote reads and writes with explicit queues, but is flexible enough to accommodate all the above idioms. Moreover, it is not limited to single program, multiple data programs common in PGAS applications, but can model ordinary concurrent programs with different processes as well.

2.2 Syntax of PGAS Programs

We define PGAS programs and their semantics in terms of automata. A (non-deterministic) *automaton* is a tuple $A = (S, \Sigma, \Delta, s_0, F)$, where S is a set of states, Σ is a finite alphabet, $\Delta \subseteq S \times (\Sigma \cup \{\varepsilon\}) \times S$ is a set of transitions, $s_0 \in S$ is an initial state, and $F \subseteq S$ is a set of final states. We call the automaton *finite* if the set of states is finite. We write $s_1 \xrightarrow{a} s_2$

$\langle cmd \rangle ::= \langle reg \rangle \leftarrow \texttt{mem[}\langle expr \rangle\texttt{]}$
| $\texttt{mem[}\langle expr \rangle\texttt{]} \leftarrow \langle expr \rangle$
| $\langle reg \rangle \leftarrow \langle expr \rangle$
| $\texttt{assume(}\langle expr \rangle\texttt{)}$
| $\texttt{read(}\langle local\text{-}addr \rangle\texttt{,}\langle rank \rangle\texttt{,}\langle remote\text{-}addr \rangle\texttt{,}\langle que\text{-}id \rangle\texttt{)}$
| $\texttt{write(}\langle local\text{-}addr \rangle\texttt{,}\langle rank \rangle\texttt{,}\langle remote\text{-}addr \rangle\texttt{,}\langle que\text{-}id \rangle\texttt{)}$
| `barrier`

Figure 2 Syntax of commands. $\langle reg \rangle$ ranges over REG; expressions $\langle expr \rangle$, local addresses $\langle local\text{-}addr \rangle$, remote addresses $\langle remote\text{-}addr \rangle$, and queue identifiers $\langle que\text{-}id \rangle$ range over expressions; ranks $\langle rank \rangle$ over $\overline{1,N}$-valued expressions.

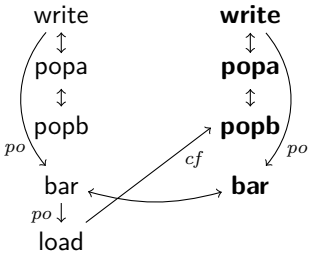


Figure 3 Happens-before relation of computation $\tau_{\texttt{1to1}}$ (Example 1). $\tau_{\texttt{1to1}}$ violates robustness.

if $(s_1, a, s_2) \in \Delta$, and extend the relation to computations $\sigma \in \Sigma^*$ in the expected way. The *language* of the automaton is $\mathcal{L}(A) := \{\sigma \in \Sigma^* \mid s_0 \xrightarrow{\sigma} s \text{ for some } s \in F\}$. We write $|\sigma|$ for the length of a computation $\sigma \in \Sigma^*$, and use $\mathsf{succ}(\sigma)$ to denote the successor relation among the letters in σ. We write $a <_\sigma b$ if $\sigma = \sigma_1 \cdot a \cdot \sigma_2 \cdot b \cdot \sigma_3$ for some $\sigma_1, \sigma_2, \sigma_3 \in \Sigma^*$.

A *PGAS program* $(\mathcal{P}, N)$ consists of a program code $\mathcal{P}$ and a fixed number $N \geq 1$ of cluster nodes. The *program code* $\mathcal{P} := (Q, \mathsf{CMD}, \mathcal{I}, q_0, Q)$ is a finite automaton with a set of control states Q, all of them final, an initial state q_0, and a set of transitions $\mathcal{I}$ labeled with *commands* CMD defined as follows.

Let DOM, ADDR, and QUE be finite domains of values (containing a value 0), addresses, and queue identifiers, respectively. Let REG be a finite set registers that take values from DOM. The grammar of commands is given in Figure 2. For simplicity, we will assume $\mathsf{DOM} = \mathsf{ADDR} = \mathsf{QUE}$. The set of expressions is defined over constants from DOM, registers from REG, and (unspecified) operators over DOM. The set of commands CMD includes local assignments and conditionals (`assume`), remote read and write API calls `read` and `write` respectively, and barriers `barrier`.

At run time, there is a process on each node $\overline{1,N}$ that executes program $\mathcal{P}$, where $\overline{M,N} := \{M, M+1, \ldots, N\}$. We will identify each process with its rank from $\mathsf{RNK} := \overline{1,N}$. For modeling purposes, one may assume there are special constant expressions that let a process learn about its rank in RNK and about the total number of processes N.

2.3 Semantics of PGAS Programs

The semantics of a PGAS program $(\mathcal{P}, N)$ is defined using a *state-space automaton* $X(\mathcal{P}, N) := (S_X, \mathsf{E}, \Delta_X, s_{0X}, F_X)$. A state $s \in S_X$ is a tuple $s = (\mathsf{st}, \mathsf{m}, \mathsf{fa}, \mathsf{fb})$, where state configuration $\mathsf{st}\colon \mathsf{RNK} \to Q$ maps each process to its current control state, memory configuration $\mathsf{m}\colon \mathsf{RNK} \times (\mathsf{REG} \cup \mathsf{ADDR}) \to \mathsf{DOM}$ maps each process to the values stored in each register and at each address, queue configuration $\mathsf{fa}\colon \mathsf{RNK} \times \mathsf{QUE} \to (\mathsf{RNK} \times \mathsf{ADDR} \times \mathsf{RNK} \times \mathsf{ADDR})^*$ maps each process to remote read and write requests that were issued, and $\mathsf{fb}\colon \mathsf{RNK} \times \mathsf{QUE} \to (\mathsf{RNK} \times \mathsf{ADDR} \times \mathsf{DOM})^*$ contains values to be transferred. The two queue configurations capture the delays between creating a request, reading data, and writing data.

The initial state is $s_{0X} := (\mathsf{st}_0, \mathsf{m}_0, \mathsf{fa}_0, \mathsf{fb}_0)$, where for all ranks $\mathsf{r} \in \mathsf{RNK}$, registers and addresses $\mathsf{a} \in \mathsf{REG} \cup \mathsf{ADDR}$, and queue identifiers $\mathsf{q} \in \mathsf{QUE}$, we have $\mathsf{st}_0(\mathsf{r}) := q_0$, $\mathsf{m}_0(\mathsf{r}, \mathsf{a}) := 0$, and $\mathsf{fa}_0(\mathsf{r}, \mathsf{q}) := \varepsilon =: \mathsf{fb}_0(\mathsf{r}, \mathsf{q})$. The set of final states is $F_X := \{(\mathsf{st}, \mathsf{m}, \mathsf{fa}, \mathsf{fb}) \in S_X \mid \mathsf{fa}(\mathsf{r}, \mathsf{q}) = \varepsilon = \mathsf{fb}(\mathsf{r}, \mathsf{q}) \text{ for all } \mathsf{r} \in \mathsf{RNK}, \mathsf{q} \in \mathsf{QUE}\}$. The semantics of commands ensures queues can always be emptied, so acceptance with empty queues is not a restriction.

The alphabet of $X(\mathcal{P}, N)$ is the set of *events* $\mathsf{E} := \mathsf{K} \times \mathsf{RNK} \times ((\mathsf{RNK} \times \mathsf{ADDR}) \cup \{\bot\})$ with *event kinds* $\mathsf{K} := \{\mathsf{load}, \mathsf{store}, \mathsf{assign}, \mathsf{assume}, \mathsf{read}, \mathsf{write}, \mathsf{popa}, \mathsf{popb}, \mathsf{bar}\}$. Consider an event

Table 1 Transition rules for $X(\mathcal{P}, N)$, given $q_1 \xrightarrow{\mathsf{cmd}} q_2$ and current state $s = (\mathsf{st}, \mathsf{m}, \mathsf{fa}, \mathsf{fb})$ with $\mathsf{st}(\mathsf{r}) = q_1$. We set $\mathsf{st}' := \mathsf{st}[\mathsf{r} := q_2]$ to update st so that process r is at q_2. $\widehat{e}$ denotes the evaluation of expression e in memory configuration m.

$$\frac{\mathsf{cmd} = r \leftarrow \mathtt{mem}[e_\mathsf{a}]}{s \xrightarrow{(\mathsf{load},\mathsf{r},(\mathsf{r},\widehat{e_\mathsf{a}}))} (\mathsf{st}', \mathsf{m}[(\mathsf{r}, r) := \mathsf{m}(\mathsf{r}, \widehat{e_\mathsf{a}})], \mathsf{fa}, \mathsf{fb})} \quad (\mathsf{load})$$

$$\frac{\mathsf{cmd} = \mathtt{write}(e_\mathsf{a}^{\mathsf{loc}}, e_\mathsf{r}^{\mathsf{rem}}, e_\mathsf{a}^{\mathsf{rem}}, e_\mathsf{q}) \qquad \mathsf{fa}(\mathsf{r}, \widehat{e_\mathsf{q}}) = \alpha}{s \xrightarrow{(\mathsf{write},\mathsf{r},\bot)} (\mathsf{st}', \mathsf{m}, \mathsf{fa}[(\mathsf{r}, \widehat{e_\mathsf{q}}) := \alpha \cdot (\mathsf{r}, \widehat{e_\mathsf{a}^{\mathsf{loc}}}, \widehat{e_\mathsf{r}^{\mathsf{rem}}}, \widehat{e_\mathsf{a}^{\mathsf{rem}}})], \mathsf{fb})} \quad (\mathsf{write})$$

$$\frac{\mathsf{fa}(\mathsf{r},\mathsf{q}) = (\mathsf{r}_\mathsf{s}, \mathsf{a}_\mathsf{s}, \mathsf{r}_\mathsf{d}, \mathsf{a}_\mathsf{d}) \cdot \alpha \qquad \mathsf{fb}(\mathsf{r},\mathsf{q}) = \beta}{s \xrightarrow{(\mathsf{popa},\mathsf{r},(\mathsf{r}_\mathsf{s},\mathsf{a}_\mathsf{s}))} (\mathsf{st}, \mathsf{m}, \mathsf{fa}[(\mathsf{r},\mathsf{q}) := \alpha], \mathsf{fb}[(\mathsf{r},\mathsf{q}) := \beta \cdot (\mathsf{r}_\mathsf{d}, \mathsf{a}_\mathsf{d}, \mathsf{m}(\mathsf{r}_\mathsf{s}, \mathsf{a}_\mathsf{s}))])} \quad (\mathsf{popa})$$

$$\frac{\mathsf{fb}(\mathsf{r},\mathsf{q}) = (\mathsf{r}_\mathsf{d}, \mathsf{a}_\mathsf{d}, v) \cdot \beta}{s \xrightarrow{(\mathsf{popb},\mathsf{r},(\mathsf{r}_\mathsf{d},\mathsf{a}_\mathsf{d}))} (\mathsf{st}, \mathsf{m}[(\mathsf{r}_\mathsf{d}, \mathsf{a}_\mathsf{d}) := v], \mathsf{fa}, \mathsf{fb}[(\mathsf{r},\mathsf{q}) := \beta])} \quad (\mathsf{popb})$$

$$\frac{\mathsf{st}(\mathsf{r}) \xrightarrow{\mathtt{barrier}} \mathsf{st}'(\mathsf{r}) \text{ for each } \mathsf{r} \in \mathsf{RNK}}{s \xrightarrow{(\mathsf{bar},1,\bot)\cdot(\mathsf{bar},2,\bot)\cdots(\mathsf{bar},N,\bot)} (\mathsf{st}', \mathsf{m}, \mathsf{fa}, \mathsf{fb})} \quad (\mathsf{bar})$$

$\mathsf{e} = (\mathsf{k}, \mathsf{r}, (\mathsf{r}_\mathsf{a}, \mathsf{a})) \in \mathsf{E}$. We use $\mathsf{kind}(\mathsf{e}) = \mathsf{k}$ to determine the kind of the event, $\mathsf{rank}(\mathsf{e}) = \mathsf{r}$ for the rank of the process that produced the event, and $\mathsf{addr}(\mathsf{e}) = (\mathsf{r}_\mathsf{a}, \mathsf{a})$ to obtain the rank and the address that are *accessed* by the event. If $\mathsf{kind}(\mathsf{e}) \in \{\mathsf{load}, \mathsf{popa}\}$, then e is said to be a *read of* (r_a, a). If $\mathsf{kind}(\mathsf{e}) \in \{\mathsf{store}, \mathsf{popb}\}$, then e is a *write of* address $\mathsf{addr}(\mathsf{e})$.

Table 1 shows a subset of the transition relation Δ_X; the remaining rules are similar. When a process executes a remote write command, Rule (write), a new item is added to a queue in fa. This item contains the source rank and source address from which the data will be copied, together with the destination rank and destination address to which the data will be copied. Eventually, the item is popped from the queue in fa, Rule (popa), the value is read from the source address, and a new item is pushed into the corresponding queue in fb. The new item contains the destination rank and destination address, and the value that was read from the source address. Eventually, this item is popped from the queue, Rule (popb), and the value is written to the destination address in the destination rank. Modeling two queue configurations yields a symmetry between remote writes and reads: a read can be interpreted as a write that comes upon request.

The semantics of a PGAS program $\mathsf{C}(\mathcal{P}, N) := \mathcal{L}(X(\mathcal{P}, N)) \subseteq \mathsf{E}^*$ is the set of computations of the state-space automaton.

▶ **Example 1.** Consider PGAS program $(\mathbf{1to1}, 2)$ with the program code from Figure 1(a) being run on two nodes. It has the following computation:

$$\tau_{\mathbf{1to1}} = \mathsf{write} \cdot \mathbf{write} \cdot \mathbf{popa} \cdot \mathsf{popa} \cdot \mathsf{bar} \cdot \mathbf{bar} \cdot \mathsf{load} \cdot \mathbf{popb} \cdot \mathsf{popb}.$$

Bold events belong to the process with rank 2, the other events — to the process with rank 1. We have $\mathsf{addr}(\mathsf{popa}) = (1, x)$, $\mathsf{addr}(\mathsf{popb}) = (2, y)$. Symmetrically, $\mathsf{addr}(\mathbf{popa}) = (2, x)$ and $\mathsf{addr}(\mathbf{popb}) = (1, y)$. The `assert` in Figure 1 is a shortcut for a combination of load and assume, and in this computation $\mathsf{addr}(\mathsf{load}) = (1, y)$.

2.4 Simulating PGAS APIs

Our formalism natively supports asynchronous data transfers and queues. Operations in the same queue are completed in the order in which they were issued. Using this, we can model

the ordering guarantees given by ARMCI and GPI – by putting ordered operations into the same queue.

To model waiting on individual operations (waiting on a handle), we associate a shadow memory address with each operation. Before issuing the operation, the value at this address is set to 0. When the operation has been issued, the process sends to the same queue a read request which overwrites the value at the shadow address to 1. Now waiting on the individual operation can be implemented by polling on the shadow address associated with the operation. Waiting on all operations in a given queue is done similarly. Synchronous data transfers are modeled by asynchronous transfers, immediately followed by a wait.

3 Robustness: A Notion of Appropriate Synchronization

We now define *robustness*, a correctness condition for PGAS programs. Robustness is a weaker criterion than requiring all computations to be sequentially consistent [18]: it allows for reordering of events as long as there are no causality cycles. As causality relation, we adopt the *happens-before relation* [26]. Fix a computation $\tau \in \mathsf{C}(\mathcal{P}, N)$. Its happens-before relation is the union of the three relations we define next, $\rightarrow_{hb}(\tau) := \rightarrow_{po} \cup \rightarrow_{cf} \cup \leftrightarrow$.

The *program order relation* $\rightarrow_{po}$ is the union of the program order relations for all processes: $\rightarrow_{po} := \bigcup_{\mathsf{r} \in \mathsf{RNK}} \rightarrow^{\mathsf{r}}_{po}$. Relation $\rightarrow^{\mathsf{r}}_{po}$ gives the order in which events were issued in process r. Formally, let τ' be the subsequence of all events e in τ such that $\mathsf{rank}(\mathsf{e}) = \mathsf{r}$ and $\mathsf{kind}(\mathsf{e}) \notin \{\mathsf{popa}, \mathsf{popb}\}$. Then $\rightarrow^{\mathsf{r}}_{po} := \mathsf{succ}(\tau')$.

The *conflict relation* $\rightarrow_{cf}$ orders conflicting accesses to the same address. Let $\tau = \alpha \cdot \mathsf{e}_1 \cdot \beta \cdot \mathsf{e}_2 \cdot \gamma$, where e_1 and e_2 access the same address, and at least one of them is a write: $\mathsf{addr}(\mathsf{e}_1) = \mathsf{addr}(\mathsf{e}_2) = (\mathsf{r}, \mathsf{a})$, $\mathsf{kind}(\mathsf{e}_1) \in \{\mathsf{store}, \mathsf{popb}\}$ or $\mathsf{kind}(\mathsf{e}_2) \in \{\mathsf{store}, \mathsf{popb}\}$. If there is no $\mathsf{e} \in \beta$ such that $\mathsf{addr}(\mathsf{e}) = (\mathsf{r}, \mathsf{a})$ and $\mathsf{kind}(\mathsf{e}) \in \{\mathsf{store}, \mathsf{popb}\}$, then $\mathsf{e}_1 \rightarrow_{cf} \mathsf{e}_2$.

The *identity relation* $\leftrightarrow$ identifies events corresponding to the same command. Let e be a remote read or write event, $\mathsf{kind}(\mathsf{e}) \in \{\mathsf{read}, \mathsf{write}\}$, and e_1 and e_2 be the corresponding requests, $\mathsf{kind}(\mathsf{e}_1) = \mathsf{popa}$ and $\mathsf{kind}(\mathsf{e}_2) = \mathsf{popb}$. Then we have $\mathsf{e} \leftrightarrow \mathsf{e}_1 \leftrightarrow \mathsf{e}_2$. In a similar way, $\leftrightarrow$ identifies matching barrier events in different processes.

We say a computation τ is *violating* if the associated happens-before relation contains a non-trivial cycle, i.e., a cycle that is not included in $\leftrightarrow$. Violating computations violate sequential consistency. The robustness problem amounts to proving the absence of violations:

ROB Given a program $(\mathcal{P}, N)$, show that no computation $\tau \in \mathsf{C}(\mathcal{P}, N)$ is violating.

▶ **Example 2.** The happens-before relation of computation $\tau_{\mathbf{1to1}}$ is depicted in Figure 3. It is cyclic, therefore $\tau_{\mathbf{1to1}}$ is violating and $(\mathbf{1to1}, 2)$ is not robust. Indeed, no sequentially consistent execution of **1to1** allows the `assert` statements to load the initial value of y.

Our main result is the following.

▶ **Theorem 3.** **ROB** *is* PSPACE*-complete.*

The PSPACE lower bound follows from PSPACE-hardness of control state reachability in sequentially consistent programs [17]. To reduce to robustness, we add an artificial happens-before cycle starting in the target control state. The rest of the paper shows a PSPACE algorithm, and hence upper bound, for the problem.

4 Normal-Form Violations

We show that a PGAS program is not robust if and only if it has a violating computation of the following normal form.

▶ **Definition 4.** Computation $\tau = \tau_1 \cdot \tau_2 \cdot \tau_3 \cdot \tau_4 \in \mathsf{C}(\mathcal{P}, N)$ is *in normal form* if all $\mathsf{e} \in \tau_2 \cdot \tau_3 \cdot \tau_4$ satisfy $\mathsf{kind}(\mathsf{e}) \in \{\mathsf{popa}, \mathsf{popb}\}$ and for all $a, b \in \tau_1$ with $\mathsf{kind}(a), \mathsf{kind}(b) \notin \{\mathsf{popa}, \mathsf{popb}\}$ and all $a', b' \in \tau_i$ with $i \in \overline{1,4}$ we have:

$$a <_{\tau_1} b,\ a \not\leftrightarrow^* b,\ a \leftrightarrow^* a',\ b \leftrightarrow^* b' \quad \text{implies} \quad a' <_{\tau_i} b'. \tag{NF}$$

We explain the normal-form requirement (NF). Consider two accesses a and b to remote processes that can be found in the first part of the computation τ_1. Assume corresponding pop events a' and b' are delayed and can both be found in a later part of the computation, say τ_2. Then the ordering of a' and b' in τ_2 coincides with the order of a and b in τ_1. Computation $\tau_{\mathbf{1to1}}$ is not in normal-form whereas $\tau^{nf}_{\mathbf{1to1}}$ in Figure 4 is. The following theorem guarantees that, in case of non-robustness, normal-form violations always exist.

▶ **Theorem 5.** *A PGAS program $(\mathcal{P}, N)$ is robust iff it has no normal-form violation.*

Phrased differently, to decide robustness our procedure should look for normal-form violations. The remainder of the section is devoted to proving Theorem 5. We make use of the following property of PGAS programs: every computation contains an event that can be deleted, in the sense that the result is again a computation of the program, i.e., in $\mathsf{C}(\mathcal{P}, N)$.

▶ **Lemma 6** (Cancellation)**.** *Consider a computation $\varepsilon \neq \tau \in \mathsf{C}(\mathcal{P}, N)$ and let e be the last event in τ with $\mathsf{kind}(\mathsf{e}) \notin \{\mathsf{popa}, \mathsf{popb}\}$. Then $\tau \setminus \mathsf{e} \in \mathsf{C}(\mathcal{P}, N)$, where computation $\tau \setminus \mathsf{e}$ is defined to remove e and all $\leftrightarrow$-related events from τ.*

Proof. All events to the right of e are unconditionally executable. Moreover, τ does not have $\rightarrow_{po}$-successors following e. Therefore, the resulting computation $\tau \setminus \mathsf{e}$ is in $\mathsf{C}(\mathcal{P}, N)$. ◀

A PGAS program is not robust if and only if it has a violating computation τ of minimal length. Let $\mathsf{e} \in \tau$ be the event determined by Lemma 6. If $\mathsf{kind}(\mathsf{e}) \notin \{\mathsf{read}, \mathsf{write}\}$, then $\tau = \tau_1 \cdot \mathsf{e} \cdot \tau_2$. Otherwise $\tau = \tau_1 \cdot \mathsf{e} \cdot \tau_2 \cdot \mathsf{e}' \cdot \tau_3 \cdot \mathsf{e}'' \cdot \tau_4$ with $\mathsf{e} \leftrightarrow \mathsf{e}' \leftrightarrow \mathsf{e}''$. Consider the latter case where $\tau \setminus \mathsf{e} = \tau_1 \cdot \tau_2 \cdot \tau_3 \cdot \tau_4$. Since $|\tau \setminus \mathsf{e}| < |\tau|$, the new computation is not violating and $\rightarrow_{hb} (\tau \setminus \mathsf{e})$ is acyclic. This acyclicity guarantees that we find a computation $\sigma \in \mathsf{E}^*$ with the same happens-before relation as $\tau \setminus \mathsf{e}$ and where pop events directly follow their remote accesses. Intuitively, σ is a sequentially consistent computation corresponding to $\tau \setminus \mathsf{e}$.

▶ **Lemma 7** ([26])**.** *There is $\sigma \in \mathsf{C}(\mathcal{P}, N)$ with $\rightarrow_{hb} (\sigma) = \rightarrow_{hb} (\tau \setminus \mathsf{e})$ and $\sigma = \sigma_1 \cdot \mathsf{e}_1 \ldots \mathsf{e}_n \cdot \sigma_2$ for all $\mathsf{e}_1 \leftrightarrow \ldots \leftrightarrow \mathsf{e}_n$.*

We now use σ to rearrange the events in $\tau \setminus \mathsf{e}$ and guarantee the normal-form requirement. The idea is to project σ to the events in τ_1 to τ_4. Reinserting e yields a normal-form violation:

$$\tau^{nf} := (\sigma \downarrow \tau_1) \cdot \mathsf{e} \cdot (\sigma \downarrow \tau_2) \cdot \mathsf{e}' \cdot (\sigma \downarrow \tau_3) \cdot \mathsf{e}'' \cdot (\sigma \downarrow \tau_4).$$

The following lemma concludes the proof of Theorem 5.

▶ **Lemma 8** (Reinsertion)**.** *$\tau^{nf} \in \mathsf{C}(\mathcal{P}, N)$, $\rightarrow_{hb} (\tau^{nf}) = \rightarrow_{hb} (\tau)$, and τ^{nf} is in normal form.*

▶ **Example 9.** Computation $\tau_{\mathbf{1to1}}$ in Example 1 is a shortest violation. The event determined by Lemma 6 is $\mathsf{e} = \mathsf{load}$. Therefore, $\tau \setminus \mathsf{e} = \tau_1 \cdot \tau_2$ with

$$\tau_1 = \mathsf{write} \cdot \mathbf{write} \cdot \mathbf{popa} \cdot \mathsf{popa} \cdot \mathsf{bar} \cdot \mathbf{bar} \qquad \text{and} \qquad \tau_2 = \mathbf{popb} \cdot \mathsf{popb}.$$

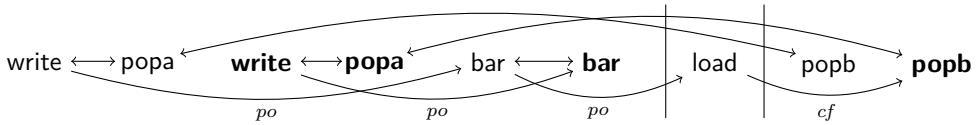


Figure 4 Normal-form violation $\tau^{nf}_{\mathbf{1to1}}$ from Example 9. The edges indicate the dependencies in the computation and coincide with the relations in Figure 3.

A sequentially consistent computation corresponding to $\tau \setminus \mathsf{e}$ is

$$\sigma = \mathsf{write} \cdot \mathsf{popa} \cdot \mathsf{popb} \cdot \mathbf{write} \cdot \mathbf{popa} \cdot \mathbf{popb} \cdot \mathsf{bar} \cdot \mathbf{bar}.$$

The normal-form violation $\tau^{nf}_{\mathbf{1to1}}$ is depicted in Figure 4. Note that $\tau^{nf}_{\mathbf{1to1}}$ is indeed in $\mathsf{C}(\mathbf{1to1}, 2)$. Moreover, popa and **popa** immediately follow write and **write**, respectively. Similarly, the popb and **popb** events in the second part of the computation respect the order of write and **write** in the first part of the computation. This means the property (NF) holds.

5 From Normal-Form Violations to Language Emptiness

We now reduce checking the absence of normal-form violations to the emptiness problem in a suitable automaton model. We introduce multiheaded automata and construct, for each program $(\mathcal{P}, N)$, a multiheaded automaton accepting all normal-form computations. To verify robustness, we check that the intersection of this automaton with regular languages accepting cyclic happens-before relations is empty.

5.1 Multiheaded Automata

Multiheaded automata are an extension of finite automata. Intuitively, instead of generating just a single computation, they generate several computations in one pass, each by a separate head. The language of the multiheaded automaton then consists of the concatenations of the computations generated by each head.

Syntactically, an *n-headed finite automaton over* Σ is a finite automaton that uses the extended alphabet $\overline{1,n}\times\Sigma$. So we have $A = (S, (\overline{1,n}\times\Sigma), \Delta, s_0, F)$. The semantics, however, is different from finite automata. Given $\sigma \in (\overline{1,n}\times\Sigma)^*$, we use $\sigma{\downarrow}k$ to project σ to the letters (k, a), and afterwards cut away the index k. So $((1,a)\cdot(2,b)\cdot(1,c)){\downarrow}1 = a\cdot c$. The *language of* A is $\mathcal{L}(A) := \{\mathsf{comp}(\sigma) \mid s_0 \xrightarrow{\sigma} s \text{ for some } s \in F\}$ where $\mathsf{comp}(\sigma) := \sigma{\downarrow}1 \cdots \sigma{\downarrow}n$.

Multiheaded automata are closed under regular intersection, and emptiness is decidable in non-deterministic logarithmic space. Indeed, checking emptiness reduces to finding a path from an initial to a final node in a directed graph.

▶ **Lemma 10.** *Consider an n-headed automaton U and a finite automaton V over a common alphabet Σ. There is an n-headed automaton W with $\mathcal{L}(W) = \mathcal{L}(U) \cap \mathcal{L}(V)$.*

▶ **Lemma 11.** *Emptiness for n-headed automata is* NL*-complete.*

Multiheaded automata are incomparable with context-free grammars, and indeed the normal-form computations of a program may be non-context-free.[1]

[1] Consider $\mathcal{P} := (\{q_0\}, \mathsf{CMD}, \{q_0 \xrightarrow{\mathtt{read}(0,0,0,0)} q_0\}, \{q_0\})$ running on a single node. The language $\mathsf{C}(\mathcal{P}, 1)$ is not context-free. To see this, let $\mathsf{kind}(a) = \mathsf{read}$, $\mathsf{kind}(b) = \mathsf{popa}$, and $\mathsf{kind}(c) = \mathsf{popb}$. Then $\mathsf{C}(\mathcal{P}, 1) \cap a^*b^*c^*$ is the non-context-free language $\{a^p b^p c^p \mid p \geq 0\}$.

Table 2 Transition rules for $Y(\mathcal{P}, N)$, given $q_1 \xrightarrow{\mathsf{cmd}} q_2$ and current state $s = (\mathsf{st}, \mathsf{m}, \mathsf{pa}, \mathsf{pb})$ with $\mathsf{st}(\mathsf{r}) = q_1$. The target is $s' = (\mathsf{st}', \mathsf{m}', \mathsf{pa}', \mathsf{pb}')$ where, unless otherwise stated, $\mathsf{st}' = \mathsf{st}$, $\mathsf{m}' = \mathsf{m}$, $\mathsf{pa}' = \mathsf{pa}$, $\mathsf{pb}' = \mathsf{pb}$. The auxiliary states $s_{aux1}, s_{aux2} \in S_Y^{\mathsf{aux}}$ are unique for each rule application.

$$(\mathsf{gpa}') \ \frac{\mathsf{pa}(\mathsf{r},\mathsf{q}) < \mathsf{pb}(\mathsf{r},\mathsf{q})}{s \xrightarrow{\varepsilon} s' \ \mathsf{pa}' := \mathsf{pa}[(\mathsf{r},\mathsf{q}) := \mathsf{pa}(\mathsf{r},\mathsf{q}) + 1]} \qquad \frac{\mathsf{pb}(\mathsf{r},\mathsf{q}) < 4}{s \xrightarrow{\varepsilon} s' \ \mathsf{pb}' := \mathsf{pb}[(\mathsf{r},\mathsf{q}) := \mathsf{pb}(\mathsf{r},\mathsf{q}) + 1]} \ (\mathsf{gpb}')$$

$$\frac{\mathsf{cmd} = \mathtt{write}(e_{\mathsf{a}}^{\mathsf{loc}}, e_{\mathsf{r}}^{\mathsf{rem}}, e_{\mathsf{a}}^{\mathsf{rem}}, e_{\mathsf{q}}) \qquad \mathsf{pa}(\mathsf{r}, \widehat{e_{\mathsf{q}}}) = m \qquad \mathsf{pb}(\mathsf{r}, \widehat{e_{\mathsf{q}}}) = n}{s \xrightarrow{1,(\mathsf{write},\mathsf{r},\perp)} s_{aux1} \xrightarrow{m,(\mathsf{popa},\mathsf{r},(\mathsf{r},\widehat{e_{\mathsf{a}}^{\mathsf{loc}}}))} s_{aux2} \xrightarrow{n,(\mathsf{popb},\mathsf{r},(\widehat{e_{\mathsf{r}}^{\mathsf{rem}}},\widehat{e_{\mathsf{a}}^{\mathsf{rem}}}))} s' \quad \mathsf{st}' := \mathsf{st}[\mathsf{r} := q_2]} \ (\mathsf{write}')$$
$$\text{if } n = 1 \text{ then } \mathsf{m}' := \mathsf{m}[(\widehat{e_{\mathsf{r}}^{\mathsf{rem}}}, \widehat{e_{\mathsf{a}}^{\mathsf{rem}}}) := \mathsf{m}(\mathsf{r}, \widehat{e_{\mathsf{a}}^{\mathsf{loc}}})]$$

Multiheaded automata can be understood as a restriction of matrix grammars [12]. In matrix grammars, productions simultaneously rewrite multiple non-terminals. Roughly, each production can be understood as a Petri net transition, and emptiness is decidable as Petri net reachability is. Since we target a PSPACE result, matrix grammars are too expressive for our purposes.

5.2 Detecting Normal-Form Computations

We define a 4-headed automaton $Y(\mathcal{P}, N) := (S_Y \uplus S_Y^{\mathsf{aux}}, \mathsf{E}, \Delta_Y, s_{0Y}, S_Y)$ that accepts all normal-form computations $\tau = \tau_1 \cdot \tau_2 \cdot \tau_3 \cdot \tau_4 \in \mathsf{C}(\mathcal{P}, N)$. In order to accept τ_1, the new automaton tracks the control and memory configurations in the way $X(\mathcal{P}, N)$ does. For the remainder of the computation, these configurations are not needed. Indeed, τ_2 to τ_4 only consist of popa and popb events that are executable independently of the control and memory configurations. However, $Y(\mathcal{P}, N)$ has to take care of the ordering of popa and popb events from the same queue. In particular, if e_1 handles a request issued before the request of e_2 with $\mathsf{kind}(\mathsf{e}_1) = \mathsf{kind}(\mathsf{e}_2)$, then it cannot be the case that $\mathsf{e}_1 \in \tau_j$ and $\mathsf{e}_2 \in \tau_i$ with $i < j$.

Guided by this discussion, we define a state $s \in S_Y$ as a tuple $s := (\mathsf{st}, \mathsf{m}, \mathsf{pa}, \mathsf{pb})$. The state and memory configurations st and m are defined as in Section 2. They reflect the state of the program after it has generated a prefix of τ_1. The functions $\mathsf{pa}, \mathsf{pb} : \mathsf{RNK} \times \mathsf{QUE} \to \overline{1,4}$ give, for each process and each queue, the part τ_1 to τ_4 of the computation where the next popa resp. popb event will be generated. The initial state is $s_0 := (\mathsf{st}_0, \mathsf{m}_0, \mathsf{pa}_0, \mathsf{pb}_0)$ with $\mathsf{pa}_0(\mathsf{r},\mathsf{q}) := 1 =: \mathsf{pb}_0(\mathsf{r},\mathsf{q})$ for all $\mathsf{r} \in \mathsf{RNK}$ and $\mathsf{q} \in \mathsf{QUE}$.

The transition relation Δ_Y is the smallest relation defined by the rules in Table 2. Rule (gpa') lets the automaton choose the part of the computation to which the next popa event will be appended. The first restriction is that the index of the part can only increase, as events from the same queue are processed in order. The second restriction is that popa events cannot be generated to the right of popb events from the same queue. Rule (gpb') is the similar rule for popb events.

By Rule (write'), the automaton appends a write event to τ_1 and the corresponding popa and popb events in one shot to the parts determined by pa and pb. Since a single transition of a multiheaded automaton can generate at most one letter, the rule makes use of intermediary states from S_Y^{aux}. If popb is added to τ_1, the memory configuration is updated accordingly. Note that the generation in one shot causes pop events within the same part τ_i to follow in the order of the corresponding read/write events in τ_1. Fortunately, this is always the case in normal-form computations by (NF). Computations that are not in normal form, e.g. $\tau_{\mathbf{1to1}}$, cannot be generated by $Y(\mathcal{P}, N)$.

The set of final states of $Y(\mathcal{P}, N)$ is S_Y. The auxiliary states S_Y^{aux} are not included in the set of final states to forbid computations with pending remote requests.

▶ **Lemma 12.** $\{\tau \in C(\mathcal{P}, N) \mid \tau \text{ is in normal form}\} = \mathcal{L}(Y(\mathcal{P}, N))$.

5.3 Detecting Violations

The multiheaded automaton accepts all normal form computations, and we would like to check if one of these computations is violating. In general, violating computations can contain complicated cycles in the happens-before relation. However, we now show that whenever a computation has a happens-before cycle, it has a cycle in which each process is entered and left at most once. Our algorithm for robustness will look for happens-before cycles of this special form that, as we will show, can be captured by a regular language.

▶ **Lemma 13.** *Computation $\tau \in C(\mathcal{P}, N)$ is violating iff there is a cycle*

$$a_1 \leftrightarrow^* b_1 \rightarrow_{po}^* c_1 \leftrightarrow^* d_1 \rightsquigarrow \ldots \rightsquigarrow a_k \leftrightarrow^* b_k \rightarrow_{po}^* c_k \leftrightarrow^* d_k \rightsquigarrow a_1 \qquad \text{(CYC)}$$

where $\mathit{rank}(x_i) = \mathit{rank}(y_j)$ iff $i = j$, for all $x_i, y_j \in \{a_1, \ldots, d_k\}$, and $\rightsquigarrow := \rightarrow_{cf} \cup \leftrightarrow$.

▶ **Example 14.** The computations $\tau_{\mathbf{1to1}}$ (Example 1) and $\tau^{nf}_{\mathbf{1to1}}$ (Example 9) have a cycle of the form (CYC) depicted in Figure 3: $k = 2$, $a_1 = b_1 = \mathsf{bar}$, $c_1 = d_1 = \mathsf{load}$, $a_2 = \mathbf{popb}$, $b_2 = \mathbf{write}$, $c_2 = d_2 = \mathbf{bar}$.

Note that $d_i \leftrightarrow a_{i+1}$ means both are barriers, $\mathsf{kind}(d_i) = \mathsf{bar} = \mathsf{kind}(a_{i+1})$. This holds as the ranks are different. In spite of the additional restrictions, cycles (CYC) are not trivial to recognize. The reason is that the events constituting the cycle are not necessarily contained in the computation in the order in which they appear in the cycle, see Figure 4. The idea of our cycle detection is to first guess the events a_i and d_i for each process and then check that $d_i \rightsquigarrow a_{i+1}$ holds. The former can be accomplished by an extension $Y^{\mathsf{M}}(\mathcal{P}, N)$ of the multiheaded automaton $Y(\mathcal{P}, N)$, the latter by a regular intersection.

The automaton $Y^{\mathsf{M}}(\mathcal{P}, N)$ accepts computations over the alphabet $\mathsf{E} \times \mathsf{M}$ with $\mathsf{M} := 2^{\{\mathsf{enter},\mathsf{leave}\}}$. The events marked by enter are the guessed a_i events in (CYC) and those marked by leave are the d_i events in (CYC). We still have to guarantee we only mark a_i and d_i that satisfy $a_i \leftrightarrow^* b_i \rightarrow_{po}^* c_i \leftrightarrow^* d_i$. This is straightforward thanks to the fact that $Y(\mathcal{P}, N)$ generates the events of each process in program order, and generates events related by $\leftrightarrow$ in one shot. The full construction of $Y^{\mathsf{M}}(\mathcal{P}, N)$ is given in [9].

▶ **Example 15.** Consider the normal-form computation $\tau^{nf}_{\mathbf{1to1}}$ (Example 9) that has the cycle (CYC) given in Figure 3. A corresponding marked computation of $Y^{\mathsf{M}}(\mathcal{P}, N)$ is

$$(\mathsf{write}, \emptyset) \cdot (\mathsf{popa}, \emptyset) \cdot (\mathbf{write}, \emptyset) \cdot (\mathbf{popa}, \emptyset) \cdot (\mathsf{bar}, \{\mathsf{enter}\}) \cdot (\mathbf{bar}, \{\mathsf{leave}\}) \cdot (\mathsf{load}, \{\mathsf{leave}\}) \cdot (\mathsf{popb}, \emptyset) \cdot (\mathbf{popb}, \{\mathsf{enter}\}).$$

Every cycle of the form (CYC) has a *cycle type* cyc, which is a sequence $\mathsf{cyc} = \mathsf{r}_1 \ldots \mathsf{r}_k$ of ranks from $\overline{1, N}$ with $\mathsf{r}_i \neq \mathsf{r}_j$ for $i \neq j$. The idea is that the events a_i, b_i, c_i, d_i belong to rank r_i. For each pair $\mathsf{r}_i, \mathsf{r}_{i+1}$ in this sequence, we construct a finite automaton $Z^{\mathsf{r}_i,\mathsf{r}_{i+1}}$ over the alphabet $\mathsf{E} \times \mathsf{M}$. It checks whether there is a conflict or identity edge from the leave-marked event of process r_i to the enter-marked event of process r_{i+1}. Consider the case of conflicts. The automaton looks for a marked event (e_i, m_i) with $\mathsf{rank}(\mathsf{e}_i) = \mathsf{r}_i$ marked by $\mathsf{leave} \in m_i$. It remembers the kind and the address of this event. Then, it seeks a marked event $(\mathsf{e}_{i+1}, m_{i+1})$ with $\mathsf{rank}(\mathsf{e}_{i+1}) = \mathsf{r}_{i+1}$ marked by $\mathsf{enter} \in m_{i+1}$. If both events are found, they touch the same address, and one of them is a write, the automaton reaches the accepting state. Since finite automata are closed under intersection, we can define the *finite automaton of cycle type* cyc as $Z^{\mathsf{cyc}} := Z^{\mathsf{r}_1,\mathsf{r}_2} \cap \ldots \cap Z^{\mathsf{r}_{k-1},\mathsf{r}_k} \cap Z^{\mathsf{r}_k,\mathsf{r}_1}$.

▶ **Theorem 16.** *$\mathcal{P}$ is robust iff $\mathcal{L}(Y^{\mathsf{M}}(\mathcal{P}, N)) \cap \mathcal{L}(Z^{\mathsf{cyc}}) = \emptyset$ for all cycle types cyc.*

We can now prove Theorem 3. To check whether $(\mathcal{P}, N)$ is robust, we go over all cycle types $\mathsf{cyc} = \mathsf{r}_1 \dots \mathsf{r}_k$. This enumeration of cycle types can be done in space that is polynomial in N. For each such sequence, we check if $\mathcal{L}(Y^{\mathsf{M}}(\mathcal{P}, N)) \cap \mathcal{L}(Z^{\mathsf{cyc}}) = \emptyset$. By Theorem 16, the program is robust iff all intersections are empty. By Lemma 10, there is a 4-headed finite state automaton W with $\mathcal{L}(W) = \mathcal{L}(Y^{\mathsf{M}}(\mathcal{P}, N)) \cap \mathcal{L}(Z^{\mathsf{cyc}})$. Since the size of W is exponential in the size of $(\mathcal{P}, N)$ and emptiness is in NL by Lemma 11, deciding $\mathcal{L}(W) = \emptyset$ can be done in space that is polynomial in $(\mathcal{P}, N)$. This shows robustness is in PSpace.

References

1 S. V. Adve and M. D. Hill. A unified formalization of four shared-memory models. *IEEE Transactions on Parallel and Distributed Systems*, 4(6):613–624, 1993.

2 J. Alglave. *A Shared Memory Poetics*. PhD thesis, University Paris 7, 2010.

3 J. Alglave and L. Maranget. Stability in weak memory models. In *CAV*, volume 6806 of *LNCS*, pages 50–66. Springer, 2011.

4 D. Bonachea. GASNet specification, v1.1. Technical Report UCB/CSD-02-1207, University of California, Berkeley, 2002.

5 A. Bouajjani, E. Derevenetc, and R. Meyer. Checking and enforcing robustness against TSO. In *ESOP*, volume 7792 of *LNCS*, pages 533–553. Springer, 2013.

6 A. Bouajjani, R. Meyer, and E. Möhlmann. Deciding robustness against Total Store Ordering. In *ICALP*, volume 6756 of *LNCS*, pages 428–440. Springer, 2011.

7 S. Burckhardt and M. Musuvathi. Effective program verification for relaxed memory models. In *CAV*, volume 5123 of *LNCS*, pages 107–120. Springer, 2008.

8 J. Burnim, C. Stergiou, and K. Sen. Sound and complete monitoring of sequential consistency for relaxed memory models. In *TACAS*, volume 6605 of *LNCS*, pages 11–25. Springer, 2011.

9 G. Calin, E. Derevenetc, R. Majumdar, and R. Meyer. A theory of partitioned global address spaces. *CoRR*, abs/1307.6590, 2013. `http://arxiv.org/abs/1307.6590`.

10 B. Chapman, T. Curtis, S. Pophale, S. Poole, J. Kuehn, C. Koelbel, and L. Smith. Introducing OpenSHMEM: SHMEM for the PGAS community. In *PGAS*, page 2. ACM, 2010.

11 UPC Consortium. UPC language specification v1.2. Technical report, 2005.

12 J. Dassow and G. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer, 1989.

13 D. Dice. A race in locksupport park() arising from weak memory models. `https://blogs.oracle.com/dave/entry/a_race_in_locksupport_park`, Nov 2009.

14 J. Dinan, P. Balaji, D. Buntinas, D. Goodell, W. Gropp, and R. Thakur. An implementation and evaluation of the MPI 3.0 one-sided communication interface. `www.mcs.anl.gov/uploads/cels/papers/P4014-0113.pdf`.

15 Global address space programming interface. `http://www.gaspi.de/`.

16 P. N. Hilfinger, D. O. Bonachea, K. Datta, D. Gay, S. L. Graham, B. R. Liblit, G. Pike, J. Zh. Su, and K. A. Yelick. Titanium language reference manual, version 2.19. Technical Report UCB/EECS-2005-15, UC Berkeley, 2005.

17 D. Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266. IEEE, 1977.

18 L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 28(9):690–691, 1979.

19 R. Machado and C. Lojewski. The Fraunhofer virtual machine: a communication library and runtime system based on the RDMA model. *Computer Science — Research and Development*, 23(3-4):125–132, 2009.

20 A. Muzahid, S. Qi, and J. Torrellas. Vulcan: Hardware support for detecting sequential consistency violations dynamically. In *MICRO*, pages 363–375. IEEE, 2012.

21 J. Nieplocha and B. Carpenter. ARMCI: A portable remote memory copy library for distributed array libraries and compiler run-time systems. In *Parallel and Distributed Processing*, volume 1586 of *LNCS*, pages 533–546. Springer, 1999.

22 The UPC NAS parallel benchmarks. `http://upc.gwu.edu/download.html`.

23 R. W. Numrich and J. Reid. Co-array Fortran for parallel programming. In *ACM Sigplan Fortran Forum*, volume 17, pages 1–31. ACM, 1998.

24 S. Owens. Reasoning about the implementation of concurrency abstractions on x86-TSO. In *ECOOP*, volume 6183 of *LNCS*, pages 478–503. Springer, 2010.

25 C.-S. Park, K. Sen, P. Hargrove, and C. Iancu. Efficient data race detection for distributed memory parallel programs. In *SC'11*, page 51. ACM, 2011.

26 D. Shasha and M. Snir. Efficient and correct execution of parallel programs that share memory. *ACM TOPLAS*, 10(2):282–312, 1988.

A Strong Direct Product Theorem for the Tribes Function via the Smooth-Rectangle Bound

Prahladh Harsha[1] and Rahul Jain[2]

1 **Tata Institute of Fundamental Research, Mumbai, India,** `prahladh@tifr.res.in`
2 **Centre for Quantum Technologies and Department of Computer Science, National University of Singapore, Singapore,** `rahul@comp.nus.edu.sg`

Abstract

The main result of this paper is an optimal strong direct product result for the two-party public-coin randomized communication complexity of the Tribes function. This is proved by providing an alternate proof of the optimal lower bound of $\Omega(n)$ for the randomised communication complexity of the Tribes function using the so-called smooth-rectangle bound, introduced by Jain and Klauck [6]. The optimal $\Omega(n)$ lower bound for Tribes was originally proved by Jayram, Kumar and Sivakumar [10], using a more powerful lower bound technique, namely the information complexity bound. The information complexity bound is known to be at least as strong a lower bound method as the smooth-rectangle bound [13]. On the other hand, we are not aware of any function or relation for which the smooth-rectangle bound is (asymptotically) smaller than its public-coin randomized communication complexity. The optimal direct product for Tribes is obtained by combining our smooth-rectangle bound for tribes with the strong direct product result of Jain and Yao [8] in terms of smooth-rectangle bound.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, G.2.1 Combinatorics

Keywords and phrases Rectangle bound, Tribes function, Strong direct product

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.141

1 Introduction

Study of lower bounds for various natural functions and relations has been a major theme of research in communication complexity from its advent; both for its own intrinsic value and for applications of these bounds towards other areas of theoretical computer science [15]. Several lower bound techniques have been developed over the years in communication complexity such as fooling sets, discrepancy method, rectangle bound, information complexity bound, partition bound etc. It is interesting to understand the relative power of these techniques and rank them against each other. Sometimes, we would like to understand what is the weakest technique required to prove a particular lower bound.

An important and extensively used technique in communication complexity is the so called rectangle bound (a.k.a. the corruption bound). In this technique, one argues that for some output value z, and all large rectangles, a constant fraction of inputs in the rectangle have a function value different from z. This helps to lower bound the distributional communication complexity of the function, which then translates to a lower bound on the public-coin communication complexity via Yao's minmax principle [20]. This technique has been successfully applied to obtain optimal lower bounds for several problems; Razborov's lower bound proof [18] for the set-disjointness function [11] is arguably the most well-known application of this technique.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 141–152

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

Another technique that has been extremely useful is the information complexity bound [17, 4]. In this method, one lower bounds the distributional communication complexity by the amount of information the transcript of the protocol reveals about the inputs of Alice and Bob. The tools from information theory then come handy to lower bound the information cost of the protocol. Bar-Yossef, Jayram, Kumar and Sivakumar [1] successfully used this technique[1] to give an alternate proof of the linear lower bound for the set-disjointness function. This method has also been useful to give an optimal linear lower bound for the Tribes function [10].

Jain and Klauck [6], using tools from linear programming and semi-definite programming gave a uniform treatment to several of the existing lower bound techniques and proposed two additional lower bound techniques, the so-called partition bound and the smooth-rectangle bound. These bounds are stronger than almost all other known lower bound techniques including the rectangle bound. The partition bound, as the name suggests, is a linear programming formulation of the number of partitions in a randomized protocol. The smooth-rectangle bound, a weakening of the partition bound, is a robust version of the rectangle bound in the following informal sense: smooth-rectangle bound for a function f under a distribution μ, is the maximum over all functions g , which are close to f under the distribution μ, of the rectangle bound of g. In other words, a function f is said to have a large smooth-rectangle bound, if it is close to some other function g (under the distribution μ) which has a large rectangle bound, even though f itself might not have a large rectangle bound. This suffices to lower bound the communication complexity of f. These new lower bound methods have been successfully applied, for example to obtain an optimal lower bound for the Gap-Hamming problem [3]. In fact we are not aware (to the best of our knowledge) of any function or relation for which the partition bound or smooth-rectangle bound is (asymptotically) smaller than its public-coin randomized communication complexity. To determine how tight these new lower bounds are, remains an important open question in communication complexity.

Recently, Kerenidis *et al.* [13] showed that the information complexity is at least as powerful as the relaxed-partition-bound, which is a bound intermediate between the partition bound and the smooth-rectangle bound. The relative strengths of the information complexity and partition bound is not yet well understood.

Another important theme in communication complexity has been the study of the so called strong direct-product and (the weaker) direct-sum conjectures; again for their own intrinsic value and also for important applications of such results in other areas of theoretical computer science [12]. A strong direct-product conjecture for the public-coin communication complexity of a relation f would state the following. Let c be the public-coin communication complexity of f (with constant error). Suppose k independent instances of f are being solved using communication less than kc, then the overall success would be exponentially small in k. In fact, the information complexity was introduced initially [4] as a tool to resolve the direct sum/product question. However, despite the considerable progress made over the last few years [2, 7], the direct product question has not yet been resolved. On the other hand, we are not aware of any function or relation for which this conjecture is false. Settling this conjecture for all relations, again is an important open question in communication complexity.

[1] The notion of information complexity was formalized by Chakrabarti, Shi, Wirth and Yao [CSWY01] in the direct sum context, however has been used by earlier works as well for example by Ponzio, Radhakrishnan and Venkatesh [17] for showing optimal lower bounds on the communication complexity of the pointer-chasing problem. Chakrabarti *et al.* [4] defined and used, what in today's language is called,"external information cost" while Bar-Yossef, Jayram, Kumar and Sivakumar [1] defined and used "internal information cost" in their proof of the disjointness lower bound.

Recently, Jain and Yao [8] proved a direct-product result for all relations in terms of the smooth-rectangle bound (srec). They show that for any relation f, if less than $k \cdot \log \mathsf{srec}(f)$ communication (c.f., Definition 2.2) is provided for solving k independent copies of f, then the overall success is exponentially small in k. This provides a recipe to arrive at strong direct-product results for any relation f: by exhibiting that $\log \mathsf{srec}(f)$ provides optimal lower bound for the public-coin communication complexity of f. Jain and Yao's result implies (and in some cases reproves) strong direct product result for many interesting functions and relations including that for the set-disjointness function (a strong direct-product result for set-disjointness was first shown by Klauck [14], again via showing that the smooth-rectangle bound of a related function is large). This also strongly motivates the search of functions for which their smooth-rectangle bound is asymptotically smaller than their public-coin communication complexity. This leads us to the study of the Tribes function as described below.

1.1 Our result

In this work we are concerned with the $\mathsf{Tribes} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ function, defined as follows.

$$\mathsf{Tribes}(x,y) \stackrel{\text{def}}{=} \bigwedge_{i=1}^{\sqrt{n}} \left(\bigvee_{j=1}^{\sqrt{n}} \left(x_{(i-1)\sqrt{n}+j} \wedge y_{(i-1)\sqrt{n}+j} \right) \right).$$

As mentioned earlier, an optimal linear lower bound for Tribes was shown by Jayram, Kumar and Sivakumar [10] using the information complexity technique. It is to be noted that the rectangle bound proves only a $\Theta(\sqrt{n})$ lower bound and thus fails to provide an optimal lower bound for Tribes. In fact, the primary motivation for Jayram *et al.* [10] to study the Tribes function was the fact that it provided the first example where information complexity techniques were provably stronger than the then known "combinatorial" lower bound techniques. Therefore it is natural to ask if Tribes also provides a separation between smooth-rectangle bound and public-coin communication complexity, in the process also implying separation between information complexity bound and smooth-rectangle bound. We consider this question in this work and answer it in the negative.

▶ **Theorem 1.1** (smooth-rectangle bound for Tribes)**.**
For sufficiently small $\varepsilon \in (0,1)$, $\mathsf{R}^{\mathrm{pub}}_{\varepsilon}(\mathsf{Tribes}) \geq \log \mathsf{srec}_{\varepsilon}(\mathsf{Tribes}) \geq \Omega(n)$.

Here, $\mathsf{R}^{\mathrm{pub}}_{\varepsilon}(f)$ refers to the ε-error public-coin randomized communication complexity of f.

Another important motivation for our work (besides answering the above question) is its consequence to strong direct product. As indicated in the recipe outlined above, combining our smooth-rectangle bound for Tribes with the result of Jain and Yao [8], we obtain the following.

▶ **Corollary 1.2** (strong direct product for Tribes)**.** $\mathsf{R}^{\mathrm{pub}}_{1-2^{-\Omega(k)}}\left(\mathsf{Tribes}^{(k)}\right) = \Omega(kn)$.

Here, $f^{(k)}$ refers to the k-wise direct product of the function f. Our result (Theorem 1.1) also exhibits for the first time, an asymptotic separation between the smooth-rectangle bound and the rectangle bound for a total function (previously a quadratic separation was known however for the Gap-Hamming partial function [3]).

It is to be noted that the information complexity lower bound for Tribes was generalised to constant depth read-once trees functions [9, 16]. Given our results, it is interesting to ask if these lower bounds can be obtained using the smooth-rectangle bound instead, which would imply a direct product for these functions. These alternate lower bounds might also help to obtain bounds for super-constant depth read-once formulae.

1.2 Our techniques

It will be convenient for us to view the Tribes function as the conjunction of $\sqrt{n}$ set-disjointness functions over $\sqrt{n}$ sized inputs[2]. We refer to the $\sqrt{n}$ sized inputs to each of the disjointness functions as a block. We consider a distribution μ on the inputs for the Tribes function which has support only on the following type of inputs: in every block, except for one block (say j), the inputs to the two parties Alice and Bob are NO instances of the disjointness function (the sets corresponding to the blocks intersect at exactly one location) and in block j, there could be $0, 1$ or 2 intersections which occur at locations k_j and l_j. Let's refer to the three types of subsets of inputs based on the number of intersections as U_0, U_1, and U_2 respectively. Recall that to show that the smooth-rectangle bound of Tribes is large, we need to demonstrate a function g, close to Tribes (under μ), whose rectangle bound is large. This function g is constructed as follows: g takes value 0 in $U_0 \cup U_2$ and value 1 in U_1. Note that Tribes takes value 0 in U_0 and value 1 in $U_1 \cup U_2$. I.e., Tribes and g disagree on the inputs in U_2. For our choice of distribution μ, this disagreement set U_2 will have weight $\mu(U_2) \approx 1/16$ while the weight of the 1-inputs will be approximately $\mu(U_1) \approx 6/16$ (i.e., U_1 is 6 times larger than U_2).

Observe that for Tribes, there are large rectangles (of size $\approx 2^{-\sqrt{n}}$ under μ) which are monochromatic. We can just fix any one coordinate in each block and force intersection there to create large 1-monochromatic rectangle. Similarly we can choose any one block and force non-intersection in that entire block to create large 0-monochromatic rectangle. Hence the rectangle bound of Tribes is at most $O(\sqrt{n})$. However, note that the 1-monochromatic rectangles described above are not monochromatic in g. Indeed, we show that there exists constants C and D such that for every large rectangle W (with $\mu(W) \geq 2^{-\Omega(n)}$), $\mu(U_1 \cap W)$ is either dominated by $C \cdot \mu(U_0 \cap W)$ (this is similar to the rectangle bound) or is dominated by $D \cdot \mu(U_2 \cap W)$. This immediately implies the rectangle bound of g is $\Omega(n)$. We will prove the above statement for D strictly smaller than 6. This fact implies that whenever $\mu(U_1 \cap W)$ is not dominated by $C \cdot \mu(U_0 \cap W)$ in W, the ratio of U_2-inputs to U_1-inputs in the rectangle W is considerably more than the similar ratio globally (which is $\approx 1/6$). This fact lets us translate the $\Omega(n)$ rectangle bound for g to a similar smooth-rectangle bound for Tribes.

We consider an exhaustive collection of sub-events such that conditioned on any such sub-event, the non-product distribution μ becomes a product distribution. Such handling of non-product distributions, by decomposing them into several product distributions, has been done several times before, for instance in Razborov's proof [18] of the optimal lower bound for the set-disjointness function. Assume such a conditioning exists for the rest of this proof outline.

How does one prove that for all large rectangles W, either $\mu(U_1 \cap W) \leq C\mu(U_0 \cap W))$ or $\mu(U_1 \cap W) \leq D\mu(U_2 \cap W)$ for some D strictly smaller than 6. Note that one cannot prove for all rectangles W, $\mu(U_1 \cap W) \leq D\mu(U_2 \cap W)$ for some D strictly less than 6, since this is false globally (i.e., $\mu(U_1) \approx 6\mu(U_2)$). Hence, one needs to do a case analysis[3]. And we do this based on the values of $\Pr\left[X_{l_j} = Y_{l_j} = 1\right]$ and $\Pr\left[X_{k_j} = Y_{k_j} = 1\right]$.

Consider the case when $\Pr\left[X_{l_j} = Y_{l_j} = 1\right] \geq \frac{3}{4}\mu(U_1 \cap W)$. Since the rectangle is large, using an entropy argument, we can argue that in most cases, conditioned on the sub-event

---

[2] By the disjointness function, we refer to the function $\bigvee_{j=1}^{\sqrt{n}} (x_j \wedge y_j)$. Strictly speaking, this is the set-intersection problem, but as is common in this literature, we will abuse notation and refer to this problem as the set-disjointness problem.

[3] Such a case analysis is not required to prove rectangle bound (c.f., proof of disjointness [18]), but is necessary while proving a smooth-rectangle bound.

$(X_{l_j} = Y_{l_j} = 1)$, both $\Pr\left[X_{k_j} = 1\right]$ and $\Pr\left[Y_{k_j} = 1\right]$ are large enough ($\approx 1/2$). Now since the distribution is product it means that conditioned on $(X_{l_j} = Y_{l_j} = 1)$, $\Pr\left[X_{k_j} = Y_{k_j} = 1\right]$ is large enough and hence $\mu(U_2 \cap W)$ is a required fraction of $\mu(U_1 \cap W)$. Similar arguments hold for the case with the roles of l and k reversed.

In the third case, when $\max\{\Pr\left[X_{l_j} = Y_{l_j} = 1\right], \Pr\left[X_{k_j} = Y_{k_j} = 1\right]\} \leq \frac{3}{4}\mu(U_1 \cap W)$, again using the same entropy argument, we can show that $\Pr\left[X_{l_j} = Y_{l_j} = 1, X_{k_j} = Y_{k_j} = 0\right]$ and $\Pr\left[X_{l_j} = Y_{l_j} = 0, X_{k_j} = Y_{k_j} = 1\right]$ are large. Now, since W is a rectangle, we can show that $\Pr\left[X_{l_j} = 1, Y_{l_j} = 0, X_{k_j} = 0, Y_{k_j} = 1\right]$ and $\Pr\left[X_{l_j} = 0, Y_{l_j} = 1, X_{k_j} = 1, Y_{k_j} = 0\right]$ are large using a *cut-and-paste* argument. This implies that $\mu(U_0 \cap W)$ is a required fraction of $\mu(U_1 \cap W)$. This concludes our proof outline.

We note that our distribution is similar to (and in fact inspired from) the distribution used by Jain and Klauck [6] while analyzing the query complexity of the Tribes function. We also note that the distribution used by Jayram, Kumar and Sivakumar [10] in their information complexity lower bound for Tribes is different from our distribution, in particular, their distribution does not put any support on U_2 inputs which have intersections of size 2 within block j. However, we do add that they also use similar in spirit, albeit different cut-and-paste arguments in their lower bound proof.

2 Preliminaries

Communication Complexity

We begin by recalling the Yao's two-party communication model [19] (see Kushilevitz and Nisan [15] for an excellent introduction to the area). Let $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ be finite non-empty sets, and let $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ be a function. A two-party protocol for computing f consists of two parties, Alice and Bob, who get inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ respectively, and exchange messages in order to compute $f(x, y) \in \mathcal{Z}$.

For a distribution μ on $\mathcal{X} \times \mathcal{Y}$, let the ε-error distributional communication complexity of f under μ (denoted by $\mathsf{D}^{\mu}_{\varepsilon}(f)$), be the number of bits communicated (for the worst-case input) by the best deterministic protocol for f with average error at most ε under μ. Let $\mathsf{R}^{\mathrm{pub}}_{\varepsilon}(f)$, the public-coin randomized communication complexity of f with worst case error ε, be the number of bits communicated (for the worst-case input) by the best public-coin randomized protocol, that for each input (x, y) computes $f(x, y)$ correctly with probability at least $1 - \varepsilon$. Randomized and distributional complexity are related by the following celebrated result of Yao [20].

▶ **Theorem 2.1** (Yao's minmax principle [20]). $\mathsf{R}^{\mathrm{pub}}_{\varepsilon}(f) = \max_{\mu} \mathsf{D}^{\mu}_{\varepsilon}(f)$.

Given a function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$, the k-wise direct product of f, denoted by $f^{(k)}$ is the function $f : \mathcal{X}^k \times \mathcal{Y}^k \to \mathcal{Z}^k$ defined as follows: $f^{(k)}((x_1, \ldots, x_k), (y_1, \ldots, y_k)) = (f(x_1, y_1), \ldots, f(x_k, y_k))$. The direct product/sum question involves relating $\mathsf{R}^{\mathrm{pub}}(f^{(k)})$ to $\mathsf{R}^{\mathrm{pub}}(f)$. More precisely, the strong direct product conjecture states that $\mathsf{R}^{\mathrm{pub}}_{1-2^{-\Omega(k)}}(f^{(k)}) = \Omega\left(k \cdot \mathsf{R}^{\mathrm{pub}}_{1/3}(f)\right)$.

The smooth rectangle bound

The smooth rectangle bound was introduced by Jain and Klauck [6], as a generalization of the rectangle bound. Informally, the smooth-rectangle bound for a function f under a distribution μ, is the maximum over all functions g , which are close to f under the distribution μ, of the rectangle bound of g. However, it will be more convenient for us to work with the following

linear programming formulation of smooth-rectangle bound. Please see [6, Lemma 2] and [8, Lemma 6] for the relations between the LP formulation and the more "natural" formulation in terms of rectangle bound. A broad connection between the two definitions is that the variable φ in the dual of the linear programming definition takes non-zero values precisely at the inputs (x, y) where f and g differ.

▶ **Definition 2.2** (smooth-rectangle bound)**.** For a total Boolean function f, the ε- smooth rectangle bound of f denoted $\mathsf{srec}_\varepsilon(f)$ is defined to be $\max\{\mathsf{srec}^z_\varepsilon(f) : z \in \{0,1\}\}$, where $\mathsf{srec}^z_\varepsilon(f)$ is given by the optimal value of the following linear program (below $\mathcal{W}$ represents the set of all rectangles in $\mathcal{X} \times \mathcal{Y}$).

Primal

$$\begin{aligned}
\min \sum_{W \in \mathcal{W}} & v_W \\
\sum_{W \ni (x,y)} & v_W \geq 1 - \varepsilon, \quad && \forall (x,y) \in f^{-1}(z) \\
\sum_{W \ni (x,y)} & v_W \leq 1, && \forall (x,y) \in f^{-1}(z) \\
\sum_{W \ni (x,y)} & v_W \leq \varepsilon, && \forall (x,y) \notin f^{-1}(z) \\
& v_W \geq 0, && \forall W \ .
\end{aligned}$$

Dual

$$\begin{aligned}
\max \sum_{(x,y) \in f^{-1}(z)} & ((1-\varepsilon)\lambda_{x,y} - \varphi_{x,y}) - \sum_{(x,y) \notin f^{-1}(z)} \varepsilon \cdot \lambda_{x,y} \\
\sum_{(x,y) \in W_z} & (\lambda_{x,y} - \varphi_{x,y}) - \sum_{(x,y) \in W_{-z}} \lambda_{x,y} \leq 1, \quad \forall W \\
& \lambda_{x,y}, \varphi_{x,y} \geq 0, \quad \forall (x,y) \\
& \text{where } W_z = W \cap f^{-1}(z) \text{ and } W_{-z} = W \setminus f^{-1}(z) \ .
\end{aligned}$$

▶ **Theorem 2.3** ([6, Theorem 1])**.** *For all functions $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ and $\varepsilon \in (0,1)$, we have $\mathsf{R}^{\mathrm{pub}}_\varepsilon(f) \geq \log(\mathsf{srec}_\varepsilon(f))$.*

Jain and Yao [8] proved the following strong direct product theorem in terms of the smooth rectangle bound.

▶ **Theorem 2.4** ([8, Theorem 1 and Lemma 6])**.** *Let $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ be a Boolean function. For every $\varepsilon \in (0,1)$, there exists small enough $\eta \in (0, 1/3)$ such that the following holds. For all integers k,*

$$\mathsf{R}^{\mathrm{pub}}_{1-(1-\eta)^{\lfloor \eta^2 k/32 \rfloor}}(f^{(k)}) \geq \frac{\eta^2}{32} \cdot k \cdot \left(11\eta \cdot \log \mathsf{srec}_\varepsilon(f) - 3 \log \frac{1}{\varepsilon} - 2\right).$$

Information theory

We need the following basic facts from information theory. Let μ be a (probability) distribution on a finite set $\mathcal{X}$ and X be a random variable distributed according to μ. Let $\mu(x)$ represent the probability of $x \in \mathcal{X}$ according to μ. The entropy of X is defined as $H(X) \stackrel{\text{def}}{=} \sum_x \mu(x) \cdot \log \frac{1}{\mu(x)}$. Entropy satisfies subadditivity: $H(XY) \leq H(X) + H(Y)$.

3 The smooth rectangle bound for Tribes

In this section, we prove a linear lower bound on the randomized communication of Tribes via the smooth-rectangle bound.

First we introduce some notation. We will prove the result for n of the form $(2r+1)^2$, where $r \geq 2$ is even. Assume the input indices $[n]$ to the Tribes function are partitioned into

$\sqrt{n}$ blocks $s_1, \ldots, s_{\sqrt{n}}$, where the i^{th} block $s_i = \{(i-1)\sqrt{n}+1, \ldots, i\sqrt{n}\}$. Thus,

$$\mathsf{Tribes}(x,y) = \bigwedge_{i=1}^{\sqrt{n}} \left(\bigvee_{j \in s_i} (x_j \wedge y_j) \right).$$

A string $x \in \{0,1\}^n$ can be viewed both as an n-bit string and as a subset $x \subseteq [n]$. We will use both these interpretations.

Consider the distribution $\mu(x,y)$ on the inputs of the Tribes function defined by the following (informal) description. As mentioned earlier, this distribution is inspired by the distribution used by Jain and Klauck [6] while analyzing the query complexity of the Tribes function. Among the $\sqrt{n}$ blocks, one of the blocks is chosen as a special block, say block j. Alice's and Bob's inputs are then chosen such that their inputs when restricted to any of the blocks (special or non-special) have exactly $(r/2+1)$ ones each. Furthermore, for each of the non special blocks, Alice's and Bob's input are chosen such that their inputs, restricted to this block, have a unique intersection (this is identical to the yes instances of Razborov's distribution for disjointness) while for the special block j, Alice's and Bob's inputs are chosen such that their inputs, restricted to the special block, have an intersection of size 0, 1 or 2. As in the case of Razborov's distribution, the variable t is used to denote the random variable containing the index of the special block j and other relevant information such that conditioned on t, the distribution (X,Y) is a product distribution. The formal description of the distribution μ is as follows:

1. Choose $j \in [\sqrt{n}]$ uniformly.
   For each $i \in [\sqrt{n}] \setminus \{j\}$, randomly partition the indices in s_i as follows: $s_i = (t_i^A, t_i^B, \{l_i\})$ into 3 disjoint sets such that $|t_i^A| = |t_i^B| = r$ and $l_i \in s_i$.
   For index j, randomly partition the indices in s_j as follows: $s_j = (\tilde{t}_j^A, \tilde{t}_j^B, \{k_j\}, \{l_j\}, \{d_j\})$ into 5 disjoint sets such that $|\tilde{t}_j^A| = |\tilde{t}_j^B| = r-1$ and $k_j, l_j, d_j \in s_j$. Set $t_j^A = \tilde{t}_j^A \cup \{k_j\}$ and $t_j^B = \tilde{t}_j^B \cup \{k_j\}$.
   Let $t = \left(j, k_j, (t_i^A, t_i^B, l_i)_{i \in [\sqrt{n}]}\right)$.
2. For each $i \neq j \in [\sqrt{n}]$, set the variables in block s_i as follows:
   - Set $x_{l_i} \leftarrow 1$ and $x_{s_i \setminus (t_i^A \cup \{l_i\})} \leftarrow \bar{0}$. Let $x_{t_i^A}$ be a random string of exactly $r/2$ ones.
   - Set $y_{l_i} \leftarrow 1$ and $y_{s_i \setminus (t_i^B \cup \{l_i\})} \leftarrow \bar{0}$. Let $y_{t_i^B}$ be a random string of exactly $r/2$ ones.
3. Set the variables in block s_j as follows:
   - Let $x_{t_j^A \cup \{l_j\}}$ be a random string of exactly $r/2+1$ ones and $x_{s_j \setminus (t_j^A \cup \{l_j\})} \leftarrow \bar{0}$.
   - Let $y_{t_j^B \cup \{l_j\}}$ be a random string of exactly $r/2+1$ ones and $y_{s_j \setminus (t_j^B \cup \{l_j\})} \leftarrow \bar{0}$.

Let (X,Y) be distributed according to μ, where X represents the input to Alice and Y represents the input to Bob. Let $T = \left(J, K_J, (T_i^A, T_i^B, L_i)_{i \in [\sqrt{n}]}\right)$ be the random variable (correlated with (X,Y)) representing t distributed as above. Observe that though (X,Y) is not a product distribution, the conditional distribution $((X,Y) \mid T = t)$ is product for each t.

Partition the set of inputs (in the support of μ) into 3 sets U_0, U_1 and U_2 as follows:

$$U_i = \{(x,y) \mid \mu(x,y) > 0 \text{ and sets } x \text{ and } y \text{ have exactly } \sqrt{n} - 1 + i \text{ intersections}\}.$$

Note that U_0 are the 0-inputs and $U_1 \cup U_2$ the 1-inputs of the Tribes function while $U_0 \cup U_2$ and U_1 are the 0- and 1-inputs respectively of the function g described in Section 1.2.

Let $\beta \stackrel{\text{def}}{=} \frac{r+2}{r+1}$. The following facts can be easily verified from the definition of the distribution μ. For all t,

$$\Pr[X_{l_j} = 1 \mid T = t] = \frac{\beta}{2}; \quad \Pr[X_{l_j} = X_{k_j} = 1 \mid T = t] = \Pr[X_{l_j} = 1, X_{k_j} = 0 \mid T = t] = \frac{\beta}{4}.$$

Given this, it can be easily checked that the weights of the sets U_0, U_1 and U_2 are as follows: $\mu(U_0) = 1 - 7\beta^2/16, \mu(U_1) = 6\beta^2/16$, and $\mu(U_2) = \beta^2/16$.

Our main lemma is the following (we have not optimized the constants).

▶ **Lemma 3.1.** *There exists a constant $\delta \in (0,1)$ such that for sufficiently large n, the following holds: for every rectangle $W = A \times B$, we have*

$$0.99\mu(U_1 \cap W) \leq \frac{16}{3(0.99)^2} \cdot \mu(U_2 \cap W) + \frac{16}{(0.99)^2}\mu(U_0 \cap W) + 2^{-\delta n/2+1}.$$

In other words, in any rectangle which contains a significant fraction of inputs from U_1 (i.e., at least $2^{-\delta n/2+1}$), the weight of the U_1 inputs is dominated by some linear function of the weights of U_0 and U_2 inputs. Before proving this lemma, let us first see how this lemma implies the smooth-rectangle bound for Tribes, which implies our Main Theorem 1.1

▶ **Theorem 3.2** (smooth-rectangle bound for Tribes)**.** *There exists $\gamma \in (0,1)$ such that for all sufficiently large n and $\varepsilon < 1/1000$, we have:* $\mathsf{srec}^1_\varepsilon(\mathsf{Tribes}) \geq 2^{\gamma \cdot n}$.

Proof. We will prove the bound using the dual formulation for smooth-rectangle bound given in Definition 2.2. Define the dual variables $\lambda_{x,y}$ and $\varphi_{x,y}$ as follows:

$$\lambda_{x,y} = \begin{cases} 0 & \text{if} \quad (x,y) \in U_2 \\ 0.99\mu(x,y)2^{\delta n/2-1} & \text{if} \quad (x,y) \in U_1 \\ \frac{16}{(0.99)^2}\mu(x,y)2^{\delta n/2-1} & \text{if} \quad (x,y) \in U_0. \end{cases}$$

$$\varphi_{x,y} = \begin{cases} \frac{16}{3(0.99)^2}\mu(x,y)2^{\delta n/2-1} & \text{if} \quad (x,y) \in U_2 \\ 0 & \text{if} \quad (x,y) \in U_1 \cup U_0. \end{cases}$$

From Lemma 3.1 we get

$$\forall \text{ rectangles } W: \sum_{(x,y) \in \mathsf{Tribes}^{-1}(1) \cap W} (\lambda_{x,y} - \varphi_{x,y}) - \sum_{(x,y) \in (W \setminus \mathsf{Tribes}^{-1}(1))} \lambda_{x,y} \leq 1.$$

The objective of the LP can be bounded as follows:

$$\begin{aligned} &\sum_{(x,y) \in \mathsf{Tribes}^{-1}(1)} ((1-\varepsilon)\lambda_{x,y} - \varphi_{x,y}) - \sum_{(x,y) \notin \mathsf{Tribes}^{-1}(1)} \varepsilon \cdot \lambda_{x,y} \\ &\geq \left((0.999)(0.99)\mu(U_1) - \frac{16}{3(0.99)^2}\mu(U_2) - \frac{16}{1000(0.99)^2}\mu(U_0) \right) 2^{\delta n/2-1} \\ &\geq 0.02 \cdot 2^{\delta n/2-1} \qquad \text{(for sufficiently large n).} \end{aligned}$$

Thus, proved. ◀

Corollary 1.2 follows by combining the above theorem and Jain-Yao's strong direct product theorem in terms of the smooth-rectangle bound (Theorem 2.4).

3.1 Proof of Lemma 3.1

Let $W = A \times B$ be the rectangle. For each $t = \left(j, k_j, (t_i^A, t_i^B, l_i)_{i \in [\sqrt{n}]}\right)$ and $a, b \in \{0,1\}$, define,

$$R(t,a,b) = \Pr\left[X \in A \mid T = t, X_{l_j} = a, X_{k_j} = b\right], R(t,a) = \Pr\left[X \in A \mid T = t, X_{l_j} = a\right],$$
$$C(t,a,b) = \Pr\left[Y \in B \mid T = t, Y_{l_j} = a, Y_{k_j} = b\right], C(t,a) = \Pr\left[Y \in B \mid T = t, Y_{l_j} = a\right].$$

Define the following random variables (we will set δ later):

$$\mathsf{BAD}_{\mathsf{A}}(t) = 1 \text{ iff } \min\{R(t,1,1), R(t,1,0)\} < 0.99\left(R(t,1) - 2^{-\delta n}\right),$$

and symmetrically,

$$\mathsf{BAD}_{\mathsf{B}}(t) = 1 \text{ iff } \min\{C(t,1,1), C(t,1,0)\} < 0.99\left(C(t,1) - 2^{-\delta n}\right).$$

For a given t, let t' denote a partition identical to t except that the role of the indices l_j and k_j are exchanged (i.e., $k'_j = l_j, l'_j = k_j, (t^A_j)' = \tilde{t}^A_j \cup \{l_j\}$ and $(t^B_j)' = \tilde{t}^B_j \cup \{l_j\}$). To define $\mathsf{BAD}(t)$, we need the following two quantities.

$$\begin{aligned}\rho_l(t) &= \Pr\left[X_{l_j} = Y_{l_j} = 1, X \in A, Y \in B, (X,Y) \in U_1 \mid T = t\right],\\ \rho_k(t) &= \Pr\left[X_{k_j} = Y_{k_j} = 1, X \in A, Y \in B, (X,Y) \in U_1 \mid T = t\right].\end{aligned}$$

Observe that $\mu(U_1 \cap W \mid T = t) = \rho_l(t) + \rho_k(t)$. Hence, it must be the case that exactly one of the following happens: (1) $\rho_l(t) > 3\mu(U_1 \cap W \mid T = t)/4$, (2) $\rho_k(t) > 3\mu(U_1 \cap W \mid T = t)/4$ or (3) $\max\{\rho_l(t), \rho_l(t)\} \leq 3\mu(U_1 \cap W \mid T = t)/4$ (equivalently, $\min\{\rho_l(t), \rho_l(t)\} \geq \mu(U_1 \cap W \mid T = t)/4$). We define $\mathsf{BAD}(t)$ based on these cases as follows.

$$\mathsf{BAD}(t) = \begin{cases} \mathsf{BAD}_{\mathsf{A}}(t) \vee \mathsf{BAD}_{\mathsf{B}}(t), & \text{if } \rho_l(t) > \frac{3\mu(U_1 \cap W \mid T=t)}{4} \\ \mathsf{BAD}_{\mathsf{A}}(t') \vee \mathsf{BAD}_{\mathsf{B}}(t'), & \text{if } \rho_k(t) > \frac{3\mu(U_1 \cap W \mid T=t)}{4} \\ \mathsf{BAD}_{\mathsf{A}}(t) \vee \mathsf{BAD}_{\mathsf{B}}(t) \vee \mathsf{BAD}_{\mathsf{A}}(t') \vee \mathsf{BAD}_{\mathsf{B}}(t'), & \text{otherwise.} \end{cases} \quad (3.1)$$

The following claim shows that the probability that $\mathsf{BAD}_{\mathsf{A}}(T)$ and $\mathsf{BAD}_{\mathsf{B}}(T)$ occurs is small.

▶ **Claim 3.3.** *There exists a small fixed constant $\delta > 0$ such that for sufficiently large n, the following holds: for any $(t^A_i, l_i)_{i \in [\sqrt{n}]}$, we have*

$$\Pr\left[\mathsf{BAD}_{\mathsf{A}}(T) = 1 \mid T^A_i = t^A_i, L_i = l_i, \text{ for each } i \in [\sqrt{n}]\right] < \frac{1}{6400}.$$

(Symmetrically, for any $(t^B_i, l_i)_i$, $\Pr\left[\mathsf{BAD}_{\mathsf{B}}(T) = 1 \mid T^B_i = t^B_i, L_i = l_i, \text{ for each } i \in [\sqrt{n}]\right] < \frac{1}{6400}$.)

Proof. We prove the inequality involving $\mathsf{BAD}_{\mathsf{A}}(T)$. The other inequality is proved similarly. We first consider the easy case when $(t^A_i, l_i)_{i \in [\sqrt{n}]}$ satisfies

$$\Pr\left[X \in A \mid X_{l_i} = 1, T^A_i = t^A_i, L_i = l_i, \text{ for each } i \in [\sqrt{n}]\right] < 2^{-\delta n}.$$

It follows from the definition of the distribution μ, that the above probability is unchanged on further conditioning by $T = t$ for any t consistent with $(t^A_i, l_i)_{i \in [\sqrt{n}]}$. In other words, this probability is equal to $R(t,1) = \Pr\left[X \in A \mid T = t, X_{l_j} = 1\right]$ for any t consistent with $(t^A_i, l_i)_{i \in [\sqrt{n}]}$. Hence, for any such t we have that $R(t,1) < 2^{-\delta n}$. Thus, in this case $\mathsf{BAD}_{\mathsf{A}}(t) = 0$ for all such t and we are done.

Now consider the other case when

$$\Pr\left[X \in A \mid X_{l_i} = 1, T^A_i = t^A_i, L_i = l_i, \text{ for each } i \in [\sqrt{n}]\right] \geq 2^{-\delta n}. \quad (3.2)$$

Consider a $t = (j, k_j, (t^A_i, t^B_i, l_i)_{i \in [\sqrt{n}]})$ consistent with $(t^A_i, l_i)_{i \in [\sqrt{n}]}$. We know that the bit $(X_{k_j} \mid T = t, X_{l_j} = 1)$ is a unbiased bit. Now, suppose $\mathsf{BAD}_{\mathsf{A}}(t) = 1$. Then, for some $a \in \{0,1\}$, we have

$$\Pr\left[X \in A \mid T = t, X_{l_j} = 1, X_{k_j} = a\right] < 0.99\left(\Pr\left[X \in A \mid T = t, X_{l_j} = 1\right]\right).$$

By a simple rewriting of the above inequality, we have

$$\Pr\big[X_{k_j} = a \mid X \in A, T = t, X_{l_j} = 1\big] < 0.99\left(\Pr\big[X_{k_j} = a \mid T = t, X_{l_j} = 1\big]\right) = 0.99/2. \quad (3.3)$$

In other words, the unbiased bit $(X_{k_j} \mid T = t, X_{l_j} = 1)$ when conditioned on the event "$X \in A$" is now more likely to be $1 - a$ than a.

Suppose, for contradiction, that

$$\Pr\big[\mathsf{BAD}_{\mathsf{A}}(T) = 1 \mid T_i^A = t_i^A, L_i = l_i, \text{ for each } i \in [\sqrt{n}]\big] \geq \frac{1}{6400}.$$

Consider the random variable

$$Z \stackrel{\text{def}}{=} (X \mid X_{l_i} = 1, T_i^A = t_i^A, L_i = l_i, \text{ for each } i \in [\sqrt{n}]).$$

Note that the distribution of Z is uniform and each string has probability $\left(\frac{1}{\binom{r}{r/2}}\right)^{\sqrt{n}}$. Consider the event $E \stackrel{\text{def}}{=}$ "$X \in A \mid T = t, X_{l_j} = 1$", which by (3.2) has probability at least $2^{-\delta n}$. Therefore the probability of each string in the distribution $(Z|E)$ would be at most $2^{\delta n} \cdot \left(\frac{1}{\binom{r}{r/2}}\right)^{\sqrt{n}}$. Therefore, using standard estimates on binomial coefficients,

$$H(Z|E) \geq \sqrt{n} \cdot \log \binom{r}{r/2} - \delta n \geq \sqrt{n} \cdot r(1 - o(1)) - \delta n.$$

Observe that conditioned on $T_i^A = t_i^A, L_i = l_i$, for each $i \in [\sqrt{n}]$, the index K_J can equally likely be any one of the $r\sqrt{n}$ indices in $\bigcup_i t_i^A$ (each resulting in a different value for T). Furthermore, from (3.3), we have that whenever $\mathsf{BAD}_{\mathsf{A}}(T) = 1$ (which by assumption happens with probability at least $1/6400$), conditioning on E causes X_{K_J} to be a biased bit and hence $H(X_{K_J}) \leq H(0.99/2)$. When $\mathsf{BAD}_{\mathsf{A}}(T) = 0$, which occurs with probability at most $1 - 1/6400$ by assumption, $H(X_{K_J})$ can be trivially bounded from above by 1. Using these facts, we can upper bound the entropy of $(Z|E)$ as follows:

$$\begin{aligned} H(Z|E) &\leq \sum_i H(Z_i|E) && \text{[By subadditivity of entropy]} \\ &\leq r\sqrt{n}\left(\frac{H(0.99/2)}{6400} + \left(1 - \frac{1}{6400}\right)\right). \end{aligned}$$

Combining the upper and lower bounds on $H(Z|E)$, we get

$$\delta n \geq (1 - H(0.99/2) - o(1)) \cdot \frac{r\sqrt{n}}{6400}.$$

Thus, if $\delta > 0$ is small enough we get a contradiction. ◀

The following claim shows that a version of Lemma 3.1 is true when $\mathsf{BAD}(t) = 0$. The proofs of this claim and the subsequent claim differ significantly from the proofs of the corresponding claims in Razborov's result [18] of linear lower bound for set-disjointness. This is because we need to consider several sub-events of U_1. Our arguments are more general and in fact can also be used in the context of set-disjointness.

▶ **Claim 3.4.** *Let n be large enough. If $\mathsf{BAD}(t) = 0$, then,*

$$\mu(U_1 \cap W \mid T = t) \leq \frac{16}{3(0.99)^2}\mu(U_2 \cap W \mid T = t) + \frac{16}{(0.99)^2}\mu(U_0 \cap W \mid T = t) + 2^{-\delta n/2}.$$

The following claim argues that not much probability is lost when $\mathsf{BAD}(T) = 1$.

▶ **Claim 3.5.** *Let n be large enough. Then,*

$$\mathop{\mathbb{E}}_{t \leftarrow T}[\mu(U_1 \cap W \mid T = t) \cdot \mathsf{BAD}(t)] \leq \frac{1}{100} \cdot \mathop{\mathbb{E}}_{t \leftarrow T}[\mu(W \cap U_1 \mid T = t)] + 2^{-\delta n + 3}.$$

For want of space, the proofs of Claims 3.4–3.5 are deferred to the full version of the paper [5].

Lemma 3.1 follows by combining Claim 3.4 and Claim 3.5 as follows.

$$\begin{aligned}
&0.99\mu(U_1 \cap W)\\
=&0.99 \mathop{\mathbb{E}}_{t \leftarrow T}[\mu(U_1 \cap W \mid T = t)]\\
\leq& \mathop{\mathbb{E}}_{t \leftarrow T}[\mu(U_1 \cap W \mid T = t) \cdot (1 - \mathsf{BAD}(t))] + 2^{-\delta n + 3} \qquad \text{(from Claim 3.5)}\\
\leq& \mathop{\mathbb{E}}_{t \leftarrow T}\left[\left(\frac{16\mu(U_2 \cap W \mid T = t)}{3(0.99)^2} + \frac{16\mu(U_0 \cap W \mid T = t)}{(0.99)^2} + 2^{-\delta n/2}\right)(1 - \mathsf{BAD}(t))\right]\\
&\qquad + 2^{-\delta n + 3} \qquad \text{(from Claim 3.4)}\\
\leq& \frac{16}{3(0.99)^2} \cdot \mu(U_2 \cap W) + \frac{16}{(0.99)^2} \cdot \mu(U_0 \cap W) + 2^{-\delta n/2 + 1}
\end{aligned}$$

◀

Acknowledgements. We thank Jaikumar Radhakrishnan for several useful discussions and the anonymous reviewers for useful comments.

References

1. Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Computer and System Sciences*, 68(4):702–732, June 2004. (Preliminary version in *43rd FOCS*, 2002). doi:10.1016/j.jcss.2003.11.006.
2. Boaz Barak, Mark Braverman, Xi Chen, and Anup Rao. How to compress interactive communication. In *Proc. 42nd ACM Symp. on Theory of Computing (STOC)*, pages 67–76, 2010. doi:10.1145/1806689.1806701.
3. Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of Gap-Hamming-distance. *SIAM J. Computing*, 41(5):1299–1317, 2012. (Preliminary version in *43rd STOC*, 2011). arXiv:1009.3460, doi:10.1137/120861072.
4. Amit Chakrabarti, Yaoyun Shi, Anthony Wirth, and Andrew Chi-Chih Yao. Informational complexity and the diect sum problem for simultaneous message complexity. In *Proc. 42nd IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 270–278, 2001. doi:10.1109/SFCS.2001.959901.
5. Prahladh Harsha and Rahul Jain. A strong direct product theorem for the tribes function via the smooth-rectangle bound. 2013. arXiv:1302.0275.
6. Rahul Jain and Hartmut Klauck. The partition bound for classical communication complexity and query complexity. In *Proc. 25th IEEE Conference on Computational Complexity*, pages 247–258, 2010. arXiv:0910.4266, doi:10.1109/CCC.2010.31.
7. Rahul Jain, Attila Pereszlényi, and Penghui Yao. A direct product theorem for the two-party bounded-round public-coin communication complexity. In *Proc. 53th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 167–176, 2012. arXiv:1201.1666, doi:10.1109/FOCS.2012.42.

8 Rahul Jain and Penghui Yao. A strong direct product theorem in terms of the smooth rectangle bound. 2012. arXiv:1209.0263.

9 T. S. Jayram, Swastik Kopparty, and Prasad Raghavendra. On the communication complexity of read-once AC^0 formulae. In *Proc. 24th IEEE Conference on Computational Complexity*, pages 329–340, 2009. doi:10.1109/CCC.2009.39.

10 T. S. Jayram, Ravi Kumar, and D. Sivakumar. Two applications of information complexity. In *Proc. 35th ACM Symp. on Theory of Computing (STOC)*, pages 673–682, 2003. doi:10.1145/780542.780640.

11 Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992. (Preliminary version in *2nd Structure in Complexity Theory Conference*, 1987). doi:10.1137/0405044.

12 Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Comput. Complexity*, 5(3/4):191–204, 1995. (Preliminary version in *6th Structure in Complexity Theory Conference*, 1991). doi:10.1007/BF01206317.

13 Iordanis Kerenidis, Sophie Laplante, Virginie Lerays, Jérémie Roland, and David Xiao. Lower bounds on information complexity via zero-communication protocols and applications. In *Proc. 53th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 500–509, 2012. arXiv:1204.1505, doi:10.1109/FOCS.2012.68.

14 Hartmut Klauck. A strong direct product theorem for disjointness. In *Proc. 42nd ACM Symp. on Theory of Computing (STOC)*, pages 77–86, 2010. arXiv:0908.2940, doi:10.1145/1806689.1806702.

15 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997. doi:10.2277/052102983X.

16 Nikos Leonardos and Michael Saks. Lower bounds on the randomized communication complexity of read-once functions. *Comput. Complexity*, 19(2):153–181, 2010. (Preliminary version in *24th IEEE Conference on Computational Complexity*, 2009). doi:10.1007/s00037-010-0292-2.

17 Stephen Ponzio, Jaikumar Radhakrishnan, and Srinivasan Venkatesh. The communication complexity of pointer chasing. *J. Computer and System Sciences*, 62(2):323–355, 2001. (Preliminary version in *31st STOC*, 1999). doi:10.1006/jcss.2000.1731.

18 Alexander A. Razborov. On the distributional complexity of disjointness. *Theoretical Comp. Science*, 106(2):385–390, 1992. doi:10.1016/0304-3975(92)90260-M.

19 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proc. 11th ACM Symp. on Theory of Computing (STOC)*, pages 209–213, 1979. doi:10.1145/800135.804414.

20 Andrew Chi-Chih Yao. Lower bounds by probabilistic arguments (extended abstract). In *Proc. 24th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 420–428, 1983. doi:10.1109/SFCS.1983.30.

Inapproximability of Rainbow Colouring

L. Sunil Chandran and Deepak Rajendraprasad*

Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India
{sunil, deepakr}@csa.iisc.ernet.in

Abstract

A *rainbow colouring* of a connected graph G is a colouring of the edges of G such that every pair of vertices in G is connected by at least one path in which no two edges are coloured the same. The minimum number of colours required to rainbow colour G is called its *rainbow connection number*. Chakraborty, Fischer, Matsliah and Yuster have shown that it is NP-hard to compute the rainbow connection number of graphs [J. Comb. Optim., 2011]. Basavaraju, Chandran, Rajendraprasad and Ramaswamy have reported an $(r + 3)$-factor approximation algorithm to rainbow colour any graph of radius r [Graphs and Combinatorics, 2012]. In this article, we use a result of Guruswami, Håstad and Sudan on the NP-hardness of colouring a 2-colourable 4-uniform hypergraph using constantly many colours [SIAM J. Comput., 2002] to show that for every positive integer k, it is NP-hard to distinguish between graphs with rainbow connection number $2k + 2$ and $4k + 2$. This, in turn, implies that there cannot exist a polynomial time algorithm to rainbow colour graphs with less than twice the optimum number of colours, unless $\mathsf{P} = \mathsf{NP}$.

The authors have earlier shown that the rainbow connection number problem remains NP-hard even when restricted to the class of chordal graphs, though in this case a 4-factor approximation algorithm is available [COCOON, 2012]. In this article, we improve upon the 4-factor approximation algorithm to design a linear-time algorithm that can rainbow colour a chordal graph G using at most 3/2 times the minimum number of colours if G is bridgeless and at most 5/2 times the minimum number of colours otherwise. Finally we show that the rainbow connection number of bridgeless chordal graphs cannot be polynomial-time approximated to a factor less than 5/4, unless $\mathsf{P} = \mathsf{NP}$.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases rainbow connectivity, rainbow colouring, approximation hardness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.153

1 Introduction

When a network (transport, communication, social, etc) is modelled as a graph, connectivity gives a way of quantifying its robustness. Due to the diverse application scenarios and manifold theoretical interests, many variants of the connectivity problem have been studied. One typical case is when there are different possible types of connections (edges) between nodes and additional restrictions on connectivity based on the types of edges that can be used in a path. In this case we can model the network as an edge-coloured graph. One natural restriction to impose on connectivity is that any two nodes should be connected by a path in which no edge of the same type (colour) occurs more than once. This is precisely the property

* Supported by Microsoft Research India PhD fellowship

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 153–162

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

called *rainbow connectivity*. Such a restriction for the paths can arise, for instance, in routing packets in a cellular network with transceivers that can operate in multiple frequency bands or in routing secret messages between security agencies using different handshaking passwords in different links [18, 5]. The problem was formalised in graph theoretic terms by Chartrand, Johns, McKeon, and Zhang [10].

An *edge colouring* of a graph is a function from its edge set to the set of natural numbers. A path in an edge coloured graph with no two edges sharing the same colour is called a *rainbow path*. An edge coloured graph is said to be *rainbow connected* if every pair of vertices is connected by at least one rainbow path. Such a colouring is called a *rainbow colouring* of the graph. A rainbow colouring using minimum possible number of colours is called *optimal*. The minimum number of colours required to rainbow colour a connected graph G is called its *rainbow connection number*, denoted by $rc(G)$. For example, the rainbow connection number of a complete graph is 1, that of a path is its length, that of an even cycle is half its length, and that of a tree is its number of edges. Note that disconnected graphs cannot be rainbow coloured and hence their rainbow connection number is left undefined. Any connected graph can be rainbow coloured by giving distinct colours to the edges of a spanning tree of the graph. Hence the rainbow connection number of any connected graph is less than its number of vertices. It is trivial to see that that $rc(G)$ is at least the diameter of G. It is easy to see that no two bridges in a graph can get the same colour under a rainbow colouring and hence $rc(G)$ is lower bounded by the number of bridges in the G.

While formalising the concept of rainbow colouring, Chartrand et al. also determined the rainbow connection number for some special graphs [10]. Subsequently, there have been various investigations towards finding good upper bounds for rainbow connection number in terms of other graph parameters [4, 20, 6, 3] and for many special graph classes [19, 6, 2]. Behaviour of rainbow connection number in random graphs is also well studied [4, 15, 21, 13]. A basic introduction to the topic can be found in Chapter 11 of the book *Chromatic Graph Theory* by Chartrand and Zhang [9] and a survey of most of the recent results in the area can be found in the article by Li and Sun [18] and also in their monograph *Rainbow Connection of Graphs* [17].

The first result showing the computational difficulty of the rainbow colouring problem was due to Chakraborty, Fischer, Matsliah, and Yuster [5]. They showed that it is NP-hard to compute the rainbow connection number of an arbitrary graph. In particular, it was shown that the problem of deciding whether a graph can be rainbow coloured using 2 colours is NP-complete. Later, Ananth, Nasre, and Sarpatwar [1] complemented the above result and now we know that for every integer $k \geq 2$, it is NP-complete to decide whether a given graph can be rainbow coloured using k colours. To the best of our knowledge, hitherto no results are reported on the hardness of approximation of rainbow connection number.

In this article we show that, for any positive integer k, it is NP-hard to distinguish between graphs with rainbow connection number $2k + 2$ and $4k + 2$ (Corollary 3). This precludes the possibility of having a polynomial time algorithm to rainbow colour graphs using less than twice the optimum number of colours, unless P = NP (Corollary 4). The proof is by a reduction from a hypergraph colouring problem. It was shown by Guruswami, Håstad, and Sudan that, for any constant c, it is NP-hard to colour a 2-colourable 4-uniform hypergraph with c colours [14]. It follows almost directly from their arguments that, for any constant $c \geq 2$, given a 4-uniform hypergraph H with chromatic number either 2 or more than c, it is NP-complete to decide whether the chromatic number of H is 2. We reduce this problem to a problem of determining whether a given bridgeless bipartite graph G with $rc(G) \in \{2k + 2, 4k + 2\}$ has $rc(G) = 2k + 2$ for any constant $k \leq (c - 1)/4$ (Theorem 2).

Currently, the best approximation guarantee for this problem on general graphs is given by an $O(nm)$-time $(r+3)$-factor approximation algorithm for rainbow colouring a graph with radius r by Basavaraju et al. [3]. The gap between the algorithmic guarantee and the hardness of approximation shown here invites a deeper investigation into the problem. We are inclined towards believing that there might exist a polynomial-time constant-factor approximation algorithm for this problem.

One large subclass of graphs for which a constant-factor approximation algorithm is known to exist for the rainbow connection number problem is the class of chordal graphs. Chandran et al. have shown that any bridgeless chordal graph can be rainbow coloured using at most $3r$ colours, where r is the radius of the graph [6]. The proof given there is constructive and can be easily extended to a polynomial-time algorithm which will colour any chordal graph G with b bridges and radius r using at most $3r+b$ colours. Since $\max\{r, b\}$ is a lower bound for $rc(G)$, this immediately gives us a 4-factor approximation algorithm. We modify this algorithm slightly using a technique used by Li and Dong in [12] to reuse bridge colours and design a linear-time algorithm to rainbow colour chordal graphs (Algorithm 1). We then do a careful analysis using distance properties of chordal graphs to show that for any chordal graph G with diameter at least 3, the above algorithm rainbow colours G using at most $\frac{5}{2}rc(G)$ colours. Further, it follows that if G is bridgeless, then this algorithm uses at most $\frac{3}{2}rc(G)+3$ colours only (Corollary 6). This brings to table the question of approximation hardness of the problem when restricted to chordal graphs.

We have shown, in an earlier work, that for every $k \geq 3$, the problem of deciding whether a given graph can be rainbow coloured using k colours remains NP-complete even when restricted to the class of chordal graphs [7]. In this article we go further and show an inapproximability result for the case. From the same hypergraph colouring problem that we used for the previous reduction, we give a different and more involved reduction to show that, for any positive integer k, given a bridgeless chordal graph G with $rc(G) \in \{4k, 5k\}$, it is NP-complete to decide whether $rc(G) = 4k$ (Corollary 8). As before, this precludes the possibility of having a polynomial-time algorithm to rainbow colour bridgeless chordal graphs with less than 5/4 times the optimal number of colours (Corollary 9). This should be contrasted with the case of split graphs, which are a proper subclass of chordal graphs. We have shown in an earlier work that it is NP-hard to determine the rainbow connection number of split graphs, but nevertheless designed a linear-time algorithm which will rainbow colour any split graph G using at most $rc(G)+1$ colours [7].

2 Preliminaries

First we recall some standard graph theoretic terminology that we will use in this article. All graphs considered here are finite, simple and undirected. For a graph G, we use $V(G)$ and $E(G)$ to denote its vertex set and edge set respectively. Let G be a connected graph. The *distance* between two vertices u and v in G, denoted by $d_G(u, v)$ is the length of a shortest path between them in G, where the *length* of a path is the number of edges in that path. The *eccentricity* of a vertex v is $ecc_G(v) := \max_{x \in V(G)} d_G(v, x)$. The *diameter* of G is $diam(G) := \max_{x \in V(G)} ecc_G(x)$ and *radius* of G is $radius(G) := \min_{x \in V(G)} ecc_G(x)$. A vertex is called *central* in G if its eccentricity is equal to the radius of G. The *boundary of G with respect to a vertex x* is $\{v \in V(G) : d_G(x, v) = ecc_G(x)\}$. If G has a unique central vertex x, then the *boundary* of G with respect to x will be referred to as just the *boundary* of G. The distance of a vertex v from a subset S of $V(G)$ is $\min_{s \in S} d_G(v, s)$. The *neighbourhood* $N(v)$ of a vertex v is the set of vertices adjacent to v but not including v. A *bridge* in a

connected graph G is an edge of G whose deletion disconnects G. A connected graph is called *bridgeless* if it has no bridges. A graph is called *chordal*, if it has no induced cycle of length greater than 3. A graph is called a *split graph*, if its vertex set can be partitioned into a clique and an independent set. It is easy to see that split graphs form a subclass of chordal graphs.

Next we define hypergraphs and their colouring. A *hypergraph* H is a tuple (V, E), where V is a finite set and E is a collection of subsets of V. Elements of V and E are called vertices and (hyper)edges respectively. A vertex of H is called *isolated* if its not part of any edge of H. The hypergraph H is called *r-uniform* if $|e| = r$ for every $e \in E$. Given a hypergraph $H(V, E)$ and a colouring $c : V \to \mathbb{N}$, an edge is called *k-coloured* if it contains vertices of k different colours. An edge is called *monochromatic* if it is 1-coloured. The colouring c is called *proper* if no edge in E is monochromatic under c. The minimum number of colours required to properly colour H is called its *chromatic number* and is denoted by $\chi(H)$.

In the two approximation hardness results that we prove in this article, we make use of the following deep result of Guruswami, Håstad, and Sudan [14].

▶ **Theorem 1** (Guruswami, Håstad, Sudan)**.** *For every constant $c \geq 2$, given a 4-uniform hypergraph H with either $\chi(H) = 2$ or $\chi(H) > c$, it is* NP*-complete to decide whether $\chi(H) = 2$.*

Theorem 1 is not explicitly stated as above in their work, but it will follow easily from Corollary 4.3 in [14]. Corollary 4.3 in [14] is a result about a problem called 4-set splitting which easily translates into the 4-uniform hypergraph colouring problem as noted in the proof of Theorem 4.4 there.

Given a minimisation problem P, an (α, β)-approximation algorithm for P is an algorithm whose output on every instance I to P is a solution of P for I with cost at most $\alpha x + \beta$, where x denotes the cost of an optimum solution of P for I. If α and β are independent of the instance I, then the (α, β)-approximation algorithm is called a *constant factor approximation algorithm* for P. An $(\alpha, 0)$-approximation algorithm will also be referred to as an α-factor algorithm.

Throughout this article, the shorthand $[n]$ denotes the set $\{1, \ldots, n\}$. The cardinality of a set S is denoted by $|S|$.

3 General graphs: Hardness of approximation

▶ **Theorem 2.** *For every positive integer k, the first problem below* (P1) *is polynomial-time reducible to the second* (P2)*.*

P1. *Given a 4-uniform hypergraph H with either $\chi(H) = 2$ or $\chi(H) \geq 4k+2$, decide whether $\chi(H) = 2$.*

P2. *Given a bipartite bridgeless graph G with $rc(G) \in \{2k + 2, 4k + 2\}$, decide whether $rc(G) = 2k + 2$.*

Proof. Let k be arbitrary and $H = (V_H, E_H)$ be the 4-uniform hypergraph given as an instance of P1. Since isolated vertices do not affect the chromatic number of a hypergraph, we can safely assume that H does not contain an isolated vertex. We construct a bridgeless graph G from H as follows.

First we construct a graph G'_0 with $V(G'_0) = \{x\} \cup V_H \cup E_H$ and $E(G'_0) = E_1 \cup E_2$ where $E_1 = \{\{x, v\} : v \in V_H\}$ and $E_2 = \{\{v, e\} : v \in V_H, e \in E_H, v \in e \text{ in } H\}$. From G'_0 we construct G_0 by replacing each edge $\{v, e\}$ in E_2 with a new k-length path between v and e. Let $G_1, \ldots, G_{4k+2}$ be copies of G_0. We obtain our desired graph G by identifying the vertex

x as common in every G_i, $i \in [4k+2]$. It is easy to see that G is bipartite, bridgeless and has radius $k+1$ with x as the unique central vertex. The $(4k+2)|V_H|$ neighbours of x in G are the vertices corresponding to the vertices of H and their collection is denoted by V_{V_H}. Similarly the $(4k+2)|E_H|$ vertices in G at distance $k+1$ from x (the boundary vertices of G) are those corresponding to the hyperedges of H and their collection is denoted by V_{E_H}. It is evident that the construction of G from H takes time at most polynomial in size of H.

We prove the theorem by establishing the following two claims. The converses of both the claims are also true since the converse of one is the contrapositive of the other.

Claim 2.1. *If* $\chi(H) = 2$ *then* $rc(G) = 2k+2$.

Claim 2.2. *If* $\chi(H) \geq 4k+2$ *then* $rc(G) = 4k+2$.

Since the diameter of G is $2k+2$ we see that $rc(G) \geq 2k+2$ always. Hence to prove Claim 2.1 it suffices to show that $rc(G) \leq 2k+2$ whenever $\chi(H) = 2$. Let $c_H : V_H \to \{a, b\}$ be a proper 2-colouring of H. For each vertex $v \in V_{V_H}$ consider the subtree T_v of G formed by all the $(k+1)$-length paths starting with the edge $\{x, v\}$ and reaching some vertex $e \in V_{E_H}$. We remark that, in this case, e will correspond to a hyperedge of H which contains v. If $c_H(v) = a$ (b) then colour every edge of T_v at a distance j from x in T_v with colour a_j (b_j) for each $j \in \{0, \ldots, k\}$. Note that all the subtrees considered above are pairwise edge-disjoint and hence every edge gets a unique colour. We argue that the above edge-colouring obtained for G is a rainbow colouring. First observe that since c_H is a proper colouring of H, every vertex u in G is part of a $(2k+2)$-cycle C_u of G containing x and with edge colours $a_0, \ldots, a_k, b_k, \ldots, b_0$ in that order starting from an edge incident on x. Given any two vertices $u, v \in V(G)$, it is not difficult to see that at least one of the four possible walks from u to v along C_u, x and C_v contains a rainbow path. Hence Claim 2.1.

In order to prove Claim 2.2, we first show the easy fact that G can always be rainbow coloured using $4k+2$ colours. Let H' be a hyperedge-maximal sub-hypergraph of H which can be properly 2-coloured. Let $V_{E_{H'}}$ be the set of $(4k+2)|E(H')|$ vertices in $V_{E_H} \subset V(G)$ which correspond to the hyperedges in H'. Let G' be the induced subgraph of G consisting of the vertices on all the $(k+1)$-length paths from x to $V_{E_{H'}}$. The maximality of H' ensures every vertex v of H is part of some hyperedge in H'. Hence $V_{V_H} \subset V(G')$ and we can rainbow colour G' using the $2k+2$ colours from $\{a_0, \ldots, a_k, b_0, \ldots, b_k\}$ as we did while showing Claim 2.1. Now contract entire G' in G to a single vertex y to obtain a minor G'' of G. The graph G'' consists of the vertex y and separate sets of 4 edge-disjoint k-length paths from y to each vertex in $V_{E_H} \setminus V_{E_{H'}}$. It is easy to edge-colour G'' using the $2k$ colours from $\{c_1, \ldots, c_k, d_1, \ldots, d_k\}$ so that every vertex u of G'' is part of a $2k$-cycle of G'' containing y and with edge colours $c_1, \ldots, c_k, d_k, \ldots, d_1$ in that order starting from an edge incident on y. This makes G'' rainbow connected as earlier. Now we can combine the above rainbow colourings of G' and G'' in the obvious manner to obtain a rainbow colouring of G using $4k+2$ colours.

Finally we show that if $\chi(H) > 4k+1$ then $rc(G) > 4k+1$. For the sake of contradiction, assume that $\chi(H) > 4k+1$ and $rc(G) \leq 4k+1$. Let c_G be a rainbow colouring of G using $rc(G)$ colours. From c_G we obtain $(4k+2)$ different vertex colourings $\{c_H^1, \ldots, c_H^{4k+2}\}$ of H as follows. For a vertex v of H and $i \in [4k+2]$, let v_i be the vertex in G_i corresponding to v. Then set $c_H^i(v) = c_G(\{x, v_i\})$. Since every one of the colourings thus obtained uses at most $4k+1$ colours and $\chi(H) > 4k+1$, none of these colourings is proper. That is, for each $i \in [4k+2]$, there exists a hyperedge $e_i \in E(H)$ which is monochromatic under c_H^i. Among them, by pigeonhole principle, there exist two hyperedges e_s and e_t, $s, t \in [4k+2]$ and $s \neq t$, such that $c_H^s(v_s) = c_H^t(v_t), \forall (v_s, v_t) \in e_s \times e_t$. Let V_s (V_t) be the set of four

vertices in G_s (G_t) corresponding to the vertices of H belonging to the edge e_s (e_t) and let f_s (f_t) be the vertex in G_s (G_t) corresponding to the hyperedge e_s (e_t). We see that every edge in $\{\{x, v\} : v \in V_s \cup V_t\}$ gets the same colour under c_G. Hence none of the 16 shortest paths between f_s and f_t in G is rainbow coloured under c_G. Any other path between f_s and f_t is of length at least $4k + 2$ and since c_G uses at most $4k + 1$ colours it is not rainbow coloured. This contradicts the fact that c_G was a rainbow colouring of G. ◀

Since Problem P1 is known to be NP-hard (Theorem 1), so is Problem P2. Further, it is easy to see that problem P2 is in NP. Hence the following corollary and hardness of approximation.

▶ **Corollary 3.** *For every positive integer k, given a bipartite bridgeless graph G with $rc(G) \in \{2k + 2, 4k + 2\}$, it is* NP*-complete to decide whether $rc(G) = 2k + 2$.*

▶ **Corollary 4.** *For any $\alpha, \beta \in \mathbb{R}$ with $\alpha > 0$ there does not exist a polynomial time $(2 - \alpha, \beta)$ approximation algorithm for determining the rainbow connection number of graphs unless* P = NP*.*

Proof. Suppose for some $\alpha > 0$, there exists a polynomial time $(2 - \alpha, \beta)$ approximation algorithm $\mathcal{A}$ to determine the rainbow connection number of graphs. Choose a positive integer k large enough so that $(2 + \beta)/(2k + 2) < \alpha$. If the input graph G to $\mathcal{A}$ has $rc(G)$ at most $2k+2$, then $\mathcal{A}$ will certify that $rc(G) \leq (2-\alpha)(2k+2)+\beta < (2-\frac{2+\beta}{2k+2})(2k+2)+\beta = 4k+2$. This contradicts Corollary 3. ◀

4 Chordal graphs: An approximation algorithm and inapproximability

▶ **Theorem 5.** *For every connected chordal graph G with b bridges and radius r, Algorithm 1* (ColourChordalGraph) *returns a rainbow colouring of G using at most $3r+\max\{0, b-3\}$ colours in linear time.*

Proof. Let B_i be the number of edges between V_{i-1} and V_i which are bridges of G. First we argue that $\forall i \in [r]$, and $\forall v \in V_i$, if $N(v) \cap V_{i-1} = \{u\}$ and $N(v) \cap V_i = \emptyset$, then the edge $e = \{v, u\}$ is a bridge in G. Otherwise, a smallest cycle C containing the edge e has vertices from V_{i-1} and V_{i+1} and hence should have a length at least 4. The cycle C does not have any chords since it is a smallest cycle containing e. This contradicts the fact that G is chordal. Hence for each i, b_i counts the number of bridges of G between V_{i-1} and V_i and thus the number of colours used in round i is at most $\max\{3, B_i\}$. Hence in total, Algorithm 1 uses at most $\sum_{i=1}^{r} \max\{3, B_i\} \leq 3r + \max\{0, b - 3\}$ colours.

Next we show that c_G makes G rainbow connected. For each $i \in [r]$, let $k_i = \max\{3, B_i\}$ and $K_i = \{c_{i,1}, \ldots, c_{i,k_i}\}$. In c_G, every bridge in G between V_i and V_{i-1} gets a distinct colour from K_i. Every vertex $v \in V_i$ which is not an end point of a bridge from V_{i-1} to V_i is contained in a 2- or 3-length path starting from V_{i-1} touching v and going back to V_{i-1} which is coloured $(c_{i,1}, c_{i,2})$ or $(c_{i,1}, c_{i,3}, c_{i,2})$, respectively. Hence for any two vertices $u, v \in V_i$, there exist rainbow paths P_u from u to V_{i-1} and P_v from v to V_{i-1} such that the colours assigned by c_G to P_u and P_v form disjoint subsets of K_i. Since this is true for each $i \in [r]$, for any two distinct vertices $u, v \in V(G)$, there exist two rainbow paths P_u from u to x and P_v from v to x such that the set of colours assigned to P_u and P_v by c_G are disjoint till the first common vertex in these two paths. Hence $P_u \cup P_v$ contains a u–v rainbow path.

Finally we show that Algorithm 1 runs in $O(m)$ time where m is the number of edges in the input graph. Chepoi and Dragan have designed an algorithm which finds a central vertex of a chordal graph in $O(m)$ time [11]. Hence the initialisation steps in Algorithm 1 runs in

Algorithm 1: ColourChordalGraph

Data: $G(V, E)$, a connected chordal graph
Result: A rainbow colouring c_G of G

Initialisation:
$x \leftarrow$ a central vertex of G
$r \leftarrow$ radius of G
$V_i \leftarrow \{v \in V(G) : d_G(v, x) = i\}$ for each $i \in [r]$
$b_i \leftarrow 0$ for each $i \in [r]$ `// ` b_i `counts the number of` V_{i-1} `to` V_i `bridges in` G.

for $i = 1$ **to** r **do**
  $c_G(\{v, v'\}) = c_{i,3}$, $\forall v, v' \in V_i$ and $\{v, v'\} \in E(G)$
  foreach $v \in V_i$ *such that* $|N(v) \cap V_{i-1}| \geq 2$ **do**
    Let $\{u_1, \ldots, u_t\} = N(v) \cap V_{i-1}$
    $c_G(\{v, u_1\}) = c_{i,1}$
    $c_G(\{v, u_s\}) = c_{i,2}, \forall s \in \{2, \ldots, t\}$
  end
  foreach $v \in V_i$ *such that* $|N(v) \cap V_{i-1}| = 1$ **do**
    Let u be the single vertex in $N(v) \cap V_{i-1}$
    if $N(v) \cap V_i = \emptyset$ **then**
      $b_i = b_i + 1$ `// ` $\{v, u\}$ `is a bridge in` G
      $c_G(\{v, u\}) = c_{i,b_i}$
    else if $\exists v' \in N(v) \cap V_i$ *and* $\exists u' \in N(v') \cap V_{i-1}$ *such that* $c_G(v', u') = c_{i,1}$ **then**
      $c_G(\{v, u\}) = c_{i,2}$
    else
      $c_G(\{v, u\}) = c_{i,1}$
    end
  end
end

return c_G

linear time. Each for-loop visits each vertex in G at most once. If we flag every vertex $v \in V_i$ when it gets a $c_{i,1}$-coloured edge to V_{i-1}, the algorithm, when it visits a vertex, needs to examine only its neighbours and incident edges. Hence the total running time in $O(m)$. ◀

Chang and Nemhauser have shown that the radius r and diameter d of any chordal graph are constrained by the inequality $r \leq d/2 + 1$ [8]. Hence we get the following corollary to Theorem 5.

▶ **Corollary 6.** *If G is a connected chordal graph with diameter at least 3, then Algorithm 1 returns a rainbow colouring of G using at most $\frac{5}{2}rc(G)$ colours. If G is bridgeless then Algorithm 1 uses only $\frac{3}{2}rc(G) + 3$ colours.*

Proof. Let G be a chordal graph with b bridges, diameter $d \geq 3$ and radius r. Let $a(G)$ be the number of colours used by Algorithm 1 in rainbow colouring G. Then by Theorem 5 and the bound by Chang and Nemhauser, we have

$$a(G) \leq \frac{3}{2}d + 3 + \max\{0, b - 3\} = \frac{3}{2}d + \max\{3, b\} \quad (1)$$

Since $d, b \leq rc(G)$ (easy observations) and $d \geq 3$ (by assumption), we get $a(G) \leq \frac{3}{2}rc(G) + rc(G) = \frac{5}{2}rc(G)$. Further if G is bridgeless then $a(G) \leq \frac{3}{2}rc(G) + 3$. ◀

▶ Remark. The requirement that $diam(G) \geq 3$ in Corollary 6 is a consequence of the generality of Algorithm 1 and not due to any inherent difficulty in the problem. If $diam(G) = 1$, then G is a clique and hence a colouring which gives every edge of G the same colour is a rainbow colouring. Li, Li, and Liu have shown that any graph G with diameter 2 has $rc(G) \leq 5$ if it is bridgeless, and $rc(G) \leq b + 2$ if it has b bridges [16].

In the context of the above approximation algorithm, we now state and prove the inapproximability result on chordal graphs.

▶ **Theorem 7.** *For every positive integer k, the first problem below* (P1) *is polynomial-time reducible to the second* (P2).

P1. *Given a 4-uniform hypergraph H with either $\chi(H) = 2$ or $\chi(H) \geq 5k$, decide whether $\chi(H) = 2$.*

P2. *Given a bridgeless chordal graph G with $rc(G) \in \{4k, 5k\}$, decide whether $rc(G) = 4k$.*

Proof. Let k be arbitrary and $H = (V_H, E_H)$ be the 4-uniform hypergraph given as an instance of P1. Since isolated vertices do not affect the chromatic number of a hypergraph, we can safely assume that H does not contain an isolated vertex. We construct a bridgeless chordal graph G from H as follows.

First we construct a graph G_H with $V(G_H) = \{x\} \cup V_H \cup E_H$ and $E(G_H) = E_1 \cup E_2 \cup E_3$ where $E_1 = \{\{x, v\} : v \in V_H\}$, $E_2 = \{\{v, e\} : v \in V_H, e \in E_H, v \in e \text{ in } H\}$, and $E_3 = \{\{v, v'\} : v, v' \in V_H\}$. The graph G_H is easily verified to be bridgeless and chordal. In fact, it is a split graph with V_H as the clique and $\{x\} \cup E_H$ as the independent set. Now we construct a graph G_1 by taking $(5k)^k + 1$ copies of G_H and identifying x as common in every copy. The common vertex x is relabelled as x_0. The vertex x_0 will serve as the unique central vertex for G_1 and all the $G_i, i \in \{2, \ldots, k\}$, to be constructed next. Once we have G_i for some $i < k$, we construct G_{i+1} by joining to every vertex e in the boundary of G_i, another copy of G_H identifying $x \in V(G_H)$ with e. Note that the boundary of G_i is the set of vertices at distance of $radius(G_i) = 2i$ from x_0, which in our case, turns out to be all the vertices corresponding to some hyperedge of H in a copy of G_H added in the i-th step. The graph $G = G_k$ constructed this way is our desired graph. Since a graph obtained from two bridgeless chordal graphs by identifying a single vertex as common to both is bridgeless and chordal, we see that G is a bridgeless chordal graph as desired. G has diameter $4k$ and radius $2k$ with x_0 as the unique central vertex. If $h = |E_H|$, then G has $\left((5k)^k + 1\right)\left(1 + h + \cdots + h^{k-1}\right)$ copies of G_H and hence the construction of G takes only a time polynomial in the size of H.

We prove the theorem by establishing the following two claims. The converses of both the claims are also true since the converse of one is the contrapositive of the other.

Claim 7.1. *If $\chi(H) = 2$ then $rc(G) = 4k$.*

Claim 7.2. *If $\chi(H) \geq 5k$ then $rc(G) = 5k$.*

Since the diameter of G is $4k$, it follows that $rc(G) \geq 4k$ always. Hence to prove Claim 7.1 it suffices to show that $rc(G) \leq 4k$ whenever $\chi(H) = 2$. We define an edge-colouring c_G of G based on a red-blue colouring c_H of H by describing the colours assigned to the edges of each copy of G_H in G. Let G_H^i be a copy of G_H added at the i-the level, that is, $x \in G_H^i$ is at a distance $2i-2$ from x_0 in G. Recall that $E(G_H^i) = E_1 \cup E_2 \cup E_3$, where $E_1 = \{\{x, v\} : v \in V_H\}$, $E_2 = \{\{v, e\} : v \in V_H, e \in E_H, v \in e \text{ in } H\}$, and $E_3 = \{\{v, v'\} : v, v' \in V_H\}$. An edge

$\{x, v\} \in E_1$ is given colour a_i (c_i) if the vertex corresponding to v in H is coloured red (blue) in c_H. An edge $\{v, e\} \in E_2, v \in V_H$ is given colour b_i (d_i) if the vertex corresponding to v in H is coloured red (blue) in c_H. This ensures that every vertex in G^i_H is part of a 4-cycle of G^i_H containing $x \in V(G^i_H)$ and with edge colours a_i, b_i, d_i, c_i in that order starting from an edge incident on x. Colours on the edges in E_3 do not matter to us and hence we can give them any colour that is already used. This colouring c_G of G thus uses $4k$ colours and has the property that if u and v are two distinct vertices of G with $\{d_G(u, x_0), d_G(v, x_0)\} \subset \{2i - 1, 2i\}, i \in [k]$, then there exist two vertices u' and v' (not necessarily distinct) with $d_G(u', x_0) = d_G(v', x_0) = 2i - 2$ and rainbow paths P_u from u to u' and P_v from v to v' such that the colours used in P_u and P_v form two disjoint subsets of $\{a_i, b_i, c_i, d_i\}$. It is easy to see that this property ensures that c_G is a rainbow colouring of G.

If we give G as an input graph to Algorithm 1, then the colouring returned by it will use 3 colours at all odd levels and 2 colours at all even levels. Hence the total number of colours used is $5k$ which means $rc(G) \leq 5k$. Hence to prove Claim 7.2 we only need to show that if $\chi(H) \geq 5k$, then $rc(G) \geq 5k$. Assume, for the sake of contradiction, that c_G is a rainbow colouring of G using less than $5k$ colours. Every copy G'_H of G_H in G then induces a vertex colouring of c'_H of H as $c'_H(v) = c_G(\{x, v\}), \{x, v\} \in E(G'_H)$ for every $v \in V_H$. Since c'_H uses less than $5k$ colours and $\chi(H) \geq 5k$, there exists a hyperedge $e \in E_H$ that is monochromatic under c'_H, which means that the second edge in all the 2-length paths from e to x in G'_H is of the same colour. Let us call such a vertex e a *trapped vertex* of G and the common colour on the second edge of all the 2-length paths from e to x the *blocking colour* of e. Since every copy of G_H in G has at least one trapped vertex we get $(5k)^k + 1$ disjoint sequences of the form $(t_1, \ldots, t_k)$ such that t_1 is a trapped vertex in G_1, and $t_i, i \geq 2$ is a trapped vertex in a copy of G_H attached to t_{i-1}. Since we have $(5k)^k + 1$ such disjoint sequences there exists at least 2 sequences $(t_1, \ldots, t_k)$ and $(s_1, \ldots, s_k)$ which induce the same sequence of blocking colours. Hence in any rainbow path P between t_k and s_k, if $d_P(t_i, t_{i-1}) = 2$, then $d_P(s_i, s_{i-1}) \geq 3$ and vice versa. Hence the length of P is at least $5k$ and so P cannot be a rainbow path in a colouring which uses less than $5k$ colours. This contradiction proves Claim 7.2. ◀

Since Problem P1 is known to be NP-hard (Theorem 1), so is Problem P2. Further, it is easy to see that problem P2 is in NP. Hence the following corollary and hardness of approximation.

▶ **Corollary 8.** *For every positive integer k, given a bridgeless chordal graph G with $rc(G) \in \{4k, 5k\}$, it is* NP*-complete to decide whether $rc(G) = 4k$.*

▶ **Corollary 9.** *For any $\alpha, \beta \in \mathbb{R}$ with $\alpha > 0$, there does not exist a polynomial time $(5/4-\alpha, \beta)$ approximation algorithm for determining the rainbow connection number of chordal graphs unless* P = NP.

Proof. Suppose for some $\alpha > 0$, there exists a polynomial time $(5/4 - \alpha, \beta)$ approximation algorithm $\mathcal{A}$ to determine the rainbow connection number of chordal graphs. Choose a positive integer $k > \beta/4\alpha$. If the input chordal graph G to $\mathcal{A}$ has $rc(G)$ at most $4k$, then $\mathcal{A}$ will certify that $rc(G) \leq (5/4 - \alpha)4k + \beta = 5k - (4\alpha k - \beta) < 5k$. This contradicts Corollary 8. ◀

References

1. Prabhanjan Ananth, Meghana Nasre, and Kanthi K. Sarpatwar. Rainbow Connectivity: Hardness and Tractability. In *FSTTCS 2011*, LIPIcs, volume 13, pages 241–251, Schloss Dagstuhl, 2011. doi: 10.4230/LIPIcs.FSTTCS.2011.241
2. Manu Basavaraju, L. Sunil Chandran, Deepak Rajendraprasad, and Arunselvan Ramaswamy. Rainbow connection number of graph power and graph products. *Accepted for publication in Graphs and Combinatorics. Preprint arXiv:1104.4190v2 [math.CO]*, 2011.
3. Manu Basavaraju, L. Sunil Chandran, Deepak Rajendraprasad, and Arunselvan Ramaswamy. Rainbow connection number and radius. *Graphs and Combinatorics*, pages 1–11, 2012.
4. Yair Caro, Arie Lev, Yehuda Roditty, Zsolt Tuza, and Raphael Yuster. On rainbow connection. *Electron. J. Combin.*, 15(1):Research paper 57, 13, 2008.
5. Sourav Chakraborty, Eldar Fischer, Arie Matsliah, and Raphael Yuster. Hardness and algorithms for rainbow connection. *J. Comb. Optim.*, 21(3):330–347, 2011.
6. L. Sunil Chandran, Anita Das, Deepak Rajendraprasad, and Nithin M. Varma. Rainbow connection number and connected dominating sets. *Journal of Graph Theory*, 71(2):206–218, 2012.
7. L. Sunil Chandran and Deepak Rajendraprasad. Rainbow colouring of split and threshold graphs. In Joachim Gudmundsson, Julián Mestre, and Taso Viglas, editors, *Computing and Combinatorics*, volume 7434 of *LNCS*, pages 181–192. Springer, 2012. doi: 10.1007/978-3-642-32241-9_16.
8. Gerard J Chang and George L Nemhauser. The k-domination and k-stability problems on sun-free chordal graphs. *SIAM Journal on Algebraic Discrete Methods*, 5(3):332–345, 1984.
9. G. Chartrand and P. Zhang. *Chromatic Graph Theory*. Chapman & Hall, 2008.
10. Gary Chartrand, Garry L. Johns, Kathleen A. McKeon, and Ping Zhang. Rainbow connection in graphs. *Math. Bohem.*, 133(1):85–98, 2008.
11. Victor Chepoi and Feodor Dragan. Linear-time algorithm for finding a central vertex of a chordal graph. In *ESA'94, Second Annual European Symposium*, pages 159–170. Springer, 1994.
12. Jiuying Dong and Xueliang Li. Rainbow connection number, bridges and radius. *Graphs and Combinatorics*, pages 1–7, 2012.
13. A. Frieze and C.E. Tsourakakis. Rainbow connectivity of $G(n,p)$ at the connectivity threshold. *Preprint arXiv:1201.4603*, 2012.
14. Venkatesan Guruswami, Johan Håstad, and Madhu Sudan. Hardness of approximate hypergraph coloring. *SIAM Journal on Computing*, 31(6):1663–1686, 2002.
15. Jing He and Hongyu Liang. On rainbow-k-connectivity of random graphs. *Information Processing Letters*, 112(10):406–410, 2012.
16. Hengzhe Li, Xueliang Li, and Sujuan Liu. Rainbow connection of graphs with diameter 2. *Discrete Mathematics*, 312(8):1453–1457, 2012.
17. X. Li and Y. Sun. *Rainbow Connections of Graphs*. Springerbriefs in Mathematics. Springer, 2012.
18. Xueliang Li, Yongtang Shi, and Yuefang Sun. Rainbow connections of graphs: A survey. *Graphs and Combinatorics*, 29(1):1–38, 2013.
19. Xueliang Li and Yuefang Sun. Upper bounds for the rainbow connection numbers of line graphs. *Graphs and Combinatorics*, pages 1–13, 2011. 10.1007/s00373-011-1034-1.
20. Ingo Schiermeyer. Rainbow connection in graphs with minimum degree three. In *Combinatorial Algorithms*, volume 5874 of *LNCS*, pages 432–437. Springer, Berlin, 2009.
21. Yilun Shang. A sharp threshold for rainbow connection of random bipartite graphs. *Int. J. Appl. Math.*, 24(1):149–153, 2011.

Primal Infon Logic: Derivability in Polynomial Time

Anguraj Baskar[1], Prasad Naldurg[2], K. R. Raghavendra[3], and S. P. Suresh[4]

1 **Institute of Mathematical Sciences, Chennai, India**
abaskar@imsc.res.in
2 **IBM Research India, Bangalore, India**
pnaldurg@in.ibm.com
3 **International Institute of Information Technology, Bangalore, India**
rkr@iiitb.ac.in
4 **Chennai Mathematical Institute, Chennai, India**
spsuresh@cmi.ac.in

Abstract

Primal infon logic (PIL), introduced by Gurevich and Neeman in 2009, is a logic for authorization in distributed systems. It is a variant of the $(\rightarrow, \wedge)$-fragment of intuitionistic modal logic. It presents many interesting technical challenges – one of them is to determine the complexity of the derivability problem. Previously, some restrictions of propositional PIL were proved to have a linear time algorithm, and some extensions have been proved to be PSPACE-complete. In this paper, we provide an $O(N^3)$ algorithm for derivability in propositional PIL. The solution involves an interesting interplay between the sequent calculus formulation (to prove the subformula property) and the natural deduction formulation of the logic (based on which we provide an algorithm for the derivability problem).

1998 ACM Subject Classification D.4.6 Access control, F.4.1 Proof theory

Keywords and phrases Authorization logics, Intuitionistic modal logic, Proof theory, Cut elimination, Subformula property

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.163

1 Introduction

Infon logic [10, 11, 12] is a version of modal intuitionistic logic specially designed to reason about trust and delegation in authorization systems. Its application is in the domain of access control design for distributed or federated systems, where principals who request access to resources need to present a set of assertions (or certificates) that encodes their right to access. This right may be conditioned upon attribute values, or encoded as a chain of delegations. A reasoning engine examines this query and uses the presented assertions, along with any local assertions, to derive whether the access is allowed or not according to the rules of inference in an underlying logic.

The work on infon logic is situated in the larger context of authorization languages, including SecPAL [5] and DKAL [10, 11]. These languages provide constructs to specify communication of assertions between principals, and to import or derive knowledge arising from these communications. In infon logic, the basic unit of an assertion is an *infon*, which is any information that can be communicated between two principals [12]. Infons range over relation terms, e.g., *CanRead*(*Alice*, *ncfile*), which represents a right for Alice to read *ncfile*, and form the basis of access control design.

In addition to basic terms, infon logic also allows one to express authorization (and delegation) using the modal operators said and implied. To illustrate, consider an administrator who has the right to decide access to a network configuration file *ncfile*, and can authorize a user Alice the right to read the file by giving her the following assertion: *Admin* said *CanRead*(*Alice*, *ncfile*). The statement *CanRead*(*Alice*, *ncfile*) is true if Admin is trusted i.e., (*Admin* said x) $\to x$. The said operator is similar in function to the says operator in the SpeaksFor calculus [2, 14]. Assertions can be conditional, e.g., the administrator can decide that Alice can be granted this right if she owns the file: *Admin* said [*CanRead*(*Alice*, *ncfile*) **if** *Owns*(*Alice*, *ncfile*)]. A reasoning engine needs to check the validity of *Owns*(*Alice*, *ncfile*) to derive an answer to the query *CanRead*(*Alice*, *ncfile*). Delegation is captured by formulas like *Alice* said $x \to$ *Bob* said x (this would be expressed as *Alice* speaksfor *Bob* in [2]). Note that the enforcement of the authorization is decoupled from the mechanism of granting access, enabling flexible design and control.

There are many other systems for authorization whose logical cores have similarities with infon logic. For instance, the SpeaksFor calculus [14], which pioneered the logical formulation of authorization decisions, uses the says modality which is similar to our said modality. The semantics and properties of the SpeaksFor calculus were further explored by Abadi and others [1, 2]. Among other authorization logics, Binder [8], SD3 [13], Delegation Logic [15], and SecPAL [4, 5] use Datalog as basis for both syntax and semantics. DKAL is an authorization logic that extends SecPAL with constructs for specifying and reasoning about localized knowledge and targeted communication of authorization statements.

In [12], Gurevich and Neeman studied the propositional core of infon logic, explored some aspects of its proof theory and semantics, and also the complexity of the deciding validity. They also introduced a *primal* version of the logic, as an alternate system which promises to be computationally more efficient. They also provided efficient algorithms for some restrictions of propositional PIL. In this paper, we show that validity in PIL can be decided in PTIME.

The rest of the paper is structured as follows. In Section 2, we formally present primal infon logic, and explore its interesting proof-theoretical properties. In Section 3, we prove the *subformula property* for PIL, which is used in the algorithm for checking derivability in Section 4. After proving the correctness of the algorithm, we devote Section 5 to the nontrivial analysis of its running time. We end with concluding remarks in Section 6.

2 Primal infon logic

We present **primal infon logic** (PIL) formally in this section. Assume a set of atomic propositions $\mathcal{P}$. The set of formulas of primal infon logic is given by:

$$\Phi ::= p \mid x \wedge y \mid x \to y \mid \Box_a x \mid \boxplus_a x$$

where $a \in \mathcal{A}$, $p \in \mathcal{P}$, and $x, y \in \Phi$. $\Box_a x$ and $\boxplus_a x$ model a said x and a implied x from [12], respectively.

The set of **subformulas** of a formula x, denoted $\mathsf{sf}(x)$, is defined to be the smallest set S such that: $x \in S$; whenever $x \wedge y \in S$ or $x \to y \in S$, $\{x, y\} \subseteq S$; and whenever $\Box_a x \in S$ or $\boxplus_a x \in S$, $x \in S$. For a set X of formulas, $\mathsf{sf}(X) = \bigcup_{x \in X} \mathsf{sf}(x)$.

The logic is defined by the derivation system in Figure 1. In the rules, X and X' stand for sets of formulas, and we use $\Box_a X$ and $\boxplus_a X$ to denote $\{\Box_a x \mid x \in X\}$ and $\{\boxplus_a x \mid x \in X\}$, respectively. We also use $\Box_a^{-1}(X)$ and $\boxplus_a^{-1}(X)$ to denote $\{x \mid \Box_a x \in X\}$ and $\{x \mid \boxplus_a x \in X\}$, respectively. We use X, X' to denote $X \cup X'$ and X, x to denote $X \cup \{x\}$. We also use $X - x$

$\dfrac{}{X, x \vdash x}\, ax$	$\dfrac{X \vdash x}{X, X' \vdash x}\, weaken$
$\dfrac{X \vdash x \quad X \vdash y}{X \vdash x \wedge y} \wedge i$	$\dfrac{X \vdash x_0 \wedge x_1}{X \vdash x_i} \wedge e_i$
$\dfrac{X \vdash y}{X \vdash x \to y} \to i$	$\dfrac{X \vdash x \to y \quad X \vdash x}{X \vdash y} \to e$
$\dfrac{X \vdash x}{\Box_a X \vdash \Box_a x} \Box_a$	$\dfrac{X, Y \vdash x}{\Box_a X, \boxplus_a Y \vdash \boxplus_a x} \boxplus_a$
$\dfrac{X \vdash x \quad Y \vdash y}{X, Y - x \vdash y}\, cut$	

Figure 1 The system PIL_{nd}.

$\dfrac{}{X, x \vdash x}\, ax$	$\dfrac{X \vdash x}{X, X' \vdash x}\, weaken$
$\dfrac{X \vdash x \quad X \vdash y}{X \vdash x \wedge y} \wedge r$	$\dfrac{X, x_i \vdash y}{X, x_0 \wedge x_1 \vdash y} \wedge \ell_i$
$\dfrac{X \vdash y}{X \vdash x \to y} \to r$	$\dfrac{X \vdash x \quad X, y \vdash z}{X, x \to y \vdash z} \to \ell$
$\dfrac{X \vdash x}{\Box_a X \vdash \Box_a x} \Box_a$	$\dfrac{X, Y \vdash x}{\Box_a X, \boxplus_a Y \vdash \boxplus_a x} \boxplus_a$
$\dfrac{X \vdash x \quad Y \vdash y}{X, Y - x \vdash y}\, cut$	

Figure 2 The system PIL_{sc}.

to denote $X \setminus \{x\}$. In a **sequent** $X \vdash x$, X is the **antecedent** and x is the **consequent**. In a rule, the sequent occurring below the line is the **conclusion** and the sequents occurring above the line are the **premises**. The formula x occurring in the *cut* rule is called the **cut formula**. We use $X \vdash_{nd} x$ to denote that there is a derivation of the sequent $X \vdash x$ in PIL_{nd}. The **derivation problem** asks, given X and x, whether $X \vdash_{nd} x$. We also introduce the sequent calculus formulation of PIL, PIL_{sc}, in Figure 2. We use $X \vdash_{sc} x$ to denote that there is a derivation of the sequent $X \vdash x$ in PIL_{sc}.

Three features of PIL_{nd} are significant: the $\to i$ rule, the presence of the *cut* rule, and the $\boxplus_a$ rule. The $\to i$ is what distinguishes PIL from **full infon logic (FIL)**, which has the following (more standard) version of the $\to i$ rule.

$$\frac{X, x \vdash y}{X \vdash x \to y} \to i$$

The implication in FIL involves *discharging* assumptions, while the implication in PIL is just a weakening of the consequent from y to $x \to y$, without discharging any assumptions. Thus, the implication in PIL is a new kind of operator. It is worth noting that the derivability problem for just the $\{\to\}$-fragment of full infon logic is PSPACE-hard (see [16]), while even with modalities, the corresponding problem for PIL is in PTIME (Theorem 11 in this paper).

The other noteworthy feature is the presence of the *cut* rule, which is usually a feature of sequent calculus formulations. In most reasonable proof systems, though, this rule is **admissible**, i.e. whenever there are cut-free proofs of $X \vdash x$ and $Y \vdash y$, there is a cut-free proof of $X, Y - x \vdash y$. The *cut* rule is easily seen to be admissible in FIL. If π_1 and π_2 are cut-free proofs of $X \vdash x$ and $Y \vdash y$, the following is a cut-free proof of $X, Y - x \vdash y$.

$$\dfrac{\dfrac{\begin{array}{c}\pi_2 \\ \vdots \\ Y \vdash y\end{array}}{Y - x \vdash x \to y} \to i \qquad \begin{array}{c}\pi_1 \\ \vdots \\ X \vdash x\end{array}}{X, Y - x \vdash y} \to e$$

But with the weaker primal $\to i$ rule and modalities, it can be shown that *cut* is not admissible in cut-free PIL_{nd}, and hence needs to be added as an explicit rule.

Consider the sequent $\Box_a x \wedge \Box_a y \vdash \Box_a(x \wedge y)$, for instance. Here is one possible derivation in PIL_{nd} (which crucially uses the *cut* rule).

$$\dfrac{\dfrac{\dfrac{}{\Box_a x \wedge \Box_a y \vdash \Box_a x \wedge \Box_a y}\,ax}{\Box_a x \wedge \Box_a y \vdash \Box_a x}\,\wedge e_1 \qquad \dfrac{\dfrac{}{\Box_a x \wedge \Box_a y \vdash \Box_a x \wedge \Box_a y}\,ax}{\Box_a x \wedge \Box_a y \vdash \Box_a x}\,\wedge e_0 \qquad \dfrac{\dfrac{}{x, y \vdash x}\,ax \quad \dfrac{}{x, y \vdash y}\,ax}{x, y \vdash x \wedge y}\,\wedge i}{\Box_a x, \Box_a y \vdash \Box_a(x \wedge y)}\,\Box_a}{\Box_a x \wedge \Box_a y, \Box_a y \vdash \Box_a(x \wedge y)}\,cut}{\Box_a x \wedge \Box_a y \vdash \Box_a(x \wedge y)}\,cut$$

We say that a proof system $\mathcal{S}$ has the **subformula property** if the following holds:

- Whenever $X \vdash_{\mathcal{S}} x$, there is a $\mathcal{S}$-derivation π of $X \vdash x$ such that every formula y occurring in π belongs to $\mathsf{sf}(X \cup \{x\})$.

It can be easily shown that cut-free PIL_{nd} has the subformula property. (A detailed proof is given in [3].) Now suppose there is a cut-free PIL_{nd} proof of $\Box_a p \wedge \Box_a q \vdash \Box_a(p \wedge q)$, for $p, q \in \mathcal{P}$. Then there is a proof π with the same conclusion such that only formulas from $\mathsf{sf}(\{\Box_a p \wedge \Box_a q, \Box_a(p \wedge q)\})$ can occur in π. It is easy to see that the last rule of π cannot be an elimination rule. The only possibility is that the last rule is *weaken*, whose premise is $\vdash \Box_a(p \wedge q)$. This can only be got by using the $\Box_a$ rule from the premise $\vdash p \wedge q$. But this is not provable, since it is not a validity (according to the semantics given in [12]). Thus there is no cut-free PIL_{nd} proof of $\Box_a p \wedge \Box_a q \vdash \Box_a(p \wedge q)$, even though it is provable in PIL_{nd}. This means that the *cut* rule is not admissible in cut-free PIL_{nd}, and therefore that that cut cannot be eliminated in PIL_{nd}. This makes it difficult to prove the subformula property for PIL_{nd}.

But the subformula property is essential for our algorithms on PIL_{nd}. How then are we to prove it? Our solution is simple. We move to the system PIL_{sc} of Figure 2 (this system was already considered in [6]). It is reasonably straightforward to prove that cut is eliminable for this system (cut elimination also holds for the sequent calculus formulation of FIL and many extensions). It is also straightforward to show that cut-free derivations in PIL_{sc} have the subformula property. We show that one can always translate between derivations in PIL_{nd} and PIL_{sc} without introducing new formulas in the process. This yields the subformula property for PIL_{nd}. The formal details are presented in the next section.[1]

A natural question now arises – why not work with PIL_{sc} throughout, since it behaves better proof-theoretically? The answer is that it is not algorithmically well-behaved, since the left-hand sides of the sequents in a proof shrink and grow in an uncontrolled manner. On the other hand, we shall exploit precisely the controlled nature of the change in the left-hand side of the sequents in a PIL_{nd} derivation to extract an algorithm for the derivation problem. In particular, Lemma 5, which solves the non-modal fragment of PIL in linear time, uses an algorithm that closely mimics the rules in PIL_{nd}.

The third feature of interest is the $\boxplus_a$ modality and the $\boxplus_a$ rule. The intention is that $\boxplus_a$ is a weaker modality than $\Box_a$ but has the same flavour. In particular, it is conjunctive:

[1] An interesting aspect of these results is that the standard translation of a cut-free sequent-calculus proof to a normal derivation does not work in the presence of modalities. The left-rules of sequent calculus usually translate to elimination rules at the level of the hypotheses in an equivalent natural deduction derivation, but the rules for modalities are non-local – they modify *both* the hypotheses and conclusion. This is the source of the proof-theoretic complexity of these systems, and makes the cut-rule in PIL_{nd} non-eliminable.

$\boxplus_a p \wedge \boxplus_a q \vdash \boxplus_a (p \wedge q)$. This is to be contrasted with the $\Diamond_a$ modality from modal logic which is not conjunctive, and which has the following rule (which looks similar to the $\boxplus_a$ rule, but is very different in spirit):

$$\frac{X, y \vdash x}{\Box_a X, \Diamond_a y \vdash \Diamond_a x}$$

Note that this rule insists that there be *exactly one* $\Diamond_a$ formula in the antecedent. Because of this difference, the algorithm in our paper does not extend to $\Diamond$-like modalities, but it is interesting to seek restrictions to which our techniques can apply.

It should be noted that PIL_{nd} and PIL_{sc} are not the only formulations of infon logic possible. In [12], after introducing PIL_{nd}, the authors consider a Hilbert-style proof system that helps develop efficient (linear time) algorithms for some special cases. In [7], the fragment of PIL without the $\boxplus_a$ modalities has been shown to have a linear time algorithm for the derivation problem.[2] The algorithm is based on the Hilbert-style formulation of PIL. But for unrestricted PIL (with the $\Box_a$ and $\boxplus_a$ modalities), it has been shown by Gurevich and Savateev in [9] that there are sequents for which all derivations in the Hilbert-style system of [12] are exponential in size. This has the consequence that the linear-time algorithm developed in [7] does not extend to unrestricted PIL. In [6], the authors study PIL with the $\vee$ and $\bot$ operators and prove that its derivability problem is PSPACE-complete. In contrast, our paper provides an $O(N^3)$ algorithm for PIL, thus settling an important question in the study of this logic.

3 The subformula property

In this section, we formally prove the equivalence between PIL_{nd} and PIL_{sc} (preserving the set of formulas occurring in the respective proofs). We then state a cut elimination theorem for PIL_{sc}, and as corollaries, derive the subformula property for both PIL_{sc} and PIL_{nd}.

▶ **Proposition 1.** 1. *Suppose π is a proof of $X \vdash x$ in PIL_{nd}. Then there is a proof π' of $X \vdash x$ in PIL_{sc} such that all formulas occurring in π' occur in π.*
2. *Suppose π' is a proof of $X \vdash x$ in PIL_{sc}. Then there is a proof π of $X \vdash x$ in PIL_{nd} such that all formulas occurring in π occur in π'.*

Proof. The proof is by induction on the structure of derivations, and an analysis of the last rule of π. Most of the cases are straightforward – the *ax*, *weaken*, *cut*, $\Box_a$, and $\boxplus_a$ rules are present in both systems; and the $\to i$ and $\wedge i$ rules have the same form as the $\to r$ and $\wedge r$ rules, respectively. We only need to look at the other rules.

1. There are two cases to consider.
   - Suppose π has the following form.

$$\dfrac{\begin{array}{cc}\pi_1 & \pi_2 \\ \vdots & \vdots \\ X \vdash x \to y & X \vdash x\end{array}}{X \vdash y} \to e$$

[2] In fact, this fragment, called *basic primal infon logic*, is now used in DKAL [11] instead of unrestricted PIL. But the $\boxplus_a$ is justified in its own right (see [12]) and makes the language richer. It is also of potential interest to other authorization logics, and its derivability problem is a technical challenge. Hence the interest in unrestricted PIL.

By induction hypothesis there are PIL_{sc} derivations π_1' of $X \vdash x \to y$ and π_2' of $X \vdash x$ such that every formula occurring in π_1' occurs in π_1, and every formula occurring in π_2' occurs in π_2. π' can be taken to be the following PIL_{sc} derivation.

$$\dfrac{\begin{array}{cc}\pi_1' & \pi_2' \\ \vdots & \vdots \\ X \vdash x \to y & \dfrac{X \vdash x \quad \dfrac{}{X, y \vdash y}\, ax}{X, x \to y \vdash y} \to \ell\end{array}}{X \vdash y}\, cut$$

- The case when the last rule of π is $\wedge e_i$ is similarly handled, using the $\wedge \ell_i$ and *cut* rules.

2. There are two cases to consider.
   - Suppose π' has the following form.

$$\dfrac{\begin{array}{cc}\pi_1' & \pi_2' \\ \vdots & \vdots \\ X \vdash x & X, y \vdash z\end{array}}{X, x \to y \vdash z} \to \ell$$

By induction hypothesis there are PIL_{nd} derivations π_1 of $X \vdash x$ and π_2 of $X, y \vdash z$ such that every formula occurring in π_1 occurs in π_1', and every formula occurring in π_2 occurs in π_2'. π can be taken to be the following PIL_{nd} derivation.

$$\dfrac{\dfrac{\dfrac{}{X, x \to y \vdash x \to y}\, ax \quad \begin{array}{c}\pi_1 \\ \vdots \\ X \vdash x\end{array}}{X, x \to y \vdash y} \to e \quad \begin{array}{c}\pi_2 \\ \vdots \\ X, y \vdash z\end{array}}{X, x \to y \vdash z}\, cut$$

- The case when the last rule of π is $\wedge \ell_i$ is similarly handled, using the $\wedge e_i$ and *cut* rules.

Clearly the translated proofs do not contain formulas not occurring in the original proof, in all these cases. ◀

The main reason to consider PIL_{sc} is the following important property.

▶ **Theorem 2** (Cut elimination for PIL_{sc} (Theorem 5.1 in [6])**.** *If $X \vdash_{sc} x$, then there is a proof π of $X \vdash x$ in PIL_{sc} such that the cut rule does not occur in π.*

▶ **Proposition 3** (Subformula property for PIL_{sc})**.** *Let π be a cut-free proof of $X \vdash x$ in PIL_{sc} and y be any formula that belongs to a sequent (either in the antecedent or in the consequent) occurring in π. Then $y \in \mathsf{sf}(X \cup \{x\})$.*

Proof. Observe that in every rule of PIL_{sc} other than *cut*, all formulas occurring in the premises are subformulas of the ones occurring in the conclusion. Thus any formula occurring in a cut-free PIL_{sc} derivation of $X \vdash x$ is in $\mathsf{sf}(X \cup \{x\})$. ◀

▶ **Theorem 4** (Subformula property for PIL_{nd})**.** *Suppose $X \vdash_{nd} x$. Then there is a proof π of $X \vdash x$ in PIL_{nd} such that any formula y occurring in π is in $\mathsf{sf}(X \cup \{x\})$.*

Proof. Since $X \vdash_{nd} x$, it follows (from Proposition 1) that $X \vdash_{sc} x$. Therefore there is a cut-free PIL_{sc} proof π' of $X \vdash x$, by Theorem 2. By the subformula property for PIL_{sc} (Proposition 3), every formula occurring in π' is from $\mathsf{sf}(X \cup \{x\})$. We use Proposition 1 again, to translate π' back to a proof π in PIL_{nd}, such that every formula occurring in π also occurs in π', and hence is in $\mathsf{sf}(X \cup \{x\})$. ◀

4 Algorithm for derivability

We present the algorithm for the derivation problem of PIL_{nd} in this section and prove its correctness. Fix a set of formulas X_0 and a formula x_0, and let Y_0 to be $\mathsf{sf}(X_0 \cup \{x_0\})$. Let $N = |Y_0|$. For any $X \subseteq Y_0$:

- $\mathit{closure}(X) = \{x \in Y_0 \mid X \vdash_{nd} x\}$.
- $\mathit{closure}'(X) = \{x \in Y_0 \mid$ there is a proof of $X \vdash x$ that does not use the $\Box$ and $\boxplus$ rules$\}$.

▶ **Lemma 5.** *For $X \subseteq Y_0$, $\mathit{closure}'(X)$ can be computed in $O(N)$ time.*

The above result is an immediate adaptation of Theorem 6.1 in [12], where a linear time algorithm for *primal constructive logic* is provided.

The algorithm for computing *closure* is presented as two mutually recursive functions $f : 2^{Y_0} \to 2^{Y_0}$ and $g : 2^{Y_0} \to 2^{Y_0}$, defined in Algorithm 1. The function g simulates *one* application of the $\Box_a$ and $\boxplus_a$ rules for each $a \in \mathcal{A}$, composed with an application of $\mathit{closure}'$. This might yield modal formulas that can be used in further $\Box_a$ and $\boxplus_a$ rules, so f makes repeated calls to g till a fixpoint is reached. f can thus be thought of as repeatedly simulating the *cut* rule after each call to g.

An application of a modal rule involves stripping the modalities from the set of formulas currently derivable, computing *closure* of the stripped set, and applying the modalities again to this set. Towards this, g makes a recursive call to f with the appropriate arguments. To make the complexity analysis easier, we keep track of the sequence of modalities stripped along each path in the recursive call tree. We call these sequences **modal contexts**, and provide them as further arguments to the functions f and g. For ease of notation, for any modal context σ, we refer to $f(\sigma, \cdot)$ and $g(\sigma, \cdot)$ as f_σ and g_σ, respectively.

Let $\Sigma = \{\Box_a, \boxplus_a \mid a \in \mathcal{A}\}$. The set of **modal contexts** of a formula x, denoted $\mathcal{C}(x)$, is a subset of Σ^*, defined by induction as follows:

- $\mathcal{C}(p) = \{\varepsilon\}$
- $\mathcal{C}(x \wedge y) = \mathcal{C}(x \to y) = \mathcal{C}(x) \cup \mathcal{C}(y)$
- $\mathcal{C}(\Box_a x) = \{\varepsilon\} \cup \{\Box_a \cdot \sigma \mid \sigma \in \mathcal{C}(x)\}$
- $\mathcal{C}(\boxplus_a x) = \{\varepsilon\} \cup \{\boxplus_a \cdot \sigma \mid \sigma \in \mathcal{C}(x)\}$.

For a set X of formulas, $\mathcal{C}(X) = \bigcup_{x \in X} \mathcal{C}(x)$. To simplify notation, we let $\mathcal{C}$ denote $\mathcal{C}(Y_0)$. Note that for any $X \subseteq Y_0$, $|\mathcal{C}(X)| \leq |\mathcal{C}| \leq |Y_0| \leq N$.

The **modal depth** of a formula x, denoted $\mathit{depth}(x)$, is defined by induction as follows:

- $\mathit{depth}(p) = 0$ for $p \in \mathcal{P}$.
- $\mathit{depth}(x \wedge y) = \mathit{depth}(x \to y) = \max(\mathit{depth}(x), \mathit{depth}(y))$.
- $\mathit{depth}(\Box_a x) = \mathit{depth}(\boxplus_a x) = \mathit{depth}(x) + 1$.

For a set X of formulas, $\mathit{depth}(X) = \max\{\mathit{depth}(x) \mid x \in X\}$.

▶ **Lemma 6.** *Suppose $X, Y \subseteq Y_0$ and $\sigma \in \mathcal{C}$. Then:*

1. $X \subseteq \mathit{closure}'(X) \subseteq \mathit{closure}(X)$.
2. $\mathit{closure}'(\mathit{closure}(X)) = \mathit{closure}(\mathit{closure}(X)) = \mathit{closure}(X)$.
3. *If $X \subseteq Y$ then $g_\sigma(X) \subseteq g_\sigma(Y)$ and $f_\sigma(X) \subseteq f_\sigma(Y)$.*
4. $X \subseteq g_\sigma(X) \subseteq g_\sigma^2(X) \subseteq \cdots Y_0$.
5. $f_\sigma(X) = g_\sigma^m(X)$ *for some* $m \leq N$.

Algorithm 1 Algorithm to compute *closure*

```
function f(σ, X)
    if σ ∉ C or X = ∅ then
        return ∅;
    end if
    Y ← X;
    while Y ≠ g(σ, Y) do
        Y ← g(σ, Y);
    end while
    return Y;
end function

function g(σ, X)
    for all a ∈ A : Y_a ← □_a^{-1}(X);
    for all a ∈ A : Z_a ← □_a^{-1}(X) ∪ ⊞_a^{-1}(X);
    return closure'(X ∪ ⋃_{a∈A} □_a f(σ□_a, Y_a) ∪ ⋃_{a∈A} ⊞_a f(σ⊞_a, Z_a));
end function
```

The last fact is true because $|Y_0| = N$ and the g^i_σ's form a nondecreasing sequence.

▶ **Proposition 7** (Soundness). *For $X \subseteq Y_0$, $\sigma \in \mathcal{C}$, and $m \geq 0$, $g^m_\sigma(X) \subseteq closure(X)$.*

Proof. We shall assume that $g^n_\tau(Y) \subseteq closure(Y)$ for all $Y \subseteq Y_0$, $\tau \in \mathcal{C}$, and all $n \geq 0$ such that $(depth(Y), n) <_{\text{lex}} (depth(X), m)$, and prove that $g^m_\sigma(X) \subseteq closure(X)$ for all $\sigma \in \mathcal{C}$.

For $a \in \mathcal{A}$, let $Y_a = \Box_a^{-1}(g^{m-1}_\sigma(X))$ and $Z_a = \Box_a^{-1}(g^{m-1}_\sigma(X)) \cup \boxplus_a^{-1}(g^{m-1}_\sigma(X))$. Further, let $X' = g^{m-1}_\sigma(X) \cup \bigcup_{a\in\mathcal{A}} \Box_a f_{\sigma\Box_a}(Y_a) \cup \bigcup_{a\in\mathcal{A}} \boxplus_a f_{\sigma\boxplus_a}(Z_a)$. Then $g^m_\sigma(X) = closure'(X')$. Note that $depth(Y_a) < depth(X)$ and $depth(Z_a) < depth(X)$. Now if $x \in X'$ we can distinguish the following three cases that can arise:

- Suppose $x \in g^{m-1}_\sigma(X)$. Since $(depth(X), m-1) <_{\text{lex}} (depth(X), m)$, by induction hypothesis, $g^{m-1}_\sigma(X) \subseteq closure(X)$, and hence $x \in closure(X)$.
- Suppose $x = \Box_a y$ for some $a \in \mathcal{A}$ and some $y \in f_{\sigma\Box_a}(Y_a)$. But $f_{\sigma\Box_a}(Y_a) = g^n_{\sigma\Box_a}(Y_a)$ for some $n \leq N$. Since $depth(Y_a) < depth(X)$, $(depth(Y_a), n) <_{\text{lex}} (depth(X), m)$, and hence by induction hypothesis, $g^n_{\sigma\Box_a}(Y_a) \subseteq closure(Y_a)$. Therefore $y \in closure(Y_a)$. Now one can use the $\Box_a$ rule and *weaken* rule to show that $\Box_a y \in closure(g^{m-1}_\sigma(X))$.
- Suppose $x = \boxplus_a y$ for some $a \in \mathcal{A}$ and some $y \in f_{\sigma\boxplus_a}(Z_a)$. But $f_{\sigma\boxplus_a}(Z_a) = g^n_{\sigma\boxplus_a}(Z_a)$ for some $n \leq N$. Since $depth(Z_a) < depth(X)$, $(depth(Z_a), n) <_{\text{lex}} (depth(X), m)$, and hence by induction hypothesis, $g^n_{\sigma\boxplus_a}(Z_a) \subseteq closure(Z_a)$. Therefore $y \in closure(Z_a)$. Now one can use the $\boxplus_a$ rule and *weaken* rule to show that $\boxplus_a y \in closure(g^{m-1}_\sigma(X))$.

Thus $X' \subseteq closure(g^{m-1}_\sigma(X))$. Also, $g^{m-1}_\sigma(X) \subseteq closure(X)$. Therefore $X' \subseteq closure(X)$. And since $g^m_\sigma(X) = closure'(X')$, $g^m_\sigma(X) \subseteq closure(X)$. ◀

We next prove that whenever $X \vdash x$, $x \in g^n_\sigma(X)$ for an appropriate σ and $n \leq N$. Because of our use of contexts, this direction is nontrivial. We illustrate the subtleties with an example. Let $X_0 = \{\Box_a \boxplus_b p, \boxplus_a \Box_b q\}$. One can easily see that $x_0 = \boxplus_a \boxplus_b (p \wedge q)$ is derivable from X_0. It is also easy to see that $x_0 \in f_\varepsilon(X_0)$. But $x_0 \notin f_{\boxplus_a}(X_0)$. This is because all the recursive subcalls to f return $\emptyset$, either because $\emptyset$ is passed as argument or because the context passed does not belong to $\mathcal{C}(X_0 \cup \{x_0\})$. Thus we need to ensure that the contexts supplied to recursive calls are proper. One way to ensure this is that the given context σ concatenated with any context in the argument set X belongs to $\mathcal{C}(X_0 \cup \{x_0\})$. But that condition is too

strong and does not apply to the recursive call $f_{\boxplus_a}(X)$ (where $X = \{\boxplus_b p, \Box_b q\}$) even though this call will be made by $f_\varepsilon(X_0)$. A weaker condition holds, though – for every context in X, we can prepend at least one of $\Box_a$ and $\boxplus_a$ to it to get a context from X_0. We formalize this intuition below.

For two contexts $\sigma = \mathsf{M}_1 \cdots \mathsf{M}_n$ and $\sigma' = \mathsf{M}'_1 \cdots \mathsf{M}'_n$, we say that σ' is a strengthening of σ (in symbols: $\sigma' \geq \sigma$) if for all $i \leq N$, either $\mathsf{M}_i = \mathsf{M}'_i$, or $\mathsf{M}_i = \boxplus_a$ and $\mathsf{M}'_i = \Box_a$ for some $a \in \mathcal{A}$. We say that $\sigma \in \mathcal{C}$ is **safe** for $X \subseteq Y_0$ if for every $\tau \in \mathcal{C}(X)$, there is some $\sigma' \geq \sigma$ such that $\sigma'\tau \in \mathcal{C}$.

▶ **Proposition 8** (Completeness). *Suppose $X \subseteq Y_0$, $x \in closure(X)$, and $\sigma \in \mathcal{C}$. If σ is safe for $X \cup \{x\}$, then there is $m \geq 0$ such that $x \in g^m_\sigma(X)$.*

Proof. Suppose $x \in closure(X)$. Then there is a proof π of $X \vdash x$. By Theorem 4, we can assume that for every subproof π' of π with conclusion $X' \vdash x'$, and all formulas y occurring in π', $y \in \mathsf{sf}(X' \cup \{x'\})$. We now prove the desired claim by induction on the structure of π.

- Suppose the last rule of π is ax, $\wedge i$, $\to i$, $\wedge e$, or $\to e$. Without loss of generality, let the last rule have two premises and let x' and x'' be the consequents in the two premises. Suppose σ is safe for $X \cup \{x\}$. It is also safe for $X \cup \{x'\}$ and $X \cup \{x''\}$. By induction hypothesis, there exist $m, n \geq 0$ such that $x' \in g^m_\sigma(X)$ and $x'' \in g^n_\sigma(X)$. Without loss of generality, let $m \geq n$. Then $x', x'' \in g^m_\sigma(X)$, and $x \in closure'(\{x', x''\}) \subseteq g^m_\sigma(X)$.
- Suppose the last rule of π is *weaken*. Suppose the last rule has premise $X' \vdash x$, for some $X' \subseteq X$. Since σ is safe for X, it is also safe for X'. Hence by induction hypothesis, there is some m such that $x \in g^m_\sigma(X') \subseteq g^m_\sigma(X)$.
- Suppose π has the following form (and $Y = \Box_a^{-1}(X)$ and $x = \Box_a y$):

$$\dfrac{\begin{array}{c}\pi' \\ \vdots \\ Y \vdash y\end{array}}{X \vdash x}\,\Box_a$$

  Now for every $\tau \in \mathcal{C}(Y \cup \{y\})$, $\Box_a\tau \in \mathcal{C}(X \cup \{x\})$. Since σ is safe for $X \cup \{x\}$, it follows that $\sigma\Box_a$ is safe for $Y \cup \{y\}$. Thus by induction hypothesis, $y \in g^n_{\sigma\Box_a}(Y) \subseteq f_{\sigma\Box_a}(Y)$, for some $n \geq 0$. Now it is immediately seen that $x \in g_\sigma(X)$, by definition of g_σ.
- Suppose π has the following form (and $Y = \Box_a^{-1}(X)$, $Z = \boxplus_a^{-1}(X)$ and $x = \boxplus_a y$):

$$\dfrac{\begin{array}{c}\pi' \\ \vdots \\ Y, Z \vdash y\end{array}}{X \vdash x}\,\boxplus_a$$

  For every $\tau \in \mathcal{C}(Y)$, $\Box_a\tau \in \mathcal{C}(X \cup \{x\})$, and for every $\tau \in \mathcal{C}(Z \cup \{y\})$, $\boxplus_a\tau \in \mathcal{C}(X \cup \{x\})$. But σ is safe for $X \cup \{x\}$. So for every $\tau \in \mathcal{C}(Y \cup Z \cup \{y\})$, there is a strengthening σ' of σ such that either $\sigma'\Box_a\tau \in \mathcal{C}$ or $\sigma'\boxplus_a\tau \in \mathcal{C}$. In other words, for every $\tau \in \mathcal{C}(Y \cup Z \cup \{y\})$, there is a strengthening $\widehat{\sigma}$ of $\sigma\boxplus_a$ such that $\widehat{\sigma}\tau \in \mathcal{C}$. Therefore $\sigma\boxplus_a$ is safe for $Y \cup Z \cup \{y\}$. Thus by induction hypothesis, $y \in g^n_{\sigma\boxplus_a}(Y \cup Z) \subseteq f_{\sigma\boxplus_a}(Y \cup Z)$, for some $n \geq 0$. Now it is immediately seen that $x \in g_\sigma(X)$, by definition of g_σ.
- Suppose π has the following form (and $X = Y' \cup (Y'' \setminus \{y\})$):

$$\dfrac{\begin{array}{cc}\pi' & \pi'' \\ \vdots & \vdots \\ Y' \vdash y & Y'' \vdash x\end{array}}{X \vdash x}\,cut$$

Since $y \in \mathsf{sf}(X \cup \{x\})$, $\mathcal{C}(Y' \cup \{y\}) \subseteq \mathcal{C}(X \cup \{x\})$ and $\mathcal{C}(Y'' \cup \{x\}) \subseteq \mathcal{C}(X \cup \{x\})$. Thus σ is safe for both $Y' \cup \{y\}$ and $Y'' \cup \{x\}$. By induction hypothesis, there is $m \geq 0$ such that $y \in g_\sigma^m(Y') \subseteq g_\sigma^m(X)$. Therefore $Y'' \subseteq X \cup \{y\} \subseteq g_\sigma^m(X)$. Also by induction hypothesis (since π'' is a smaller proof than π), $x \in g_\sigma^n(Y'')$ for some $n \geq 0$. Therefore $x \in g_\sigma^{m+n}(X)$. ◀

▶ **Theorem 9.** *For all $X \subseteq Y_0$, $f_\varepsilon(X) = closure(X)$.*

Proof. On the one hand, $f_\varepsilon(X) = g_\varepsilon^N \subseteq closure(X)$ by soundness. Conversely, for any $x \in closure(X)$, ε is safe for $X \cup \{x\}$ and hence there is $m \leq N$ such that $x \in g_\varepsilon^m(X) \subseteq f_\varepsilon(X)$, by completeness. ◀

5 Complexity

Fix a set of formulas X_0 and a formula x_0 as before, and let Y_0 to be $\mathsf{sf}(X_0 \cup \{x_0\})$. Let $N = |Y_0|$. We focus on a call of $f(\varepsilon, X_0)$ and all the recursive invocations of f and g in the course of that computation. We use intuitive notions like *parent call*, *later call*, *earlier call*, which formally refer to the call tree of the computation of $f(\varepsilon, X_0)$. We use the notation $(\sigma, X) \to_f (\tau, Y)$ to denote that $f(\sigma, X)$ is an earlier recursive call and $f(\tau, Y)$ is a later recursive call in the computation of $f(\varepsilon, X_0)$. The notation $(\sigma, X) \to_g (\tau, Y)$ has a similar interpretation.

The following lemma is the first step towards analyzing the complexity of the algorithm.

▶ **Lemma 10.** *Suppose $\sigma \in \mathcal{C}$, and $X, Y \subseteq Y_0$.*
1. *If $(\sigma, X) \to_f (\sigma, Y)$ then $f_\sigma(X) \subseteq Y$.*
2. *If $(\sigma, X) \to_g (\sigma, Y)$ then $g_\sigma(X) \subseteq Y$.*

Proof. We prove both the above statements together, by induction on $|\sigma|$. There are two cases to consider.

Case $|\sigma| = 0$: In this case, $\sigma = \varepsilon$.

1. There is only one call of f with first argument ε. So the statement is vacuously true.
2. Suppose $(\varepsilon, X) \to_g (\varepsilon, Y)$. This means that $X = g_\varepsilon^i(X_0)$ and $Y = g_\varepsilon^j(X_0)$ for some i, j with $i < j$. Thus $g_\varepsilon(X) = g_\varepsilon^{i+1}(X_0) \subseteq g_\varepsilon^j(X_0) = Y$.

Case $|\sigma| > 0$: We prove the statement about f assuming the statement about g for a prefix of σ, and then prove the statement about g assuming the statement about f (for σ).

1. There are two subcases to consider.
   - **Case $\sigma = \tau\Box_a$:** Suppose $(\sigma, X) \to_f (\sigma, Y)$. This means that there are sets X', Y' such that $X = \Box_a^{-1}(X')$, $Y = \Box_a^{-1}(Y')$, and parent calls $g(\tau, X')$ and $g(\tau, Y')$ such that $(\tau, X') \to_g (\tau, Y')$. Thus by induction hypothesis $g_\tau(X') \subseteq Y'$. But then, by definition of g, we have that $\Box_a f_\sigma(X) = \Box_a f_{\tau\Box_a}(\Box_a^{-1}(X')) \subseteq g_\tau(X') \subseteq Y'$. Therefore $f_\sigma(X) \subseteq \Box_a^{-1}(Y') = Y$.
   - **Case $\sigma = \tau \boxplus_a$:** Suppose $(\sigma, X) \to_f (\sigma, Y)$. This means that there are sets X', Y' such that $X = \Box_a^{-1}(X') \cup \boxplus_a^{-1}(X')$, $Y = \Box_a^{-1}(Y') \cup \boxplus_a^{-1}(Y')$, and parent calls $g(\tau, X')$ and $g(\tau, Y')$ such that $(\tau, X') \to_g (\tau, Y')$. Thus $g_\tau(X') \subseteq Y'$. But then, by definition of g, $\boxplus_a f_\sigma(X) = \boxplus_a f_{\tau\boxplus_a}(\Box_a^{-1}(X') \cup \boxplus_a^{-1}(X')) \subseteq g_\tau(X') \subseteq Y'$. Therefore $f_\sigma(X) \subseteq \boxplus_a^{-1}(Y') \subseteq Y$.
2. Suppose $(\sigma, X) \to_g (\sigma, Y)$. There are two cases to consider.
   - There is one parent call $f(\sigma, Z)$ of which $g(\sigma, X)$ is a subcall and $g(\sigma, Y)$ is a later subcall. From the definition of f, it follows that there are i, j with $i < j$ such that $X = g_\sigma^i(Z)$ and $Y = g_\sigma^j(Z)$. Thus $g_\sigma(X) = g_\sigma^{i+1}(Z) \subseteq g_\sigma^j(Z) = Y$.

- There are parent calls $f(\sigma, X')$ and $f(\sigma, Y')$ such that $(\sigma, X') \rightarrow_f (\sigma, Y')$. By induction hypothesis $f_\sigma(X') \subseteq Y'$. But by definition of f it follows that there are $i > 0$ and $j > 0$ such that $X = g_\sigma^i(X')$ and $Y = g_\sigma^j(Y')$. Therefore

$$g_\sigma(X) = g_\sigma^{i+1}(X') \subseteq g_\sigma^N(X') = f_\sigma(X') \subseteq Y' \subseteq g_\sigma^j(Y') = Y.$$

◀

Algorithm 2 Improved algorithm to compute *closure*

Initialization: for all $\sigma \in \mathcal{C} : G_\sigma \leftarrow \emptyset$;

function $f(\sigma, X)$
  if $\sigma \notin \mathcal{C}$ or $X = \emptyset$ **then**
    return $\emptyset$;
  end if
  $Y \leftarrow X$;
  while $Y \neq G_\sigma$ **do** ▷ $G_\sigma = g(\sigma, G_\sigma)$ before the start of the loop.
    $G_\sigma \leftarrow Y$;
    $Y \leftarrow g(\sigma, Y)$;
  end while ▷ $G_\sigma = g(\sigma, G_\sigma)$ at the end of the loop.
  return G_σ;
end function

▶ **Theorem 11.** *It can be checked in* $O(N^3)$ *time whether* $x_0 \in closure(X_0)$.

Proof. For each $\sigma \in \mathcal{C}$, if $g_\sigma(X)$ is a recursive call and if $g_\sigma(Y)$ is a later recursive call, $X \subseteq g_\sigma(X) \subseteq Y$. Thus the arguments to g_σ (in temporal order of the calls) constitutes a nondecreasing sequence of subsets of Y_0. Such a sequence can have at most N *distinct* elements. But it is possible that there are many invocations of g_σ with the same argument, which constitutes wasteful work. We thus present an improved algorithm using **memoization** in Algorithm 2. (We only redefine the function $f(\sigma, \cdot)$. The function $g(\sigma, \cdot)$ is the same as in Algorithm 1.) In this implementation, for any $\sigma \in \mathcal{C}$, the *total number of calls* to $g(\sigma, \cdot)$ is N. We achieve this by storing the last argument to g_σ in the variable G_σ, preserving this across invocations from different calls to f_σ. The code for f_σ in Algorithm 2 reveals that across different invocations of f_σ, the same argument is never passed to subcalls of g_σ. Thus there are at most N calls of g_σ. Since there is only one call of f_ε and since for every context $\sigma\mathsf{M}$, there is at most one subcall to $f_{\sigma\mathsf{M}}$ from each invocation of g_σ, the *total number of calls* of f_σ is also N, for any fixed σ.

Further, each invocation of g involves computing $closure'(\cdot)$, which takes $O(N)$ time, and each invocation of f involves looking up (and updating) each distinct value assumed by G_σ once. Thus overall, there are N lookups and updates of the variable G_σ, which can each be achieved in $O(N)$ time. Across all contexts, there are N^2 computations of $closure'(\cdot)$ and N^2 lookups and updates. Thus the overall time taken is $O(N^3)$. ◀

6 Conclusions and Future Work

We have provided an $O(N^3)$ algorithm for the derivability problem for propositional PIL. The interesting aspects of our solution are the proof of the subformula property by going over to PIL_{sc} and back, and exploiting the controlled change in the antecedents of sequents in a

PIL_{nd} proof to derive an efficient algorithm. We believe that these techniques are general, and not specific to authorization logics or PIL. We plan to adapt our techniques to many variants of intuitionistic modal logic. One plan of study is to find (proof-theoretic) variants for other operators like disjunction, with the view of deriving efficient algorithms. Another interesting possibility is to keep the rules standard but restrict the structure of formulas in $X \cup \{x\}$ in such a way that the techniques in our paper can be adapted to the problem of checking if $X \vdash x$. It is also essential to consider extensions of these systems with $\Diamond$-like modalities, as mentioned in Section 2.

References

1 M. Abadi. Logic in access control. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science*, pages 228–233, 2003.

2 M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

3 A. Baskar, Prasad Naldurg, K.R. Raghavendra, S.P. Suresh. Primal infon logic: derivability in polynomial time. Technical Report. Available at `http://www.cmi.ac.in/~spsuresh/pdffiles/pil-fsttcs2013-tr.pdf`.

4 Moritz Y. Becker. Information flow in credential systems. In *Proc. 23rd IEEE Computer Security Foundations Symposium*, CSF '10, pages 171–185, Washington, DC, USA, 2010. IEEE Computer Society.

5 Moritz Y. Becker, Cedric Fournet, and Andrew D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

6 Lev Beklemishev and Yuri Gurevich. Propositional primal logic with disjunction. *Journal of Logic and Computation* 22(2012),

7 Carlos Cotrini and Yuri Gurevich. Basic primal infon logic. *Journal of Logic and Computation*, Special issue devoted to Arnon Avron.

8 John DeTreville. Binder, a logic-based security language. In *Proc. 2002 IEEE Symposium on Security and Privacy*, 2002.

9 Yuri Gurevich. Two notes on propositional primal logic. Microsoft Research Technical Report MSR-TR-2011-70, May 2011.

10 Yuri Gurevich and Itay Neeman. DKAL: Distributed-knowledge authorization language. In *CSF*, pages 149–162, 2008.

11 Yuri Gurevich and Itay Neeman. DKAL2 – a simplified and improved authorization language. Microsoft Research Technical report MSR-TR-2009-11, 2009.

12 Yuri Gurevich and Itay Neeman. Logic of infons: The propositional case. *ACM Transactions of Computational Logic*, 12, January 2011.

13 Trevor Jim. Sd3: A trust management system with certified evaluation. In *IEEE Symposium on Security and Privacy*, 2001.

14 B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

15 Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information Systems Security*, 6(1):128–171, February 2003.

16 Richard Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9: 67–72, 1979.

Composition Problems for Braids

Igor Potapov

University of Liverpool, Computer Science Department, Liverpool, UK
potapov@liverpool.ac.uk

Abstract

In this paper we investigate the decidability and complexity of problems related to braid composition. While all known problems for a class of braids with 3 strands, B_3, have polynomial time solutions we prove that a very natural question for braid composition, the membership problem, is NP-hard for braids with only 3 strands. The membership problem is decidable for B_3, but it becomes harder for a class of braids with more strands. In particular we show that fundamental problems about braid compositions are undecidable for braids with at least 5 strands, but decidability of these problems for B_4 remains open. The paper introduce a few challenging algorithmic problems about topological braids opening new connections between braid groups, combinatorics on words, complexity theory and provides solutions for some of these problems by application of several techniques from automata theory, matrix semigroups and algorithms.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.4.2 Grammars and Other Rewriting Systems, F.4.3 Formal Languages

Keywords and phrases Braid group, automata, group alphabet, combinatorics on words, matrix semigroups, NP-hardness, decidability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.175

1 Introduction

In this paper we investigate the decidability and complexity for a number of problems related to braid composition. Braids are classical topological objects that attracted a lot of attention due to their connections to topological knots and links as well as their applications to polymer chemistry, molecular biology, cryptography, quantum computations and robotics [1, 11, 14].

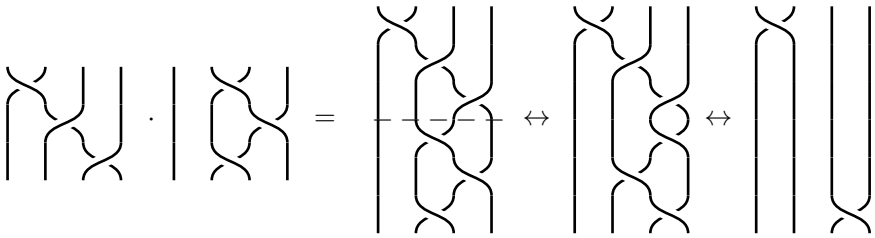
The discovery of a various cryptosystems based on the braid group inspired a new line of research about the complexity analysis of decision problems for braids, including the word problem, the generalized word problem, root extraction problem, the conjugacy problem and the conjugacy search problem. For many problems the polynomial time solutions were found, but it was surprisingly shown by M. S. Paterson and A. A. Razborov in 1991 that another closely related problem, the *non-minimal braid problem*, to be NP-complete [16]

> **Non-minimal braid problem:** Given a word ω in the generators $\sigma_1, \ldots, \sigma_{n-1}$ and their inverses, determine whether there is a shorter word ω' in the same generators which represents the same element of the n-strand braid group B_n?

The main result of this paper is to show another hard problem for braids in B_3, i.e. with only three strands. The problem can be naturally formulated in terms of composition (or concatenation) of braids which is one of the fundamental operations for the Braid Group.

Given two geometric braids, we can compose them, i.e. put one after the other making the endpoints of the first one coincide with the starting points of the second one. There is a neutral element for the composition: it is the trivial braid, also called identity braid, i.e. the class of the geometric braid where all the strings are straight. Two geometric braids are

isotopic if there is a continuous deformation of the ambient space that deforms one into the other, by a deformation that keeps every point in the two bordering planes fixed.



In this paper we study several computational problems related to composition of braids: Given a set of braids with n strands $B = b_1, \ldots, b_k \in B_n$. Let us denote a semigroup of braids, generated by B and the operation of composition, by $\langle B \rangle$.

- **Membership problem.** Check whether exist a composition of braids from a set B that is isotopic to a given braid b. I.e. is b in $\langle B \rangle$?
- **Identity problem.** Check whether exist a composition of braids from a set B that is isotopic to a trivial braid.
- **Group problem.** Check whether for any braid $b \in B$ we can construct the inverse of b by composition of braids from B. I.e. is a semigroup $\langle B \rangle$ a group?

	B_3	B_4	B_5
Membership	Decidable, NP-hard	?	Undecidable
Group/Identity	Decidable	?	Undecidable

In contrast to many polynomial time problems we show that the Membership problem for B_3 is NP-hard[1] by using a combination of new and existing encoding techniques from automata theory, group theory, matrix semigroups [4, 5] and algebraic properties of braids [1]. Then we prove decidability result for the membership problem for B_3 which is the first non-trivial case where composition is associative, but it is non-commutative. The membership problem for braids in B_3 has a very close connection with other non-trivial computational problems in matrix semigroups. since the braid group B_3 is the universal central extension of the modular group $PSL(2, \mathbb{Z})$. The idea of decidability in B_3 was inspired by the work of several authors on the membership problem for 2×2 matrix semigroups [9, 13, 4, 5]. We also show that fundamental problems about the braid compositions are undecidable for braids with at least 5 strands, but decidability of these problems for B_4 remains open.

2 Preliminaries

2.1 Words and Automata

Given an alphabet $\Gamma = \{1, 2, \ldots, m\}$, a word w is an element $w \in \Gamma^*$. We denote the concatenation of two words u and v by either $u \cdot v$ or uv if there is no confusion. For a letter $a \in \Gamma$, we denote by $\overline{a}$ or a^{-1} the inverse letter of a, such that $a\overline{a} = \varepsilon$ where ε is the empty word. We also denote $\overline{\Gamma} = \Gamma^{-1} = \{\overline{1}, \overline{2}, \ldots, \overline{m}\}$ and for a word $w = w_1 w_2 \cdots w_n$, we denote $\overline{w} = w^{-1} = w_n^{-1} \cdots w_2^{-1} w_1^{-1}$.

The free group over a generating set H is denoted by $\mathrm{FG}(H)$, i.e., the free group over two elements a and b is denoted as $\mathrm{FG}(\{a, b\})$. For example, the elements of $\mathrm{FG}(\{a, b\})$ are all

[1] Note that proposed NP-hardness construction is not directly applicable for Identity Problem.

the words over the alphabet $\{a, b, a^{-1}, b^{-1}\}$ that are reduced, i.e., that contain no subword of the form $x \cdot x^{-1}$ or $x^{-1} \cdot x$ (for $x \in \{a, b\}$). Note that $x \cdot x^{-1} = x^{-1} \cdot x = \varepsilon$.

Let $\Sigma = \Gamma \cup \overline{\Gamma}$. Using the notation of [2], we shall also introduce a reduction mapping which removes factors of the form $a\overline{a}$ for $a \in \Sigma$. To that end, we define the relation $\vdash \subseteq \Sigma^* \times \Sigma^*$ such that for all $w, w' \in \Sigma^*$, $w \vdash w'$ if and only if there exists $u, v \in \Sigma^*$ and $a \in \Sigma$ where $w = ua\overline{a}v$ and $w' = uv$. We may then define by $\vdash^*$ the reflexive and transitive closure of $\vdash$.

▶ **Lemma 1** ([2]). *For each $w \in \Sigma^*$ there exists exactly one word $r(w) \in \Sigma^*$ such that $w \vdash^* r(w)$ does not contain any factor of the form $a\overline{a}$, with $a \in \Sigma$.*

The word $r(w)$ is called the reduced representation of word $w \in \Sigma^*$. As an example, we see that if $w = 132\overline{2}1\overline{1}\,\overline{3}\,\overline{1} \in \Sigma^*$, then $r(w) = \varepsilon$.

Using standard notations, a deterministic finite automaton (DFA) is given by quintuple $(Q, \Sigma', \delta, q_0, F)$ where Q is the set of states, Σ' is the *input alphabet*, $\delta : Q \times \Sigma' \to Q$ is the *transition function*, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final states of the automaton. We may extend δ in the usual way to have domain $Q \times \Sigma'^*$. Given a deterministic finite automaton A, the language recognized by A is denoted by $L(A) \subseteq \Sigma'^*$, i.e. for all $w \in L(A)$, it holds that $\delta(q_0, w) \in F$.

▶ **Lemma 2.** *For any given $n \in \mathbb{Z}, n \geq 3$ there is a DFA P_n over a group alphabet Σ, $|\Sigma| = 2n$, with $n + 2$ states and $2n$ edges such that the only word $w \in L(P_n)$ and $r(w) = \epsilon$, has length $|w| = 2^n$.*

Proof. We adapt the proof of a related result over *deterministic finite automata* (DFA) recently shown in [2]. Define alphabets $\Gamma = \{1, 2, \ldots, n\}$, $\overline{\Gamma} = \{\overline{1}, \overline{2}, \ldots, \overline{n}\}$ and $\Sigma = \Gamma \cup \overline{\Gamma}$. It is shown in [2] that for any $n \geq 3$, there exists a DFA A_n, with $n + 1$ states over Σ, such that for any word $w \in \Sigma^*$ where $w \in L(A_n)$ and $r(w) = \varepsilon$ then $|w| \geq 2^{n-1}$. Their proof is constructive and we shall now show an adaption of it. Let $Q = \{q_0, \ldots, q_{n+2}\}$ and q_0 be the initial state and $\{q_{n+2}\}$ is the final state. We define the transition function $\delta : Q \times \Sigma^* \to Q$ of the DFA such that:

$$\delta(q_a, c) = \begin{cases} q_1, & \text{if } c = 1 \text{ and } a = 0; \\ q_{a+1}, & \text{if } c = \overline{a} \text{ and } 1 \leq a \leq n; \\ q_0, & \text{if } c = a \text{ and } 2 \leq a \leq n-1, \\ q_{n+2}, & \text{if c = n and } a = n+1; \end{cases}$$

All other transitions are not defined. The structure of this DFA can be seen in Figure 1. The only path leading to a state q_n, for any $n \geq 3$ with an empty reduced word has length

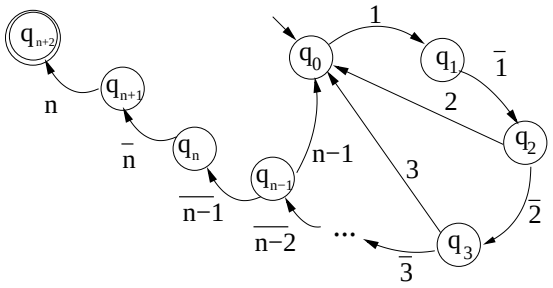


Figure 1 A deterministic finite automaton such that the minimal non empty word w such that $r(w) = \varepsilon$ and $\delta(q_0, w) \in F$ is of length 2^n.

$2^n - 2$. The path for reaching state q_2 with an empty reduced word has length 2 and there are no other paths leading to q_2 with an empty reduced word. Let us assume that another path is leading to q_2 via a path where the larger index of a reachable state on this path is j. Then at least one symbol j is not canceled in the reduced word leading to q_2. Consider a path from q_i to g_{i+1} which corresponds to reduced word v then it should be of the form $v = i \cdot u \cdot \bar{i}$ where a word u is an empty word and it corresponds to a path from a state q_0 to q_i otherwise the reduced word of v is not empty.

Let us assume that the path leading to a state q_i with an empty reduced word, i.e $r(w) = \epsilon$ has length $2^i - 2$. Then the path for reaching state $i + 1$ with a reduced word equal to the empty word can be represented as a path $w \cdot \bar{i} \cdot ui$ where $r(u) = \epsilon$. Since w is the only path to reach q_i from q_0 then we have the required path has a form $w \cdot \bar{i} \cdot wi$ and its length is $(2^i - 2) + 1 + (2^i - 2) + 1 = 2^{i+1} - 2$. Finally we add two extra transitions to make the length of a path to be 2^n. ◀

▶ **Lemma 3.** *For any given $s \in \mathbb{Z}$ which has a binary representation of size m, i.e. $m = \lceil log_2(s) \rceil$, there is a DFA M_s over a group alphabet Σ, $|\Sigma| = O(m^2)$, with $O(m^2)$ states such that the only word $w \in L(M_s)$ and $r(w) = \epsilon$, has a length $|w| = s$.*

Proof. Let us represent s as the following power series

$$\alpha_m 2^m + \alpha_{m-1} 2^{m-1} + \ldots + \alpha_2 2^1 + \alpha_1 2^0, \text{ where } \alpha_i \in \{0, 1\}.$$

For each non-zero α_i and $i \geq 3$ we will contract the automaton P_i from Lemma 2 using unique non-intersecting alphabets for each automaton to avoid any possible cancellation of words between different parts of our final automaton. Also for non-zero α_1, α_2 and α_3 we define three different automata P_1, P_2, P_3 having a linear structure with one ϵ transition, two consecutive ϵ transitions and four consecutive ϵ transitions, which will give us paths of length 2^0, 2^1 and 2^2.

Then we will use a resulting set of automata $P_{i_1}, P_{i_2}, \ldots P_{i_l}$ to build a single automaton by merging the initial state of P_{i_t} with the final state of $P_{i_{t+1}}$ for all $t = 1 \ldots l - 1$ and defining the initial state of P_{i_1} as the initial state of automaton M_s and the final state of P_{i_l} as the final state of M_s. It is easy to see that following the Lemma 2 each P_{i_t} will reach its own final state having an empty word iff the number of executed transition is 2^{i_t}. So finally we build a DFA M_s over a group alphabet, such that the only word $w \in L(M_s)$ and $r(w) = \epsilon$, has a length $|w| = s$.

The DFA M_s over a group alphabet Σ, will have $|\Sigma| = O(m^2)$, $O(m^2)$ states and $O(m^2)$ transitions, since there are no more then m parts $P_{i_1}, P_{i_2}, \ldots P_{i_l}$ and each part P_{i_t} has only $i_t + 2$ states. Moreover the only word $w \in L(M_s)$ and $r(w) = \epsilon$, has a length $|w| = s$. ◀

2.2 Braids

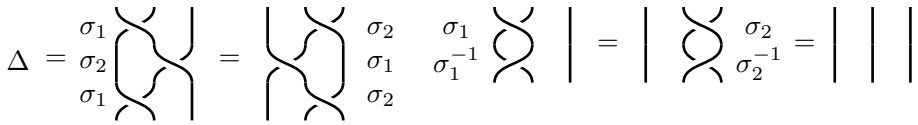
The braid groups can be defined in many ways including geometric, topological, algebraic and algebro-geometrical definitions [17]. Here we provide algebraic definition of the braid group.

▶ **Definition 4.** The n-strand braid group B_n is the group given by the presentation with $n - 1$ generators $\sigma_1, \ldots, \sigma_{n-1}$ and the following relations $\sigma_i \sigma_j = \sigma_j \sigma_i$, for $|i - j| \geq 2$ and $\sigma_i \sigma_{i+1} \sigma_i = \sigma_{i+1} \sigma_i \sigma_{i+1}$ for $1 \leq i \leq n - 2$. These relations are called Artrin's relation.

Words in the alphabet $\{\sigma\ ,\ \sigma^{-1}\}$ will be referred to as braid words [2].

We say that a braid word w is positive if no letter σ_i^{-1} occurs in w. The positive braids form a semigroup denoted by B_n^+. There is one very important positive braid known as the fundamental n-braid, Δ_n. The fundamental braid of the group B_n (also known as Garside element) can be written with $n(n-1)/2$ Artin generators as: $\Delta_n = (\sigma_{n-1}\sigma_{n-2}\dots\sigma_1)(\sigma_{n-1}\sigma_{n-2}\dots\sigma_2)\dots\sigma_{n-1}$.

Geometrically, the fundamental braid is obtained by lifting the bottom ends of the identity braid and flipping (right side over left) while keeping the ends of the strings in a line. The inverse of the fundamental braid Δ_n is denoted by Δ_n^{-1}.



Let $B_3 = \{\sigma_1, \sigma_2 | \sigma_1\sigma_2\sigma_1 = \sigma_2\sigma_1\sigma_2\}$ be the group with three braids. Let Δ be the Garside element: $\Delta = \sigma_1\sigma_2\sigma_1$. Let $\tau : B_3 \to B_3$ be automorphism defined by $\sigma_1 \to \sigma_2$, $\sigma_2 \to \sigma_1$. It is straightforward to check that

$$\Delta b = \tau(b)\Delta, \qquad \Delta^{-1}b = \tau(b)\Delta^{-1}, \qquad b \in B_3. \tag{1}$$

▶ **Lemma 5** ([15]). *Two positive words are equal in B_3 if and only if they can be obtained from each other by applying successively the relation $\sigma_1\sigma_2\sigma_1 = \sigma_2\sigma_1\sigma_2$. A positive word is left or right divisible by Δ if and only if it contains the subword $\sigma_1\sigma_2\sigma_1$ or $\sigma_2\sigma_1\sigma_2$.*

▶ **Lemma 6** ([12, 1]). *Garside normal form – Every braid word $w \in B_n$ can be written uniquely as $\Delta^k b$, where k is an integer and b is a positive braid of which Δ is not a left divisor.*

▶ **Definition 7.** Two braids are isotopic if their braid words can be translated one into each other via the relations from the Defintion 4 plus the relations $\sigma_i\sigma_i^{-1} = \sigma_i^{-1}\sigma_i = 1$, where 1 is the identity (trivial braid).

Let us define a set of natural problems for semigroups and groups in the context of braid composition. Given a finite set of braids B. A multiplicative semigroup $\langle B\rangle$ is a set of braids that can be generated by any finite composition of braids from B.

Membership Problem: Given a braid $b \in B_n$ and a finite set of braids $B \subseteq B_n$, does there exist a composition $Y_1Y_2\cdots Y_r$, with each $Y_i \in B$ such that $Y_1Y_2\cdots Y_r = b$? In other words, is $b \in \langle M\rangle$? In the Membership Problem, when braid b is the trivial braid, we call this problem the Identity Problem.

The Identity Problem for semigroups is a well-known challenging problem which is also computationally equivalent to another fundamental problem in Group Theory: given a finitely generated semigroup S, decide whether a subset of the generator of S generates a nontrivial group (Group Problem) [9].

[2] Whenever a crossing of strands i and $i+1$ is encountered, σ_i or ${\sigma_i}^{-1}$ is written down, depending on whether strand i moves under or over strand $i+1$.

3 NP-hardness of the Membership Problem in B_3

In this section we show that the Membership is NP-hard for braids in B_3. Our reduction will use the following well-known NP-complete problem. Subset Sum Problem: Given a positive integer x and a finite set of positive integer values $S = \{s_1, s_2, \ldots, s_k\}$, does there exist a nonempty subset of S which sums to x?

We will require the following encoding between words over an arbitrary group alphabet and a binary group alphabet, which is well known from the literature.

▶ **Lemma 8.** *Let $\Sigma' = \{z_1, z_2, \ldots, z_l\}$ be a group alphabet and $\Sigma_2 = \{c, d, \overline{c}, \overline{d}\}$ be a binary group alphabet. Define the mapping $\alpha : \Sigma' \to \Sigma_2^*$ by:*

$$\alpha(z_i) = c^i d \overline{c}^i, \alpha(\overline{z_i}) = c^i \overline{d} \overline{c}^i,$$

where $1 \leq i \leq l$. Then α is a monomorphism [3] (see [8] for more details). Note that α can be extended to domain Σ'^* in the usual way.

▶ **Lemma 9** ([7])**.** *Let $\Sigma_2 = \{c, d, \overline{c}, \overline{d}\}$ be a binary group alphabet and define $f : \Sigma_2^* \to B_3$ by: $f(c) = {\sigma_1}^4, f(\overline{c}) = {\sigma_1}^{-4}, f(d) = {\sigma_2}^4, f(\overline{d}) = {\sigma_2}^{-4}$. Then mapping f is a monomorphism.*

The above two morphisms give a way to map words from an arbitrary sized alphabet into the set braid words in B_3. We will later require the following corollary concerning mappings f and α to allow us to argue about the size of braid words constructed by $f \circ \alpha$.

▶ **Corollary 10.** *Let α and f be mappings as defined in Lemma 8 and Lemma 9, then:*

$$f(\alpha(z_j)) = f(c^j d \overline{c}^j) = {\sigma_1}^{4j} {\sigma_2}^4 {\sigma_1}^{-4j}$$

and the length of a braid word from B_3 corresponding a symbol $z_j \in \Sigma'$ is $8j + 4$.

▶ **Theorem 11.** *The* Membership Problem *is NP-hard for braids from B_3*

Proof. We shall use an encoding of the Subset Sum Problem into a set of braids from B_3. Define an alphabet $\Sigma = \Sigma' \cup \{\Delta, \overline{\Delta}\}, \Sigma' = \{1, 2, \ldots, k+2, \overline{1}, \overline{2}, \ldots, \overline{k+2}\}$ that will be extended during the construction.

We now define a set of words W which will encode the Subset Sum Problem (SSP) instance. Note that the length of words in the following set is not bounded by a polynomial of the size of the SSP instance, however this is only a transit step and will not cause a problem in the final encoding. In particular the unary representation of a number s by a word Δ^{2s} will be substituted by a set of words of a polynomial size of i, j and s that will generate a unique word $i \cdot \Delta^{2s} j$.

$$W = \begin{array}{ll} \{1 \cdot \Delta^{2s_1} \cdot \overline{2}, & 1 \cdot \varepsilon \cdot \overline{2}, \\ 2 \cdot \Delta^{2s_2} \cdot \overline{3}, & 2 \cdot \varepsilon \cdot \overline{3}, \\ \vdots & \vdots \\ k \cdot \Delta^{2s_k} \cdot \overline{(k+1)}, & k \cdot \varepsilon \cdot \overline{(k+1)}, \\ (k+1) \cdot \overline{\Delta}^{2x} \cdot \overline{(k+2)}\} \subseteq \Sigma^* & \end{array}$$

Figure 2 shows the way in which the words of W can be combined to give the identity for the reduced word on labels in the graph structure. The above assumption will mean that we

[3] A monomorphism is an injective homomorphism.

start from node 1 of the graph and choose either a^{s_1} or ε to move to node 2. This corresponds to w_1 being equal to either $1 \cdot \Delta^{2s_1} \cdot \overline{2}$ or $1 \cdot \varepsilon \cdot \overline{2}$. We follow such non-deterministic choices from node 1 until we reach a node s_{k+2}. At this point, if we chose $s_{i_1}, s_{i_2}, \ldots, s_{i_l}$, such that they sum to x, then the reduced representation of w will equal $1 \cdot \overline{k+2}$. If there does not exist a solution to the subset sum problem, then it will not be possible to reach the empty word concatenating the labels on a graph structure so it would be possible to get a word $1 \cdot \overline{k+2}$, since it will be only $1 \cdot w' \cdot \overline{k+2}$, where $w' \neq \epsilon$.

Using the encoding idea from Lemma 3 we replace each transition from state i to state j labelled with $\Delta^{2s} j$ by the automaton M_{2s} and then will encode each transition form M_{2s} from a state x to state y with the label $z \in \Sigma$ by the braid word $f(\alpha(x)) \cdot (\sigma_1\sigma_2\sigma_3)^2 \cdot f(\alpha(z)) \cdot f(\alpha(\overline{y}))$ following Corollary 10. We use $\Delta^2 = (\sigma_1\sigma_2\sigma_3)^2$ rather then Δ to have unchanged structure of words since Δ^2 is commutative with any word in B_3. Also each word of the following type $i \cdot \varepsilon \cdot \overline{j}$, where $i, j \in \Sigma'$ can be directly encoded by a braid $f(\alpha(i)) \cdot f(\alpha(\overline{i}))$.

The number of states, the alphabet size and the number of edges for each M_{2s_i} automaton are of the order $O(m^2)$, where m is $log_2 s_i$. Thus we have that the whole automaton after replacing all Δ^{2s_i} transitions by M_{2s_i} will be encoded with the finite number of words of the order $O(k \cdot log_2^2 2s)$, where s is the maximal element of $\{s_1, s_2, \ldots, s_k\}$ and the length of each braid word is of the order $O(k \cdot log_2^2 2s)$. In addition to that we add k words representing ϵ transitions.

Using Lemma 8, we encode the set of words W into a set of braids words over the alphabet $\{\sigma_1, \sigma_1^{-1}, \sigma_2, \sigma_2^{-1}\}$, where the total number of letters will be only polynomially increased. So finally the SSP has a solution if and only iff the braid $f(\alpha(1)) \cdot f(\alpha(\overline{k+2}))$ belongs to the defined semigroup of braid words . ◀

4 Decidability of the Membership problem in B_3

▶ **Theorem 12.** *The membership problem is decidable for braids from B_3.*

Proof. Let us given a set of braid words $\{b_1, b_2, \ldots b_n\}$ from B_3. First let us convert them into Garside normal form $\Delta^k b$ where k is an integer and b is a positive braid of which Δ is not a left divisor.

In order to find the unique Garside decomposition we need to replace each occurrence of σ_1^{-1} with $\sigma_2\sigma_1\Delta^{-1}$, and σ_2^{-1} with $\sigma_1\sigma_2\Delta^{-1}$, and then push all Δ^{-1} to the right using (1). After that iteratively one should successively replace all subwords $\sigma_1\sigma_2\sigma_1$ and $\sigma_2\sigma_1\sigma_2$ with Δ and push them to the right using (1).

Then we construct a finite state automaton A with n multi-states loops representing n braid words in the Garside normal form. For each braid word b_i in the Garside form $\Delta^k b$, where $b = \sigma_{j_1}\sigma_{j_2}\ldots\sigma_{j_{|b_i|}}$ we define a sequence of $|b_i|$ transitions

$$s_{1,i} \xrightarrow{\Delta^k} s_{2,i} \xrightarrow{\sigma_{j_1}} s_{3,i} \xrightarrow{\sigma_{j_2}} s_{4,i} \to \ldots \to s_{|b_i|-1,i} \xrightarrow{\sigma_{j_{|b_i|}}} s_{|b_i|,i}$$

After that we merge all states $s_{1,i}$ and $s_{|b_i|,i}$ for all $i's$ into a single state s_0, which will be

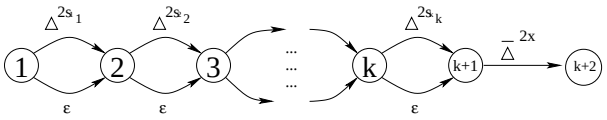


Figure 2 The initial structure of a product which forms the identity on labels.

the initial and the final state of the automaton A. Thus A has n multi-states loops from the initial/final state s_0 representing n braid words.

If the automaton A has q states we will first show that any path from a state s to t of the length greater than $3 \cdot 2^{(q^2-3q)}$ and equal to Δ^k, for some $k \in \mathbb{Z}$ should contain a path of shorter length from s to t which is equal to $\Delta^{k'}$. Suppose A has a path u from state s to t which is equal to Δ^k. Then u can be decomposed in at least one of two ways. Either in *Case 1* there exist two words $v_1 = \Delta^{k_1}$ and $v_2 = \Delta^{k_2}$, $k_1, k_2 \in \mathbb{Z}$ such that $u = v_1 \cdot v_2$ or in *Case 2* there exist two words $v_1 = \Delta^{\mu_1}$ and $v_2 = \Delta^{\mu_2}$ such that

- if μ_1, μ_2 are even numbers then $u = \sigma_1 v_1 \sigma_2 v_2 \sigma_1$ or $u = \sigma_2 v_1 \sigma_1 v_2 \sigma_2$ or $u = \sigma_1^{-1} v_1 \sigma_2^{-1} v_2 \sigma_1^{-1}$ or $u = \sigma_2^{-1} v_1 \sigma_1^{-1} v_2 \sigma_2^{-1}$;
- if μ_1, μ_2 are odd numbers then $u = \sigma_1 v_1 \sigma_1 v_2 \sigma_1$ or $u = \sigma_2 v_1 \sigma_2 v_2 \sigma_2$ or $u = \sigma_1^{-1} v_1 \sigma_1^{-1} v_2 \sigma_1^{-1}$ or $u = \sigma_2^{-1} v_1 \sigma_2^{-1} v_2 \sigma_2^{-1}$;
- if μ_1 is even and μ_2 is odd then $u = \sigma_1 v_1 \sigma_2 v_2 \sigma_2$ or $u = \sigma_2 v_1 \sigma_1 v_2 \sigma_1$ or $u = \sigma_1^{-1} v_1 \sigma_2^{-1} v_2 \sigma_2^{-1}$ or $u = \sigma_2^{-1} v_1 \sigma_1^{-1} v_2 \sigma_1^{-1}$,
- if μ_1 is odd and μ_2 is even then $u = \sigma_2 v_1 \sigma_2 v_2 \sigma_1$ or $u = \sigma_1 v_1 \sigma_1 v_2 \sigma_2$ or $u = \sigma_2^{-1} v_1 \sigma_2^{-1} v_2 \sigma_1^{-1}$ or $u = \sigma_1^{-1} v_1 \sigma_1^{-1} v_2 \sigma_2^{-1}$;

Any subword u' of u such that $u' = \Delta^{k'}$ can also be decomposed in at least one of these two ways, so we can recursively decompose u and the resulting subwords until we have decomposed u into single symbols. So, we can specify a certain type of parse tree such that the automaton A has a path u from state s to t which is equal to Δ^k if and only if we can build this type of parse tree for u.

Let us define a parse tree for a given word u equal to Δ^k from a state s to t as follows. Every internal node corresponds to a subword u' of u, such that u' is a power of the fundamental braid Δ and the root of the whole tree corresponds to u. The leaves store individual symbols $\sigma_1, \sigma_1^{-1}, \sigma_2, \sigma_2^{-1}$. When read from left to right, the symbols in the leaves of any subtree form the word that corresponds to the root of the subtree. Following Lemma 5 each internal node is

1. either the node that has two children, both of which are internal nodes that serve as roots of subtrees (corresponds to Case 1).
2. or the node that has five children, where the first, third and fifth (from the left) children are single symbols and the second and fourth children in the middle can be either empty or an internal node that is the root of another subtree which is equal to Δ^r, $r \in \mathbb{Z}$ (corresponds to Case 2).

We label each internal node γ with a pair of states (s_{j_1}, s_{j_2}) such that if u' is the subword of u that corresponds to the subtree rooted at γ, and $u = v_1 \cdot u' \cdot v_2$ then $s_{j_1} \in \delta(s, v_1)$, $s_{j_2} \in \delta(s_{j_1}, u')$. are the states reached after reading the input prefixes v_1 and $v_1 u'$, respectively, during the accepting computation under consideration. This implies that $s \in \delta(s_{j_2}, v_2)$ and (s,t) is the label associated with the root of the tree.

If the parse tree of u has two nodes ξ_1 and ξ_2 with the same state-pair label such that ξ_2 is a descendant of ξ_1, then there exists a word shorter than u which is $\Delta^{k'}$. This is because we can replace the subtree rooted at ξ_1 with the subtree rooted at ξ_2. Furthermore, if an internal node ξ_1 is labeled with a pair (p,q), for some $p, q \in Q$ then the subword u' corresponding to the subtree rooted at ξ_1 can be removed from u, obtaining a shorter path. Therefore the height of the subtree corresponding to the shortest subword equal to a power of Δ is at most $q^2 - q$ and the number of leaves of a parse tree of height h is at most $3(2^{q^2-q} - 2)$. In the maximal case we have the complete binary tree of depth $q - 1$ with three extra leaves in each internal node and on the last level every node has three leaves. Assume that we have a path

with some larger length from a state s to t with a braid word w equal to Δ^k then according to above proof it should be two states p and q which will appear twice in the path with the following order $s \xrightarrow{A} p \xrightarrow{B} p \xrightarrow{C} q \xrightarrow{D} q \xrightarrow{E} t$ and decomposing it into five parts A, B, C, D, E, where $w = A \cdot B \cdot C \cdot D \cdot E = \Delta^k$, $C = \Delta^{k_1}$, $B \cdot C \cdot D = \Delta^{k_2}$. From this follows that any path from a state s to t in A, which is equal to Δ^k, can be represented by a linear combination of shorter Δ paths each of which has length at most $3(2^{q^2-q} - 2)$. This gives us a bound on the number of values for expressing a power of Δ's during modification of automata A and also will guarantee the termination of the following procedure, where we add a number of new transitions between states s and t to get a direct edge labelled by a power of Δ:

1. For any of the following sequences, where $even_1$ and $even_2$ are any even numbers or 0, which means that in case of Δ^0 the transition is not there:
   $s \xrightarrow{\sigma_1} \xrightarrow{\Delta^{even_1}} \xrightarrow{\sigma_2} \xrightarrow{\Delta^{even_2}} \xrightarrow{\sigma_1} t$; $\quad s \xrightarrow{\sigma_2} \xrightarrow{\Delta^{even_1}} \xrightarrow{\sigma_1} \xrightarrow{\Delta^{even_2}} \xrightarrow{\sigma_2} t$ we add
   $s \xrightarrow{\Delta^{even_1+even_2+1}} t$
2. For any of the following sequences, where odd is any odd number:

   $$s \xrightarrow{\sigma_1} \xrightarrow{\Delta^{odd}} \xrightarrow{\sigma_1} \xrightarrow{\sigma_2} t; s \xrightarrow{\sigma_2} \xrightarrow{\Delta^{odd}} \xrightarrow{\sigma_2} \xrightarrow{\sigma_1} t; s \xrightarrow{\sigma_1} \xrightarrow{\sigma_2} \xrightarrow{\Delta^{odd}} \xrightarrow{\sigma_2} t; s \xrightarrow{\sigma_2} \xrightarrow{\sigma_1} \xrightarrow{\Delta^{odd}} \xrightarrow{\sigma_1} t$$

   we add $s \xrightarrow{\Delta^{odd+1}} t$
3. For any of the following sequences, where odd_1 and odd_2 are any odd numbers:
   $s \xrightarrow{\sigma_1} \xrightarrow{\Delta^{odd_1}} \xrightarrow{\sigma_1} \xrightarrow{\Delta^{odd_2}} \xrightarrow{\sigma_1} t$; $\quad s \xrightarrow{\sigma_2} \xrightarrow{\Delta^{odd_1}} \xrightarrow{\sigma_2} \xrightarrow{\Delta^{odd_2}} \xrightarrow{\sigma_2} t$
   we add $s \xrightarrow{\Delta^{odd_1+odd_2+1}} t$
4. For any of the following sequences, where odd is any odd number and $even$ is any even number, we add $s \xrightarrow{\Delta^{odd+even}} t$:

   $$s \xrightarrow{\sigma_1} \xrightarrow{\Delta^{odd}} \xrightarrow{\sigma_1} \xrightarrow{\Delta^{even}} \xrightarrow{\sigma_2} t; s \xrightarrow{\sigma_2} \xrightarrow{\Delta^{odd}} \xrightarrow{\sigma_2} \xrightarrow{\Delta^{even}} \xrightarrow{\sigma_1} t;$$

   $$s \xrightarrow{\sigma_1} \xrightarrow{\Delta^{even}} \xrightarrow{\sigma_2} \xrightarrow{\Delta^{odd}} \xrightarrow{\sigma_2} t; s \xrightarrow{\sigma_2} \xrightarrow{\Delta^{even}} \xrightarrow{\sigma_1} \xrightarrow{\Delta^{odd}} \xrightarrow{\sigma_1} t$$

5. For any of the following sequences, where z_1, z_2 are integer numbers:
   $s \xrightarrow{\Delta^{z_1}} \xrightarrow{\Delta^{z_2}} t$; we add $s \xrightarrow{\Delta^{z_1+z_2}} t$
6. If $s = t$, for any of the above cases 1-5, then the new edge should make a cycle labelled by some power of Δ, i.e Δ^a, where $a \in \mathbb{Z}$. In this case if there are no other cyclic edges from a state s then we add a cyclic edge from state with an expression $\Delta^{x_{new} \cdot a}$, where x_{new} is a new symbol from some infinite alphabet $\{x_i | i \in \mathbb{N}\}$, which will be used later as unknown in a system of equations. Assume that there is a cyclic edge with Δ^z where z is represented by a linear expression $Expr(x_{i_1}, x_{i_2}, ..., x_{i_j})$ then no extra cyclic edges are added, but the expression $Expr(x_{i_1}, x_{i_2}, ..., x_{i_j})$ will be replaces by $Expr(x_{i_1}, x_{i_2}, ..., x_{i_j}) + x_{new} \cdot a$.

Now we will generalize the cases 1-5 to incorporate the idea of expressions into the new set of rules in the straightforward way: for any Δ^x, x is an expressions $Expr()$ that can be a constant of a linear function. In case 6, if a in Δ^a is already some expression of the form $Expr_1(...) + c$, where $c \in \mathbb{Z}$, we replace the expression on the original cycle by $Expr(x_{i_1}, x_{i_2}, ..., x_{i_j}) + Expr_1(...) + x_{new} \cdot c$. [4]

[4] Formally we should also multiply $Expr_1(...)$ by x_{new} which can be avoided since the variables in the linear expression $Expr_1(...)$ are independent from x_{new} and in this case can be simply renamed.

7. For $s \xrightarrow{\Delta^{Expr_1()}} p \xrightarrow{\Delta^{Expr_2()}} p \xrightarrow{\Delta^{Expr_3()}} t$; we add $s \xrightarrow{\Delta^{Expr_1()+Expr_2()+Expr_3()}} t$, where any of the expressions $Expr_i()$ can be a constant.
8. For any of the following sequences, where $\alpha, \beta \in \mathbb{Z}$ $s \xrightarrow{\sigma_{j_1}} p_0 \xrightarrow{\Delta^{Expr_0()}} p_0 \xrightarrow{\Delta^{\alpha}} p_1 \xrightarrow{\Delta^{Expr_1()}} p_1 \xrightarrow{\sigma_{j_2}} p_2 \xrightarrow{\Delta^{Expr_2()}} p_2 \xrightarrow{\Delta^{\beta}} p_3 \xrightarrow{\Delta^{Expr_3()}} p_4 \xrightarrow{\sigma_{j_3}} t$ we add the transition $s \xrightarrow{\Delta^{Expr_0()+Expr_1()+Expr_2()+Expr_3()+\alpha+\beta+1}} t$ if the values of the σ's indices and even/odd constrains for Δ in between them will match with one of the cases 1-4. In order to record the information about even/odd case for the expressions we can add one of the following equations:
   - if $Expr()$ should have an even value then we add: $Expr() = 2 \cdot x_{new}$
   - if $Expr()$ should have an odd value then we add: $Expr() = 2 \cdot x_{new} + 1$

Now let us modify the automaton A following rules 1-8 in the following way. We apply rules 1-5 in any order until it is possible then if no cycles are created then the process will terminate since there will be only a finite number of such sequences. In this case no other rules 1-8 are applicable. If there is at least one cycle after applying rules 1-5 we iteratively apply rules 6, 7 and 8: as many times as possible each, then the process is starting again. When no extra transition can be added according to the above rules the process will terminate.

Finally in order to check the membership for a braid $b = \Delta^k \cdot w$ we will need to find a path from the initial to the final state of A that consists of a set of $|w|$ edges in the automaton A connected by Δ's such that total sum for powers of Δ's will be equal to k and the positive word after moving all Δ's to the left will be equal to w.

If the length of a braid word b is equal to l then in the canonical form $\Delta^k \cdot w$ that is isotopic to b we have that the length of a positive word w , i.e $h = |w|$, is limited by $2 \cdot l$, following the process of rewriting. Then the absolute value of the negative power of Δ is $2 \cdot l$ and the number of positive Δ's that can be derived from w is bounded by $\frac{2l}{3}$. So the absolute value of k is bounded by $|\frac{2l}{3} - l| = \frac{l}{3}$.

Let w be a positive word $\sigma_{i_1} \cdot \sigma_{i_2} \cdot \ldots \cdot \sigma_{i_h}$. If b is in the semigroup of braids generated by $b_1, b_2, \ldots, b_n$ then it should be a word in A that consists of a set of h edges with σ labels which are connected by Δ's such that total sum for powers of Δ's will be equal to k and the positive word after moving all Δ's to the left will be equal to w:

$$\Delta^{j_1}\sigma_{i_1}\Delta^{j_2}\sigma_{i_2}\ldots\Delta^{j_h}\sigma_{i_h} = \Delta^k w; i \in \{1,2\}, j \in \mathbb{Z}, \sum_{d=1}^{h} j_d = k$$

Now we nondeterministically choose a particular pattern of σ's and even/odd values of $j's$ and checking whether the resulting positive word after moving all Δ's to the left will be equal w. Obviously it can be converted into deterministic algorithm since any of the non-deterministic guesses will be made from some finite sets. Then we also define a system of linear Diophantine equations that will correspond to the Δ transitions which are connecting $\sigma_{i_1}, \sigma_{i_2} \ldots \sigma_{i_h}$ transitions. Let us assume that the subset of σ's transitions are connected via a Δ transitions in modified automaton A with the following expressions $R_1, R_2, \ldots, R_{h+1}$, where each R_l can be

- ϵ, i.e. empty, which corresponds to direct connection of σ_{l-1} and σ_l
- Δ_l^k for some $k_l \in \mathbb{Z}$
- $\Delta^{Expr_l(x_{z_1}, x_{i_2}, \ldots, x_{z_\mu})}$ - which is a linear expression with μ variables, $x_i \in \mathbb{N}$.

The system of linear Diophantine equations and inequalities will consists of the equation $\sum_{d=1}^{h} R_d = k$ and several equations representing a number of constraints for even/odd properties as well as the restrictions on variables to be positive integers. Since this system is known to be decidable [3] we can decide the membership problem by checking the existence of a solution for our system of linear Diophantine equations and inequalities. ◀

The composition problems become harder with a larger number of strands.

Here we illustrate the technique to show undecidability of the fundamental problem whether a semigroup of braids is a group.

▶ **Lemma 13.** *[7] Subgroups* $\langle {\sigma_1}^4, {\sigma_2}^4\rangle$, $\langle {\sigma_4}^2, d\rangle$ *of the group* B_5 *are free and* B_5 *contains the direct product* $\langle {\sigma_1}^4, {\sigma_2}^4\rangle \times \langle {\sigma_4}^2, d\rangle$ *of two free groups of rang 2 as a subgroup, where* $d = \sigma_4\sigma_3\sigma_2\sigma_1^2\sigma_2\sigma_3\sigma_4$.

▶ **Theorem 14.** *The Identity problem and the Group Problem are undecidable for braids in* B_5.

Proof. It was recently proved in [6] the undecidability of the following Identity Correspondence Problem (ICP) which asks whether a finite set of pairs of words (over a group alphabet) can generate an identity pair by a sequence of concatenations:

Identity Correspondence Problem (ICP) - Let $\Sigma = \{a, b\}$ be a binary alphabet and $\Pi = \{(s_1, t_1), (s_2, t_2), \ldots, (s_m, t_m)\} \subseteq \mathrm{FG}(\Sigma) \times \mathrm{FG}(\Sigma)$. Is it decidable to determine if there exists a nonempty finite sequence of indices $l_1, l_2, \ldots, l_k$ where $1 \leq l_i \leq m$ such that $s_{l_1}s_{l_2}\cdots s_{l_k} = t_{l_1}t_{l_2}\cdots t_{l_k} = \varepsilon$, where ε is the empty word (identity)?

We can directly use the Lemma 13 to encode Identity Correspondence Problem in terms of braid words. We shall use a straightforward encoding to embed an instance of the Identity Correspondence Problem into a set of braids. Given an instance of ICP say $W \subseteq \Sigma^* \times \Sigma^*$ where $\Sigma = \{a, b, a^{-1}, b^{-1}\}$ generates a free group. Define two morphisms ϕ and ψ, $\Sigma \to B_5$:

$$\phi(a) = {\sigma_1}^4, \phi(b) = {\sigma_2}^4, \phi(a^{-1}) = {\sigma_1}^{-4}, \phi(b^{-1}) = {\sigma_2}^{-4}.$$

$$\psi(a) = {\sigma_4}^2, \psi(b) = \sigma_4\sigma_3\sigma_2\sigma_1^2\sigma_2\sigma_3\sigma_4,$$

$$\psi(a^{-1}) = {\sigma_4}^{-2}, \psi(b^{-1}) = \sigma_4^{-1}\sigma_3^{-1}\sigma_2^{-1}\sigma_1^{-2}\sigma_2^{-1}\sigma_3^{-1}\sigma_4^{-1}.$$

Both ϕ and ψ can be naturally extended for words $\Sigma^* \to B_5$:

$$\phi(a_{i_1} \ldots a_{i_j}) = \phi(a_{i_1}) \cdot \ldots \cdot \phi(a_{i_j}); \quad \psi(a_{i_1} \ldots a_{i_j}) = \psi(a_{i_1}) \cdot \ldots \cdot \psi(a_{i_j})$$

For each pair of words $(s, t) \in \Pi$, define the braid word $\phi(s) \cdot \psi(t)$. Let S be a semigroup generated by these braid words. If there exists a solution to ICP, i.e., $(\varepsilon, \varepsilon)$, then we see that $\phi(\varepsilon) \cdot \psi(\varepsilon) = 1 \in S$ where 1 is the trivial braid. Otherwise, since ψ and ϕ are injective homomorphisms, $1 \notin S$.

Thus we have that the problem whether a trivial braid can be expressed by any finite length composition of braids from B_5 is undecidable. The ICP problem is also computationally equivalent to the following Group Problem: is the semigroup generated by a finite set of pairs of words (over a group alphabet) a group. Using the same morphisms ϕ and ψ we can encode the Group Problem for words by braids, having that the Group Problem for braids in B_5 is also undecidable. ◀

5 Conclusion

The paper introduce a few challenging algorithmic problems about topological braids opening new connections between braid groups, combinatorics on words, complexity theory and provides solutions for some of these problems by application of several techniques from automata theory, matrix semigroups and algorithms.

We show that the membership problem for B_3 is decidable. The complexity of the problem is at least NP-hard[5] and the basic upper bound for the time complexity of proposed construction is exponential. The question about the exact complexity of the membership problem in B_3 is left open and may require a further study in terms of improving the lower bound or designing a more efficient algorithm. We believe that the proposed technique for deciding the membership problem in B_3 can also be used to design the algorithm for the Freeness Problem: Given a set of braids with n strands $B = b_1, \ldots, b_k \in B_n$. Let us denote a semigroup of braids, generated by B and the operation of composition, by $\langle B \rangle$. Check whether any two different concatenations of braids from B are not isotopic. I.e. is a semigroup of braids $\langle B \rangle$ free? One of the possibilities for solving above problem with our techniques is to follow ideas proposed in [10], but arranging a more sophisticated procedure of dealing with powers of Δ's. Finally in this paper we show that fundamental problems about the braid compositions are undecidable for braids with at least 5 strands, but decidability of these problems for B_4 remains open.

Acknowledgements. The author is grateful for many fruitful discussions with Sergei Chmutov and Victor Goryunov on the computational problems in tolopogy.

References

1 P. Dehornoy, I. Dynnikov, D. Rolfsen, B. Wiest. Ordering braids. Mathematical Surveys and Monographs 148, Providence, R.I.: American Mathematical Society, ISBN 978-0-8218-4431-1, MR 2463428, (2008).

2 T. Ang, G. Pighizzini, N. Rampersad, and J. Shallit. Automata and reduced words in the free group. In *arXiv:0910.4555*, 2009.

3 Farid Ajili and Evelyne Contejean. Avoiding slack variables in the solving of linear Diophantine equations and inequations. Theoret. Comput. Sci., 173:183–208, 1997.

4 Paul C. Bell, Mika Hirvensalo, Igor Potapov. Mortality for 2×2 Matrices Is NP-Hard. MFCS 2012:148–159.

5 Paul C. Bell, Igor Potapov. On the Computational Complexity of Matrix Semigroup Problems. Fundam. Inform. 116(1–4):1–13 (2012).

6 Paul C. Bell, Igor Potapov. On the Undecidability of the Identity Correspondence Problem and its Applications for Word and Matrix Semigroups. Int. J. Found. Comput. Sci. 21(6): 963-978 (2010).

7 V. N. Bezverkhnii, I. V. Dobrynina. Undecidability of the conjugacy problem for subgroups in the colored braid group R_5. Math. Notes, 65:1 (1999), 13–19.

8 J.-C. Birget and S. Margolis. Two-letter group codes that preserve aperiodicity of inverse finite automata. *Semigroup Forum*, 76(1):159–168, 2008.

9 Christian Choffrut, Juhani Karhumaki. Some decision problems on integer matrices. ITA 39(1):125–131 (2005).

10 Julien Cassaigne, François Nicolas: On the decidability of semigroup freeness. RAIRO – Theor. Inf. and Applic. 46(3):355–399 (2012).

11 David B. A. Epstein, M. S. Paterson, J. W. Cannon, D. F. Holt, S. V. Levy, W. P. Thurston. Word Processing in Groups. A K Peters/CRC Press, 352p, 1992.

12 F.A. Garside. The braid group and other groups. In *Quart. J. Math. Oxford Ser. (2), 20:235–254*, 1969.

[5] The NP-hardness result is in line with the best current knowledge about similar problem in $SL(2, \mathbb{Z})$.

13 Yuri Gurevich, Paul Schupp. Membership Problem for the Modular Group. SIAM J. Comput. 37(2): 425–459 (2007).

14 Samuel J. Lomonaco, Louis H. Kauffman. Quantizing braids and other mathematical structures: the general quantization procedure. Proc. SPIE 8057, Quantum Information and Computation IX, 805702 (June 02, 2011); doi:10.1117/12.883681.

15 S.Yu. Orevkov. Quasipositivity problem for 3-braids Proceedings of 10th Gokova Geometry-Topology Conference 2004, pp. 89–93. Turkish Journal of Math., 28(2004), 89–93.

16 Mike Paterson, Alexander A. Razborov. The Set of Minimal Braids is co-NP-Complete. J. Algorithms 12(3): 393–408 (1991).

17 Braid group, Wikipedia. `http://en.wikipedia.org/wiki/Braid_group`, July 2013.

DLOGTIME Proof Systems

Andreas Krebs[1] and Nutan Limaye[2]

1 **University of Tübingen, Germany,** `mail@krebs-net.de`
2 **Indian Institute of Technology, Bombay, India,** `nutan@cse.iitb.ac.in`

Abstract

We define DLOGTIME proof systems, DLTPS, which generalize NC^0 proof systems. It is known that functions such as $Exact_k$ and Majority do not have NC^0 proof systems. Here, we give a DLTPS for $Exact_k$ (and therefore for Majority) and also for other natural functions such as Reach and $Clique_k$. Though many interesting functions have DLTPS, we show that there are languages in NP which do not have DLTPS. We consider the closure properties of DLTPS and prove that they are closed under union and concatenation but are not closed under intersection and complement. Finally, we consider a hierarchy of polylogarithmic time proof systems and show that the hierarchy is strict.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Proof systems, DLOGTIME, NC^0

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.189

1 Introduction

In a seminal paper by Cook and Reckhow [6] a *proof system* for a language was defined as a polynomial time computable function, called a *verifier*, whose range is all the words in the language. For any language in NP, there exists such a proof system. This therefore gives a way of viewing the complexity classes in the framework of proof systems. Polynomial time verifiers naturally characterize languages in NP. To understand complexity classes that contain NP, some works have considered proof systems with more powerful verifiers. (See for example [9, 5, 4, 3].)

Taking a dual approach, proof systems with very weak verifiers have been recently studied in [2, 8]. It is known that for any language in NP there exists a uniform AC^0 proof system, i.e. a proof system in which the verifier is a function computable by a uniform AC^0 circuit. In [2], a restriction of AC^0 proof systems, namely NC^0PS was considered, where the verifier is a (possibly non-uniform) NC^0 circuit. They observed that there are NP complete languages which have NC^0PS. However, there are even regular languages for which there are no NC^0PS. It is natural to define a proof system that generalizes NC^0PS, but is not as general as proof systems for NP.

In this work we investigate a proof system which generalizes uniform NC^0PS (i.e. the verifier is a uniform NC^0 circuit) but is more restrictive than AC^0PS. We consider a proof system in which the verifier is a deterministic log-time Turing machine.

Deterministic log-time Turing machines can compute an AND of $\omega(1)$ bits and therefore are more powerful than uniform NC^0 circuits. However, they cannot compute an AND of $\Theta(n)$ bits, and therefore are less powerful than uniform AC^0 circuits. Also NC^0 verifiers can make only $O(1)$ queries to the *proof bits*[1] as opposed to DLOGTIME verifiers which can

[1] The input to the verifier is called a proof and it is a string over some fixed alphabet.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 189–200
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

make $O(\log n)$ queries. (See Section 2 for formal definitions.) We see that this makes DLTPS much more powerful than uniform NC^0PS. Note, however, that the DLOGTIME verifier is by default uniform. At first, this may seem like a big restriction. However, it is interesting to note that most languages which are shown to have NC^0 proof systems in [2], in fact have uniform proof systems.

Our results

- In the same spirit as in [2, 8], we prove that many interesting natural functions have DLTPS. In [2], it was proved that functions such as ExactOR, Majority, Exact_k do not have NC^0PS. We prove (Section 3) that for any DLOGTIME computable function $k : \mathbb{N} \to \mathbb{N}$, Exact_k has DLTPS, and hence ExactOR and Majority also have DLTPS.
- We consider some well-known graph problems. We prove that Reachability on directed graphs has DLTPS. We also prove that for any DLOGTIME computable function $k : \mathbb{N} \to \mathbb{N}$, Clique_k has a DLTPS (Section 4).
- As a part of our study of DLTPS we analyze the closure properties of DLTPS (Section 5). We prove that it is closed under union and concatenation. That is, if L_1 and L_2 are two languages which have DLTPS then $L_1 \cup L_2$ and $L_1 L_2$ also have DLTPS. On the other hand, we prove that DLTPS are not closed under intersection and complement.
- We prove that there is a language in NP for which there is no DLTPS. We prove this by showing that any language which has DLTPS can be recognized by a non-deterministic Turing machine in time $O(n^2 \log^3 n)$ (Theorem 10). A similar lemma was proved in [2] for NC^0PS. Our proof is similar in spirit, however we have to be more careful in our argument due to the adaptive nature of the queries made by the DLOGTIME verifier.
- Finally, we consider a hierarchy of polylogarithmic time proof systems. We show that this hierarchy is strict (Section 6). Our proof of the hierarchy theorem uses lazy diagonalization as in the proof of non-deterministic time hierarchy theorem [10]. However, our proof is more delicate due to some technical reasons: there are two crucial parameters in our hierarchy theorem– the proof length and the running time of the DLOGTIME machine. A naive implementation of non-deterministic time hierarchy theorem gives a hierarchy on these two parameters simultaneously. However, we observe that DLOGTIME machine cannot access too many proof bits and that diagonalization does not need to compute the full output of the proof system, but only the length of the output. We believe that our proof may be of independent interest.

Related work. Krebs et al. in [8] defined poly log AC^0 proof systems. The verifier for this proof system is a bounded fan-in $O(\log \log n)$ depth circuit (possibly non-uniform) with at most $O(1)$ alternations. By definition, poly log AC^0 proof systems generalize NC^0PS and are a restriction of AC^0 proof systems.

Though DLTPS and poly log AC^0 proof systems are both generalizations of NC^0PS, they both are very different. A poly log AC^0 verifier of depth $c \log \log n$ can query $O(\log^c n)$ proof bits; whereas a DLOGTIME verifier can query at most $O(\log n)$ bits. In this sense, poly log AC^0 proof system is more powerful than DLTPS. However, a DLOGTIME verifier can make adaptive queries to the proof bits[2]. This means that for computing one output bit, a DLOGTIME verifier can potentially query all proof bits.

[2] The queries made to the proof bits are said to be adaptive if having read a few bits of the proof, the locations of the bits to be read subsequently depend on the values of those bits.

Interestingly, [8] gave poly log depth AC^0 proof systems for Exact_k, ExactOR, and Majority, but the the DLTPS and poly log AC^0 proof systems for these functions are fundamentally different. .

2 Proof Systems

DLOGTIME Turing machines have been studied in the past. (See for example [1, 7].) We use the definition from [1]. A DLOGTIME Turing machine is a deterministic Turing machine which has an input tape, a constant number of read-write tapes, and an index tape. We assume that none of the tapes have an end marker. The machine runs in time $O(\log n)$ time, where n is the length of the input. In one step, the machine can read a bit from the input indexed by the index tape.

▶ **Definition 1.** A function $f : \Gamma^* \to \Sigma^*$ is computable in DLT if there exists a DLOGTIME Turing machine M such that $M(\sigma, i, x)$ halts and accepts iff $w_i = \sigma$, where $w_0 \dots w_{|x|-1} = f(x)$.

▶ Remark. Not every DLT describes a function. Consider a machine M that accepts no letter at the first position but a letter at the second position.

We assume that the index i is given in binary notation. In general, whether i is encoded in binary or unary is not important to the power of the functions considered here, since we can always assume that $i \leq |x|$.

A language is said to be accepted by a Turing machine if it accepts all the words in the language and nothing else. For proof systems, this notion is reversed. A language is said to have a Turing machine as its proof system if the output of the machine is exactly all the words of the language, while the input ranges over all possible strings. Formally,

▶ **Definition 2** (Proof System)**.** Let Σ, Γ be alphabets. A proof system for $L \subseteq \Sigma^*$ is a map $f : \Gamma^* \to L$, that is onto. A proof system f is polynomial bounded if there is a polynomial p such that for every $y \in L$ there exists an x with $f(x) = y$ and $|x| < p(|y|)$.

▶ Remark. Our proof systems cannot accept the empty language, i.e. no word at all. The "smallest" language we can accept is the language which contains only the empty word, by an oracle that always rejects. The input alphabet can be assumed to be $\{0, 1\}$ without loss of generality (by binary encoding of Γ).

In the definition there are two properties required for f to be a proof system of L. First, for all inputs x the output $f(x)$ must be in L. We refer to this property as *correctness*. Second, for every $y \in L$ there is an input x such that $f(x) = y$, i.e. f is surjective.

Here we will study proof systems where the function f is computable in DLT. Given that a machine that runs in logarithmic time cannot output a long string, we say that a function f is computable in DLT if every bit (or letter) of the output is computable in DLT.

Combining the last two definitions we say the language L has a DLT proof system (DLTPS) if there is a polynomial bounded proof system f for L such that f is computable in DLT. Note that, as input and output lengths in DLTPS are polynomially related, and as the computational power of DLTPS is bounded by logtime in terms of the input length, it is also bounded by logtime in terms of the output length.

In [1], simple functions were shown to be computable in DLOGTIME. They showed that given an input x, a DLOGTIME machine can compute $|x|$ by a double binary search. Addition and subtraction of two $O(\log n)$ bit numbers can be done in DLOGTIME. Logarithm of a $O(\log n)$ bit number can be computed in DLOGTIME.

3 Word Problems

In this section we give DLTPS for various languages which are subsets of $\{0,1\}^*$. ExactOR is a set of all string from $\{0,1\}^*$ with exactly one bit set to 1. For $k : \mathbb{N} \to \mathbb{N}$, $\text{Exact}_k \subset \{0,1\}^*$, is a set of all strings with exactly k bits set to 1. And Majority $\subset \{0,1\}^*$ is a set of all strings with at least as many 1s as 0s.

In [2], the above functions were considered. They proved that ExactOR, Majority and ExMaj do not have NC^0PS. Here, we prove that all these languages have DLTPS. In fact we prove slightly more: we show that for every function k computable in DLT, the language Exact_k has a DLTPS.

Before we start to give a general proof for arbitrary functions k, we will look at the specific case when $k(n) = 1$. So we need a proof system which outputs all strings with exactly one occurrence of 1, i.e. the language ExactOR.

By definition a proof system is a function $f : \Gamma^* \to \Sigma^*$, but in order to explain how a proof system works it is helpful to give some interpretation to the proof x, when outputting $f(x)$. In the case of ExactOR the proof should encode the position of the unique one in the string and the length of the output.

On (σ, i, x), the machine interprets the first $\log |x|$ bits of x as the prescribed position α for the unique 1 in the output string, and the length of x will be the length of the output. Let α denote the value of the first $\log |x|$ bits of x. The function computed by the machine is: $f(x) = 0^\alpha 10^{|x|-\alpha-1}$.

The machine can be described formally as follows:

$M(\sigma, i, x)$

Let $n = |x|$.
if $i < n$ **then**
  if $i = \alpha$ **then**
    If $\sigma = 1$ Accept.
  else
    If $\sigma = 0$ Accept.
  end if
end if
Reject.

$$x = \overbrace{\boxed{\alpha}}^{\log n} \underbrace{\cdots\cdots\cdots\cdots}_{n}$$

Note that α need not be computed explicitly: to check whether $i = \alpha$, we only need to compare i with the first $\log |x|$ bits of x. Also note that $n = |x|$ can be computed in DLOGTIME. It is easy to see that the machine outputs strings with exactly one 1. As we cycle through all x, α gets all values in the range $[0, n-1]$. This ensures that the range of the function defined by the machine is ExactOR and it is onto.

▶ **Lemma 3.** ExactOR *has a* DLTPS.

Our next goal to prove the generalization for every function k computable in DLT. As a first step we give another simple proof system. Given a function $k : \mathbb{N} \to \mathbb{N}$ computable by a DLOGTIME Turing machine such that $\forall n : k(n) < n$, there is a DLTPS for the language $\{1^{k(n)}0^{n-k(n)} \mid n \in \mathbb{N}\}$, i.e. the language has exactly one word of length n that consists of $k(n)$ ones followed by zeros. This language clearly has a DLTPS, nevertheless we will give the exact DLTPS which we will then extend to a proof system for Exact_k:

$M(\sigma, i, x)$

$x = \underbrace{\cdots\cdots\cdots\cdots\cdots}_{n}$

Let $n = |x|$.
if $i > n$ **then**
  Reject.
else
  if $\sigma = 1$ and $i < k(n)$ **then** Accept.
  if $\sigma = 0$ and $i \geq k(n)$ **then** Accept.
  Reject.
end if

Note that in the algorithm above we ignore all bits of x and only use the length of x to determine the output. The machine maps every input word of length n to the word $1^{k(n)}0^{n-k(n)}$. And therefore is the proof system for this language.

Every word in the output is already in Exact_k. However, Exact_k contains every permutation of these words too. To give a proof system for Exact_k, we will modify the above ensuring surjectivity. We interpret the input x as a list of numbers $\alpha_0, \ldots, \alpha_{n-1}$ between 0 and $n-1$, where n is the length of the word we want to output. To access the i-th number in this list in this notation will require multiplication of s and i, where $s = \lceil \log n \rceil$. However, we do not know how to do this in DLOGTIME. Therefore, we store each number in DLOGTIME, as an s bit number where s is the smallest power of 2 greater of equal to $\log n$. Then we compute $s \cdot i$ by a simple bit shift in DLOGTIME. Also since we will allow arbitrary proofs, the number in the list might have a value larger than $n-1$, in this case we will interpret this number as the largest suitable number. Since these are only technical details we will simply write $\log n$ bit numbers in the rest of the paper.

Let $x = (\alpha_0 \ldots \alpha_{n-1} \ldots)$. We use the first n elements, i.e. $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, to come up with the output of length n and ignore the rest of the bits of x.

We interpret this list as a bijective map $m_{\vec{\alpha}}$. We say $m_{\vec{\alpha}}(i) = j$ if $\alpha_i = j$ and $\alpha_j = i$, if there is no such j we say $m_{\vec{\alpha}}(i) = i$. For every list α the map $m_{\vec{\alpha}}$ will be a bijection. Assume that $m(i) = j = m(i')$ then $i = \alpha_j = i'$ and hence m is injective. Since the map is between finite sets it is also surjective. Also note that every involution, i.e. a map m' such that $m'(m'(x)) = x$ for all x, can be represented in this way.

In order to have enough space in the proof for the whole list we will check that the proof has at least quadratic length l compared to the length n of string we want to output. Since we cannot exactly compute the square root we will approximate it. To approximate $\lfloor \sqrt{l} \rfloor$, let $j = \lfloor \log l \rfloor$ and let $k = \lfloor j/2 \rfloor$. As logarithm can be computed in DLOGTIME (see for example [1]) and division by 2 simply involves a shift by one bit to the right, j, k can be computed in DLOGTIME. Now, let n be the first j bits of the binary representation of l. Then n has the property $n^2 \leq l$, and for all $n \in \mathbb{N}$ there exists an $l \in \mathbb{N}$ such that n is the result of this operation. We define the function $\widehat{\text{sqrt}}(l) = n$, where n is obtained by the procedure above.

▶ **Theorem 4.** *For every function* $k : \mathbb{N} \to \mathbb{N}$ *computable in* DLT, Exact_k *has* DLTPS.

Proof. Consider the following proof system.

$M(\sigma, i, x)$

Let $n = \widehat{\text{sqrt}}(|x|)$.
if $i < n$ **then**
  Let $j = \alpha_i$
  if $\alpha_j = i$ **then**
    Let $t = j$.
  else
    Let $t = i$.
  end if
  if $\sigma = 1$ and $t < k(n)$ **then** Accept.
  if $\sigma = 0$ and $t \geq k(n)$ **then** Accept.
end if
Reject.

$$x = \overbrace{\boxed{\alpha_0}\ldots\boxed{\alpha_{n-1}}\ldots\ldots\ldots\ldots}^{\approx n^2}$$
(with α_0 over $\log n$ and α_{n-1} over $\log n$)

We now prove the correctness and surjectivity of M. In order to see that for a fixed x the output contains exactly $k(n)$ ones, note that we output a 1 if $t < k(n)$ and t is the image of i under a bijection. Given a word $w = w_0 \ldots w_{n-1}$, we need to show that there is an x that produces the output w. Let I be the set of 1s in w. We will assign to each index in I a unique value β_i in the following way: Let $I_0 = \{i \in I \mid i < k(n)\}$. We define $\beta_i = i$ for $i \in I_0$ and for the remaining values of I are assigned the remaining numbers less than $k(n)$ in an arbitrary and one-to-one manner. The x which produces this w can now be described by specifying $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$: $\alpha_i = \beta_i$ if $i \in I$, $\alpha_i = j$ if $\beta_j = i$, and $\alpha_i = i$ otherwise. It is easy to see that this input produces the output w. ◀

Since we can add additional 1s to the output as in [2], we get:

▶ **Corollary 5.** Majority *admits a* DLTPS.

Proof. As in [2] we can take a proof system of a language and add additional 1's to the output. Consider the proof system of Exact_k where $k = \lfloor n/2 \rfloor + 1$. We extend the proof system by n additional bits $\gamma_1, \ldots, \gamma_n$. Whenever would output a 0 at position i in the proof system of Exact_k we will check if $\gamma_i = 1$, in which case we output a 1 instead. ◀

4 Problems on Graphs

In this section, we consider three problems on graphs. For every $n \in \mathbb{N}$, a directed graph on n vertices is in Reach if there exists a path from a vertex labelled 0 to a vertex labelled $n-1$ in the graph. For a fixed k, and for every $n \in \mathbb{N}$, a graph on n vertices is in Clique_k if it has a clique of size at least k. For graph problems like Reach, Clique_k, the output of the proof system will be all graphs of these languages encoded as the adjacency matrix. The nodes of the graph are labeled by $1, \ldots, n$ and hence the adjacency matrix has size $n \times n$. We assume that the positions of the words are indexed by (i, j) and $w_{(i,j)} = 1$ iff there is an edge from i to j.

If we were to index the positions by a single number k, we could use any DLOGTIME computable encoding. Though the usual encoding $k = i \cdot n + j$ is not immediately in

DLOGTIME, a slight modification $k = i \cdot 2^{\lceil \log n \rceil} + j$ is computable in DLOGTIME. For such an encoding we would pad all positions not in the image of the tuple-function by a special character. In the following we use (i, j) to index the positions of the word.

▶ **Theorem 6.** Reach *has a* DLTPS

Proof. For an output of size $n \times n$ we require an input of length at most $n \log n + 2n^2 \leq 4n^2$. So we let $n = \widehat{\text{sqrt}}(|x|/4)$. We think of the beginning of x as a list of $\log n$-bit numbers $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$. Also we pick any tuple function computable in DLOGTIME and think of the end of x as an $n \times n$ matrix of single bits β_{ij}. All other bits of x are ignored.

We interpret $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ as a path of length at most n, and $(\beta_{ij})_{i,j \in \{0,1,\ldots,n-1\}}$ as a graph G. For an arbitrary input, we have no guarantee that the path α actually exists in the graph G. If the graph indeed has this path, we wish to preserve it in the output. On the other hand, if α is not a path in G, we wish to output a graph with a path. However, in the computation of a single output bit corresponding to the edge from i to j we cannot check whether α is a path in G, since the computation time is log-time bounded. So we spread this test among many output bits.

The smallest proof for existence of a path between 0 and $n-1$ is of constant size: for any i, adding edges $(0, i)$ and $(i, n-1)$ creates a positive instance. In DLT we can check for one i if α_i and α_{i+1} are the same nodes or they are connected in G. If they are not connected we will ensure that one of the short paths exists in the graph. If α indeed encodes a valid path, then we will simply copy G on the output tape. This way, we will generate all graphs that have paths from 0 to $n-1$.

It is easy to see that the following deterministic machine runs in $O(\log n)$ time. Observe that we output only positive instances of Reach. Suppose G is a graph with a path from 0 to $n-1$, then by repeating some of the vertices we get a sequence of nodes $(\alpha_0, \ldots, \alpha_{n-1})$, which encodes this path. For this input, we will output exactly G. That is, for every positive instance of Reach, there is an input to the algorithm which outputs that instance.

$M(\sigma, (i, j), x)$

$$x = \overbrace{\boxed{\alpha_0}}^{\log n} \ldots \overbrace{\boxed{\alpha_{n-1}}}^{\log n} \ldots \overbrace{\boxed{\beta_{00} \ldots \beta_{n-1,n-1}}}^{\leq 2n^2} \quad \underbrace{\qquad\qquad}_{\approx 4n^2}$$

Let $n = \widehat{\text{sqrt}}(|x|/4)$
if $i < n$ and $j < n$ **then**
  if $i = 0$ or ($i < n-1$ and $j = n-1$) **then**
    if $j = n-1$ **then** let $k = i$ **else** let $k = j$
    if $\beta_{\alpha_k, \alpha_{k+1}} \neq 1$ **then**
      {Check if α_k and α_{k+1} are connected}
      If $\sigma = 1$ Accept. {add the short path}
      If $\sigma = 0$ Reject.
    end if
  end if
  {No check needed or check succeeded, so output the graph G.}
  If $\sigma = \beta_{ij}$ Accept.
end if
Reject.

◀

▶ **Theorem 7.** *For any function* $k : \mathbb{N} \to \mathbb{N}$ *computable in* DLT, Clique_k *is in* DLTPS

Proof. We begin with an easy case, where a graph with n nodes has a $k(n)$ clique among the nodes $0, \ldots, k(n) - 1$. The input consists of an arbitrary graph $G = (\beta_{ij})$. We output any graph in which if two vertices have labels less than $k(n)$ then they share an edge. All these graphs belong to Clique_k.

As in the previous proof one can use a bijection to permute the nodes (all the nodes including 0 and $n - 1$ this time), which will give a correct and surjective proof system. The details of the proof can be worked out as in the previous proof. ◀

5 Closure Properties and Complexity

Closure under union seems to be naturally hold for all proof systems.

▶ **Lemma 8.** *If* L_1, L_2 *have* DLTPS, *then* $L_1 \cup L_2$ *has* DLTPS.

Proof. Let f_1, f_2 be the DLTPScorresponding to L_1, L_2. Then we define the function $f(x) = f_1(x_1 \ldots x_{n-1})$ if $x_0 = 0$, and $f(x) = f_2(x_1 \ldots x_{n-1})$ otherwise. Since f_1, f_2 are computable in DLT, so is f, and f is a proof system for $L_1 \cup L_2$.

The algorithm for union:

$M(\sigma, i, x)$

if $b = 0$ **then**
  Run $M_1(\sigma, i, x')$.
else
  Run $M_2(\sigma, i, x')$.
end if

$$x = \overbrace{\boxed{b}}^{1}\underbrace{\boxed{\quad x'\quad}}_{n}$$

◀

The closure under concatenation of proof systems is not so obvious.

▶ **Lemma 9.** *If* L_1, L_2 *have* DLTPS, *then* $L_1 L_2$ *has* DLTPS.

Proof. By definition neither L_1 nor L_2 can be empty. Let $w_1 \in L_1$ and $w_2 \in L_2$. Let f_1, f_2 be the proof systems corresponding to L_1 and L_2, and M_1, M_2 the corresponding DLOGTIME machines.

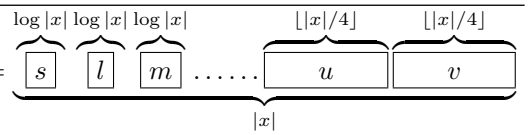
We will construct a proof system for $L_1 L_2$. The idea is that the proof consists of s, l, m, u, v, where s, l, m are of length $\log |x|$ and u, v are of length $|x|/4$. To concatenate $f_1(u_0 \ldots u_{l-1})$ and $f_2(v_0 \ldots v_{m-1})$, we will use s to "guess" the length of $f_1(u_0 \ldots u_{l-1})$.

Suppose we output the bit at position i. First we check if $|f_1(u_0 \ldots u_{l-1})| = s$, if this is not the case we output the i-th letter of the fixed default word $w_1 w_2$. Otherwise if $i < s$ we output the i-th letter of $f_1(u_0 \ldots u_{l-1})$ and if $i \geq s$ we output the s-i-th letter of $f_2(v_0 \ldots v_{m-1})$.

Clearly the proof system will be correct since we only output words in $L_1 L_2$. For surjectivity note that the input $u_0 \ldots u_{l-1}$ and $v_0 \ldots v_{m-1}$ are independent. Also since f_1, f_2 are polynomial bounded so is the constructed proof system. The algorithm for concatenation:

$M(\sigma, i, x)$

if $|f_1(u_0 \dots u_{l-1})| = s$ **then**
  if $i < s$ **then**
    Run $M_1(\sigma, i, u_0 \dots u_{l-1})$.
    {Accept or Reject depending on M_1}
  else
    Run $M_2(\sigma, i - s, v_0 \dots v_{m-1})$.
    {Accept or Reject depending on M_2}
  end if
end if
{Output default word otherwise}
if $(w_1 w_2)_i = \sigma$ **then**
  Accept.
else
  Reject.
end if



In the algorithm we can test $|f_1(u_0 \dots u_{l-1})| = s$ by testing if there is a σ such that $M_1(\sigma, s-1, u_0 \dots u_{l-1})$ accepts and for all σ we have that $M_1(\sigma, s, u_0 \dots u_{l-1})$ rejects. This requires to simulate M_1 on $2|\Sigma|$ different inputs which is possible in logarithmic time. ◀

In [2], it was proved that every language L that has $\mathrm{NC}^0\mathrm{PS}$ is recognised in $\mathrm{NTIME}(n)$. In the case of $\mathrm{NC}^0\mathrm{PS}$ it is enough to guess a proof and then evaluate the circuit on this guessed proof. This suffices because the NC^0 circuit queries the proof bits in a non-adaptive manner. However, in the case of DLTPS the deterministic log-time machine may read bits on the proof in an adaptive manner, i.e. it may read a location, say i, of the proof and depending on the value of that proof bit, may decide to read the next bit. Therefore, the simulation of such a DLTPS needs to remain consistent with respect to such adaptive queries.

▶ **Theorem 10.** *If a language L has* DLTPS *then it can be recognised in NTIME*$(n^2 \log^3 n)$.

Proof. As L has a DLTPS there exists a $f : \Gamma^* \to \Sigma^*$ computable in DLT by a Turing machine M. Also since a DLTPS is polynomial bounded there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$, such that $w \in L$ iff $w \in f(\Sigma^{<p(|w|)})$.

Assume we guess the word x such that $w = f(x)$, then we could check that $M(\sigma, i, x)$ accepts iff $w_i = \sigma$. Since we want to show an upper bound of $n^2 \log^2 n$ we cannot simply guess x which might have size larger than $n^2 \log^2 n$.

But M is a DLT machine, so it cannot access all of the bits of x, but only $O(\log n)$ of these bits. Since we simulate M on $O(n)$ different inputs we require only $O(n \log n)$ of these bits. We guess and store only the bits of x accessed by M together with their indices on the tape which requires $O(n \log^2 n)$ length.

Then we can check that $f(x) = w$ by simulating M for all $i = 0, \dots, |w|$. For the simulation of a single step we might need to search on the tape of the bit accessed which requires $O(n \log^2 n)$ time. Since we simulate the machine $O(n)$ times with a runtime of $O(\log n)$ steps each, so we require time $O((n \log^2 n) \cdot n \cdot \log n)$. ◀

Using the non-deterministic time hierarchy theorem and Theorem 10 we get:

▶ **Corollary 11.** *There exists a language L in* NP *for which there is no* DLTPS.

We can use the previous theorem to show that DLTPS are not closed under intersection.

▶ **Theorem 12.** *The languages which have* DLTPS *are not closed under intersection.*

Proof. We show that if DLTPS are closed under intersection then all languages in NP have DLTPS. For this let $L \subset \Sigma^*$ be any problem in NP, i.e. in NTIME(n^c). We will construct two language L_a, L_b over the alphabet $\Sigma' = \Sigma \cup \{x, a, b\}$, such that both L_a, L_b have DLTPS and their intersection is L.

Let $L^{=n} = L \cap \Sigma^n$, and $B_a = \{x, a\}^* a \{x, a\}^*$, and $B_a^n = B_a \cap \Sigma^n$. Similar $B_b = \{x, b\}^* b \{x, b\}^*$, and $B_b^n = B_b \cap \Sigma^n$. Consider the languages $L_a = \bigcup_n L^{=n} x^{n^{2c}} \cup \Sigma^n B_a^{n^{2c}}$, $L_b = \bigcup_n L^{=n} x^{n^{2c}} \cup \Sigma^n B_b^{n^{2c}}$.

Each of the languages consists of each words of L padded by a certain number of xs, union some "bad" part which has at least one a for the first language or at least one b for the second language. If we take the intersection of $L_a \cap L_b$ we get the language L padded by some xs.

We will show that we have a proof system for L_a. So the proof will consist of the computation of the TM for L. This computation has at most length n^{2c}. While the DLOGTIME for L_a outputs the input to the proof in the first n positions, we output at the remaining positions a x if the computation at this position is consistent, otherwise an a. Additionally we can output all words in $\Sigma^n B_a^{n^{2c}}$ and hence have a proof system for L_a.

Similarly we can construct a proof system for L_b. Assuming that DLTPS are closed under intersection we have a proof system for $L_a \cap L_b$. Assuming $x \notin \Sigma$, we can modify the DLOGTIME machine such that it always rejects when $\sigma = x$. This is only possible since no letter other than x can appear at a position behind x. (Otherwise the output would not be a word, but contains holes inside the word). The modified proof system is a proof system for L. Since L was any language in NP this is a contradiction. ◀

▶ **Corollary 13.** DLTPS *are not closed under complement.*

6 Hierarchy Theorems

▶ **Definition 14.** A function $f : \Gamma^* \to \Sigma^*$ is computable in DLT^k if there exists a deterministic Turing machine M such that $M(\sigma, i, x)$ halts and accepts in time $O((\log |x|)^k)$ if $w_i = \sigma$, where $w_0 \dots w_{n-1} = f(x)$. We let $\mathrm{DL^kTPS}$ be the polynomial bounded proof systems that are computable in DLT^k.

▶ **Theorem 15** (Time Hierarchy for Proof Systems)**.** $\mathrm{DL^tTPS} \subsetneq \mathrm{DL}^{2t+1}\mathrm{TPS}$

Proof. The basic idea is to apply diagonalization as in the nondeterministic time hierarchy theorem. The machines that we consider are deterministic but we need to "guess" the proof.

We will only show the proof for $\mathrm{DLTPS} \subsetneq \mathrm{DL}^3\mathrm{TPS}$, the proof for any k is similar. The idea is to define a language $L \subseteq 1^*$ and show that L has a $\mathrm{DL}^3\mathrm{TPS}$ but no DLTPS. Since we will be working over the unary alphabet we will ignore the parameter σ in our proof systems. Also it suffices in general to consider proofs in $\{0, 1\}^*$. Let $M_0, M_1, M_2, \dots$ be an enumeration of all DLOGTIME machines (which also includes DLOGTIME machines which are not DLTPS). The idea is to divide the natural numbers in intervals such that the left border of the interval is much smaller than the right border. For this we define a function $f : \mathbb{N} \to \mathbb{N}$ by:

$$f(0) = 2, f(l+1) = 2^{2^{2^{f(l)}}}$$

We will define the language L: Let k be any number and $t = \log^{1.1} k$.

1. L contains the empty word.
2. If $f(l) < k < f(l+1)$ for some l, then $1^k \in L$ iff there exists a word $w \in \{0,1\}^*$ of length less than or equal to 2^t such that $M_l(k, w)$ halts and accepts within t steps and $M_l(k+1, w)$ halts and rejects within t steps.
3. If $k = f(l+1)$ for some l, then $1^k \in L$ iff there exists a word $w \in \{0,1\}^*$ of length less than or equal to $2^{\log^{1.1} f(l)}$ such that $M_l(f(l), w)$ halts and rejects within t steps or $M_l(f(l)+1, w)$ halts and accepts within t steps.

For a DLOGTIME machine M_l, the definition basically says that:

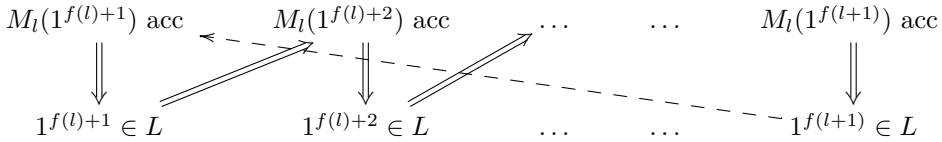
1. If $f(l) < k < f(l+1)$ for some l, then $1^k \in L$ iff the proof system M_l outputs 1^{k+1}.
2. If $k = f(l+1)$ for some l, then $1^k \in L$ iff the proof system M_l does not output $1^{f(l)+1}$.

But keep in mind that this is not our definition, just the the intention for the definition.

We will show that if M_l is a DLTPS then the language corresponding to M_l is different from the language L. Since M_l is a DLOGTIME proof system we can assume that there is some c such that M_l halts after at most $c \log n$ steps. And since M_l is polynomial bounded, if there is a w such that M_l on input w will output 1^n there is a word w of length at most $n^{c'}$. We assume that for all $n \geq f(l)$: $c \log n \leq \log^{1.1} n$ and $n^{c'} \leq 2^{\log^{1.1} n}$ (since there are infinite many l that represent the same DLOGTIME machine we can ensure this).

Assume by contradiction that the language of M_l equals L. By equality, we have that M_l outputs $1^{f(l)+1}$ iff L contains $1^{f(l)+1}$. By definition of L, for any k in $f(l) < k < f(l+1)$, 1^k is in L iff there exists a word $w \in \{0,1\}^*$ of length less or equal than $2^{\log^{1.1} k}$ such that $M_l(k, w)$ halts and accepts within t steps and $M_l(k+1, w)$ halts and rejects within t steps. Assuming that M_l is a proof system with the bounds on computation time and proof length as above, this happens iff M_l outputs 1^{k+1} (for some input). Hence by induction L contain $1^{f(l)+1}$ iff M_l outputs $1^{f(l+1)}$.

But by definition, L contains $1^{f(l+1)}$ iff M_l does not output $1^{f(l)+1}$ on any input (again by the assumption on the bounds of computation time and proof length). This is a contradiction.



So L clearly has no DLTPS, it remains to show that there is a proof system D in DL^{3}TPS for L. We will construct a proof system D in DL^{3}TPS for L. The proof system will use a proof of length k to either output 1^k or an empty string.

For an input x we let $k_0 = |x|$ be the length of the input. We compute $k_{i+1} = \lfloor \log\log\log k_i \rfloor$, and repeat this process till we reach l such that $k_l = 0$. Then $f(l) < k \leq f(l+1)$, where equality occurs exactly if we never needed to round down somewhere in the process. We need at most $\log^* |x|$ repetitions, and except for the first repetition (which requires $O(\log n)$ steps) we only require $O(\log\log\log |x|)$ steps for the computation of one repetition, so clearly we can compute this in $O(\log n)$ steps.

Given an input x of length k such that there is an l with $f(l) < k < f(l+1)$. We want to output the word 1^k iff there exists a word w such that $M_l(k, w)$ halts and accepts within t steps and $M_l(k+1, w)$ halts and rejects within $t = \log^{1.1} k$ steps. First note that only two instance M_l are run each for t steps on w. Hence M_l will access only $2t$ positions of w. This implies that we can modify all bits other than these $2t$ positions without changing the acceptance behaviour of $M_l(k, w)$ and $M_l(k+1, w)$. In particular, we can set all positions other than these $2t$ positions to 0. Hence there is a word w such that $M_l(k, w)$ halts and

accepts within t steps and $M_l(k+1,w)$ halts and rejects within t steps iff there is a word w with at most $2t$ bits set to 1 such that $M_l(k,w)$ halts and accepts within t steps and $M_l(k+1,w)$ halts and rejects within t steps.

This will greatly limit the amount of words w that we need to simulate. We interpret the proof x as a list of $\log^{1.1}|x|$ bit numbers: $s, \alpha_0, \ldots, \alpha_{2t}$ (we ignore all other bits). We will let $w = \phi(x)$ where $\phi(x)$ is a word of length s that has a bit set to 1 at the positions $\alpha_0, \ldots, \alpha_{2t}$. Then there is a word x of length k such that $M_l(k,\phi(x))$ halts and accepts within t steps and $M_l(k+1,\phi(x))$ halts and rejects within t steps iff there exists a word $w \in \{0,1\}^*$ of length less or equal than 2^t such that $M_l(k,w)$ halts and accepts within t steps and $M_l(k+1,w)$ halts and rejects within t steps. Hence for our proof system it suffices to simulate $M_l(k,\phi(x))$ and $M_l(k+1,\phi(x))$ for t steps, and output 1^k accordingly to the definition of L.

Now assume that $k = f(l+1)$, then the proof system on an input x of length k should output 1^k iff there exists a word $w \in \{0,1\}^*$ of length less or equal than $2^{\log^{1.1}(f(l)+1)}$ such that $M_l(f(l),w)$ halts and rejects within t steps or $M_l(f(l)+1,w)$ halts and accepts within t steps. So here we need to check if M_l outputs a word of length $f(l)+1$, where $f(l)$ is much smaller than $f(l+1)$. That is, the simulating machine runs for $\log^{1.1}(f(l)+1)$ steps and for each word of length at most $2^{\log^{1.1}(f(l)+1)}$. Therefore, the total running time of the simulating machine is upper bounded by $2^{2^{f(l)}}$. As the input length (as well as the output length) in this case is $2^{2^{2^{f(l)}}}$, our machine has enough time to actually simulate all possible words and check the output of M_l (in this case M_l ignores the bits of x, we only require a long x to have sufficient time for the simulation).

This completes the proof showing that L has a $\mathrm{DL}^3\mathrm{TPS}$ (in fact $\mathrm{DL}^{2+\varepsilon}\mathrm{TPS}$). ◀

Acknowledgements. We thank Klaus-Jörn Lange for helpful comments on this draft.

References

1 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On Uniformity within NC^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990.

2 Olaf Beyersdorff, Samir Datta, Andreas Krebs, Meena Mahajan, Gido Scharfenberger-Fabian, Karteek Sreenivasaiah, Michael Thomas, and Heribert Vollmer. Verifying proofs in constant depth. In F. Murlak and P. Sankowski, editors, *Full version on ECCC TR12-079, a preliminary version in MFCS*, volume 6907 of *LNCS*, pages 84–95. Springer, 2011.

3 Olaf Beyersdorff, Johannes Köbler, and Sebastian Müller. Proof systems that take advice. *Inf. Comput.*, 209(3):320–332, 2011.

4 Olaf Beyersdorff and Sebastian Müller. A tight karp-lipton collapse result in bounded arithmetic. In M. Kaminski and S. Martini, editors, *Computer Science Logic*, volume 5213 of *LNCS*, pages 199–214. Springer Berlin Heidelberg, 2008.

5 Stephen A. Cook and Jan Krajícek. Consequences of the provability of NP subset of or equal to P/poly. *J. Symb. Log.*, 72(4):1353–1371, 2007.

6 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):pp. 36–50, 1979.

7 Birgit Jenner, Pierre McKenzie, and Jacobo Torán. A note on the hardness of tree isomorphism. In *IEEE Conference on Computational Complexity*, pages 101–105, 1998.

8 Andreas Krebs, Nutan Limaye, Meena Mahajan, and Karteek Sreenivasaiah and. Small depth proof systems. In *a preliminary version to appear in MFCS*, 2013.

9 Pavel Pudlák. Quantum deduction rules. *Annals of Pure and Applied Logic*, 157(1):16–29, 2009.

10 Stanislav Žák. A turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, 1983.

On Improved Degree Lower Bounds for Polynomial Approximation

Srikanth Srinivasan

Department of Mathematics, IIT Bombay, Mumbai, India
srikanth@math.iitb.ac.in

Abstract

A polynomial $P \in \mathbb{F}[X_1, \ldots, X_n]$ is said to *ε-approximate* a boolean function $F : \{0,1\}^n \to \{0,1\}$ under distribution $\mathcal{D}$ over $\{0,1\}^n$ if $\Pr_{x \sim \mathcal{D}}[P(x) \neq F(x)] \leq \varepsilon$. Smolensky (1987) showed that for any constant primes $p \neq q$, any polynomial $P \in \mathbb{F}_p[x_1, \ldots, x_n]$ that $(\frac{1}{2q} - \Omega(1))$-approximates the boolean function $\mathrm{MOD}_q : \{0,1\}^n \to \{0,1\}$ – which accepts its input iff the number of ones is non-zero modulo q – under the uniform distribution must have degree $\Omega(\sqrt{n})$.

We consider the problem of finding an explicit function $f : \{0,1\}^n \to \{0,1\}$ that has no ε-approximating polynomial of degree less than $n^{1/2+\Omega(1)}$ under *some distribution*, for some constant $\varepsilon > 0$. We show a number of negative results in this direction: specifically, we show that many interesting classes of functions including symmetric functions and linear threshold functions do have approximating polynomials of degree $O(\sqrt{n} \cdot \mathrm{polylog}(n))$ under *every distribution*. This demonstrates the power of this model of computation.

The above results, in turn, provide further motivation for this lower bound question. Using the upper bounds obtained above, we show that finding such a function f would have applications to: lower bounds for $\mathrm{AC}^0 \circ \mathcal{F}$ where $\mathcal{F} = \mathrm{SYM} \cup \mathrm{THR}$; stronger lower bounds for 1-round compression by $\mathrm{ACC}^0[p]$ circuits; improved correlation lower bounds against low degree polynomials; and (under further conditions) showing that the Inner Product (over $\mathbb{F}_2$) function does not have small $\mathrm{AC}^0 \circ \mathrm{MOD}_2$ circuits.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes

Keywords and phrases Polynomials, Approximation, Compression, Circuit lower bounds

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.201

1 Introduction

Quite often, when trying to understand a model of computation (such as, say, a class of Boolean circuits), it is profitable to see if functions computed by that model can be approximated by functions that can be computed using a different, simpler, and in general, "nicer" model. Low-degree polynomials are an important example of such a nice model. Given the versatility of polynomials and the wealth of algebraic and combinatorial understanding we have of them, it is not surprising that approximations by low-degree polynomials have been used to prove a variety of results in Complexity theory [22, 24, 1, 4, 5], Learning Theory [19, 18], Derandomization [3, 8], etc..

The different applications listed above use various notions of "approximation", of what it means to be "low-degree", and which ring to choose our low-degree polynomials from. Here, we study one such means of approximation that has been useful in proving circuit lower bounds [22, 24]. We say that a polynomial $P \in \mathbb{F}[X_1, \ldots, X_n]$ (for some field $\mathbb{F}$) *ε-approximates* a Boolean function $F : \{0,1\}^n \to \{0,1\}$ over a probability distribution $\mathcal{D}$ on $\{0,1\}^n$ if $\Pr_{x \sim \mathcal{D}}[P(x) \neq F(x)] \leq \varepsilon$. We typically think of $\varepsilon > 0$ as a small constant, though

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 201–212

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

the regime of ε close to $1/2$ is also interesting and is dealt with later on.

A seminal result of Razborov [22] showed, in the case $\mathbb{F} = \mathbb{F}_2$, that a certain symmetric function $F : \{0,1\}^n \to \{0,1\}$ has no ε-approximating polynomial of degree $o(\sqrt{n})$ over the uniform distribution $\mathcal{U}$ for some constant $\varepsilon > 0$; this fact was used to prove a lower bound for $\mathrm{ACC}^0[2]$ circuits computing the Majority function. Smolensky [24] further showed that for constant primes $p \neq q$ and $\mathbb{F} = \mathbb{F}_p$, the function $\mathrm{MOD}_q : \{0,1\}^n \to \{0,1\}$ – which accepts its input x iff $\sum_i x_i \not\equiv 0 \pmod q$ – does not have a $((1/2q) - \varepsilon)$-approximating polynomial over $\mathcal{U}$ of degree $o(\sqrt{n})$. In another work [25], Smolensky also showed a similar degree lower bounds for polynomials that $((1/2) - \varepsilon)$-approximate the Majority function over $\mathcal{U}$ for *any* field $\mathbb{F}$. All these lower bounds are known to be tight under $\mathcal{U}$ [6, 27]. Moreover, as far as we are aware, these remain the best degree lower bounds that we know of today for polynomials approximating *explicit functions*[1], even if we allow the probability distribution $\mathcal{D}$ to be arbitrary (even non-explicit).

Therefore, we ask if Smolensky's lower bound can be strengthened considerably, even just in this weaker regime. More formally,

▶ Problem 1. Come up with an explicit function $F : \{0,1\}^n \to \{0,1\}$ that has no $(1/10)$-approximating polynomials of degree less than $n^{\frac{1}{2}+\Omega(1)}$ under *some* distribution $\mathcal{D}$.

The reasons for considering a lower bound under an arbitrary (as opposed to the uniform) distribution are twofold: the first is that the circuit lower bound results mentioned above [22, 24] and other applications we will see later on only require this weaker lower bound result; and secondly, the above lower bound notion, by LP duality, has a nice dual *upper bound* notion as well. More precisely, if a Boolean function F has ε-approximating polynomials of degree d under *every* distribution $\mathcal{D}$, then there is a probability distribution over *polynomials* of degree d that computes F correctly on *every input* with probability $(1 - \varepsilon)$. Such a distribution is called a *probabilistic polynomial* [26]. Probabilistic polynomials for a function F are much easier to reason with than polynomials that approximate function F w.r.t. a fixed distribution; this is the same kind of distinction that exists between, say, randomized and distributional algorithms. Combination, composition, error-reduction, etc. are much easier with probabilistic polynomials than with their distributional counterparts.

Results. We begin our search for a suitable lower bound candidate $F : \{0,1\}^n \to \{0,1\}$ with the class of *symmetric* functions. Recall that $F : \{0,1\}^n \to \{0,1\}$ is symmetric if $F(x)$ depends only on the Hamming weight of x, denoted $|x|$. The degree lower bounds mentioned above [22, 24, 25] were all proved for symmetric functions over the uniform distribution. A natural question to ask is if we can improve these lower bounds for functions in this class.

It has been observed [6, 27] in the literature that as long as we only consider the uniform distribution, the lower bounds of Razborov and Smolensky are indeed tight. In other words, for any constant $\varepsilon \in (0, 1/2)$, any symmetric function $F : \{0,1\}^n \to \{0,1\}$ has an ε-approximating polynomial of degree $O(\sqrt{n})$ under the uniform distribution. The way to prove this is to note that the uniform distribution puts all but an ε-fraction of its mass on the central $O(\sqrt{n})$ layers of the hypercube $\{0,1\}^n$. Thus, by interpolating a polynomial of degree $O(\sqrt{n})$ that computes F exactly on these points of the hypercube (possible since F is symmetric), we obtain an $O(\sqrt{n})$ degree polynomial that ε-approximates F.

However, the strategy used above is crucially dependent on the underlying distribution being uniform (or at least having a nice product structure) and thus it is conceivable that

[1] Throughout, we are not very formal about what we mean by an *explicit* function, since any reasonable notion will do. The reader may take it to mean a function from a function family computable in polynomial time.

under other distributions, symmetric functions might not have such low-degree polynomial approximations. Somewhat surprisingly, this turns out to be false. We show that symmetric functions in fact have small error *probabilistic polynomials* of degree $O(\sqrt{n} \cdot \mathrm{polylog}(n))$ and hence they have small error approximations of the same degree over any distribution $\mathcal{D}$.

We describe a simple case of the proof here for intuition. Assume that F is the MOD_2 function and that the underlying field is $\mathbb{R}$. As we outlined above, F has an ε-approximation $P \in \mathbb{R}[X_1, \ldots, X_n]$ of degree $O(\sqrt{n})$ under the uniform distribution. We obtain an ε-error probabilistic polynomial for F as follows. Suppose $x \in \{0,1\}^n$ is an arbitrary input. Note that for any $y \in \{0,1\}^n$, we can write $\mathrm{MOD}_2(x) = \mathrm{MOD}_2(x \oplus y) \bigoplus \mathrm{MOD}_2(y)$, where $x \oplus y \in \{0,1\}^n$ is obtained by taking the bitwise sum of x and y modulo 2. Now say $\mathbf{y} \in \{0,1\}^n$ is chosen uniformly at random; we have $\mathrm{MOD}_2(x) = \mathrm{MOD}_2(x \oplus \mathbf{y}) \bigoplus \mathrm{MOD}_2(\mathbf{y})$. Since $\mathbf{y}$ is uniformly random, we see that $x \oplus \mathbf{y}$ is also uniformly distributed over $\{0,1\}^n$. Hence, we have $P(x \bigoplus \mathbf{y}) = \mathrm{MOD}_2(x \oplus \mathbf{y})$ with probability at least $1 - \varepsilon$. Thus, the probabilistic polynomial $\mathbf{Q}(x) := P(x \oplus \mathbf{y}) \bigoplus \mathrm{MOD}_2(\mathbf{y})$ computes $\mathrm{MOD}_2(x)$ correctly with probability at least $1 - \varepsilon$. It is easily argued that the degree of $\mathbf{Q}$ for every fixed value of $\mathbf{y}$ is at most the degree of P and this concludes the proof.

It turns out that the above idea, suitably modified, also gives low-degree polynomials for MOD_q as long as q is not too large. After having done this, we can handle the case of general symmetric functions by using the Chinese Remainder Theorem (Theorem 15), which tells us that the Hamming weight $|x| \in \{0, \ldots, n\}$ is completely determined by the values $|x| \pmod q$, where q ranges over all "small" primes. This allows us to obtain $O(\sqrt{n} \cdot \mathrm{poly}(\log n))$-degree probabilistic polynomials for all symmetric functions using the probabilistic polynomials we construct for MOD_q. In fact, we obtain a more general result: we get probabilistic polynomials of low degree for all functions that are determined by "small-weight" sums of the bits of the input x. This result is proved in Section 3.2.

In Section 3.3, we extend these results to certain functions that depend on "large weight" sums of the bits of x. The class of functions that we consider are the Linear Threshold functions that have received quite some attention in the literature [14, 15, 17]. Using the above result in conjunction with ideas due to Hofmeister [17], we show that any Linear Threshold function of n Boolean variables has a probabilistic polynomial of degree $O(\sqrt{n} \cdot \mathrm{poly}(\log n))$.

The above results show that probabilistic polynomials of degree $n^{1/2+o(1)}$ are a surprisingly powerful and robust model of computation. This suggests that solving Problem 1 might be a significant challenge. On the other hand, however, these results also imply that solving Problem 1 would have significant rewards. In Section 4, we show that an explicit function F as above would have applications to many problems including lower bounds for compression by constant-depth circuits [11], lower bounds for some strong constant-depth circuit classes, improved correlation lower bounds against low-degree polynomials over small fields [27], and to showing that the Inner Product function does not have small $\mathrm{AC}^0 \circ \mathrm{MOD}_2$ circuits [23] (if the function F has some additional "nice" properties).

Due to lack of space, many proofs are postponed to the full version of the paper.

2 Definitions and preliminaries

Let $\mathbb{F}$ be an arbitrary field and $n \in \mathbb{N}$ be a growing parameter. We denote by $\mathcal{B}_n$ the set of all functions $f : \{0,1\}^n \to \{0,1\}$.

▶ **Definition 2.** A function $f \in \mathcal{B}_n$ is said to be *symmetric* if there exists a function $h : \{0, \ldots, n\} \to \{0,1\}$ such that for any $x \in \{0,1\}^n$, we have $f(x_1, \ldots, x_n) = h\left(\sum_{i=1}^n x_i\right)$.

That is, the value of $f(x)$ is determined by the Hamming weight of x, denoted $|x|$.

More generally, given $W \in \mathbb{N}$, we say that f is *W-sum determined* if there exist $w_1, w_2, \ldots, w_n \in \mathbb{N}$ such that $\sum_i w_i \leq W$ and a function $h : \{0, \ldots, W\} \to \{0,1\}$ such that for any $x \in \{0,1\}^n$, we have $f(x_1, \ldots, x_n) = h\left(\sum_{i=1}^n w_i x_i\right)$. Note that symmetric functions in $\mathcal{B}_n$ are n-sum determined. It can also be seen that *every* $f \in \mathcal{B}_n$ is $(2^n - 1)$-sum determined.

Some examples of symmetric functions in $\mathcal{B}_n$ are:

- the Majority function MAJ^n, which accepts inputs x such that $|x| > n/2$,
- the $\mathrm{MOD}^n_{m,r}$ function for $m \geq 2$ and $r \in \mathbb{N}$, which accepts x such that $|x| \equiv r \pmod m$.

We omit the superscript n above when it is clear from context.

▶ **Definition 3.** A function $f \in \mathcal{B}_n$ is said to be a *Linear Threshold function* if there exist $a_1, \ldots, a_n, \theta \in \mathbb{R}$ such that for any $x \in \{0,1\}^n$ $f(x) = 1$ iff $\sum_{i=1}^n a_i x_i \geq \theta$.

Notation. We denote by $\mathrm{SUM}^n_W, \mathrm{THR}^n$ the set of all W-sum determined functions and linear threshold functions in $\mathcal{B}_n$ respectively. When n is clear from context, we use $\mathrm{SUM}_W, \mathrm{THR}$ instead of $\mathrm{SUM}^n_W, \mathrm{THR}^n$.

▶ **Definition 4.** Let $n, d \in \mathbb{N}$ and $\varepsilon \in (0, \frac{1}{2})$. Fix a probability distribution $\mathcal{D}$ over $\{0,1\}^n$. We say that a function $f \in \mathcal{B}_n$ is *$(d, \varepsilon, \mathcal{D})_{\mathbb{F}}$-approximable* if there is a polynomial $P \in \mathbb{F}[X_1, \ldots, X_n]$ of degree at most d such that $\Pr_{x \sim \mathcal{D}}[P(x) \neq f(x)] \leq \varepsilon$.

Smolensky [24] proved the following degree lower bounds on ε-approximating polynomials for the $\mathrm{MOD}^n_{q,0}$ function, where q is a constant prime.

▶ **Theorem 5** ([24]). *For any $\varepsilon < 1/10q$, any polynomial that ε-approximates the $\mathrm{MOD}^n_{q,0}$ function w.r.t. the uniform distribution over $\{0,1\}^n$ has degree $\Omega(\sqrt{n \log(1/\varepsilon)})$.*

▶ **Definition 6** (Probabilistic polynomial [26])**.** A *probabilistic polynomial* of degree d in $\mathbb{F}[X_1, \ldots, X_n]$ is a probability distribution $\mathbf{P}$ over polynomials of degree at most d in $\mathbb{F}[X_1, \ldots, X_n]$. Given $f \in \mathcal{B}_n$ and an $\varepsilon \in (0, \frac{1}{2})$, we say that $\mathbf{P}$ is an *ε-error probabilistic polynomial* for f if $\forall x \in \{0,1\}^n$, we have $\Pr_{\mathbf{P}}[\mathbf{P}(x) \neq f(x)] \leq \varepsilon$. The *$\varepsilon$-error probabilistic degree of f* over $\mathbb{F}$ is the least d s.t. f has an ε-error probabilistic polynomial of degree d.

The following simple facts will be useful. The first is an easy consequence of LP duality and the second follows, for example, from polynomial interpolation or Möbius Inversion.

▶ Fact 7. Fix any $\varepsilon \in (0, \frac{1}{2})$, $n \in \mathbb{N}$, and any field $\mathbb{F}$. Then, for any $f \in \mathcal{B}_n$, f has ε-error probabilistic degree d iff f is $(d, \varepsilon, \mathcal{D})_{\mathbb{F}}$-approximable for *every* probability distribution $\mathcal{D}$.

▶ Fact 8. Every $f \in \mathcal{B}_n$ can be *exactly* represented by a polynomial in $\mathbb{F}[X_1, \ldots, X_n]$ of degree at most n. That is, there is a polynomial $P \in \mathbb{F}[X_1, \ldots, X_n]$ of degree at most n such that for any $x \in \{0,1\}^n$, we have $P(x) = f(x)$.

▶ Fact 9. Assume $f \in \mathcal{B}_s$ and $g_i \in \mathcal{B}_t$ $(i \in [s])$. Define $F \in \mathcal{B}_t$ as follows: $F(x) := f(g_1(x), \ldots, g_s(x))$. Say $\mathbf{P}$ is an ε-error probabilistic polynomial of degree d for f and $\mathbf{Q}_i$ $(i \in [s])$ is a δ_i-error probabilistic polynomial of degree d_i for g_i. Then, $\mathbf{P}(\mathbf{Q}_1, \ldots, \mathbf{Q}_s)$ is an $(\varepsilon + \sum_{i=1}^s \delta_i)$-error probabilistic polynomial for F of degree $d \cdot (\max_{i \in [s]} d_i)$.

We also recall here the definitions of some well-known constant-depth circuit classes. Throughout, the size of a circuit is the number of *wires* in the circuit. The class of polynomial sized constant-depth circuits made up of AND, OR, and NOT gates is called AC^0; for p a *constant* prime, $\mathrm{ACC}^0[p]$ is the class of polynomial sized constant depth circuits containing AND, OR, NOT and MOD_p gates (a MOD_p gate computes a $\mathrm{MOD}_{p,r}$ function applied to

its input bits, for some r). We will abuse notation and use "AC^0 circuits of size s" to refer to constant-depth circuits of size s with AND, OR, NOT gates (similarly, we will also say "$\mathrm{ACC}^0[p]$ circuits of size s").

The following is implied by works of Razborov [22], Smolensky [24], and Tarui [26].

▶ **Theorem 10** ([22, 24, 26]). *Let $s \in \mathbb{N}$ and $\varepsilon \in (0, 1/2)$ be arbitrary parameters. Any AC^0 circuit of size s and depth d has an ε-error probabilistic polynomial of degree $(\log s \cdot \log(1/\varepsilon))^{O(d)}$ over any field $\mathbb{F}$. Any $\mathrm{ACC}^0[p]$ circuit of size s and depth d has an ε-error probabilistic polynomial of degree $(\log s \cdot \log(1/\varepsilon))^{O(d)}$ over the field $\mathbb{F}_p$.*

We also consider constant-depth circuits with other gate types: a SYM gate can compute an arbitrary symmetric function of its inputs, a THR gate can compute an arbitrary linear threshold function, and a SYMTHR gate can do either. For a gate type $\mathcal{G}$ (such as SYM, THR, MOD_p, etc.), we use $\mathrm{AC}^0 \circ \mathcal{G}$ to denote the class of polynomial constant-depth circuits where the inputs feed into gates of type $\mathcal{G}$, which in turn feed into an AC^0 circuit.

3 Probabilistic polynomials for SUM_W and THR

Let $\mathbb{F}$ be an arbitrary field and $n \in \mathbb{N}$ a growing parameter. In this section, we prove the following theorems.

▶ **Theorem 11.** *Fix $\varepsilon \in (0, \frac{1}{2})$ and $W \in \mathbb{N}$. Any $f \in \mathrm{SUM}_W^n$ has ε-error probabilistic degree at most $O(\sqrt{n \log(1/\varepsilon)} \cdot (\log W)^{O(1)})$.*

▶ **Theorem 12.** *Fix $\varepsilon \in (0, \frac{1}{2})$. Any $f \in \mathrm{THR}^n$ has ε-error probabilistic degree at most $O(\sqrt{n} \cdot (\log n \log\left(\frac{1}{\varepsilon}\right))^{O(1)})$.*

We will prove these two theorems in three steps:

- We will first show that certain functions in $\mathcal{B}_n$ (variants of the modular functions $\mathrm{MOD}_{m,r}$) have ε-error probabilistic degree $O(\sqrt{n \log(1/\varepsilon)} \cdot (\log n)^{O(1)})$. This will establish a weak form of Theorem 11.
- Using some Chinese Remaindering ideas, we will then derive Theorem 11.
- Using some more Chinese Remaindering ideas due to Hofmeister [17] and Theorem 11, we will prove Theorem 12.

We first introduce a generalization of the symmetric $\mathrm{MOD}_{m,r}^n$ function. Given $w \in \mathbb{Z}_m^n$, we denote by $\mathrm{MOD}_{m,r,w}^n$ (or just $\mathrm{MOD}_{m,r,w}$ if n is clear from context) the Boolean function in $\mathcal{B}_n$ that accepts its input $x \in \{0,1\}^n$ iff $\sum_{i=1}^n w_i x_i \equiv r \pmod m$. We identify $\mathbb{Z}_m$ with $\{0, 1, \ldots, m-1\}$.

3.1 Probabilistic polynomials for $\mathrm{MOD}_{m,r,w}^n$

The main result of this section is the following.

▶ **Lemma 13.** *Fix integers $m \geq 2$ and $r \in \{0, 1, \ldots, m-1\}$. For any $\varepsilon \in (0, \frac{1}{2})$ and $w \in \mathbb{Z}_m^n$, the function $\mathrm{MOD}_{m,r,w}^n$ has an ε-error probabilistic degree $O(\sqrt{n \log(m/\varepsilon)} \cdot m)$.*

In particular for $m = (\log n)^{O(1)}$, the probabilistic degree is $O(\sqrt{n \log(1/\varepsilon)} \cdot (\log n)^{O(1)})$.

We first need a lemma regarding the approximability of symmetric functions in $\mathcal{B}_n$ with respect to some simple product distributions over $\{0,1\}^n$. For $\rho \in [0,1]$, let $\mathcal{D}_\rho$ denote the distribution over $\{0,1\}^n$ where each bit is set to 1 independently with probability ρ.

The following lemma is a slight extension of one from a survey of Viola [27] – where the proof is attributed to Avi Wigderson – based on a lemma due to Bhatnagar, Gopalan,

and Lipton [6]. Though the proof in [27] only works in finite characteristic, the lemma can actually be proved for any field $\mathbb{F}$. The proof is postponed to the full version of the paper.

▶ **Lemma 14.** *Let $f \in \mathcal{B}_n$ be an arbitrary symmetric function, $\rho \in [0,1]$, and $\varepsilon \in (0, \frac{1}{2})$. Then, f is $(O(\sqrt{n \log(1/\varepsilon)}), \varepsilon, \mathcal{D}_\rho)_{\mathbb{F}}$-approximable.*

Proof of Lemma 13. The idea behind the proof is to exploit the *random self reducibility* of $\mathrm{MOD}_{m,r,w}$. Let $x \in \{0,1\}^n$ be *arbitrary* and $w \circ x \in \mathbb{Z}_m^n$ be the vector whose i-th entry is $w_i x_i$. For any $y \in \mathbb{Z}_m^n$ and $t \in \mathbb{Z}_m$, we define $S_{y,t}(x) \in \{0,1\}^n$ to be such that the ith bit of $S_{y,t}(x)$ is 1 iff the i-th entry of the vector $(w \circ x) + y$ is congruent to t modulo m; that is, $w_i x_i + y_i \equiv t \pmod m$. The basic observation we will use is that to check if $\sum_i w_i x_i \equiv r \pmod m$, it suffices to check that $\sum_i (w_i x_i + y_i) \equiv r + \sum_i y_i \pmod m$, which in turn reduces to computing $\mathrm{MOD}_{m,r'}(S_{y,t}(x))$ for various r', t.

Formally, note that for *any* $y \in \mathbb{Z}_m^n$, we have

$$\mathrm{MOD}_{m,r,w}(x) = \sum_{(r_1,\ldots,r_{m-1}) \in T} \bigwedge_{t=1}^{m-1} \mathrm{MOD}_{m,r_t}(S_{y,t}(x)) \tag{1}$$

where $T = \left\{ (r_1, \ldots, r_{m-1}) \in \mathbb{Z}_m^{m-1} \;\middle|\; \sum_j j \cdot r_j \equiv r + \sum_i y_i \pmod m \right\}$. (Note that the function on the left hand side above is $\mathrm{MOD}_{m,r,w}$, where as on the right hand side, we have the symmetric functions $\mathrm{MOD}_{m,r'}$ for various r'.) For any fixed y, the i-th bit of $S_{y,t}(x)$ depends only on x_i and by Fact 8, can be represented exactly by a degree 1 polynomial in x_i. Now, for a uniformly random $\mathbf{y} \in \mathbb{Z}_m^n$, $(w \circ x) + \mathbf{y}$ is a uniformly random element of $\mathbb{Z}_m^n$. Hence, $S_{\mathbf{y},t}(x)$ is a random element from $\{0,1\}^n$ chosen according to the distribution $\mathcal{D}_{\frac{1}{m}}$ and its individual bits are degree 1 *probabilistic* polynomials in x.

Let $\delta \in (0, \frac{1}{2})$ be a parameter whose value we will fix later. Since the function $\mathrm{MOD}_{m,r'}$ is symmetric, by Lemma 14, we know that for all m, r' there is a polynomial $P_{m,r'}$ of degree $O(\sqrt{n \log(1/\delta)})$ such that $\Pr_{\mathbf{x} \sim \mathcal{D}_{\frac{1}{m}}}[P_{m,r'}(\mathbf{x}) \neq \mathrm{MOD}_{m,r'}(\mathbf{x})] \leq \delta$. Consider the *probabilistic* polynomial $\mathbf{Q}_{m,r,w}(X_1, \ldots, X_n)$ defined (based on the choice of $\mathbf{y} \in \mathbb{Z}_m^n$) as follows:

$$\mathbf{Q}_{m,r,w}(X_1, \ldots, X_n) = \sum_{(r_1,\ldots,r_{m-1}) \in T} \prod_{t=1}^{m-1} P_{m,r_t}(S_{\mathbf{y},t}(X_1, \ldots, X_n))$$

.

Observe that the degree of $\mathbf{Q}_{m,r,w}$ is $O(m\sqrt{n \log(1/\delta)})$.

We claim that for suitably small δ, $\mathbf{Q}_{m,r,w}$ is an ε-error probabilistic polynomial for $\mathrm{MOD}_{m,r,w}$. To see this, note that if $\mathbf{y} \in \mathbb{Z}_m^n$ satisfies $P_{m,r'}(S_{\mathbf{y},t}(x)) = \mathrm{MOD}_{m,r'}(S_{\mathbf{y},t}(x))$ for all choices of r' and t, then by (1), we have $\mathbf{Q}_{m,r,w}(x) = \mathrm{MOD}_{m,r,w}(x)$. Thus, we have

$$\Pr_{\mathbf{Q}}[Q_{m,r,w}(x) \neq \mathrm{MOD}_{m,r,w}(x)] \leq \Pr_{\mathbf{y}}[\exists r', t \; P_{m,r'}(S_{\mathbf{y},t}(x)) \neq \mathrm{MOD}_{m,r'}(S_{\mathbf{y},t}(x))]$$

For any fixed r', t, by our choice of $P_{m,r'}$, the probability that $P_{m,r'}(S_{\mathbf{y},t}(x))$ does not equal $\mathrm{MOD}_{m,r'}(S_{\mathbf{y},t}(x))$ is at most δ. By a union bound over all t, r' and using the inequality above, we have $\Pr_{\mathbf{Q}}[Q_{m,r,w}(x) \neq \mathrm{MOD}_{m,r,w}(x)] \leq m^2 \cdot \delta$. Setting $\delta = \varepsilon/m^2$, we get that $\mathbf{Q}_{m,r,w}$ is an ε-error probabilistic polynomial for $\mathrm{MOD}_{m,r,w}$. The degree of $\mathbf{Q}_{m,r,w}$ is $O(m\sqrt{n \log(1/\delta)}) = O(\sqrt{n \log(m/\varepsilon)} \cdot m)$. ◀

3.2 Proof of Theorem 11

By the definition of SUM_W, there exist $w_1, \ldots, w_n \in \mathbb{N}$ such that $\sum_{i=1}^n w_i \leq W$ and for any $x \in \{0,1\}^n$, the output $f(x)$ is determined by the value $\sum_{i=1}^n w_i x_i \in \{0, 1, \ldots, W\}$.

We reduce the problem of constructing small error probabilistic polynomials for f to that of constructing small error probabilistic polynomials for $\mathrm{MOD}_{m,r,\overline{w}}$ (for suitable $m, \overline{w}$) by using the Chinese Remainder Theorem (see, e.g., [13]), a special case of which follows.

▶ **Theorem 15.** *Fix a positive $W \in \mathbb{N}$ and any distinct primes $p_1, \ldots, p_\ell$ ($\ell \geq 1$) such that $\prod_{j=1}^{\ell} p_j > 2W$. Then, given any congruence classes $a_j \in \{0, \ldots, p_j - 1\}$ ($j \in [\ell]$) modulo these primes, there is at most one $k \in \{-W, -(W-1), \ldots, W\}$ s.t. $k \equiv a_j \pmod{p_j}$ for each $j \in [\ell]$.*

Let $\ell = \lceil \log W \rceil + 2$ and $p_1 \leq \cdots \leq p_\ell$ be the first ℓ distinct primes. By the Prime Number Theorem [13], $\max_{j \in [\ell]} p_j = p_\ell = O(\log W \log\log W)$. Since $\prod_{j=1}^{\ell} p_j \geq 2^\ell > 2W$, we see using Theorem 15 that each integer $k \in \{0, \ldots, W\}$ is uniquely determined (among the integers $\{0, \ldots, W\}$) by its congruence classes modulo each of the p_j ($j \in [\ell]$); that is, by the tuple $(k \pmod{p_j})_{j \in [\ell]}$. In particular, for any $x \in \{0,1\}^n$, the integer $\sum_{i=1}^{n} w_i x_i$ is uniquely determined by $((\sum_i w_i x_i) \pmod{p_j})_{j \in [\ell]}$. Moreover, note that for any $j \in [\ell]$, the congruence class of $\sum_i w_i x_i$ modulo p_j is determined uniquely by the tuple $(\mathrm{MOD}_{p_j,r,\overline{w}^j}(x))_{r \in \mathbb{Z}_{p_j}}$, where $\overline{w}^j$ is obtained from w by dropping each of its entries modulo p_j. Hence, for any $x \in \{0,1\}^n$, the value of $f(x)$ is determined by the tuple $(\mathrm{MOD}_{p_j,r,\overline{w}^j}(x))_{j \in [\ell], r \in \mathbb{Z}_{p_j}}$.

We summarize the discussion above in the form of the following claim. Let $s = \sum_{j=1}^{\ell} p_j$.

▶ Claim 16. Let $f, w_1, \ldots, w_n, p_1, \ldots, p_\ell$ be as above. Then, there is a function $g \in \mathcal{B}_s$ such that for all $x \in \{0,1\}^n$, we have $f(x) = g\left(\mathrm{MOD}_{p_j,r,\overline{w}^j}(x) : j \in [\ell], r \in \mathbb{Z}_{p_j}\right)$.

Let $\delta \in (0, \frac{1}{2})$ be a parameter that we will fix below. Now, by Lemma 13, we know that for each $j \in [\ell]$ and $r \in \mathbb{Z}_{p_j}$, the function $\mathrm{MOD}_{p_j,r,\overline{w}^j}$ has a δ-approximating probabilistic polynomial of degree $O(p_j \sqrt{n \log(p_j/\delta)}) = O(\log W \log\log W \sqrt{n \log(\log W/\delta)}) = O(\log^2 W \sqrt{n \log(1/\delta)})$. Moreover, by Fact 8, the function g from Claim 16 can be represented *exactly* by a polynomial of degree $s = \sum_j p_j = O(\log^3 W)$. Thus, by Fact 9, we see that f has a $((\sum_j p_j) \cdot \delta)$-error probabilistic polynomial of degree $O((\log W)^5 \sqrt{n \log(1/\delta)})$ for any $\delta \in (0, 1/2)$.

Since $\sum_j p_j = O(\log^3 W)$, we may set $\delta = \varepsilon / C \log^3 W$ for some large absolute constant C and thus obtain an ε-error probabilistic polynomial for f. The degree of this polynomial is $(\log W)^{O(1)} \sqrt{n \log(1/\varepsilon)}$. This completes the proof of Theorem 11.

3.3 Proof of Theorem 12

In this section, we construct probabilistic polynomials for an arbitrary $f \in \mathrm{THR}^n$. We use many ideas and observations of Hofmeister [17] (see also [14, 15]) regarding the structure of Linear Threshold functions.

Recall that given $f \in \mathrm{THR}^n$, there exist $a_1, \ldots, a_n, \theta \in \mathbb{R}$ such that for all $x \in \{0,1\}^n$, we have $f(x) = 1$ iff $\sum_{i=1}^{n} a_i x_i - \theta \geq 0$. Moreover, it is known by a result of Muroga [21] that we may in fact choose $a_1, \ldots, a_n$ and θ to be integers of magnitude at most $M = 2^{O(n \log n)}$. We fix such integers $a_1, \ldots, a_n, \theta$. Let N denote $M(n+1)$. Let $I = \{i \in \mathbb{Z} \mid -(n+1) \leq i \leq n+1\}$ and $I^{\geq 0}$ be the non-negative members of I.

Similar to [17], we define the following.

- For $j \in \{0, \ldots, \ell = \lceil \log N \rceil\}$, we define integers $a_i^{(j)}$ ($i \in [n]$) and $\theta^{(j)}$ as follows. We define $\theta^{(0)} = \theta$ and $a_i^{(0)} = a_i$ for $i \in [n]$. For $j \in [\ell]$, we define $\theta^{(j)} = \mathrm{trunc}(\theta^{(j-1)}/2)$ and $a_i^{(j)} = \mathrm{trunc}(a_i^{(j-1)}/2)$ for $i \in [n]$. Here, $\mathrm{trunc}(z) = \lfloor z \rfloor$ for $z \geq 0$ and $\lceil z \rceil$ for $z < 0$.
- For $j \in \{0, \ldots, \ell\}$ and any prime $p \in \mathbb{N}$, define

- $\mathrm{INS}_j \in \mathcal{B}_n$ so that $\mathrm{INS}_j(x) = 1$ iff $\sum_{i=1}^n a_i^{(j)} x_i - \theta^{(j)} \in I$.
- $\mathrm{INS}_j^p \in \mathcal{B}_n$ so that $\mathrm{INS}_j^p(x) = 1$ iff $\exists k \in I$ such that $\sum_{i=1}^n a_i^{(j)} x_i - \theta^{(j)} \equiv k \pmod p$.
- $\mathrm{POS}_j \in \mathcal{B}_n$ so that $\mathrm{POS}_j(x) = 1$ iff $\sum_{i=1}^n a_i^{(j)} x_i - \theta^{(j)} \in I^{\geq 0}$.
- $\mathrm{POS}_j^p \in \mathcal{B}_n$ so that $\mathrm{POS}_j^p(x) = 1$ iff $\exists k \in I^{\geq 0}$ such that $\sum_{i=1}^n a_i^{(j)} x_i - \theta^{(j)} \equiv k \pmod p$.

The following is implicit in [17]. We omit the proof.

▶ **Lemma 17.** *Let $f, \mathrm{POS}_j, \mathrm{INS}_j$ ($j \in \{0, \ldots, \ell\}$) be as defined above. Then, for any $x \in \{0,1\}^n$, $f(x) = \mathrm{POS}_0(x) + \sum_{j=1}^{\ell} \left(\overline{\mathrm{INS}_{j-1}(x)} \wedge \mathrm{POS}_j(x)\right)$, where the sum is taken over the integers (and hence, the equality also holds modulo the characteristic of $\mathbb{F}$).*

We construct a small error probabilistic polynomial for f by constructing small error probabilistic polynomials for the functions $\mathrm{INS}_j, \mathrm{POS}_j$ ($j \in \{0, \ldots, \ell\}$). The following lemma assures us that this is possible.

▶ **Lemma 18.** *For $j \in \{0, \ldots, \ell\}$ and any $\delta \in (0, 1/2)$, the functions INS_j and POS_j have δ-error probabilistic polynomials of degree $O(\sqrt{n} \cdot (\log n \log(1/\delta))^{O(1)})$.*

Assuming Lemma 18, we can prove Theorem 12 easily as follows. Fix $\delta = \varepsilon/(2\ell+1)$. Using Lemma 18, we have δ-error probabilistic polynomials $\mathbf{P}_j$ for POS_j and $\mathbf{Q}_j$ for INS_j of degree $O(\sqrt{n} \cdot (\log n \log(1/\varepsilon))^{O(1)})$. Consider the probabilistic polynomial $\mathbf{P} = \mathbf{P}_0 + \sum_{j=1}^{\ell} (1 - \mathbf{Q}_{j-1}) \cdot \mathbf{P}_j$. By Lemma 17, for any $x \in \{0,1\}^n$, we have $\mathbf{P}(x) = f(x)$ unless there is a $j \in \{0, \ldots, \ell\}$ such that $\mathbf{P}_j(x) \neq \mathrm{POS}_j(x)$ or a $j \in [\ell]$ such that $\mathbf{Q}_j(x) \neq \mathrm{INS}_j(x)$. By a union bound, the probability that this occurs is at most $(2\ell+1) \cdot \delta = \varepsilon$. Hence, $\mathbf{P}$ is an ε-error probabilistic polynomial for f of degree $O(\sqrt{n} \cdot (\log n \log(1/\varepsilon))^{O(1)})$. This finishes the proof of Theorem 12.

Proof of Lemma 18. Fix a $j \in \{0, \ldots, \ell\}$. We prove the claim for INS_j only. The proof for POS_j is very similar with $I^{\geq 0}$ replacing I throughout.

Recall that $\mathrm{INS}_j(x) = 1$ iff $\sum_{i=1}^n a_i^{(j)} x_i - \theta^{(j)} \in I$. Hence, $\mathrm{INS}_j(x)$ depends only on the sum $\sum_i a_i^{(j)} x_i$ and this might indicate that Theorem 11, that constructs probabilistic polynomials for functions in SUM_W, might be useful. However, the value of W here is too large: it may be as large as $M \cdot (n+1) = 2^{\Omega(n \log n)}$. We therefore first reduce the problem to the case of small-weight sums by going modulo small primes. This idea has been used before by Hofmeister [17], albeit for a different purpose from ours.

Consider any prime $p \in \mathbb{N}$. Note that $\mathrm{INS}_j^p(x)$ is a function only of $\sum_i b_i x_i$ where $b_i \in \{0, \ldots, p-1\}$ is chosen so that $b_i \equiv a_i^{(j)} \pmod p$. Thus, $\mathrm{INS}_j^p \in \mathrm{SUM}_W$ where $W = O(pn)$. Moreover, observe that:

- If $\mathrm{INS}_j(x) = 1$, then $\mathrm{INS}_j^p(x) = 1$ for any prime p.
- If $\mathrm{INS}_j(x) = 0$, then though $\mathrm{INS}_j^p(x)$ could be 1 for some values of p, this does not happen too often. The following makes this precise:

  ▶ Claim 19. If $\mathrm{INS}_j(x) = 0$, then $\left| \left\{ p \in \mathbb{N} \,\middle|\, p \text{ prime and } \mathrm{INS}_j^p(x) = 1 \right\} \right| \leq \log((2N)^{2n+3}) \leq Cn^2 \log n$ for some constant $C > 0$.

  Proof. Since $\mathrm{INS}_j(x) = 0$, the integer $K := \sum_i a_i^{(j)} x_i - \theta^{(j)}$ does not belong to I. Also, we know that $|K| \leq M(n+1) = N$. Now, let $S_x = \left\{ p \in \mathbb{N} \,\middle|\, p \text{ prime and } \mathrm{INS}_j^p(x) = 1 \right\}$. By the definition of S_x, $p \in S_x$ iff there is some $k \in I$ such that $K \equiv k \pmod p$ or in other words, $p | (K - k)$. Thus, every $p \in S_x$ divides $K - k$ for some $k \in I$ and hence the product $\prod_{k \in I}(K-k)$. Therefore, $|S_x| \leq \log(\prod_k |K-k|) \leq \log((2N)^{2n+3})$ as claimed. ◀

Now fix $r = \lceil (2Cn^2 \log n)/\delta \rceil$ where C is the constant from Claim 19. Let $p_1, \ldots, p_r$ be the first r primes. By our reasoning above, for any $x \in \{0,1\}^n$, we have

- $\mathrm{INS}_j(x) = 1 \Rightarrow \mathrm{INS}_j^{p_k}(x) = 1 \quad \forall k \in [r]$,
- $\mathrm{INS}_j(x) = 0 \Rightarrow \Pr_{k \in [r]}[\mathrm{INS}_j^{p_k}(x) = 1] \leq \frac{Cn^2 \log n}{r} \leq \frac{\delta}{2}$

We claim that the functions $\mathrm{INS}_j^{p_k}$ ($k \in [r]$) have probabilistic polynomials of low degree. As we argued above, the function $\mathrm{INS}_j^{p_k}(x) \in \mathrm{SUM}_{W_k}$ for $W_k = O(p_k n)$. Moreover, by the Prime Number Theorem [13], we have $p_k \leq O(r \log r)$ for each $k \in [r]$. Hence, for each $k \in [r]$, the function $\mathrm{INS}_j^{p_k} \in \mathrm{SUM}_W$ for $W = O(nr \log r)$. Applying Theorem 11, we see that the function $\mathrm{INS}_j^{p_k}$ has a $(\delta/2)$-error probabilistic polynomial $\mathbf{Q}_k$ of degree $O(\sqrt{n \log(1/\delta)}(\log W)^{O(1)}) = \sqrt{n} \cdot (\log n \log(1/\delta))^{O(1)}$, where the final equality follows from our choice of r.

Now, consider the probabilistic polynomial $\mathbf{Q}$: that is, we first pick $\mathbf{k} \in [r]$ uniformly at random and then sample a polynomial from the distribution $\mathbf{Q_k}$. We claim that $\mathbf{Q}$ is a δ-error probabilistic polynomial for INS_j. To see this, note that for any $x \in \{0,1\}^n$, we must have $\mathbf{Q}(x) = \mathrm{INS}_j(x)$ unless one of the following two events occurs:

- $\mathrm{INS}_j^{p_\mathbf{k}}(x) \neq \mathrm{INS}_j(x)$. As we saw above, this happens with probability at most $\delta/2$.
- $\mathbf{Q_k}(x) \neq \mathrm{INS}_j^{p_\mathbf{k}}(x)$. By our choice of $\mathbf{Q_k}$, this happens with probability at most $\delta/2$.

Hence, the probability that $\mathbf{Q}(x) \neq \mathrm{INS}_j(x)$ is at most δ. Thus, $\mathbf{Q}$ is a δ-error probabilistic polynomial for INS_j of degree $\sqrt{n} \cdot (\log n \log(1/\delta))^{O(1)}$. ◀

4 Connections to other problems

1-round Compression by $\mathrm{ACC}^0[p]$ circuits. Motivated by applications in Cryptography, parameterized complexity, and PCPs, Chattopadhyay and Santhanam [11] study the problem of proving lower bounds for compression by constant-depth circuits. We briefly describe the setup here, referring to [11] for details.

We define a *compression game*, between two players Alice and Bob, as follows. Let $f \in \mathcal{B}_n$ be known to both players. Alice, a computationally bounded player, is given $x \in \{0,1\}^n$ and wishes to compute $f(x)$ with the aid of Bob, who is computationally unbounded. The aim is to minimize the amount of communication between the players. We consider the case when Alice's computational power is restricted to a class of constant-depth circuits $\mathcal{C}$, which we call a $\mathcal{C}$-compression game. We also consider the special case of 1-*round compression games*, where Alice sends a message to Bob based on the input x and Bob declares the value $f(x)$. Note that for any reasonable $\mathcal{C}$, there is always a 1-round communication protocol with communication n: Alice simply sends x to Bob, who then outputs $f(x)$.

In the case that $\mathcal{C} = \mathrm{AC}^0$, [11] showed a close to optimum communication lower bound of $n/(\log n)^{O(1)}$ for the MOD_2 function. In the case that $\mathcal{C} = \mathrm{ACC}^0[p]$ (p prime),using Theorem 5, they show that any 1-*round* compression protocol for $\mathrm{MOD}_{q,0}$ must involve $\Omega(\sqrt{n}/(\log n)^{O(1)})$ bits of communication. Using the same idea, we can show that stronger inapproximability results (w.r.t. *any* distribution) would imply stronger communication lower bounds for 1-round $\mathrm{ACC}^0[p]$-compression game. The proof is omitted.

▶ **Theorem 20.** *Assume $f \in \mathcal{B}_n$ has a 1-round $\mathrm{ACC}^0[p]$-compression protocol with communication c. Then, f has a $(1/10)$-error probabilistic polynomial over $\mathbb{F}_p$ of degree $O(c \cdot (\log n \log c)^{O(1)})$. In particular, if f has no $(1/10)$-error probabilistic polynomial of degree $n^{1/2+\varepsilon}$ over $\mathbb{F}_p$ for some $\varepsilon > 0$, then any 1-round $\mathrm{ACC}^0[p]$-compression protocol for f requires $\Omega(n^{1/2+\varepsilon}/(\log n)^{O(1)})$ bits of communication.*

Lower bounds for $\mathrm{AC}^0 \circ \mathrm{SYMTHR}$. There has been considerable work on proving lower bounds for AC^0 augmented with more powerful gates in some specific way, as a way of making progress towards proving lower bounds for stronger circuit classes, such as TC^0. Some of these works [16, 9, 20] have considered the model where the AC^0 circuits are allowed to have a "few" SYM, THR, or other gates. We consider a different variant where there are no bounds on the *number* of SYM and THR gates but on their *position* in the circuit: we require that these gates appear just above the input variables. As far as we know, lower bounds even for simple special cases of this model such as $\mathrm{OR} \circ \mathrm{AND} \circ \mathrm{SYM}$ and $\mathrm{OR} \circ \mathrm{AND} \circ \mathrm{THR}$ are unknown. In the case that the SYM gates at the bottom involve modular computation, some recent progress [12, 10] has been made. We can show that any function in the circuit class $\mathrm{AC}^0 \circ \mathrm{SYMTHR}$ has small error probabilistic polynomials of degree $O(\sqrt{n} \cdot \mathrm{polylog}(n))$, and hence proving that an explicit function f does not have probabilistic polynomials of this degree will prove an explicit lower bound for this circuit class. The proof is omitted.

▶ **Theorem 21.** *Let $F \in \mathcal{B}_n$ be computed by an $\mathrm{AC}^0 \circ \mathrm{SYMTHR}$ circuit of size s. Then, F has a $(1/10)$-error probabilistic polynomial of degree $O(\sqrt{n}(\log n \log s)^{O(1)})$ over any field $\mathbb{F}$. In particular, if F has no $(1/10)$-error probabilistic polynomial of degree less than $n^{1/2+\Omega(1)}$ over some field, then any $\mathrm{AC}^0 \circ \mathrm{SYMTHR}$ circuit for F must have size $2^{n^{\Omega(1)}}$.*

Correlation bounds for low-degree polynomials. Let $p \in \mathbb{N}$ be a constant prime. We consider the problem of proving *correlation bounds* over $\mathbb{F}_p$: i.e., showing that an explicit Boolean function $f \in \mathcal{B}_n$ is $(d, \delta, \mathcal{D})_{\mathbb{F}_p}$-inapproximable for δ close to $1/2$. For brevity, we say that $f \in \mathcal{B}_n$ is $(d, \varepsilon, \mathcal{D})_{\mathbb{F}_p}$-*correlated* (resp. uncorrelated) if it is $(d, \frac{1}{2} - \varepsilon, \mathcal{D})_{\mathbb{F}_p}$-approximable (resp. inapproximable). Showing strong correlation bounds against low-degree polynomials, (say, for $\varepsilon \ll 1/n$, $d = \mathrm{polylog}(n)$, and $\mathcal{D}$ the uniform distribution) would have applications to constructing PRGs for $\mathrm{ACC}^0[p]$ (see [27]).

Smolensky [25] showed that the MAJ^n function is $(d, O\left(\frac{d}{\sqrt{n}}\right), \mathcal{U})_{\mathbb{F}_p}$-uncorrelated for any $d \in \mathbb{N}$ where $\mathcal{U}$ denotes the uniform distribution. For $d \ll \log n$, this bound has been strengthened by results of Bourgain [7] and Viola and Wigderson [28]. However, when $d \geq \log n$, Smolensky's bound has not been improved for any probability distribution $\mathcal{D}$. We can show that proving probabilistic polynomial degree lower bounds of $n^{1/2+\Omega(1)}$ over $\mathbb{F}_p$ would improve on Smolensky's correlation lower bound for *some* (possibly non-explicit) distribution. The proof is omitted.

▶ **Theorem 22.** *Assume that for some $d \in \mathbb{N}$ and $\delta > 0$, the function $f \in \mathcal{B}_n$ is $(d, \delta, \mathcal{D})_{\mathbb{F}_p}$-correlated for* every *distribution $\mathcal{D}$ over $\{0,1\}^n$. Then, for any $\varepsilon \in (0, \frac{1}{2})$, f has an ε-error probabilistic polynomials of degree $\frac{d \cdot (\log(1/\delta) \log(1/\varepsilon))^{O(1)}}{\delta}$ over $\mathbb{F}_p$. In particular, if f has no $(1/10)$-error probabilistic polynomial of degree $n^{1/2+\eta}$ for some constant $\eta > 0$, than there is some distribution $\mathcal{D}$ such that f is $(d, O\left(\frac{d \cdot (\log n)^{O(1)}}{n^{1/2+\eta}}\right), \mathcal{D})_{\mathbb{F}_p}$-uncorrelated.*

Lower bounds for $\mathrm{AC}^0 \circ \mathrm{MOD}_2$ circuits computing Inner Product. We define the Inner product function $\mathrm{IP}^n \in \mathcal{B}_{2n}$ as follows: $\mathrm{IP}^n(x_1, \ldots, x_n, y_1, \ldots, y_n) = \bigoplus_{i=1}^n x_i \wedge y_i$. By definition, IP^n has a depth-2 $\mathrm{ACC}^0[2]$ circuit of linear size, which is made up of $2n$ AND gates feeding into a $\mathrm{MOD}_{2,1}$ gate. Servedio and Viola [23] consider the question of whether IP^n can be computed by a polynomial sized $\mathrm{AC}^0 \circ \mathrm{MOD}_2$ circuit. This question has relations to Matrix Rigidity (see [23]) and Communication complexity [2].

Here, we show the following: if there is a constant-degree polynomial from $\mathbb{F}_2[X_1, \ldots, X_n]$ that does not have a $(1/10)$-error probabilistic polynomial of degree $n^{1/2+\Omega(1)}$ over *some*

field $\mathbb{F}$, then IP^n does not have polynomial-sized $\mathrm{AC}^0 \circ \mathrm{MOD}_2$ circuits. Note that by Lemma 13, any degree-1 polynomial over $\mathbb{F}_2$ has a (1/10)-error probabilistic polynomial of degree $O(n^{1/2})$ over any field $\mathbb{F}$. For polynomials of degree 2 and above, as far as we are aware, there are no such results known.

▶ **Theorem 23.** *If* IP^n *has an* $\mathrm{AC}^0 \circ \mathrm{MOD}_2$ *circuit of size* s*, then any constant-degree polynomial* $P \in \mathbb{F}_2[X_1, \ldots, X_n]$ *has a* (1/10)*-error probabilistic polynomial* $\mathbf{Q}$ *of degree* $O(\sqrt{n} \cdot (\log n \log s)^{O(1)})$ *over* any *field. In particular, if there is a constant-degree polynomial* $P \in \mathbb{F}_2[X_1, \ldots, X_n]$ *that has no* (1/10)*-error probabilistic polynomial of degree less than* $n^{1/2+\Omega(1)}$ *over some field, then any* $\mathrm{AC}^0 \circ \mathrm{MOD}_2$ *circuit for* IP^n *must have size* $2^{n^{\Omega(1)}}$.

The proof of the above is postponed to the full version of the paper. Note that as opposed to the explicit lower bounds required for the applications in previous sections, the above theorem says that a degree lower bound for probabilistic polynomials over some $\mathbb{F}$ even for a somewhat *non-explicit* function suffices to prove a lower bound for the explicit Inner Product function.

Acknowledgements. I would like to thank Arkadev Chattopadhyay, Parikshit Gopalan, Kristoffer Hansen, Prahladh Harsha, Swastik Kopparty, Nutan Limaye, and Emanuele Viola for their valuable feedback and encouragement. Arkadev Chattopadhyay also suggested the application to 1-round compression (Section 4) and I thank him for his permission to include it here. I am also grateful to the anonymous referees for their corrections and suggestions. Finally, I acknowledge the support of DST Inspire grant IFA12-ENG-14.

References

1 James Aspnes, Richard Beigel, Merrick L. Furst, and Steven Rudich. The expressive power of voting polynomials. *Combinatorica*, 14(2):135–148, 1994.

2 László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *FOCS*, pages 337–347. IEEE Computer Society, 1986.

3 Louay M. J. Bazzi. Polylogarithmic independence can fool dnf formulas. *SIAM J. Comput.*, 38(6):2220–2272, 2009.

4 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, 2001.

5 Richard Beigel. The polynomial method in circuit complexity. In *In Proceedings of the 8th IEEE Structure in Complexity Theory Conference*, pages 82–95. IEEE Computer Society Press, 1995.

6 Nayantara Bhatnagar, Parikshit Gopalan, and Richard J. Lipton. Symmetric polynomials over z_m and simultaneous communication protocols. *J. Comput. Syst. Sci.*, 72(2):252–285, 2006.

7 Jean Bourgain. Estimation of certain exponential sums arising in complexity theory. *Comptes Rendus Mathematique*, 340(9):627 – 631, 2005.

8 Mark Braverman. Polylogarithmic independence fools AC^0 circuits. *J. ACM*, 57(5), 2010.

9 Arkadev Chattopadhyay and Kristoffer Arnsfelt Hansen. Lower bounds for circuits with few modular and symmetric gates. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 994–1005. Springer, 2005.

10 Arkadev Chattopadhyay and Shachar Lovett. Linear systems over finite abelian groups. In *IEEE Conference on Computational Complexity*, pages 300–308. IEEE Computer Society, 2011.

11 Arkadev Chattopadhyay and Rahul Santhanam. Lower bounds on interactive compressibility by constant-depth circuits. In *FOCS*, pages 619–628. IEEE Computer Society, 2012.

12 Arkadev Chattopadhyay and Avi Wigderson. Linear systems over composite moduli. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 43–52. IEEE Computer Society, 2009.

13 G. Everest and T. Ward. *An Introduction to Number Theory*. Graduate Texts in Mathematics. Springer, 2005.

14 Mikael Goldmann, Johan Håstad, and Alexander A. Razborov. Majority gates vs. general weighted threshold gates. *Computational Complexity*, 2:277–300, 1992.

15 Mikael Goldmann and Marek Karpinski. Simulating threshold circuits by majority circuits. *SIAM J. Comput.*, 27(1):230–246, 1998.

16 Kristoffer Arnsfelt Hansen and Peter Bro Miltersen. Some meet-in-the-middle circuit lower bounds. In Jirí Fiala, Václav Koubek, and Jan Kratochvíl, editors, *MFCS*, volume 3153 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2004.

17 Thomas Hofmeister. A note on the simulation of exponential threshold weights. In Jin yi Cai and C. K. Wong, editors, *COCOON*, volume 1090 of *Lecture Notes in Computer Science*, pages 136–141. Springer, 1996.

18 Adam R. Klivans and Rocco A. Servedio. Learning dnf in time $2^{\tilde{O}(n^{1/3})}$. *J. Comput. Syst. Sci.*, 68(2):303–318, 2004.

19 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993.

20 Shachar Lovett and Srikanth Srinivasan. Correlation bounds for poly-size AC^0 circuits with $n^{1-o(1)}$ symmetric gates. In Leslie Ann Goldberg, Klaus Jansen, R. Ravi, and José D. P. Rolim, editors, *APPROX-RANDOM*, volume 6845 of *Lecture Notes in Computer Science*, pages 640–651. Springer, 2011.

21 Saburo Muroga. *Threshold logic and its applications*. Wiley, 1971.

22 Alexander A. Razborov. Lower bounds on the size of constant-depth networks over a complete basis with logical addition. *Mathematicheskie Zametki*, 41(4):598–607, 1987.

23 Rocco A. Servedio and Emanuele Viola. On a special case of rigidity. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:144, 2012.

24 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 77–82, 1987.

25 Roman Smolensky. On representations by low-degree polynomials. In *FOCS*, pages 130–138. IEEE Computer Society, 1993.

26 Jun Tarui. Probablistic polynomials, ac0 functions, and the polynomial-time hierarchy. *Theoretical Computer Science*, 113(1):167–183, 1993.

27 Emanuele Viola. On the power of small-depth computation. *Foundations and Trends in Theoretical Computer Science*, 5(1):1–72, 2009.

28 Emanuele Viola and Avi Wigderson. Norms, XOR lemmas, and lower bounds for polynomials and protocols. *Theory of Computing*, 4(1):137–168, 2008.

Implementing Realistic Asynchronous Automata

S. Akshay[1], Ionut Dinca[2], Blaise Genest[3], and Alin Stefanescu[2]

1 **Indian Institute of Technology, Bombay, India**
2 **University of Pitesti, Romania**
3 **CNRS, IRISA, Rennes, France**

Abstract

Zielonka's theorem, established 25 years ago, states that any regular language closed under commutation is the language of an *asynchronous automaton* (a tuple of automata, one per process, exchanging information when performing common actions). Since then, constructing asynchronous automata has been simplified and improved [6, 19, 7, 12, 8, 4, 2, 20, 21].

We first survey these constructions and conclude that the synthesized systems are not *realistic* in the following sense: existing constructions are either plagued by deadends, non deterministic guesses, or the acceptance condition or choice of actions are not distributed. We tackle this problem by giving (effectively testable) necessary and sufficient conditions which ensure that deadends can be avoided, acceptance condition and choices of action can be distributed, and determinism can be maintained. Finally, we implement our constructions, giving promising results when compared with the few other existing prototypes synthesizing asynchronous automata.

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

Keywords and phrases Asynchronous automata, Zielonka construction, Implementability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.213

1 Introduction

Designing distributed systems is notoriously difficult and prone to bugs. Verification algorithms are very useful to detect and report bugs, but the discovered issues must be solved by the designer. An alternative is to use automatic implementation tools, which directly *synthesize* an implementation that is guaranteed to be correct by construction. As the complexity of automatic implementation is quite high in the general case of *open* distributed systems (distributed games) [9], we focus on closed systems in this paper.

Here, the specification is given as a regular language L over an alphabet Σ where every action (i.e., letter in Σ) is associated with the set of processes managing that action. Such

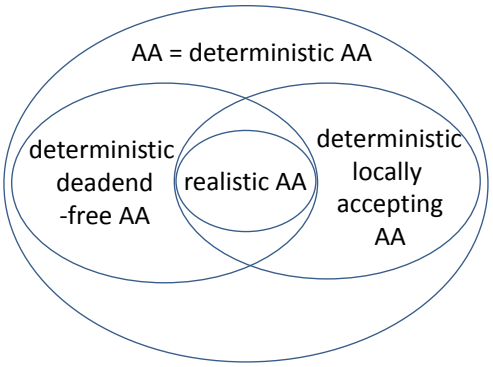


Figure 1 Expressivity of different types of asynchronous automata (AA).

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 213–224
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a specification allows to reason globally about the requirements, instead of having to deal carefully with partial views of each process in a distributed manner (which is one of the error-prone tasks). The problem is then to automatically implement a (truly) distributed control that will globally have the same behavior as the given specification language L. Of course, not all languages can be implemented with such a distributed control. For instance, if ab is the only word in the specification language, with a an action local to a process and b local to another process, then it cannot be implemented in a truly distributed manner. Indeed, any distributed implementation will also feature the word ba, since a process is unable to know when another process performs an (independent) action.

Zielonka's theorem, established 25 years ago [22], states that this is sufficient: every language closed by this commutation relation can be implemented in the form of an *asynchronous automaton*, that is, a network of automata where the control is *mostly* distributed, and two processes can exchange information whenever they perform a common action. Initially, this was merely an expressiveness result and was believed to be rather impractical due to its prohibitive complexity. During subsequent years, this construction has been simplified and improved in several works [6, 19, 7, 12, 8]. Also, different constructions [4, 2, 20] and heuristics [21] have been proposed to handle the complexity blow-up.

However, none of these constructions gives a general *realistic* distributed implementation: either the constructions are plagued by deadends [22, 6, 19, 7, 12, 8, 2, 4], non-deterministic guesses [23, 5, 2], or the acceptance condition or choice of actions are not distributed [22, 6, 19, 7, 12, 8]. Further, while the initial state is trivially distributed in Zielonka's construction (since it is unique, due to determinism), this is not the case in [23, 5]. One cannot always obtain an implementation satisfying all these conditions: we schematically depict in Figure 1 the relations between corresponding subclasses, proved in Proposition 6.

Thus, our main goal is to characterize the class of regular languages that can be implemented by a *realistic asynchronous automaton*, i.e., one which is deterministic, deadend-free, and has distributed final states and choice of actions. This notion strictly subsumes the class of deadend-free synchronized product of automata [17]. Our central result provides semantical and syntactical characterizations of languages of realistic asynchronous automata, together with algorithms to check these characterizations. Thus, given a global regular specification passing these algorithmic tests, we build a realistic asynchronous automaton which distributedly implements the specification. Finally, we implement our procedure, based on the latest, state of the art variant of Zielonka's construction [8]. On a variety of distributed programs, we show that this gives realistic distributed implementations of a size which is reasonable compared to existing implementations.

Asynchronous automata model shared-memory systems directly. However, even for message passing systems, Zielonka's theorems continue to remain interesting: [15] and [10] build bounded message passing automata using Zielonka's construction (see [3] for a survey). We are confident that combining the techniques in [15] with our results would lead to the automatic implementation of realistic bounded message passing automata.

The paper is structured as follows. In section 2, we define (realistic) asynchronous automata, and restate the different implementation theorems. In section 3, we come up with semantical and syntactical characterizations of realistic asynchronous automata. In section 4, we exhibit algorithms to test the characterizations and analyze their complexity. In section 5, we experiment and compare the automatic distributed implementation of different specifications. A long version of this paper with complete proofs can be found at `http://perso.crans.org/~genest/ADGS13.pdf`.

2 Realistic Asynchronous Automata

Let $\mathcal{P}$ be a fixed set of processes. A distributed alphabet (Σ, dom) is a finite set Σ of actions together with the domain function $\mathrm{dom} : \Sigma \to 2^{\mathcal{P}} \setminus \emptyset$, which associates to each action a the set $\mathrm{dom}(a)$ of processes executing a. For any $p \in \mathcal{P}$, we also denote $\Sigma_p = \{a \in \Sigma \mid p \in \mathrm{dom}(a)\}$. We say that actions a and b are *independent*, denoted $(a,b) \in I$, iff $\mathrm{dom}(a) \cap \mathrm{dom}(b) = \emptyset$. This gives rise to an equivalence relation on words: first, for all words $v, w \in \Sigma^*$ and actions $(a,b) \in I$, we define $vabw \equiv_1 vbaw$. Then, the transitive reflexive closure of $\equiv_1$, denoted $\equiv$, is an equivalence relation. The equivalence class containing v, denoted $[v]$, is called a (Mazurkiewicz) *trace* [7]. Given a word $w \in \Sigma^*$ and a process $p \in \mathcal{P}$, the *p-view* of w, denoted $\mathrm{view}_p(w)$, is the shortest trace $[v]$ such that: there exists v' with $w \equiv vv'$, and each action $a \in \Sigma_p$ occurs as many times in v as in w. Finally, for a language $L \subseteq \Sigma^*$, $\mathrm{pref}(L)$ will denote its set of prefixes and ϵ will denote the empty string.

An *asynchronous automaton* is a tuple $((S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, In, Fin)$, where for all $p \in \mathcal{P}$, S_p is the set of *local states of process p*, and for all $a \in \Sigma$, $\Delta_a \subseteq \prod_{p\in\mathrm{dom}(a)} S_p \times \prod_{p\in\mathrm{dom}(a)} S_p$ defines the (partial) *transition relation*. Note that while we define the transition relation on letters for ease of presentation, it is equivalent to a corresponding definition on processes. Any $s = (s_p)_{p\in\mathcal{P}} \in \prod_{p\in\mathcal{P}} S_p$ is called a *global state* and $In, Fin \subseteq (S_p)_{p\in\mathcal{P}}$ denote the set of global initial and final states, respectively.

The *semantics* of an asynchronous automaton $AA = ((S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, In, Fin)$ is given by the (sequential) automaton $S(AA) = (C, \to, In, Fin)$ over Σ, where $C = \prod_{p\in\mathcal{P}} S_p$ is the set of global states, and the global transition relation is given by $\to$: $C \to C$ with $(s_p)_{p\in\mathcal{P}} \xrightarrow{a} (s'_p)_{p\in\mathcal{P}}$ iff $(s'_p)_{p\in\mathrm{dom}(a)} \in \Delta_a((s_p)_{p\in\mathrm{dom}(a)})$ and $s'_p = s_p$ for all $p \notin \mathrm{dom}(a)$. As usual, we extend $\to$ to words by fixing for ϵ the empty word: for all $s, s' \in C$, $s \xrightarrow{\epsilon} s'$ iff $s' = s$ and $s \xrightarrow{aw} s'$ iff there exists $s'' \in C$ with $s \xrightarrow{a} s''$ and $s'' \xrightarrow{w} s'$. In case $\to$ is deterministic (which is the case for deterministic asynchronous automata), we will denote $\delta_w(s)$ for the unique state $s' \in C$ (if it exists) such that $s \xrightarrow{w} s'$. The language $\mathcal{L}(AA)$ accepted by AA is by definition $\mathcal{L}(S(AA))$, the language accepted by $S(AA)$.

An automaton $A = (C, \to, In, Fin)$ is *diamond* [7] if for all $s, s', t \in C$ and all $(a,b) \in I$, if $s \xrightarrow{a} s' \xrightarrow{b} t$, then there exists t' with $s \xrightarrow{b} t' \xrightarrow{a} t$. For any given asynchronous automaton AA, $S(AA)$ is *diamond* [7], which implies that $\mathcal{L}(AA)$ is *closed by commutation*: for all $v \in \mathcal{L}(AA)$ and $w \equiv v$, we also have $w \in \mathcal{L}(AA)$.

An asynchronous automaton, as defined above, cannot always be implemented in a distributed manner, without adding further restrictions. For instance, the set of final states is currently given globally. To obtain purely distributed implementations, we now introduce several restrictions on asynchronous automata.

▶ **Definition 1** (determinism). We call an asynchronous automaton $AA = ((S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, In, Fin)$ *deterministic*, if $|In| = 1$ and $|\Delta_a(s)| \leq 1$ for all $a \in \Sigma$ and $s \in \prod_{p\in\mathrm{dom}(a)} S_p$.

Non-determinism allows a process to guess what another process is doing concurrently. Note that every asynchronous automaton can be transformed into a deterministic asynchronous automaton, albeit with an unavoidable blow-up in the number of states [14].

▶ **Definition 2** (deadend-freeness). A global state s is called a *deadend*, if there does not exist a word $w \in \Sigma^*$ and global state $s' \in Fin$ with $s \xrightarrow{w} s'$. An asynchronous automaton is *deadend-free* iff no global state reachable from an initial state is a deadend: for all $v \in \Sigma^*, s_0 \in In$, and all s with $s_0 \xrightarrow{v} s$, the state s is not a deadend.

Deadend-freeness prevents a process from performing actions that will not be observable in terms of the language. For instance, consider two processes p, q and actions a, b, c

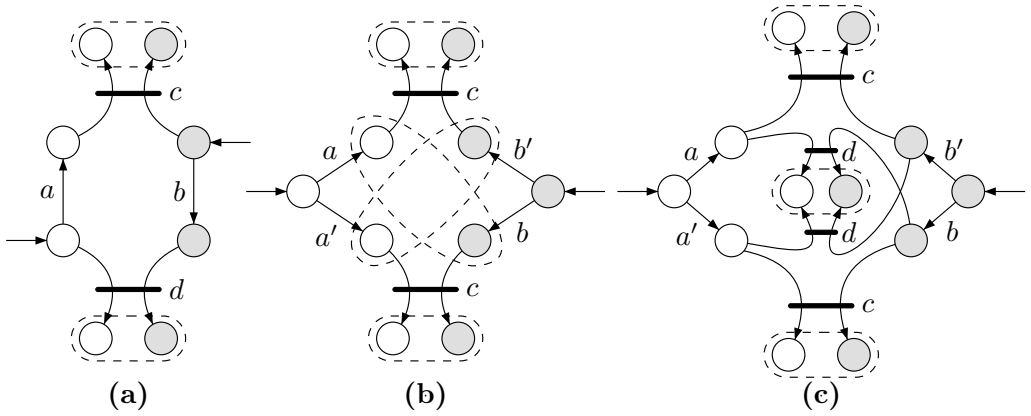


Figure 2 Examples of "unrealistic" asynchronous automata accepting respectively L_1, L_2, L_3. States of process p (resp. q) are unshaded (resp. shaded). Dashed lines mark global final states.

such that $\mathrm{dom}(a) = p$, $\mathrm{dom}(b) = q$ and $\mathrm{dom}(c) = \mathrm{dom}(d) = \{p, q\}$. Then, the language $L_1 = \{ac, bd\}$ cannot be implemented deterministically and without deadends. Indeed, both a and b are allowed from the initial state (which is unique, if the implementation is deterministic), and thus any realistic implementation would also allow ab (and ba), as $\mathrm{dom}(a) \cap \mathrm{dom}(b) = \{p\} \cap \{q\} = \emptyset$. However, an asynchronous automaton with deadends can implement this language as shown in Figure 2(a): the state reached after reading the trace $[ab]$ is a deadend.

▶ **Definition 3** (local acceptance)**.** An asynchronous automaton $((S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, In, Fin)$ is said to be *locally accepting* or *have local final states*, if $Fin = \prod_{p\in\mathcal{P}} Fin_p$ for some $Fin_p \subseteq S_p$ for all $p \in \mathcal{P}$.

Local final states ensure that processes can stop locally, and there is no supervisor which looks at all processes at the same time to choose to stop them. Note that the asynchronous automaton in Figure 2(a) has local final states. Now, the language $L_2 = \{ab, ba, a'b', b'a', a'bc, ba'c, ab'c, b'ac\}$ with $\mathrm{dom}(a) = \mathrm{dom}(a') = p$, $\mathrm{dom}(b) = \mathrm{dom}(b') = q$, and $\mathrm{dom}(c) = \{p, q\}$ cannot be accepted by a deterministic asynchronous automaton having local final states as local states reached on p after a, a' and local states reached on q after b, b' can all be final, depending what the other process did. However, there is a deadend-free deterministic asynchronous automaton with global final states accepting this language, as shown in Figure 2(b). Here, the global final states reached after reading $[ab]$ and $[a'b']$ cannot be expressed as a product of local final states (without also accepting $[ab'], [a'b]$).

▶ **Definition 4** (locally enabled)**.** An asynchronous automaton $((S_p)_{p\in\mathcal{P}}, \Delta, In, Fin)$ is called *locally enabled*, if for all reachable global states $s = (s_p)_{p\in\mathcal{P}}$, $s' = (s'_p)_{p\in\mathcal{P}}$, and $s'' = (s''_p)_{p\in\mathcal{P}}$, if there exist $a \in \Sigma$ and global states t, t' with $s''_p \in \{s_p, s'_p\}$ for all $p \in \mathrm{dom}(a)$ and $s \xrightarrow{a} t$ and $s' \xrightarrow{a} t'$, then there exists a global state t'' with $s'' \xrightarrow{a} t''$.

Local enabledness prevents the processes from taking into account the state of other processes to decide whether they should propose an action or not. In terms of distributed control, process based controllers [9] have this property, while action based controllers [11] do not. The asynchronous automata in Figure 2(a,b) are locally enabled. In a distributed implementation, non-local enabledness is not realistic. For instance, consider the language $L_3 = \{abd, bad, a'bc, ba'c, ab'c, b'ac, a'b'd, b'a'd\}$, with $\mathrm{dom}(a) = \mathrm{dom}(a') = p$, $\mathrm{dom}(b) = \mathrm{dom}(b') = q$ and $\mathrm{dom}(c) = \mathrm{dom}(d) = \{p, q\}$. Intuitively, processes p, q should synchronize

with d if they did both a, b or a', b', and with c if they did a, b' or a', b. This language cannot be realized by a *deadend-free and locally enabled* asynchronous automaton. However, the deadend-free and locally accepting asynchronous automaton shown in Figure 2(c) accepts L_3, but it is not locally-enabled.

Ideally, we would like a realistic distributed implementation to satisfy *all* the properties of determinism, deadend-freeness, local acceptance and local enabledness and not just *some* of them. Thus, by combining all the above desired properties of a distributed implementation we arrive at our proposal for a realistic asynchronous automaton.

▶ **Definition 5.** An asynchronous automaton AA is said to be *realistic*, if AA is deterministic, deadend-free, has local final states, and is locally enabled.

With this definition, language L_3 above cannot be accepted by a realistic asynchronous automaton (because of local enabledness). Using languages L_1, L_2, L_3, we conclude:

▶ **Proposition 6.** *The inclusions schematically represented in Figure 1, between the expressive powers of the above introduced restrictions of asynchronous automata, are strict. Further, the classes of deterministic deadend-free and deterministic locally accepting asynchronous automata have incomparable expressive power.*

We remark here that the notion of realistic automata as defined above strictly subsumes the notion of (deadend-free) synchronized product of automata [17]. Such an automaton is given by a tuple of automata $A = (A_p)_{p\in\mathcal{P}}$, one for each process p on alphabet $\Sigma_p = \Sigma \cap \mathrm{dom}^{-1}(p)$, such that $u \in L(A)$ iff $\pi_p(u) \in L(A_p)$ for all $p \in \mathcal{P}$, where $\pi_p(u)$ is the projection of u on Σ_p, that is u where actions not in Σ_p have been deleted.

▶ **Proposition 7.** *Let $A = (A_p)_{p\in\mathcal{P}}$ be a (possibly non-deterministic) deadend-free synchronized product of automata. Then there exists a realistic asynchronous automaton B with $L(B) = L(A)$. However, the converse does not hold.*

2.1 Survey of the different constructions

In the past 25 years, several attempts have been made to construct asynchronous automata from regular (commutation-closed) specifications which preserve *some* (but not all) of these above mentioned properties. We summarize them below.

▶ **Theorem 8.** *Let L be a regular language closed by commutation. Then, there exists an asynchronous automaton AA over (Σ, dom) with $\mathcal{L}(AA) = L$ such that either:*
1. *AA is deterministic [22, 6, 7, 19, 12, 8], or*
2. *AA is deadend-free [23] (see also [5] for a proof for message-passing systems), or*
3. *AA has local initial and final states [2].*

We provide here the worst case space complexities (the number of local states) to obtain a deterministic or non deterministic asynchronous automaton (Det AA, Non Det AA), given a deterministic or non deterministic diamond automaton A over a set of processes $\mathcal{P}$:

complexity	Det AA	Non Det AA
Det A	$\lvert A\rvert^{O(\lvert\mathcal{P}\rvert^2)} \cdot 2^{2^{\lvert\mathcal{P}\rvert^4}}$ [8]	–
Non Det A	$2^{O(\lvert A\rvert\cdot\lvert\mathcal{P}\rvert^2+\lvert\mathcal{P}\rvert^4)}$ [12]	$\lvert A\rvert^{O(\lvert\mathcal{P}\rvert^2)}$ [2]

The complexities stated to obtain a deterministic asynchronous automaton from [12, 8] are optimal, while optimality is not proven for obtaining a non deterministic asynchronous automaton (using [2] for instance). Note that [2] uses a construction not based on Zielonka's. Determinizing an asynchronous automaton is possible, but the blow-up is doubly exponential [14]: constructing a deterministic asynchronous automaton directly is preferable.

3 Obtaining Realistic Asynchronous Automata

We now turn to the question of characterizing regular languages L for which there exists a *realistic* asynchronous automaton AA such that $\mathcal{L}(AA) = L$. We will give necessary and sufficient semantical conditions on L to have a realistic distributed implementation AA accepting L. Further, we will provide syntactical conditions on automata to be equivalent to realistic distributed implementations, and prove that a diamond automaton with such conditions is always constructible. Our proofs are constructive, in that they provide realistic asynchronous automata. We also offer characterizations for subsets of realistic properties.

Our main proof can use any of the variants of the Zielonka construction from [22, 6, 7, 19, 12, 8] as a "black box", without having to reprove them. Moreover, the changes we make to the implementation obtained from the Zielonka construction do not add states.

3.1 A Theoretical Characterization of Realistic AA

Before stating the main theoretical result of the paper, we first define the syntactical and semantical restrictions which will enable realistic asynchronous automata. Recall that we defined the notion of $\text{view}_p(u)$ in Section 2, which stands for all actions of u that p has seen directly or indirectly (through a common action). For instance, let $\text{dom}(a) = p, \text{dom}(b) = q$ and $\text{dom}(c) = \{p, q\}$. Then $\text{view}_p(abcb) \equiv abc$ since c is "seen" by p ($p \in \text{dom}(c)$) and b is "before" c, b and c are not independent as $\text{dom}(b) \cap \text{dom}(c) = \{q\} \neq \emptyset$.

▶ **Definition 9** (Semantical conditions). For language L, we define the following conditions:

(LC1) *forward diamond*: Whenever $w \in \Sigma^*, (a, b) \in I$ and $wa, wb \in \text{pref}(L)$, we have $wab \in \text{pref}(L)$.

(LC2) *causally closed*: Whenever $w \in \Sigma^*$, if for all $p \in \mathcal{P}$ there exists $v_p \in L$ with $\text{view}_p(v_p) = \text{view}_p(w)$, then $w \in L$.

(LC3) *locally closed*: Whenever $w \in \text{pref}(L)$, if for all actions c and all $p \in \text{dom}(c)$, there exists $v_p c \in \text{pref}(L)$ with $\text{view}_p(v_p) = \text{view}_p(w)$, then $wc \in \text{pref}(L)$.

The first two *language conditions* (LC1,LC2) have been defined before (in the different setting of *Message Sequence Charts* for (LC2) [1]), and their names are standard in the Mazurkiewicz trace community. However, they have only been considered separately; and the third notion (LC3) is new.

▶ **Definition 10** (Syntactical conditions). For a sequential diamond deterministic automaton $A = (C, \to, In, Fin)$, we define the following conditions:

($AC1$) *forward diamond*: Whenever $s, s', t' \in C$, $(a, b) \in I$ with $s \xrightarrow{a} s'$ and $s \xrightarrow{b} t'$, there exists a state t with $s' \xrightarrow{b} t$ and $t' \xrightarrow{a} t$.

($AC2$) Whenever $s \in C$, if for all $p \in \text{dom}(a)$ there exist $r^p, t^p \in C$ and words $w^p, (w')^p \in (\Sigma \setminus \Sigma_p)^*$, such that $r^p \xrightarrow{w^p} s$, $r^p \xrightarrow{(w')^p} t^p$ and $t^p \in Fin$, then $s \in Fin$.

($AC3$) Whenever $s \in C$ and $a \in \Sigma$, if for all $p \in \text{dom}(a)$ there exist $r^p, t^p, x^p \in C$ and words $w^p, (w')^p \in (\Sigma \setminus \Sigma_p)^*$, such that $r^p \xrightarrow{w^p} s$ and $r^p \xrightarrow{(w')^p} t^p \xrightarrow{a} x^p$, then there exists $t' \in C$ with $s \xrightarrow{a} t'$.

The first *automaton condition* (AC1) has been defined earlier, while the two others are new. Our main theorem below shows that these local syntactical conditions have a global semantical implication.

To illustrate (LC3) and (AC3), consider the language L_3 in Section 2 (Figure 2(c)). We observe that L_3 does not meet (LC3) as $w = ab \in \text{pref}(L_3)$, $\text{dom}(c) = \{p, q\}$, $ab'c \in \text{pref}(L_3)$

with $\text{view}_p(ab') = [a] = \text{view}_p(w)$ and $ad'bc \in \text{pref}(L_3)$ with $\text{view}_q(a'b) = [b] = \text{view}_q(w)$ but $wc \notin \text{pref}(L_3)$. Further, if A_3 is a deterministic automaton with $L(A_3) = L_3$, denoting by s_w the state reached after w, we consider state s_{ab} and action c. Then, letting $r^p = s_a, w^p = b', t^p = s_{ab}$ and $r^q = s_b, w^q = a', t^q = s_{a'b}$ it follows that A_3 does not satisfy (AC3).

▶ **Theorem 11.** *Let L be a regular language. Then, the following are equivalent:*

1. *There is a (sequential, finite) deterministic diamond automaton $A = (C, \to, \{s_0\}, Fin)$ satisfying (AC1,AC2,AC3), with $\mathcal{L}(A) = L$, such that every state is reachable from s_0 and every state can reach Fin.*
2. *L is closed under commutation and satisfies (LC1,LC2,LC3).*
3. *There exists a realistic asynchronous automaton AA with $\mathcal{L}(AA) = L$.*

The construction of a realistic asynchronous automaton first builds a deterministic asynchronous automaton by applying the algorithm from [8]. Then, a realistic asynchronous automaton is obtained by following the transformation described in the next section, which does not add any state or transition to [8] (though it may result in the removal of some states). For complexity issues, we expect that L is given by a deterministic diamond automaton A satisfying (AC1,AC2,AC3). Indeed, checking that A fulfills (AC1,AC2,AC3) is doable in polynomial time (see section 4).

3.2 Proof of Theorem 11

Theorem 11 is shown by proving (1 $\implies$ 2), then (2 $\implies$ 3), and last (3 $\implies$ 1). In this short version, we only show (2 $\implies$ 3): if L is closed under commutation and satisfies (LC1,LC2,LC3), then there exists a realistic AA with $\mathcal{L}(AA) = L$.

Our basic strategy is to use Theorem 8 (part 1.) to construct a *deterministic* AA from a given language L and then refine this AA to obtain a *realistic* AA which accepts the same language. For this, we will use as our template the recent construction from [8], and hence we begin by stating the relevant result and a definition that we need from this paper.

▶ **Definition 12** ([8])**.** We call a deterministic asynchronous automaton $AA = ((S_p)_{p\in\mathcal{P}}, \Delta, \{s_0\}, Fin)$ *locally rejecting* if for every process p, there is a set of states $R_p \subseteq S_p$ such that for each word w: $\text{view}_p(w) \notin \text{pref}(\mathcal{L}(AA))$ iff the p-local state reached by AA on w is in R_p.

Notice that if AA reaches R_p on a word w, then it does so on every extension of w, i.e., every word w' such that w is a prefix of w'. Obviously, no reachable global final state of AA has a (projected) component in R_p, which justifies why the states in R_p are called rejecting. Any Zielonka construction gives a naturally locally rejecting asynchronous automaton. In particular:

▶ **Theorem 13** ([8])**.** *Let A be a deterministic diamond automaton over alphabet (Σ, dom). We can construct a deterministic locally rejecting asynchronous automaton AA with at most $|A|^{|\mathcal{P}|^2} \cdot 2^{2|\mathcal{P}|^4}$ states such that $\mathcal{L}(A) = \mathcal{L}(AA)$.*

Now we can prove our result as follows. Given a regular language L closed by commutation under (Σ, dom), we first build its minimal deterministic automaton A. It is then easy to check that A has the diamond property [7]. Now, we apply Theorem 13 to obtain a deterministic asynchronous automaton AA such that $\mathcal{L}(AA) = \mathcal{L}(A) = L$. Of course, AA may still have deadends (or global final states or not be locally enabled). Henceforth, for $s \xrightarrow{w} t$ with $t = (t_p)_{p\in\mathcal{P}}$, we will denote the (local) state t_p by $\delta^p_w(s)$. Notice that as the asynchronous automaton is deterministic, $\delta^p_w(s)$ is unique (if it exists) for each p, w, s.

First, we show that deadends can be avoided by using the locally rejecting property of AA. We remove all states of R_p from $AA = ((S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, \{s_0\}, Fin)$. That is, we define the asynchronous automaton $AA' = ((S'_p)_{p\in\mathcal{P}}, (\Delta'_a)_{a\in\Sigma}, \{s_0\}, Fin')$ with $S'_p = S_p \setminus R_p$ for all $p \in \mathcal{P}$, and $\Delta'_a = \Delta_a \cap \prod_{p\in\mathrm{dom}(a)} S'_p \times \prod_{p\in\mathrm{dom}(a)} S'_p$ for all $a \in \Sigma$, $Fin' = Fin \setminus R$, where $R = \{(s_p)_{p\in\mathcal{P}} \in \prod_{p\in\mathcal{P}} S_p \mid \exists q, s_q \in R_q\}$. We assume for convenience that $s_0 \notin R$ (else $L = \emptyset$ is trivial to deal with).

▶ **Lemma 14.** *AA' is deadend-free and $\mathcal{L}(AA') = \mathcal{L}(AA) = L$.*

Now, AA' may still not be realistic due to final states that are global. To obtain local final states, we define $Fin_p = \{\delta^p_w(s_0) \in S_p \mid w \in L\}$ for all $p \in \mathcal{P}$ and let $Fin'' = \prod_{p\in\mathcal{P}} Fin_p$. Note that Fin_p can be computed in time $O(|\mathcal{P}| \cdot |A|)$. Thus, we obtain a new asynchronous automaton $AA'' = ((S'_p)_{p\in\mathcal{P}}, (\Delta'_a)_{a\in\Sigma}, \{s_0\}, Fin'')$, differing from AA' only in its final states.

▶ **Lemma 15.** *AA'' is a realistic asynchronous automaton such that $\mathcal{L}(AA'') = \mathcal{L}(AA') = L$.*

Proof. By definition, AA'' is locally accepting, and it is deterministic since AA' and AA were deterministic. Also as $Fin' \subseteq Fin''$, setting the final states to be Fin'' does not add a deadend. Next, we show that $\mathcal{L}(AA'') = \mathcal{L}(AA') = L$. Take a word $w \in \mathcal{L}(AA'')$. Hence $\delta^p_w(s_0) \in Fin_p$ for all p. By definition of Fin_p, for all p there exists $v_p \in \mathcal{L}(AA') = L$ with $\delta^p_w(s_0) = \delta^p_{v_p}(s_0)$. We want to use (LC2) to conclude, but so far, there is no reason that $\mathrm{view}_p(v_p) = \mathrm{view}_p(w)$ for any p. We will thus build $v'_p \in L$ such that $\mathrm{view}_p(v'_p) = \mathrm{view}_p(w)$ for every p. Let $p \in \mathcal{P}$. It suffices to decompose $[v_p] = \mathrm{view}_p(v_p)\,[y_p]$. We then set $v'_p = \mathrm{view}_p(w)\,y_p$ and so $\mathrm{view}_p(v'_p) = \mathrm{view}_p(w)$ for all p. To obtain that $v'_p \in L$, we use a property of the Zielonka's construction from a deterministic automaton A: for all words w, w' such that $\delta^p_w(s_0) = \delta^p_{w'}(s_0)$, the state of A reached from the initial state after reading $\mathrm{view}_p(w)$ is the same as the state reached after reading $\mathrm{view}_p(w')$ (in other words, the p-state maintains the information about the state of A reached by the p-view of the executed trace). Now, let s be the state of the minimal deterministic automaton A for L reached after reading $\mathrm{view}_p(v'_p) = \mathrm{view}_p(w)$. This is also the state reached after reading $\mathrm{view}_p(v_p)$ because $\delta^p_w(s_0) = \delta^p_{v_p}(s_0)$ and by the property above. Reading y_p from s thus leads to a final state of A, as $\mathrm{view}_p(v_p)y_p \in L$ and the automaton is deterministic. Thus $v'_p = \mathrm{view}_p(w_p)y_p \in L$ too. Applying (LC2), we get $w \in L = \mathcal{L}(AA')$, and thus $L = \mathcal{L}(AA') = \mathcal{L}(AA'')$. □

3.3 Corollaries

Analyzing the proofs, one can find that in many (but not *all*) cases, there is an automaton for L satisfying (ACi) as soon as L is (LCi), for $i = 1, 2, 3$. Now, one may consider the cases where all states are final (see [21]), in which case (LC2) and (AC2) are not useful. Removing (LC2) and (AC2) from Theorem 11, we obtain:

▶ **Corollary 16.** *Let L be a regular language. Then, the following are equivalent:*

1. *There exists a deterministic diamond automaton $A = (C, \to, \{s_0\}, Fin)$ with $\mathcal{L}(A) = L$, every state is reachable from s_0 and can reach Fin, and satisfying (AC1) and (AC3).*
2. *L is closed under commutation and satisfies (LC1) and (LC3).*
3. *There exists a deterministic, deadend-free and locally enabled asynchronous automaton AA with $\mathcal{L}(AA) = L$.*

The following results are useful for testing if a given asynchronous automaton is realistic, that is, for testing if each of the conditions (LC1),(LC2),(LC3) holds (see next section). The next corollary is slightly more powerful than what we proved earlier, as it states that we can choose A to be the minimal automaton. This will serve as the basis to the test for (LC1):

▶ **Corollary 17.** *Let L be a regular language. Then, the following are equivalent:*
1. *The minimal deterministic diamond automaton A of L satisfies (AC1).*
2. *L is closed under commutation and satisfies (LC1).*
3. *There exists a deterministic, deadend-free AA with $\mathcal{L}(AA) = L$.*

Finally, *both* (AC1) and (AC2) are used to prove (LC2). However, if deadends are allowed, one can instead use a complete sequential automaton A. We recall that A is *complete* if for any word $w \in \Sigma^*$ and every $s_0 \in In$, $s_0 \xrightarrow{w} s$ is defined. The following corollary is helpful for implementing supervisors for the mutual exclusion problem that we will present in the experimentation section.

▶ **Corollary 18.** *Let L be a regular language. Then, the following are equivalent:*
1. *There is a deterministic* complete *diamond automaton A s.t. $\mathcal{L}(A) = L$ satisfying (AC2).*
2. *L is closed under commutation and satisfies (LC2).*
3. *There exists a deterministic, (locally enabled) asynchronous automaton AA with local final states and $\mathcal{L}(AA) = L$.*

Another corollary of Theorem 11 is that languages implementable by realistic asynchronous automata are closed by intersection, which can be shown using the syntactical characterization. However, they are not closed by union as $L = \{a, b\}$ cannot be implemented by any deadend-free asynchronous automaton, while both $\{a\}$ and $\{b\}$ can be implemented by realistic ones. Hence, they are also not closed under complementation.

4 Testing for Realistic Asynchronous Automata

We now explain how to check each property (LCi) and (ACi) for all $i = 1, 2, 3$.

Testing automata restrictions (ACi): Let A be an automaton, possibly non deterministic. To test (AC1), for each state s we need to check if it has a pair of outgoing transitions on actions that are independent, and if so, test for the existence of a common state that can be reached, giving a complexity quadratic in the number of states and transitions of A.

To test (AC2), we perform one graph search (e.g. DFS) from each state $s \notin Fin$ and for each process $p \in \mathcal{P}$ to return set R_p^s of states r with $r \xrightarrow{w'_p} s$ for some $w'_p \in (\Sigma \setminus \Sigma_p)^*$. We then perform another graph search from R_p^s to compute the set T_p^s of *final* states t such that $r \xrightarrow{w_p} t$, for some $r \in R_p^s$ and $w_p \in (\Sigma \setminus \Sigma_p)^*$. Now A does not satisfy (AC2) iff $\exists s, \forall p, T_p^s \neq \emptyset$. Hence, this takes time $O(|\mathcal{P}| \cdot |A|^2)$. The test of (AC3) is similar, with the same complexity.

Testing language restrictions (LCi): We now describe how to test language restrictions. We assume that the language L to be tested is given as an automaton (possibly non deterministic). First, using Corollary 17, one has a simple way to test for (LC1): compute the minimal deterministic automaton A with $\mathcal{L}(A) = L$, and test (AC1) using the polynomial procedure given above at the beginning of the section. This gives a PSPACE algorithm. The complexity is polynomial if the starting automaton is deterministic.

In order to test for (LC2), we use Corollary 18. Indeed, we build the asynchronous automaton AA from A as if $\mathcal{L}(A)$ satisfies (LC2). This can only add executions to the language, as final states are possibly added. Then we test whether $\mathcal{L}(AA) \subseteq \mathcal{L}(A)$. If the inclusion holds, then A satisfies (LC2), else A does not satisfy (LC2). This gives a PSPACE algorithm. If $\mathcal{P}$ is not part of the input and A is deterministic, then it is polynomial time. Notice that one cannot resort, as in the case of (LC1), to using the minimal automaton associated to L. This minimal automaton may not necessarily satisfy (AC2), even if L satisfies (LC2).

For instance, consider the language $L_4 = \{\epsilon, a_1, b_1, a_1b_1, b_1a_1\} \cup \{a_ib_jc, b_ja_ic \mid i, j \in \{1, 2\}\}$ with $\mathrm{dom}(a_i) = p, \mathrm{dom}(b_i) = q$ for all $i, j \in \{1, 2\}$ and $\mathrm{dom}(c) = \{p, q\}$. There is a state t in the minimal automaton with $s_0 \xrightarrow{a_1} s \xrightarrow{b_2} t$ and $s_0 \xrightarrow{b_1} s' \xrightarrow{a_2} t$, with s, s' final, meaning if (AC2) holds that t is final, a contradiction. Finally, to test (LC3), we again implement L into an AA and test if $S(AA)$ satisfies (AC3). As described in the proof of Theorem 11, if $\mathcal{L}(A)$ satisfies (LC3), then $S(AA)$ satisfies (AC3). Conversely, if $S(AA)$ satisfies (AC3), the proof also shows that $\mathcal{L}(A)$ satisfies (LC3). This gives a PSPACE algorithm. The complexity is polynomial if $\mathcal{P}$ is not part of the input and A is deterministic.

Note that while the algorithms to test for (LC1),(LC2),(LC3) may be PSPACE, they are actually polynomial in the size of the asynchronous automaton AA we want to obtain. As shown below, obtaining the global state space for AA is actually feasible in a number of examples, and hence testing for (LC1), (LC2) and (LC3) is also doable in these cases.

5 Experiments

In this section, we report our experiments on the implementation of the results in this paper, based on the construction from [8], which has not been implemented before. To give a point of comparison, we also report results obtained using the only previous implementation prototype for Zielonka constructions from [21], which implements the original synthesis algorithms from [22] and the heuristic in [21].

We report below the results of several systems that are (distributively) implemented using these three algorithms: We will denote by *heuristic* the heuristic from [21], by *original* the original Zielonka's construction from [22], and by *local* and *global* two different metrics for our new implementation as described below. *heuristic* takes into account the structure of the automaton (using ideas from the theory of regions [17]) to identify small asynchronous automata before generating the whole global state space. Since such structural properties cannot be found for every regular commutation-closed language, it uses the equivalence in *original* as an upper bound. Hence, the state space produced by *heuristic* is never bigger than the one of *original*. On the other hand, *original* uses a generic construction which always produces an asynchronous automaton. In contrast to these two algorithms producing global state spaces, our implementation produces the local state space directly. Further, ours is an on-the-fly symbolic algorithm. As argued in [19], on-the-fly computation allows to implement distributed algorithms whose global state space cannot be explicitly enumerated: with 4 processes, the timestamping used in [22, 8] can give rises to 10^7 global states, and to 10^{16} global states with 5 processes. But for symbolic algorithms (e.g., the one from [8] that we implement), 5 processes means maintaining 128 bits of information, which can be updated in time polynomial in the number of bits. To produce and compare the results of all algorithms, we report global state spaces, thus limiting ourselves to less than 4 processes.

The results are compiled in the table below. The first column gives the names of the input systems, while second and third provide their number of states $|A|$ and processes $|P|$, respectively. The fourth column states the syntactical properties (ACi) of the automaton A. The next three columns give the number of *global* states produced by each of the algorithms. As noted earlier, the new prototype does not need to compute the global state space, unlike [21, 22]. The column *local* reports the total number of *local* states generated by our algorithm, which is closer to what would be used in practice (but is still larger than what is explored on-the-fly). The last row describes the properties (DF for deadend-free, LA for locally accepting, LE for locally enabled, and R for realistic) of the obtained asynchronous automaton using the new implementation, all being deterministic.

The first four systems come directly from distributed algorithms: a mutual exclusion

protocol with semaphores with 2 different distribution alphabets referred to as mutex-a and mutex-b; a simple program with 3 processes denoted simple; and a dining philosopher protocol phil. All these examples except simple, from [21], satisfy AC1,AC2,AC3.

	$\|A\|$	$\|\mathcal{P}\|$	satisfies	*heuristic*	*original*	*global*	*local*	satisfies
mutex-a	13	3	(AC1,AC2,AC3)	13	1493	271	126	(R)
mutex-b	14	4	(AC1,AC2,AC3)	14	34	22	16	(R)
simple	3	3	(AC1,AC2)	5	12	12	9	(DF+LA)
phil	5	4	(AC1,AC2,AC3)	5	70	71	60	(R)
prop2	6	2	(AC2)	188	188	36	21	(LA+LE)
prop3	11	3	(AC2)	639	639	240	92	(LA+LE)
L4	8	2	(AC1,AC3)+(LC2)	n/a	10	10	5	(R)

For these first 4 systems, the new prototype gives an implementation with lesser states than *original* (up to 10 times), although not as good as *heuristic*. Adapting ideas from *heuristic* [21] might reduce the size of the produced implementation. The two systems propN correspond to a distributed supervisor which detects whether a critical section has been accessed by 2 processes in parallel among N processes. On each process, it observes entry and exit of the critical section and synchronization between processes, and detects if a process which enters the critical section has been informed that other processes have exited it. This supervisor works on any possible (correct or not) mutual exclusion protocol, and detects on-the-fly whether the critical section was accessed by 2 processes concurrently. On this example, *heuristic* does not do better than *original*. The number of local states is around 8 times smaller, while global states are around 4 times smaller than previous implementations. As (LC1) does not hold, a realistic implementation is not possible here.

Notice that *heuristic* is guaranteed to return correct results only when all states are final [21], which is the case for the first 6 systems. The last system we experiment on is the minimal automaton L4 for language L_4 from the previous section (L4 does not satisfy (AC2), although L_4 satisfies (LC2)). Some states of this automaton are not final and the implementation created by *heuristic* is incorrect: its language is strictly larger than L_4. On the other hand, implementations produced by *original* and the new prototype accept exactly L_4. Details on the experiments can be found online at: `http://is.gd/fsttcs13_benchmark`.

6 Related Work and Conclusion

In this paper, we have provided syntactical and semantical characterizations of languages corresponding to several variants of *realistic* asynchronous automata. We designed algorithms to obtain the distributed implementation, test for the different characterizations and showed their experimental effectiveness. Our results subsume past results and answer several open questions. Corollary 17 subsumes what was claimed in [17] and proved in [21] (Theorem 2) in the subcase where the language is prefix closed. It is also worth mentioning that [1] had introduced the notion of causal closure for Message Sequence Graphs, which are a distributed model using message passing for communication. Our notion of causal closure is directly adapted from theirs. However, unlike in Corollary 18, only one direction was proved for their model. Also, they lack the syntactical characterization using (AC2) which holds by Theorem 11. Also, Corollary 18 answers an open question in the conclusion of [2].

As future work, it would be interesting to consider alternative ways of inputing the language, e.g., by giving a set of representatives to represent the language. This would avoid starting from a large automaton, and may lead to a smaller distributed implementation.

Acknowledgments. This work was supported by Romanian NASR project PN-II-ID-PCE-2011-3-0688 (MuVeT), INRIA Associated team DISTOL and DST/INSPIRE faculty award [IFA12-MA-17].

References

1 B. Adsul, M. Mukund, K. Narayan Kumar and V. Narayanan. Causal closure for MSC languages. Proc. of *FSTTCS'05*, LNCS 3821, pp. 335-347, 2005.

2 N. Baudru. Compositional synthesis of asynchronous automata *Theor. Comput. Sci.*, 412(29):3701-3716, 2011.

3 B. Bollig, J.-P. Katoen, C. Kern and M. Leucker. Learning Communicating Automata from MSCs. *IEEE Trans. Software Eng.* 36(3):390-408, 2010.

4 N. Baudru and R. Morin. Unfolding Synthesis of Asynchronous Automata. Proc. of *CSR'06*, LNCS 3967, pp. 46-57, 2006.

5 N. Baudru and R. Morin. Synthesis of Safe Message-Passing Systems. Proc. of *FSTTCS'07*, LNCS 4855, pp. 277-289, 2007.

6 R. Cori, Y. Métivier and W. Zielonka. Asynchronous Mappings and Asynchronous Cellular Automata. *Inf. and Comput.*, 106(2):159-202, 1993.

7 V. Diekert and G. Rozenberg, editors. *The Book of Traces.* In particular, Chapter 8 by V. Diekert and A. Muscholl. World Scientific, Singapore, 1995.

8 B. Genest, H. Gimbert, A. Muscholl and I. Walukiewicz. Optimal Zielonka-like Construction. Proc. of *ICALP'10*, LNCS 6199, pp. 52-63, 2010.

9 B. Genest, H. Gimbert, A. Muscholl, I. Walukiewicz. Asynchronous Games over Tree Architectures. Proc. of *ICALP'13*, LNCS 7966, pp. 275-286, 2013.

10 B. Genest, D. Kuske and A. Muscholl. A Kleene Theorem and Model Checking for a Class of Communicating Automata. *Inf. and Comput.*204(6):920-956, 2006.

11 P. Gastin, B. Lerman and M. Zeitoun. Distributed Games with Causal Memory Are Decidable for Series-Parallel Systems. Proc. of *FSTTCS'04*, LNCS 3328, 2004.

12 B. Genest and A. Muscholl. Constructing Exponential-size Deterministic Zielonka Automata. Proc. of *ICALP'06*, LNCS 4051, pp. 565-576, 2006.

13 J. G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni and P. S. Thiagarajan. A Theory of Regular MSC Languages. In *Inf. and Comput.* 202(1):1-38, 2005.

14 N. Klarlund, M. Mukund and M. Sohoni. Determinizing Asynchronous Automata. Proc. of *ICALP'94*, LNCS 820, pp. 130-141, 1994.

15 D. Kuske. Regular sets of infinite message sequence charts. In *Inf. and Comput.*, 187(1):80-109, 2003.

16 A. Mazurkiewicz. Concurrent program schemes and their interpretation. Technical report, DAIMI Report PB-78, Aarhus University, 1977.

17 M. Mukund. From global specification to local implementations. In *Synthesis and Control of Discrete Event Systems*, Kluwer, pp. 19-34, 2002.

18 M. Mukund, K. Narayan Kumar and M. Sohoni. Synthesizing Distributed Finite-State Systems from MSCs. *TCS* 290(1):221-239, 2003.

19 M. Mukund and M. Sohoni. Keeping Track of the Latest Gossip in a Distributed System. In *Distr. Computing* 10(3):137-148, 1997.

20 G. Pighizzini. Synthesis of Nondeterministic Asynchronous Automata. In *Algebra, Logic and Applications* 5, pp. 109-126, 1993.

21 A. Stefanescu, J. Esparza, and A. Muscholl. Synthesis of distributed algorithms using asynchronous automata. Proc. of *CONCUR'03*, LNCS 2761, pp. 27-41, 2003.

22 W. Zielonka. Notes on finite asynchronous automata. In *R.A.I.R.O. - Informatique Théorique et Applications*, 21:99-135, 1987.

23 W. Zielonka. Safe Executions of Recognizable Trace Languages by Asynchronous Automata. *Proc. of Logic at Botik 1989*, LNCS 363, pp. 278-289, 1989.

Computation of Summaries Using Net Unfoldings

Javier Esparza[1], Loïg Jezequel[2], and Stefan Schwoon[3]

1 **Institut für Informatik, Technische Universität München, Germany**
2 **ENS Cachan Bretagne, Rennes, France**
3 **LSV, ENS Cachan & CNRS, INRIA Saclay, France**

Abstract

We study the following *summarization problem*: given a parallel composition $\mathbf{A} = \mathcal{A}_1 \parallel \ldots \parallel \mathcal{A}_n$ of labelled transition systems communicating with the environment through a distinguished component $\mathcal{A}_i$, efficiently compute a *summary* $\mathcal{S}_i$ such that $\mathbf{E} \parallel \mathbf{A}$ and $\mathbf{E} \parallel \mathcal{S}_i$ are trace-equivalent for every environment $\mathbf{E}$. While $\mathcal{S}_i$ can be computed using elementary automata theory, the resulting algorithm suffers from the state-explosion problem. We present a new, simple but subtle algorithm based on net unfoldings, a partial-order semantics, give some experimental results using an implementation on top of Mole, and show that our algorithm can handle *divergences* and compute *weighted summaries* with minor modifications.

1998 ACM Subject Classification D.2.2 Design Tools and Techniques, F.1.2 Modes of Computation

Keywords and phrases Net unfoldings, Concurrent systems, Petri nets

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.225

1 Introduction

We address a fundamental problem in automatic compositional verification. Consider a parallel composition $\mathbf{A} = \mathcal{A}_1 \parallel \ldots \parallel \mathcal{A}_n$ of processes, modelled as labelled transition systems, which is itself part of a larger system $\mathbf{E} \parallel \mathbf{A}$ for some environment $\mathbf{E}$. Assume that $\mathcal{A}_i$ is the interface of $\mathbf{A}$ with the environment, i.e., $\mathbf{A}$ communicates with the outer world only through actions of $\mathcal{A}_i$. The task consists in computing a new interface $\mathcal{S}_i$ with the same set of actions as $\mathcal{A}_i$ such that $\mathbf{E} \parallel \mathbf{A}$ and $\mathbf{E} \parallel \mathcal{S}_i$ have the same behaviour. In other words, the environment E cannot distinguish between $\mathbf{A}$ and $\mathcal{S}_i$. Since $\mathcal{S}_i$ usually has a much smaller state space than $\mathbf{A}$ (making $\mathbf{E} \parallel \mathbf{A}$ easier to analyse) we call it a *summary*.

We study the problem in a CSP-like setting [13]: parallel composition is by rendez-vous, and the behaviour of a transition system is given by its trace semantics.

It is easy to compute $\mathcal{S}_i$ using elementary automata theory: we first compute the transition system of $\mathbf{A}$, whose states are tuples $(s_1, \ldots, s_n)$, where s_i is a state of $\mathcal{A}_i$. Then we hide all actions except those of the interface, i.e., we replace them by ε-transitions (τ-transitions in CSP terminology). We can then eliminate all ε-transitions using standard algorithms, and, if desired, compute the minimal summary by applying e.g. Hopcroft's algorithm. The problem of this approach is the state-space explosion: the number of states of $\mathbf{A}$ can grow exponentially in the number of sequential components. While this is unavoidable in the worst case (deciding whether $\mathcal{S}_i$ has an empty set of traces is a PSPACE-complete problem, and the minimal summary $\mathcal{S}_i$ may be exponentially larger than $\mathcal{A}_1, \ldots, \mathcal{A}_n$ in the worst case, see e.g. [11]) the combinatorial explosion happens already in trivial cases: if the components $\mathcal{A}_1, \ldots, \mathcal{A}_n$ do not communicate at all, we can obviously take $\mathcal{S}_i = \mathcal{A}_i$, but the algorithm we have just described will need exponential time and space.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 225–236

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

We present a technique to palliate this problem based on net unfoldings (see e.g. [4]). Net unfoldings are a partial-order semantics for concurrent systems, closely related to event structures [22], that provides very compact representations of the state space for systems with a high degree of concurrency. Intuitively, an unfolding is the extension to parallel compositions of the notion of unfolding a transition system into a tree. The unfolding is usually infinite. We show how to algorithmically construct a finite prefix of it from which the summary can be easily extracted. The algorithm can be easily implemented re-using many components of existing unfolders like PUNF and MOLE. However, its correctness proof is surprisingly subtle. This proof is the main contribution of the paper. However, we also evaluate the algorithm on some classical benchmarks [2]. We then show that – with minor modifications – the algorithm can be extended so that the summary obtained contains information about the possible divergences, that is whether or not after a given finite trace of the interface $\mathcal{A}_i$ it is possible that $\mathbf{A}$ evolves silently forever (i.e. without using any action of $\mathcal{A}_i$). And finally, we show how to extend the algorithm to deal with weighted systems: $\mathcal{S}_i$ then also gives for each of its finite traces the minimum cost in $\mathbf{A}$ to execute this trace.

Related work. The summarization problem has been extensively studied in an interleaving setting (see e.g. [10, 21, 23]), in which one first constructs the transition system of $\mathbf{A}$ and then reduces it. We study it in a partial-order setting.

Net unfoldings, and in general partial-order semantics, have been used to solve many analysis problems: deadlock [18, 15], reachability and model-checking questions [6, 3, 14, 4, 1], diagnosis [7], and other specific applications [17, 12]. To the best of our knowledge we are the first to explicitly study the summarization problem.

Our problem can be solved with the help of Zielonka's algorithm [24, 19, 9], which yields an asynchronous automaton trace-equivalent to $\mathbf{A}$. The projection of this automaton onto the alphabet of $\mathcal{A}_i$ yields a summary $\mathcal{S}_i$. However, Zielonka's algorithm is notoriously complicated and, contrary to our algorithm, requires to store much additional information for each event [19]. In [8], the complete tuple $\mathcal{S}_1, \ldots, \mathcal{S}_n$ is computed – possibly in a weighted context – with an iterative message-passing algorithm that transfers information between components until a fixed point is reached. However, termination is only guaranteed when the communication graph is acyclic.

2 Preliminaries

Transition systems. A *labelled transition system* (LTS) is a tuple $\mathcal{A} = (\Sigma, S, T, \lambda, s^0)$ where Σ is a set of *actions*, S is a set of *states*, $T \subseteq S \times S$ is a set of *transitions*, $\lambda\colon T \to \Sigma$ is a *labelling function*, and $s^0 \in S$ is an *initial state*. An a-transition is a transition labelled by a. We use this definition – excluding the possibility to have two transitions with different labels between the same pair of states – for simplicity. However, the results presented in this paper would still hold if this possibility was not excluded. A (finite or infinite) action sequence $\sigma = a_1 a_2 a_3 \ldots \in \Sigma^* \cup \Sigma^\omega$ is a *trace* of $\mathcal{A}$ if there is a (finite or infinite) sequence $s_0 s_1 s_2 \ldots$ of states such that $s_0 = s^0$, $t_i = (s_{i-1}, s_i) \in T$ and $\lambda(t_i) = a_i$ for every i. The path $s_0 s_1 s_2 \ldots$ is a *realization* of σ. The set of traces of $\mathcal{A}$ is denoted by $Tr(\mathcal{A})$. Figure 1 shows (on its left) three transition systems.

Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be LTSs where $\mathcal{A}_i = (\Sigma_i, S_i, T_i, \lambda_i, s_i^0)$. The *parallel composition* $\mathbf{A} = \mathcal{A}_1 \parallel \ldots \parallel \mathcal{A}_n$ is the LTS defined as follows. The set of actions is $\mathbf{\Sigma} = \Sigma_1 \cup \ldots \cup \Sigma_n$. The states, called *global states*, are the tuples $\mathbf{s} = (s_1, \ldots, s_n)$ such that $s_i \in S_i$ for every $i \in \{1..n\}$. The *initial global state* is $\mathbf{s}^0 = (s_1^0, \ldots, s_n^0)$. The transitions, called *global transitions*, are the tuples $\mathbf{t} = (t_1, \ldots, t_n) \in (T_1 \cup \{\star\}) \times \cdots \times (T_n \cup \{\star\}) \setminus \{(\star, \ldots, \star)\}$ such that there is

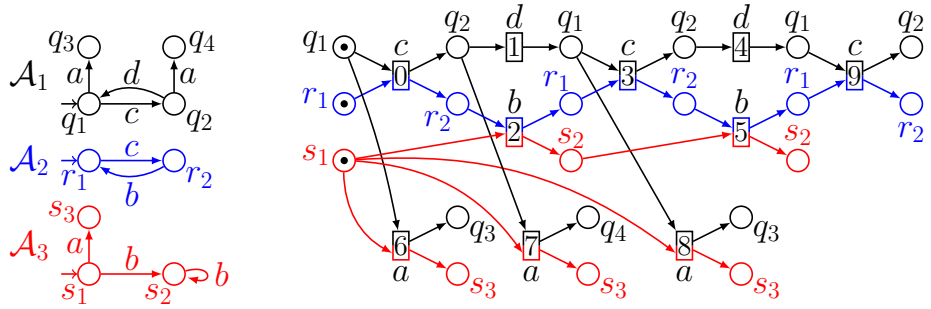


Figure 1 Three labeled transition systems (left) and a branching process (right).

an action $a \in \mathbf{\Sigma}$ satisfying for every $i \in \{1..n\}$: if $a \in \Sigma_i$, then t_i is an a-transition of T_i, otherwise $t_i = \star$; the label of $\mathbf{t}$ is the action a. If $t_i \neq \star$ we say that $\mathcal{A}_i$ *participates* in $\mathbf{t}$. It is easy to see that $\sigma \in \mathbf{\Sigma}^* \cup \mathbf{\Sigma}^\omega$ is a trace of $\mathbf{A}$ iff for every $i \in \{1..n\}$ the projection of σ on Σ_i, denoted by $\sigma_{|\Sigma_i}$, is a trace of $\mathcal{A}_i$.

Petri nets. A *labelled net* is a tuple $(\Sigma, P, T, F, \lambda)$ where Σ is a set of *actions*, P and T are disjoint sets of *places* and *transitions* (jointly called *nodes*), $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs*, and $\lambda\colon P \cup T \to \Sigma$ is a *labelling function*. For $x \in P \cup T$ we denote by ${}^\bullet x = \{\, y \mid (y,x) \in F \,\}$ and $x^\bullet = \{\, y \mid (x,y) \in F \,\}$ the sets of *inputs* and *outputs* of x, respectively. A set M of places is called a *marking*. A *labelled Petri net* is a tuple $\mathcal{N} = (\Sigma, P, T, F, \lambda, M_0)$ where $(\Sigma, P, T, F, \lambda)$ is a labelled net and $M_0 \subseteq P$ is the *initial marking*. A marking M *enables* a transition $t \in T$ if ${}^\bullet t \subseteq M$. In this case t can *occur* or *fire*, leading to the new marking $M' = (M \setminus {}^\bullet t) \cup t^\bullet$. An *occurrence sequence* is a (finite or infinite) sequence of transitions that can occur from M_0 in the order specified by the sequence. A *trace* is the sequence of labels of an occurrence sequence. The set of traces of $\mathcal{N}$ is denoted by $Tr(\mathcal{N})$.

Branching processes. The finite *branching processes* of $\mathbf{A} = \mathcal{A}_1 \parallel \ldots \parallel \mathcal{A}_n$ are labelled Petri nets whose places are labelled with states of $\mathcal{A}_1, \ldots, \mathcal{A}_n$, and whose transitions are labelled with global transitions of $\mathbf{A}$. Following tradition, we call the places and transitions of these nets *conditions* and *events*, respectively. (Since global transitions are labelled with actions, each event is also implicitly labelled with an action.) We say that a marking M of these nets *enables* a global transition $\mathbf{t}$ of $\mathbf{A}$ if for every state $s \in {}^\bullet\mathbf{t}$ some condition of M is labelled by s. The set of *finite branching processes* of $\mathbf{A}$ is defined inductively as follows:

1. A labelled Petri net with conditions $b_1^0, ..., b_n^0$ labelled by $s_1^0, \ldots, s_n^0$, no events, and with initial marking $\{b_1^0, ..., b_n^0\}$, is a branching process of $\mathbf{A}$.
2. Let $\mathcal{N}$ be a branching process of $\mathbf{A}$ such that some reachable marking of $\mathcal{N}$ enables some global transition $\mathbf{t}$. Let M be the subset of conditions of the marking labelled by ${}^\bullet\mathbf{t}$. If $\mathcal{N}$ has no event labelled by $\mathbf{t}$ with M as input set, then the Petri net obtained by adding to $\mathcal{N}$: a new event e, labelled by $\mathbf{t}$; a new condition for every state s of $\mathbf{t}^\bullet$, labelled by s; new arcs leading from each condition of M to e, and from e to each of the new conditions, is also a branching process of $\mathbf{A}$.

Figure 1 shows on the right a branching process of the parallel composition of the LTSs on the left. Events are labelled with their corresponding actions.

The set of all branching processes of a net, finite and infinite, is defined by closing the finite branching processes under countable unions (after a suitable renaming of conditions and events) [4]. In particular, the union of all finite branching processes yields the *unfolding*

of the net, which intuitively corresponds to the result of exhaustively adding all extensions in the definition above.

A *trace* of a branching process $\mathcal{N}$ is the sequence of action labels of an occurrence sequence of events of $\mathcal{N}$. In Figure 1, firing the events on the top half of the process yields any of the traces *cbdcbd*, *cdbcbd*, *cbdcdb*, or *cdbcdb*. The sets of traces of $\mathbf{A}$ and of its unfolding coincide.

Let x, y be nodes of a branching process. We say that x is a *causal predecessor* of y, denoted by $x < y$, if there is a non-empty path of arcs from x to y; further, $x \leq y$ denotes that either $x < y$ or $x = y$. If $x \leq y$ or $x \geq y$, then x and y are *causally related*. We say that x and y are *in conflict*, denoted by $x \# y$, if there is a condition z (different from x and y) from which one can reach both x and y, exiting z by different arcs. Finally, x and y are *concurrent* if they are neither causally related nor in conflict.

A set of events E is a *configuration* if it is *causally closed* (that is, if $e \in E$ and $e' < e$ then $e' \in E$) and *conflict-free* (that is, for every $e, e' \in E$, e and e' are not in conflict). The *past* of an event e, denoted by $[e]$, is the set of events e' such that $e' \leq e$ (so it is a configuration). For any event e, we denote by $M(e)$ the unique marking reached by any occurrence sequence that fires exactly the events of $[e]$. Notice that, for each component $\mathcal{A}_i$ of $\mathbf{A}$, $M(e)$ contains exactly one condition labelled by a state of $\mathcal{A}_i$. We denote this condition by $M(e)_i$. We write $\mathbf{St}(e) = \{\, \lambda(x) \mid x \in M(e) \,\}$ and call it the *global state reached by* e.

3 The Summary Problem

Let $\mathbf{A} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$ be a parallel composition with a distinguished component $\mathcal{A}_i$, called the *interface*. An *environment* of $\mathbf{A}$ is any LTS $\mathbf{E}$ (possibly a parallel composition) that only communicates with $\mathbf{A}$ through the interface, i.e, $\Sigma_{\mathbf{E}} \cap (\Sigma_1 \cup \ldots \cup \Sigma_n) = \Sigma_{\mathbf{E}} \cap \Sigma_i$. We wish to compute a *summary* $\mathcal{S}_i$, i.e., an LTS with the same actions as $\mathcal{A}_i$ such that $Tr(\mathbf{E} \parallel \mathbf{A})|_{\Sigma_{\mathbf{E}}} = Tr(\mathbf{E} \parallel \mathcal{S}_i)|_{\Sigma_{\mathbf{E}}}$ for every environment $\mathbf{E}$, where $X|_\Sigma$ denotes the projection of the traces of X onto Σ. It is well known (and follows easily from the definitions) that this holds iff $Tr(\mathcal{S}_i) = Tr(\mathbf{A})|_{\Sigma_i}$ [13]. We therefore address the following problem:

▶ **Definition 1** (Summary problem). Given LTSs $\mathcal{A}_1, \ldots, \mathcal{A}_n$ with interface $\mathcal{A}_i$, compute an LTS $\mathcal{S}_i$ satisfying $Tr(\mathcal{S}_i) = Tr(\mathbf{A})|_{\Sigma_i}$, where $\mathbf{A} = \mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n$.

The problem can be solved by computing the LTS $\mathbf{A}$, but the size of $\mathbf{A}$ can be exponential in $\mathcal{A}_1, \ldots, \mathcal{A}_n$. So we investigate an unfolding approach.

The *interface projection* $\mathcal{N}_i$ of a branching process $\mathcal{N}$ of $\mathbf{A}$ onto $\mathcal{A}_i$ is the following labelled subnet of $\mathcal{N}$: (1) the conditions of $\mathcal{N}_i$ are the conditions of $\mathcal{N}$ with labels in S_i; (2) the events of $\mathcal{N}_i$ are the events of $\mathcal{N}$ where $\mathcal{A}_i$ participates; (3) (x, y) is an arc of $\mathcal{N}_i$ iff it is an arc of $\mathcal{N}$ and (x, y) are nodes of $\mathcal{N}_i$. Obviously, every event of $\mathcal{N}_i$ has exactly one input and one output condition, and $\mathcal{N}_i$ can therefore be seen as an LTS; thus, we sometimes speak of the LTS $\mathcal{N}_i$. The interface projection $\mathcal{N}_1$ for the branching process of Figure 1 is the subnet given by the black conditions and their input and output events, and its LTS representation is shown in the left of Figure 2.

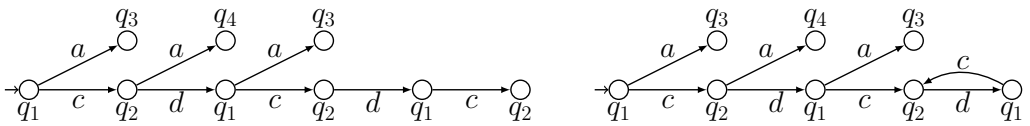


Figure 2 Projection of the branching process of Figure 1 on $\mathcal{A}_1$ (left) and a folding (right).

The projection $\mathcal{U}_i$ of the full unfolding of $\mathbf{A}$ onto $\mathcal{A}_i$ clearly satisfies $Tr(\mathcal{U}_i) = Tr(\mathbf{A})_{|\Sigma_i}$; however, $\mathcal{U}_i$ can be infinite. In the rest of the paper we show how to compute a *finite* branching process $\mathcal{N}$ and an equivalence relation $\equiv$ between the conditions of $\mathcal{N}_i$ such that the result of *folding* $\mathcal{N}_i$ into a finite LTS by merging the conditions of each equivalence class yields the desired $\mathcal{S}_i$. The *folding* of $\mathcal{N}_i$ is the LTS whose states are the equivalence classes of $\equiv$, and every transition (s, s') of $\mathcal{N}_i$ yields a transition $([s]_\equiv, [s']_\equiv)$ of the folding. Figure 2 shows on the right the result of folding the LTS on the left when the only equivalence class with more than one member is formed by the two rightmost states labelled by q_2.

We construct $\mathcal{N}$ by starting with the branching processes without events and iteratively add one event at a time. Some events are marked as *cut-offs* [4]. An event e added to $\mathcal{N}$ becomes a cut-off if $\mathcal{N}$ already contains an e', called the *companion* of e, satisfying a certain, yet to be specified *cut-off criterion*. Events with cut-offs in their past cannot be added. The algorithm terminates when no more events can be added. The equivalence relation $\equiv$ is determined by the *interface cut-offs*: the cut-offs labelled with interface actions. If an interface cut-off e has companion e', then we set $M(e)_i \equiv M(e')_i$. Algorithm 1 is pseudocode for the unfolding, where $Ext(\mathcal{N}, co)$ denotes the *possible extensions*: the events which can be added to $\mathcal{N}$ without events from the set co of cut-offs in their past.

Algorithm 1 Unfolding procedure for a product $\mathbf{A}$.

let $\mathcal{N}$ be the unique branching process of $\mathbf{A}$ without events and let $co = \emptyset$
While $Ext(\mathcal{N}, co) \neq \emptyset$ **do**
  choose e in $Ext(\mathcal{N}, co)$ and extend $\mathcal{N}$ with e
  If e is a cut-off event **then** let $co = co \cup \{e\}$
For every $e \in co$ with companion e' **do** merge $[M(e)_i]_\equiv$ and $[M(e')_i]_\equiv$

Notice that the algorithm is nondeterministic: the order in which events are added is not fixed (though it necessarily respects causal relations). We wish to find a definition of cut-offs such that the LTS $\mathcal{S}_i$ delivered by the algorithm is a correct solution to the summary problem. Several papers have addressed the problem of defining cut-offs such that the branching process delivered by the algorithm contains all global states of the system (see [4] and the references therein). In [5] we show that these approaches do not "unfold enough".

4 Two Attempts

The solution turns out to be remarkably subtle, and so we approach it in a series of steps.

4.1 First attempt

In the following we shall call events in which $\mathcal{A}_i$ participates *i-events* for short; analogously, we call *i-conditions* the conditions labelled by states of $\mathcal{A}_i$.

The simplest idea is to declare an i-event e a cut-off if the branching process already contains another i-event e' with $\mathbf{St}(e) = \mathbf{St}(e')$. Intuitively, the behaviours of the interface after the configurations $[e]$ and $[e']$ is identical, and so we only explore the future of $[e']$.

Cut-off definition 1. An event e is a cut-off event if it is an i-event and $\mathcal{N}$ contains an i-event e' such that $\mathbf{St}(e) = \mathbf{St}(e')$.

It is not difficult to show that this definition is correct for *non-divergent* systems.

▶ **Definition 2.** A parallel composition $\mathbf{A}$ with interface $\mathcal{A}_i$ is *divergent* if some infinite trace of $\mathbf{A}$ contains only finitely many occurrences of actions of Σ_i.

▶ **Theorem 3.** *Let* $\mathbf{A}$ *be non-divergent. The instance of Algorithm 1 with cut-off definition 1 terminates with a finite branching process* $\mathcal{N}$*, and the folding* $\mathcal{S}_i$ *of* $\mathcal{N}_i$ *is a summary of* $\mathbf{A}$*.*

The proof of this theorem is given in [5].

The system of Figure 1 is non-divergent. Algorithm 1 computes the branching process on the right of Figure 1. The only cut-off is event 9 with companion 3. The folding is shown in Figure 2 (right) and is a correct summary. However, cut-off definition 1 *never* works if $\mathbf{A}$ is divergent because the unfolding procedure does not terminate. Indeed, if the system has divergent traces then we can easily construct an infinite firing sequence of the unfolding such that none of the finitely many i-events in the sequence is a cut-off. Since no other events can be cut-offs, Algorithm 1 adds all events of the sequence. This occurs for instance for the system of Figure 3 with interface $\mathcal{A}_1$, where the occurrence sequence of the unfolding for the trace $i(fcd)^\omega$ contains no cut-off.

4.2 Second attempt

To ensure termination for divergent systems, we extend the definition of cut-off. For this, we define for each event e its *i-predecessor*. Intuitively, the i-predecessor of an event e is the last condition that e "knows" has been reached by the interface.

▶ **Definition 4.** The *i-predecessor* of an event e, denoted by $ip(e)$, is the condition $M(e)_i$.

Assume now that two events $e_1 < e_2$, neither of them interface event, satisfy $ip(e_1) = ip(e_2)$ and $\mathbf{St}(e_1) = \mathbf{St}(e_2)$. Then any occurrence sequence σ that executes the events of the set $[e_2] \setminus [e_1]$ leads from a marking to itself and contains no interface events. So σ can be repeated infinitely often, leading to an infinite trace with only finitely many interface actions. It is therefore plausible to mark e_2 as cut-off event, in order to avoid this infinite repetition.

Cut-off definition 2. An event e is a cut-off if
(1) e is an i-event, and $\mathcal{N}$ contains an i-event e' with $\mathbf{St}(e) = \mathbf{St}(e')$, or
(2) e is not an i-event, and some event $e' < e$ satisfies $\mathbf{St}(e) = \mathbf{St}(e')$ and $ip(e) = ip(e')$.

We give an example showing that this natural definition does not work: the algorithm always terminates but can yield a wrong result. Consider the parallel composition at the left of Figure 3, with interface $\mathcal{A}_1$. Clearly $Tr(\mathcal{A})|_{\Sigma_1} = Tr(A_1) = iab^*e$. For any strategy the algorithm generates the branching process $\mathcal{N}$ at the top right of the figure (without the dashed part). $\mathcal{N}$ has two cut-off events: the interface event 6, which is of type (1), and event 8, a non-interface event, of type (2). Event 6 has 5 as companion, with $\mathbf{St}(5) = \mathbf{St}(6) = \{q_2, r_2, s_2\}$. Event 8 has 0 as companion, with $\mathbf{St}(0) = \{q_1, r_1, s_1\} = \mathbf{St}(8)$; moreover, $0 < 8$ and $ip(0) = ip(8)$. The folding of $\mathcal{N}_1$ is shown at the bottom right of the figure. It is clearly not trace-equivalent to $\mathcal{A}_1$ because it "misses" the trace $iabe$. The dashed event at the bottom right, which would correct this, is not added by the algorithm because it is a successor of 8.

5 The Solution

Intuitively, the reason for the failure of our second attempt on the example of Figure 3 is that $\mathcal{A}_1$ can only execute $iabe$ if $\mathcal{A}_2$ and $\mathcal{A}_3$ execute $ifcd$ first. However, when the algorithm observes that the markings before and after the execution of $ifcd$ are identical, it declares 8

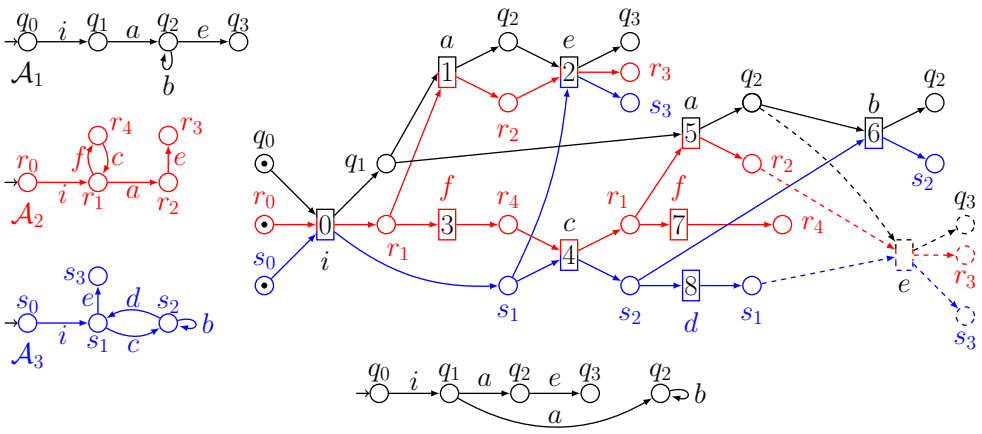


Figure 3 Cut-off definition 2 produces an incorrect result on $\mathbf{A} = \mathcal{A}_1 \parallel \mathcal{A}_2 \parallel \mathcal{A}_3$.

a cut-off event, and so it cannot "use" it to construct event e. So, on the one hand, 8 should not be a cut-off event. But, on the other hand, *some* event of the trace $i(fcd)^\omega$ must be declared cut-off, otherwise the algorithm does not terminate.

The way out of this dilemma is to introduce *cut-off candidates*. If an event is declared a cut-off candidate, the algorithm does not add any of its successors, just as with regular cut-offs. However, cut-off candidates may stop being candidates if the addition of a new event *frees* them. (So, an event is a cut-off candidate *with respect to the current branching process*.) A generic unfolding procedure using these ideas is given in Algorithm 2, where $Ext(\mathcal{N}, co, coc)$ denotes the possible extensions of $\mathcal{N}$ that do not have any event of co or coc in their past. Assuming suitable definitions of cut-off candidates and freeing, the algorithm would, in our example, declare event 8 a cut-off candidate, momentarily stop adding any of its successors, but later free event 8 when event 5 is discovered.

Algorithm 2 Unfolding procedure for a product $\mathbf{A}$.

let $\mathcal{N}$ be the unique branching process of $\mathbf{A}$ without events; let $co = \emptyset$ and $coc = \emptyset$
While $Ext(\mathcal{N}, co, coc) \neq \emptyset$ **do**
  choose e in $Ext(\mathcal{N}, co, coc)$ according to the search strategy
  If e is a cut-off event **then** let $co = co \cup \{e\}$
  Elseif e is a cut-off candidate of $\mathcal{N}$ **then** let $coc = coc \cup \{e\}$
  Else for every $e' \in coc$ **do**
    If e frees e' **then** $coc = coc \setminus \{e'\}$
  extend $\mathcal{N}$ with e
For every $e \in co$ with companion e' **do** merge $[M(e)_i]_\equiv$ and $[M(e')_i]_\equiv$

The main contribution of our paper is the definition of a correct notion of cut-off candidate for the projection problem. We shall declare event e a cut-off candidate if e is not an interface event, and $\mathcal{N}$ contains a companion $e' < e$ such that $\mathbf{St}(e') = \mathbf{St}(e)$, $ip(e) = ip(e')$, and, additionally, no interface event e'' of $\mathcal{N}$ is concurrent with e without being concurrent with e'. As long as this condition holds, the successors of e are put "on hold". In the example of Figure 3, if the algorithm first adds events 0, 3, 4, and 8, then event 8 becomes a cut-off candidate with 0 as companion. However, the addition of the interface event 5 frees event 8, because 5 is concurrent with 8 and not with 0.

However, we are not completely done yet. The parallel composition at the left of Figure 4

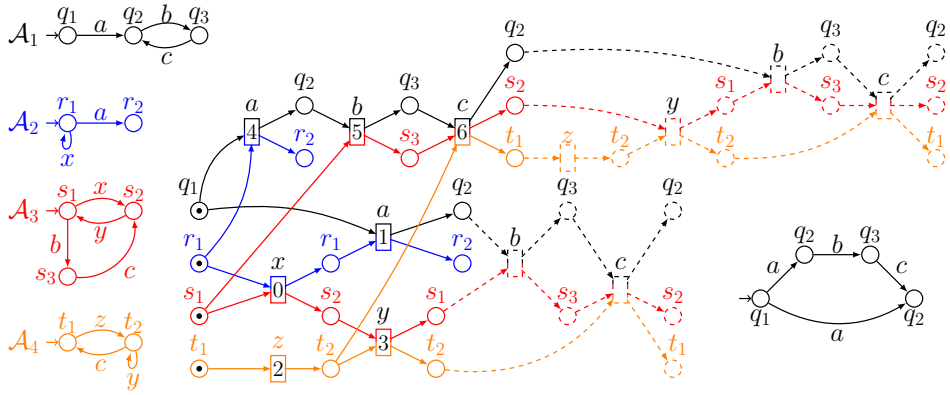


Figure 4 An example illustrating the use of strong causality.

gives an example in which even with this notion of cut-off candidate the result is still wrong. $\mathcal{A}_1$ is the interface. One branching process is represented at the top right of the figure. Event 3 (concurrent with 1) is a cut-off candidate with 2 (concurrent with 1, 4, and 5) as companion. This prevents the lower dashed part of the net to be added. Event 6 is cut-off with 1 as companion. This prevents the upper dashed part of the net to be added. The refolding obtained then (bottom right) does not contain the word *abcb*.

If we wish a correct algorithm for all strategies, we need a final touch: replace the condition $e' < e$ by $e' \ll e$, where $\ll$ is the *strong causal relation*:

▶ **Definition 5.** Event e' is a *strong cause* of event e, denoted by $e' \ll e$, if $e' < e$ and $b' < b$ for every $b \in M(e) \setminus M(e'), b' \in M(e') \setminus M(e)$.

Using this definition, event 3 is no longer a cut-off candidate in the branching process of Figure 4 as it is not in strong causal relation with its companion 2 (because the t_2-labelled condition just after 2 belongs to $M(2) \setminus M(3)$ and is not causally related with the r_1-labelled condition just after 0 which belongs to $M(3) \setminus M(2)$).

We are now in a position to provide adequate definitions for Algorithm 2.

▶ **Definition 6** (Cut-off and cut-off candidate)**.** Let $Ico_{\mathcal{N}}(e)$ denote the set of non cut-off interface events of $\mathcal{N}$ that are concurrent with e. An event e

- is a cut-off if it is an i-event, and $\mathcal{N}$ contains an i-event e' such that $\mathbf{St}(e) = \mathbf{St}(e')$.
- is a cut-off candidate of $\mathcal{N}$ if it is not an i-event, and $\mathcal{N}$ contains $e' \ll e$ such that $\mathbf{St}(e) = \mathbf{St}(e')$, $ip(e') = ip(e)$, and $Ico_{\mathcal{N}}(e) \subseteq Ico_{\mathcal{N}}(e')$.
- frees a cut-off candidate e_c of $\mathcal{N}$ if e_c is not a cut-off candidate of the branching process obtained by adding e to $\mathcal{N}$.

▶ **Theorem 7.** *Let $\mathbf{A} = \mathcal{A}_1 \parallel \ldots \parallel \mathcal{A}_n$ with interface $\mathcal{A}_i$. The instance of Algorithm 2 given by Definition 6 terminates and returns a branching process $\mathcal{N}$ such that the folding $\mathcal{S}_i$ of $\mathcal{N}_i$ is a summary of $\mathbf{A}$.*

The proof of this theorem is involved. It is given in [5]. We sketch the main ideas. Termination follows from a lemma showing that every infinite chain of causally related events contains an infinite subchain of strongly causally related events. The equality $Tr(\mathcal{S}_i) = Tr(\mathbf{A})|_{\Sigma_i}$ is proved in two parts. $Tr(\mathcal{S}_i) \subseteq Tr(\mathbf{A})|_{\Sigma_i}$ follows easily from the definitions. The proof of $Tr(\mathcal{S}_i) \supseteq Tr(\mathbf{A})|_{\Sigma_i}$ is more involved. For every trace of $\mathbf{A}$ we identify a *strongly succinct* occurrence sequence in the unfolding with this trace as projection. Intuitively, in

such a sequence, interface events occur as early as possible, and the number of non-interface events occurring between them is minimal. The main point in the proof is to show that every cut-off in strongly succinct sequences is necessarily an interface event, which allows one to conclude the proof as in the non-divergent case. This is proved by contradiction: If e is a cut-off candidate with companion e', we show that (1) e and e' are located between the same two interface events (this uses $Ico_{\mathcal{N}}(e) \subseteq Ico_{\mathcal{N}}(e')$), (2) there is no i-event in $[e] \setminus [e']$, and (3) every event of $[e] \setminus [e']$ is also located between the same two interface events (this is ensured by $e' \ll e$). The events from $[e] \setminus [e']$ can then be removed from the sequence, contradicting the definition of strongly succinct sequence.

6 Implementation and Experiments

As an illustration of the previous results, we report in this section on an implementation of Algorithm 2 as a modification of the existing unfolding tool Mole. All programs and data used are publicly available. [1] Many components of Mole could be re-used. The main work consisted in determining cut-off candidates and the "freeing" condition of Definition 6. This required two main algorithmic additions discussed in detail in [5]: (i) an efficient traversal of $[e]$, for a given event e, that allows to determine the conditions for cut-off candidates; (ii) computing $Ico_N(e)$ for an event e. Both additions could be obtained by extending existing components of the tool. While the additions were not always trivial, they could be obtained with small additional overhead.

We tested our implementation on well-known benchmarks used widely in the unfolding literature, see e.g. [2, 6, 16]. The input is the set of components $\mathcal{A}_1, \ldots, \mathcal{A}_n$, which are converted into an equivalent Petri net. For each example, we report the number of events (including cut-offs) in the prefix. Notice that this prefix is computed in less than one second in most cases (more detailed experimental results are given in [5]). We also report the number of reachable markings (taken from [20] where available, and computed combinatorially for DpSyn).

The experiments are summarized in Table 1. We used the following families of examples [2]: the CyclicC and CyclicS families are a model of Milner's cyclic scheduler with n consumers and n schedulers; in one case we compute the folding for a consumer, in the other for a scheduler. The Dac family represents a divide-and-conquer computation. Ring is a mutual-exclusion protocol on a token-ring. The tasks are not entirely symmetric, we report the results for the first. Finally, Dp, Dpsyn, and Dpd are variants of Dining Philosophers. In Dp, philosophers take and release forks one by one, whereas in Dpsyn they take and release both at once. In Dpd, deadlocks are prevented by passing a dictionary.

Table 1 Experimental results.

Test case	Events	Markings
CyclicC(6)	426	639
CyclicC(9)	3347	7423
CyclicC(12)	26652	74264
CyclicS(6)	303	639
CyclicS(9)	2328	7423
CyclicS(12)	18464	74264
Dac(9)	86	1790
Dac(12)	134	14334
Dac(15)	191	114686
Dp(6)	935	729
Dp(8)	5121	6555
Dp(10)	31031	48897
Dpd(4)	2373	601
Dpd(5)	23789	3489
Dpd(6)	245013	19861
Dpsyn(10)	176	123
Dpsyn(20)	701	15127
Dpsyn(30)	1576	1860498
Ring(5)	511	1290
Ring(7)	3139	17000
Ring(9)	16799	211528

[1] http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/summaries.tar.gz

In all cases except one (DPD) our algorithm needs clearly fewer events than there are reachable markings; in some families (DAC, DPSYN, RING) there are far fewer events. A comparison of DP and DPSYN is instructive. In DP, neighbours can concurrently pick and drop forks. Intuitively, this leads to fewer cases in which the condition $Ico_{\mathcal{N}}(e) \subseteq Ico_{\mathcal{N}}(e')$ for cut-off candidates is satisfied. On the other hand, in DPSYN both forks are picked and dropped synchronously, and so no event in $\mathcal{A}_i$ is concurrent to any event in the neighbouring components, making the unfolding procedure much more efficient.

7 Extensions: Divergences and Weights

We conclude the paper by showing that our algorithm can be extended to handle more complex semantics than traces. Indeed, the divergences of the system can be captured by the summaries, as well as the minimal weights of the finite traces from $Tr(\mathbf{A})|_{\Sigma_i}$ when $\mathcal{A}_1 \dots \mathcal{A}_n$ are weighted systems.

7.1 Divergences

We first extend our algorithm so that the summary also contains information about *divergences*. Intuitively, a divergence is a finite trace of the interface after which the system can "remain silent" forever.

▶ **Definition 8.** Let $\mathcal{A}_1, \dots, \mathcal{A}_n$ be LTSs with interface $\mathcal{A}_i$. A *divergence* of $\mathcal{A}_i$ is a finite trace $\sigma \in Tr(\mathcal{A}_i)$ such that $\sigma = \tau_{|\Sigma_i}$ for some infinite trace $\tau \in Tr(\mathbf{A})$. A *divergence-summary* is a pair $(\mathcal{S}_i, D)$, where $\mathcal{S}_i$ is a summary and D is a subset of the states of $\mathcal{S}_i$ such that $\sigma \in Tr(\mathcal{S}_i)$ is a divergence of $\mathcal{A}_i$ iff some realization of σ in $\mathcal{S}_i$ leads to a state of D.

We define the set of divergent conditions of the output of Algorithm 2, and show that it is a correct choice for the set D.

▶ **Definition 9.** Let $\mathcal{N}$ be the output of Algorithm 2. A condition s of $\mathcal{N}_i$ is *divergent* if after termination of the algorithm there is $e \in coc$ with companion e' such that s is concurrent to both e and e'. We denote the set of divergent conditions by DC.

▶ **Theorem 10.** *A finite trace $\sigma \in Tr(\mathcal{S}_i)$ is a divergence of $\mathcal{A}_i$ iff there is a divergent condition s of $\mathcal{N}_i$ such that some realization of σ leads to $[s]_{\equiv}$. Therefore, $(\mathcal{S}_i, [DC]_{\equiv})$ is a divergence-summary.*

The proof of this theorem is given in [5].

7.2 Weights

We now consider weighted systems, e.g parallel compositions of weighted LTS. More formally, a weighted LTS $\mathcal{A}^w = (\mathcal{A}, c)$ consists of an LTS $\mathcal{A} = (\Sigma, S, T, \lambda, s^0)$ and a weight function $c : T \to \mathbb{R}_+$ associating a weight to each transition. A *weighted trace* of $\mathcal{A}^w$ is a pair (σ, w) where $\sigma = a_1 \dots a_k$ is a finite trace of $\mathcal{A}$ and w is the minimal weight among the paths realizing σ, i.e:

$$w = \min_{\substack{s_0 \dots s_k \in S^{k+1}, s_0 = s^0, \\ t_i = (s_{i-1}, s_i) \in T, \lambda(t_i) = a_i}} \sum_{j=1}^{k} c(t_j).$$

We denote by $Tr(\mathcal{A}^w)$ the set of all the weighted traces of $\mathcal{A}^w$. The parallel composition $\mathbf{A}^w = (\mathbf{A}, \mathbf{c}) = \mathcal{A}_1^w \,||_w \cdots ||_w \, \mathcal{A}_n^w$ of the LTS $\mathcal{A}_1^w, \dots, \mathcal{A}_n^w$ is such that $\mathbf{A} = \mathcal{A}_1 || \dots || \mathcal{A}_n$ and

the weight of a global transition $\mathbf{t} = (t_1, \ldots, t_n)$ is:

$$\mathbf{c}(\mathbf{t}) = \sum_{t_i \neq \star} c_i(t_i).$$

Similarly a *weighted labelled Petri net* is a tuple $\mathcal{N}^w = (\mathcal{N}, c)$ where $\mathcal{N} = (\Sigma, P, T, F, \lambda, M_0)$ is a labelled Petri net and $c : T \to \mathbb{R}_+$ associates weights to transitions. A weighted trace in $\mathcal{N}^w$ is a pair (σ, w) with σ a finite trace of $\mathcal{N}$ and w the minimal weight of an occurrence sequence corresponding to σ, where the weight of an occurrence sequence is the sum of the weights of its transitions. By $Tr(\mathcal{N}^w)$ we denote the set of all the weighted traces of $\mathcal{N}^w$.

The branching processes of $\mathcal{A}_1^w || _w \ldots || _w \mathcal{A}_n^w$ are defined as weighted labelled Petri nets like in the non-weighted case, where each event is implicitly labelled by an action (as before) and a cost. Given a finite set of weighted traces W we define its restriction to alphabet Σ as

$$W|_\Sigma = \{\, (\sigma, w) : \exists (\sigma', w') \in W, \sigma = \sigma'|_\Sigma \wedge w = \min_{\substack{(\sigma', w') \in W \\ \sigma'|_\Sigma = \sigma}} w' \,\}.$$

As in the non-weighted case we are interested in solving the following summary problem:

▶ **Definition 11** (Weighted summary problem)**.** Given weighted LTSs $\mathcal{A}_1^w, \ldots, \mathcal{A}_n^w$ with interface $\mathcal{A}_i^w$, compute a weighted LTS $\mathcal{S}_i^w$ satisfying $Tr(\mathcal{S}_i^w) = Tr(\mathbf{A}^w)|_{\Sigma_i}$, where $\mathbf{A}^w = \mathcal{A}_1^w || _w \ldots || _w \mathcal{A}_n^w$.

This section aims at showing that the approach to the summary problem proposed in the non-weighted case still works in the weighted one. In other words, $\mathcal{S}_i^w$ can be obtained by computing a finite branching process $\mathcal{N}^w$ of $\mathbf{A}^w$ (using Definition 6 of cut-off and cut-off candidates and Algorithm 2) and then taking the interface projection $\mathcal{N}_i^w$ of $\mathcal{N}^w$ on $\mathcal{A}_i^w$ and folding it. The notion of interface projection needs to be slightly modified to take weights into account. The conditions, events, and arcs of $\mathcal{N}_i^w$ are defined exactly as above, and the weight of an event e of $\mathcal{N}_i^w$ is $c_i(e) = c([e]) - c([e'])$ if the predecessor e' of e in $\mathcal{N}_i^w$ exists and $c_i(e) = c([e])$ else, where c is the weight function of $\mathcal{N}^w$ and $c([e]) = \sum_{e_k \in [e]} c(e_k)$, where $[e]$ is the past of e in the weighted branching process $\mathcal{N}^w$.

▶ **Theorem 12.** *Let $\mathbf{A}^w = \mathcal{A}_1^w || _w \ldots || _w \mathcal{A}_n^w$ with interface $\mathcal{A}_i^w$. The instance of Algorithm 2 given by Definition 6 terminates and returns a weighted branching process $\mathcal{N}^w$ such that the folding S_i^w of $\mathcal{N}_i^w$ is a weighted summary of $\mathbf{A}^w$.*

The proof of this theorem is given in [5].

8 Conclusions

We have presented the first unfolding-based solution to the summarization problem for trace semantics. The final algorithm is simple, but its correctness proof is surprisingly subtle. We have shown that it can be extended (with minor modifications) to handle divergences and weighted systems.

The algorithm can also be extended to other semantics, including information about failures or completed traces; this material is not contained in the paper because, while laborious, it does not require any new conceptual ideas.

We conjecture that the condition $e' \ll e$ in the definition of cut-off candidate can be replaced by $e' < e$, if at the same time the algorithm is required to add events in a suitable order. Similar ideas have proved successful in the past (see e.g. [6, 16]).

Acknowledgement. Thanks to the anonymous reviewers for their valuable comments.

References

1 P. Baldan, A. Bruni, A. Corradini, B. König, C. Rodríguez, and S. Schwoon. Efficient unfolding of contextual Petri nets. *TCS*, 449(1):2–22, 2012.

2 J. C. Corbett. Evaluating deadlock detection methods for concurrent software. *IEEE Transactions on Software Engineering*, 22:161–180, 1996.

3 J-M. Couvreur, S. Grivet, and D. Poitrenaud. Designing a LTL model-checker based on unfolding graphs. In *Proceedings of ICATPN*, pages 123–145, 2000.

4 J. Esparza and K. Heljanko. *Unfoldings – A Partial-Order Approach to Model Checking*. Springer, 2008.

5 J. Esparza, L. Jezequel, and S. Schwoon. Computation of summaries using net unfoldings. Technical report, arXiv:1310.2143, 2013.

6 J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Proceedings of TACAS*, pages 87–106, 1996.

7 E. Fabre, A. Benveniste, S. Haar, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. *DEDS*, 15(1):33–84, 2005.

8 E. Fabre and L. Jezequel. Distributed optimal planning: an approach by weighted automata calculus. In *Proceedings of CDC*, pages 211–216, 2009.

9 B. Genest, H. Gimbert, A. Muscholl, and I. Walukiewicz. Optimal Zielonka-type construction of deterministic asynchronous automata. In *Proceedings of ICALP*, pages 52–63, 2010.

10 S. Graf and B. Steffen. Compositional minimization of finite state systems. In *Proceedings of CAV*, pages 186–196, 1990.

11 D. Harel, O. Kupferman, and M. Y. Vardi. On the complexity of verifying concurrent transition systems. In *Proceedings of CONCUR*, pages 258–272, 1997.

12 S. Hickmott, J. Rintanen, S. Thiébaux, and L. White. Planning via Petri net unfolding. In *Proceedings of IJCAI*, pages 1904–1911, 2007.

13 C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

14 V. Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, Newcastle University, 2003.

15 V. Khomenko and M. Koutny. LP deadlock checking using partial-order dependencies. In *Proceedings of CONCUR*, pages 410–425, 2000.

16 V. Khomenko, M. Koutny, and W. Vogler. Canonical prefixes of Petri net unfoldings. *Acta Informatica*, 40(2):95–118, 2003.

17 V. Khomenko, A. Madalinski, and A. Yakovlev. Resolution of encoding conflicts by signal insertion and concurrency reduction based on STG unfoldings. In *Proceedings of ACSD*, pages 57–68, 2006.

18 K. McMillan. A technique of state space search based on unfolding. *FMSD*, 6(1):45–65, 1995.

19 M. Mukund and M. Sohoni. Gossiping, asynchronous automata and Zielonka's theorem. Technical Report TCS-94-2, SPIC Science Foundation, 1994.

20 S. Römer. *Theorie und Praxis der Netzentfaltungen als Grundlage für die Verifikation nebenläufiger Systeme*. Phd thesis, TU München, 2000.

21 A. Valmari. Compositionality in state space verification methods. In *Proceedings of ICATPN*, pages 29–56, 1996.

22 G. Winskel. Events, causality and symmetry. *Computer Journal*, 54:42–57, 2011.

23 F. Zaraket, J. Baumgartner, and A. Aziz. Scalable compositional minimization via static analysis. In *Proceedings of ICCAD*, pages 1060–1067, 2005.

24 W. Zielonka. Notes on finite asynchronous automata. *RAIRO – Theoretical Informatics and Applications*, 21(2):99–135, 1987.

Faster Deterministic Algorithms for r-Dimensional Matching Using Representative Sets

Prachi Goyal[1], Neeldhara Misra[1], and Fahad Panolan[2]

1 **Indian Institute of Science, Bangalore, India**
`prachi.goyal|neeldhara@csa.iisc.ernet.in`
2 **Institute of Mathematical Sciences, Chennai, India**
`fahad@imsc.res.in`

Abstract

Given a universe $U := U_1 \uplus \cdots \uplus U_r$, and a r-uniform family $\mathcal{F} \subseteq U_1 \times \cdots \times U_r$, the r-DIMENSIONAL MATCHING problem asks if $\mathcal{F}$ admits a collection of k mutually disjoint sets. The special case when $r = 3$ is the classic 3D-MATCHING problem. Recently, several improvements have been suggested for these (and closely related) problems in the setting of randomized parameterized algorithms. Also, many approaches have evolved for deterministic parameterized algorithms. For instance, for the 3D-MATCHING problem, a combination of color coding and iterative expansion yields a running time of $\mathcal{O}^*(2.80^{(3k)})$, and for the r-DIMENSIONAL MATCHING problem, a recently developed derandomization for known algebraic techniques leads to a running time of $\mathcal{O}^*(5.44^{(r-1)k})$.

In this work, we employ techniques based on dynamic programming and representative families, leading to a deterministic algorithm with running time $\mathcal{O}^*(2.85^{(r-1)k})$ for the r-DIMENSIONAL MATCHING problem. Further, we incorporate the principles of iterative expansion used in the literature [TALG 2012] to obtain a better algorithm for 3D-MATCHING, with a running time of $\mathcal{O}^*(2.003^{(3k)})$. Apart from the significantly improved running times, we believe that these algorithms demonstrate an interesting application of representative families in conjunction with more traditional techniques.

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity

Keywords and phrases 3-Dimensional Matching, Fixed-Parameter Algorithms, Iterative Expansion

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.237

1 Introduction

Given a universe $U := U_1 \uplus \cdots \uplus U_r$, and a r-uniform family $\mathcal{F} \subseteq U_1 \times \cdots \times U_r$, the r-DIMENSIONAL MATCHING problem asks if $\mathcal{F}$ admits a collection of k mutually disjoint sets. The special case when $r = 3$ can be viewed as an immediate generalization of the matching problem on bipartite graphs to three-partite, three-uniform hypergraphs. The question of finding the largest 3D-Matching is a classic optimization problem, and the decision version is listed as one of the six fundamental NP-complete problems in Garey and Johnson [9]. These problems may also be thought as restricted versions of the more general SET PACKING questions, where no restrictions are assumed on the universe.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 237–248

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

r-Dimensional Matching. The question of r-DIMENSIONAL MATCHING has enjoyed substantial attention in the context of exact algorithms, and there have been several deterministic and randomized approaches to the problem (see Table 1). One of the earliest approaches [6] used the color-coding technique [1]. Further, in [7], an $\mathcal{O}(k^3)$ kernel was developed and the color-coding was combined with dynamic programming on the structure of the kernel to obtain an improvement. In [10], Koutis used an algebraic formulation of SET PACKING and proposed a randomized algorithm (derandomized with hash families). The randomized approaches saw further improvements in subsequent work [12, 11, 2], also based on algebraic techniques. The common theme in these developments is to express a parameterized problem in an algebraic framework by associating multilinear monomials with the combinatorial structures that are sought, ultimately arriving at multilinear monomial testing problem or polynomial identity testing problem.

In a recent development [4], the authors propose a derandomization method for these algebraic approaches, leading to a determinstic algorithm that solves the r-DIMENSIONAL MATCHING problem in time $\mathcal{O}^*(5.44^{(r-1)k})$[1].

Table 1 Algorithms for r-DIMENSIONAL MATCHING.

References	Randomized Algorithms	Deterministic Algorithms
Fellows *et al.*, 1999 [6]		$\mathcal{O}^*((rk)!(rk)^{3rk+1})$
Fellows *et al.*, 2008 [7]		$\mathcal{O}^*(2^{O(rk)})$
Koutis, 2005 [10]	$\mathcal{O}^*((4e)^{rk})$	$\mathcal{O}^*(25.6^{rk})$ (See also [3, 13])
Koutis, 2008 [12]	$\mathcal{O}^*(2^{rk})$	
Koutis and Williams, 2009 [11]	$\mathcal{O}^*(2^{(r-1)k})$	
Björklund *et al.*, 2010 [2]	$\mathcal{O}^*(2^{(r-2)k})$	
Chen and Chen, 2013 [4]		$\mathcal{O}^*(5.44^{(r-1)k})$
This work (Theorem 3.4)		$\mathcal{O}^*(2.851^{(r-1)k})$

3D-Matching. The 3D-MATCHING problem admits of even better algorithms than those obtained by using the algorithms above for the particular case of $r = 3$. Indeed, the works [13, 3] consider 3D-MATCHING separately and obtain better algorithms with determinstic running times $\mathcal{O}^*(2.77^{3k})$ and $\mathcal{O}^*(2.80^{3k})$. In fact, even in the present work, we will explore an improvement that is specific to 3D-MATCHING, resulting in a significantly faster algorithm. The approach in [3] uses a clever combination of dynamic programming (embedded in a color coding framework) and iterative expansion, derandomized using hash families. In our work, we obtain a deterministic algorithm with running time $\mathcal{O}^*(2.0034^{3k})$.

The Method of Representative Families and Our Contributions

We first consider the general problem of finding a maximum weight r-dimensional k-packing (that is, the sets have integer weights and the goal is to find a k-packing of weight at least W). Next, we focus on the 3D-MATCHING question separately, improving the general algorithm further for this special case. Our algorithms are an improvization of the work in [3], in that

[1] The $\mathcal{O}^*$ notation is used to suppress polynomial factors in the running time.

the main dynamic programming and iterative expansion wireframes stay intact, however, the color coding engine is replaced by an application of *representative families*.

In using representative families, which is a general approach based on uniform matroids, our attempt is somewhat different from the previous approaches to the problem. Matroids have been applied to design fixed-parameter tractable algorithms as early as [14], where the main tool was the notion of representative families. The idea of using representative families was introduced by Monien in [15]. We briefly describe the basic notion here, and the reader is referred to Section 2 for the details.

In a dynamic programming framework, one might think of every intermediate stage of the algorithm as a location storing several partial solutions, with the hope that one of them would "lead to" a final solution. In many settings, a solution (when it exists) can be viewed as a split into two disjoint parts — where one can be thought of as the partial part that we would like to store, and the other is the part that we are hoping to encounter down the line. The inherent disjointness of these parts suggests that perhaps all partial solutions need not be maintained, and instead, as long as there is some witness that can be evolved into a complete solution, correctness is guaranteed.

We note that the color coding approach is a rather clever way of capturing this notion, albeit with an element of randomization. The notion of representative families, on the other hand, formalizes this intuition in a combinatorial fashion. The definition of a representative family is as follows. Let $M = (E, \mathcal{I})$ be a matroid and let $\mathcal{S} = \{S_1, \ldots, S_t\}$ be a family of subsets of E of size p. A subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is *q-representative* for $\mathcal{S}$ if for every set $Y \subseteq E$ of size at most q, if there is a set $X \in \mathcal{S}$ disjoint from Y with $X \cup Y \in \mathcal{I}$, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y with $\widehat{X} \cup Y \in \mathcal{I}$.

For this definition to be useful, we will need to know that small representative families exist. A classical result due to Bollobás indicates that uniform matroids, involving sets of size at most p, admit a q-representative family with at most $\binom{p+q}{p}$ sets (subsequently generalized by Lovász to representable matroids). The next natural issue is that of computational overhead, which brings us to the question of whether these representative families be found, and at what cost. Fortunately, the combinatorial proofs turn out to be constructive, and algorithmic versions have been established. The most recent development in this line of work is in [8], where the authors describe an algorithm constructing a q-representative family of size at most $\binom{p+q}{p}$ in time bounded by a polynomial in $\binom{p+q}{p}$, t, and the time required for field operations, where t is the size of the input family.

Representative families can be applied in a very natural way to a large class of "matching and packing" problems. For instance, consider the DP table for 3-Set Packing that stores in $S[i]$ packings of size i. This approach can be improved by keeping only $(3k - 3i)$-representative families at every stage, leading to an algorithm with running time $\mathcal{O}^*(2.851^{3k})$.

Our focus is to improve this naive approach by more careful dynamic programming, leading to an algorithm with running time $\mathcal{O}^*(2.851^{(r-1)k})$ for r-dimensional matching (see Theorem 3.4). For 3D-Matching, we are able to apply iterative expansion as used in [3]. The idea of iterative expansion is to focus on the "improvement" version of the question, where the input includes a matching of size k and the question is if there is a matching of size $k+1$. Clearly, an efficient algorithm for this question can be run k times (starting with $k = 1$) to find a matching of size k. One of the reasons it is useful to have a matching of size k as input is Lemma 3.4 in [3], which states if there is an improved matching, then there

is one that overlaps the given matching in a substantial way. The guarantee of this overlap can be exploited suitably to attain better running times. We use this result as well, except that we incorporate the advantages in the setting of representative families. The result is an algorithm with running time $\mathcal{O}^*(2.0034^{3k})$ for 3D-Matching (see Theorem 4.5).

Another advantage of our first approach is the fact that it works for the weighted version of the r-dimensional matching problem with only an additional factor of $\log W$ (where W is the target weight). This factor is linear in the input, and is thus essentially optimal. To the best of our knowledge, the known approaches either are not scalable to the weighted version of the question or incur an additional factor of W in the running time.

Organization. In Section 2, we discuss some of the definitions associated with matroids and representative families. In Section 3, we describe our first algorithm for the weighted version of r-dimensional matching. Finally, in Section 4, we show how iterative expansion can be used to obtain an improved algorithm for 3D-Matching. Owing to limitations of space, some proofs have been omitted. Such results are marked with a $\star$.

2 Preliminaries

In this section we summarize the important definitions. A more detailed introduction may be found in the full version. To begin with, we define the notion of a (min/max) representative family and state the theorems associated with efficient computations of such families.

▶ **Definition 2.1** (**q-Representative Family**,[8]). Given a matroid $M = (E, \mathcal{I})$ and a family $\mathcal{S}$ of subsets of E, we say that a subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is *q-representative* for $\mathcal{S}$ if the following holds: for every set $Y \subseteq E$ of size at most q, if there is a set $X \in \mathcal{S}$ disjoint from Y with $X \cup Y \in \mathcal{I}$, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y with $\widehat{X} \cup Y \in \mathcal{I}$. If $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is q-representative for $\mathcal{S}$ we write $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$. Also, we say that $\widehat{X}$ is a q-representative for X with respect to Y.

▶ **Definition 2.2** (**Min/Max q-Representative Family**,[8]). Given a matroid $M = (E, \mathcal{I})$, a family $\mathcal{S}$ of subsets of E and a non-negative weight function $w : \mathcal{S} \to \mathbb{N}$, we say that a subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is *min q-representative* (*max q-representative*) for $\mathcal{S}$ if the following holds: for every set $Y \subseteq E$ of size at most q, if there is a set $X \in \mathcal{S}$ disjoint from Y with $X \cup Y \in \mathcal{I}$, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y with $\widehat{X} \cup Y \in \mathcal{I}$; and $w(\widehat{X}) \leq w(X)$ ($w(\widehat{X}) \geq w(X)$). We use $\widehat{\mathcal{S}} \subseteq^q_{minrep} \mathcal{S}$ ($\widehat{\mathcal{S}} \subseteq^q_{maxrep} \mathcal{S}$) to denote a min q-representative (max q-representative) family for $\mathcal{S}$. Also, we say that $\widehat{X}$ is a min q-representative (respectively, max q-representative) for X with respect to Y.

▶ **Theorem 2.3** ([8]). *Let $\mathcal{S} = \{S_1, \ldots, S_t\}$ be a family of sets of size p over a universe of size n. For a given q, a q-representative family $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ for $\mathcal{S}$ with at most $\binom{p+q}{p} \cdot 2^{o(p+q)} \cdot \log n$ sets can be computed in time $\mathcal{O}((\frac{p+q}{q})^q \cdot 2^{o(p+q)} \cdot t \cdot \log n)$.*

▶ **Theorem 2.4** ([8]). *There is an algorithm that given a p-family $\mathcal{S}$ of sets over a universe U of size n, an integer q, and a non-negative weight function $w : \mathcal{S} \to \mathbb{N}$ with maximum value at most W, computes in time $\mathcal{O}(|\mathcal{S}| \cdot (\frac{p+q}{q})^q \cdot \log n + |\mathcal{S}| \cdot \log |\mathcal{S}| \cdot \log W)$ a subfamily $\widehat{\mathcal{S}}$ such that $|\widehat{\mathcal{S}}| = \binom{p+q}{p} \cdot 2^{o(p+q)} \cdot \log n$ and $\widehat{\mathcal{S}} \subseteq^q_{minrep} \mathcal{S}$ ($\widehat{\mathcal{S}} \subseteq^q_{maxrep} \mathcal{S}$).*

The problems we consider are the following:

r-DIMENSIONAL MATCHING	**Parameter:** k
Input: A universe $U := U_1 \uplus \cdots \uplus U_r$, a family $\mathcal{F} \subseteq U_1 \times \cdots \times U_r$, and $k \in \mathbb{Z}^+$.	
Question: Does $\mathcal{F}$ have a collection of at least k mutually disjoint sets?	

WEIGHTED EXACT r-DIMENSIONAL MATCHING	**Parameter:** k
Input: A universe $U := U_1 \uplus \cdots \uplus U_r$, a family $\mathcal{F} \subseteq U_1 \times \cdots \times U_r$, a non-negative weight function $w : \mathcal{F} \to \mathbb{N}$ and positive integers k, W.	
Question: Does there exist $\mathcal{M} \subseteq \mathcal{F}$, of k mutually disjoint sets such that $w(\mathcal{M}) \geq W$?	

3 A Dynamic Programming Approach

In this section, we describe an algorithm with running time $\mathcal{O}^*(2.851^{(r-1)k})$ for the WEIGHTED EXACT r-DIMENSIONAL MATCHING problem. For ease of presentation, we will describe the algorithm for WEIGHTED EXACT 3-DIMENSIONAL MATCHING here. The generalization to the problem of finding a weighted r-dimensional matching of size k appears in the full version of this work.

Let $(U_1, U_2, U_3, \mathcal{F}, w : \mathcal{F} \to \mathbb{N}, k, W)$ be an instance of WEIGHTED EXACT 3D-MATCHING. Recall that $\mathcal{F} \subseteq U_1 \times U_2 \times U_3$, and the problem involves finding k mutually disjoint sets in $\mathcal{F}$ whose weight is at least W. For a set $S \in \mathcal{F}$, we let $S[i]$ denote $S \cap U_i$, that is, $S[i]$ is the element of S that is from U_i. We sometimes refer to the element $S[i]$ as the i^{th} coordinate of S.

The improved dynamic programming approach involves iterating over the elements in U_3 (for the discussion in this section, this is an arbitrary and fixed choice). In particular, let $U_3 := \{c_1, c_2, \ldots, c_n\}$. Let $\mathcal{M} := \{S_1, S_2, \ldots, S_t\}$ be a 3D-Matching. Let $\mathcal{M}_3 := \{S_i[3] \mid i \in [t]\} \subseteq U_3$. We define the *maximum last index* of $\mathcal{M}$, denoted by $\lambda(\mathcal{M})$, as the largest index i for which $c_i \in \mathcal{M}_3$. For the empty matching $\emptyset$, $\lambda(\emptyset) = 0$. In the i^{th} iteration of our algorithm, we would like to store all matchings whose maximum last index is at most i, and we stop at our first encounter with a matching of size k and weight at least W. However, it turns out that storing all possible matchings at every stage is exponentially expensive — we potentially run into storing $O(n^{2i})$ matchings at the i^{th} step. In [3], this problem (for the unweighted case) is mitigated using color coding. We will now discuss how we can use the notion of max representative families instead, which has the dual advantage of being faster and deterministic.

We first introduce some notation. We let $\mathcal{Q}^{(i)} = \{\mathcal{M} \mid \lambda(\mathcal{M}) \leq i\}$ and $\mathcal{Q}^{(i)}_j := \{\mathcal{M} \mid \mathcal{M} \in \mathcal{Q}^{(i)}, |\mathcal{M}| = j\}$. Notice that the $\mathcal{Q}^{(i)}_j$'s constitute a partition of the set $\mathcal{Q}^{(i)}$ based on matching size, in other words, $\mathcal{Q}^{(i)} := \bigcup_{j=0}^{n} \mathcal{Q}^{(i)}_j$. Recall that a naive application of the method of representative families would lead to a running time of $\mathcal{O}(2.85^{3k})$. To use the representative families more efficiently, we would like to splice the elements of the matchings into two parts. We will first collect the parts of the matching that come from $(U_1 \cup U_2)$, and then store separately a map that completes the first part to the complete matching. This will allow us to apply the dynamic programming approach suggested above. To this end, for a matching $\mathcal{M}$, we let $\mathcal{M}_{12}$ denote the subset of $(U_1 \cup U_2)$ obtained by projecting the elements of $\mathcal{M}$ on their first two co-ordinates, that is, $\mathcal{M}_{12} := \bigcup_{S \in \mathcal{M}} \{S[1], S[2]\}$. On other hand, let

$$\gamma := \{(\mathcal{M}, \mathcal{M}_{12}) \mid \mathcal{M} \text{ is a matching in } \mathcal{F}\}.$$

Algorithm 1: A Simple DP Implementation for Weighted Exact 3D-Matching using Representative Families

Input: $(U_1, U_2, U_3, \mathcal{F}, w : \mathcal{F} \to \mathbb{N}, k, W)$, where $\mathcal{F} \subseteq U_1 \times U_2 \times U_3$, and $k, W \in \mathbb{N}$.
Output: A 3D-Matching of size k and weight at least W if it exists, No otherwise.

$\mathcal{R}_0^{(0)} \leftarrow \{\emptyset\}$
$\mathcal{R}_j^{(0)} \leftarrow \emptyset$, for all $1 \leq j \leq k$
$\mathcal{L}^{(0)} \leftarrow \{\emptyset\}$
for $i \in \{1, 2, \ldots, n\}$ **do**
  $\mathcal{L}^{(i)} \leftarrow \{\mathcal{M} \cup S \mid \mathcal{M} \in \mathcal{L}^{(i-1)}, S \in \mathcal{F}, \mathcal{M} \cap S = \emptyset, S[3] = i\} \cup \mathcal{L}^{(i-1)}$
  for $j \in \{0, 1, \ldots, k\}$ **do**
    $\mathcal{L}_j^{(i)} \leftarrow \{\mathcal{M} \mid \mathcal{M} \in \mathcal{L}^{(i)}, |\mathcal{M}| = j\}$
    if $\mathcal{L}_k^{(i)} \neq \emptyset$ *and* $\exists \mathcal{M} \in \mathcal{L}_k^{(i)}$ *such that* $w(\mathcal{M}) \geq W$ **then**
      return $\mathcal{M}$.
    $\mathcal{P}_j^{(i)} \leftarrow \{\mathcal{M}_{12} \mid \mathcal{M} \in \mathcal{L}_j^{(i)}\}$;
    $\gamma \leftarrow \{(\mathcal{M}, \mathcal{M}_{12}) \mid \mathcal{M}$ is a matching in $\mathcal{L}_j^{(i)}\}$
    $w' \leftarrow \{(\mathcal{M}_{12}, w(\gamma^\dagger(\mathcal{M}_{12}))) \mid \mathcal{M}_{12} \in \mathcal{P}_j^{(i)}\}$
    Let $\mathcal{R}_j^{(i)} \subseteq_{maxrep}^{2k-2j} \mathcal{P}_j^{(i)}$
    $\mathcal{L}_j^{(i)} \leftarrow \{\gamma^\dagger(\mathcal{M}_{12}) \mid \mathcal{M}_{12} \in \mathcal{R}_j^{(i)}\}$
  $\mathcal{L}^{(i)} \leftarrow \bigcup_{j=0}^{k} \mathcal{L}_j^{(i)}$
return No;

In γ, we are merely storing the associations of $\mathcal{M}_{12}$ with the matchings that they "came from". Observe that γ might (by definition) contain multiple entries with the same second coordinate. On the other hand, when the algorithm stores the associations in γ, we will see that it will be enough to maintain one maximum weighted entry for each $\mathcal{M}_{12}$. To this end, we define the function $\gamma^\dagger$ as follows. Let $\leq$ be an arbitrary total order on the set of all matchings in $\mathcal{F}$. For a set $S \subseteq U_1 \cup U_2$, we define $\gamma^\dagger(S)$ as the smallest matching $\mathcal{M}$ (with respect to $\leq$) among the maximum weighted matchings in the set $\{\mathcal{M}' \mid (\mathcal{M}', S) \in \gamma\}$. If $\gamma^\dagger(S)$ is $\mathcal{M}$, then we say that $\mathcal{M}$ is the matching associated with S. Finally, for a set S and a matching $\mathcal{M}$, we abuse notation and say that $\mathcal{M}$ is disjoint from S (notationally, $\mathcal{M} \cap S = \emptyset$) to mean that $T \cap S = \emptyset$ for all $T \in \mathcal{M}$.

We are now ready to describe our first algorithm (see also Algorithm 1). In $\mathcal{L}^{(i)}$, we store carefully chosen 3D-Matchings whose maximum last index is at most i. We compute $\mathcal{L}^{(i)}$ by considering all elements $\mathcal{M}$ of $\mathcal{L}^{(i-1)}$, and checking if they can be extended to a matching whose maximum last index is at most i. For a fixed $\mathcal{M}$, this check is performed by iterating over all elements $S \in \mathcal{F}$ such that $S[3] = i$ and extending $\mathcal{M}$ to $\mathcal{M} \cup S$ whenever $\mathcal{M} \cap S = \emptyset$. Subsequently, we will compute a representative family of $\mathcal{L}^{(i)}$. Since $\mathcal{L}^{(i)}$ is a collection of matchings of varying sizes, we will need an appropriate version of $\mathcal{L}^{(i)}$ that will be suitable for Theorem 2.4. To this end, we first partition the set $\mathcal{L}^{(i)}$ — the part denoted by $\mathcal{L}_j^{(i)}$ contains all matchings from $\mathcal{L}^{(i)}$ of size j. Note that this is simply done to ensure uniformity of size. Of course, it is easily seen that if we have a matching of size k with weight at least W at this point, then we are done.

Next, we associate with every matching $\mathcal{M} \in \mathcal{L}_j^{(i)}$, a set that consists of the first two indices of every set in $\mathcal{M}$. Recall that this is denoted by $\mathcal{M}_{12}$. The collection of sets that

correspond to matchings in $\mathcal{L}_j^{(i)}$ is denoted by $\mathcal{P}_j^{(i)}$. We use γ to store the associations between the sets and the original matchings. Note that γ might have multiple pairs with the same second index and same weight w for first index, but this will be irrelevant (it would simply mean that $\mathcal{M}_{12}$ can be "pulled back" to multiple matchings, each of which would be equally valid). Now we define a weight function $w' : \mathcal{P}_j^{(i)} \to \mathbb{N}$. $\forall \mathcal{M}_{12} \in \mathcal{P}_j^{(i)}$, we set $w'(\mathcal{M}_{12}) = w(\gamma^\dagger(\mathcal{M}_{12}))$. Now, the central step of the algorithm is to compute a max $(2k-2j)$-representative family $\mathcal{R}_j^{(i)}$ for $\mathcal{P}_j^{(i)}$. Once we have the representative family, we revise $\mathcal{L}_j^{(i)}$ to only include the associated matchings with sets in $\mathcal{R}_j^{(i)}$.

The correctness of the algorithm relies on the fact that at each step, instead of the complete family of partial solutions $\mathcal{Q}^{(i)}$, it suffices to store only a representative family of $\mathcal{Q}^{(i)}$. Also, we will show that the family computed by the algorithm, $\mathcal{L}^{(i)}$, is indeed a representative family for $\mathcal{Q}^{(i)}$. The analysis of the running time will be quite straightforward in the light of Theorem 2.4, and we will discuss it after arguing the correctness.

Let $\mathcal{X}_j^{(i)} := \{\mathcal{M}_{12} \mid \mathcal{M} \in \mathcal{Q}_j^{(i)}\}$. We assign a weight function $w' : \mathcal{X}_j^{(i)} \to \mathbb{N}$ as follows. $\forall \mathcal{M}_{12} \in \mathcal{X}_j^{(i)}$, $w'(\mathcal{M}_{12}) = \max\{w(\mathcal{M}^\star) \mid \mathcal{M}^\star \in \mathcal{Q}_j^{(i)}, \mathcal{M}_{12} = \mathcal{M}^\star_{12}\}$. Our first claim is the following.

▶ **Lemma 3.1** ($\star$). *For all $0 \leq i \leq n$, and $0 \leq j \leq k-1$, the set $\mathcal{R}_j^{(i)}$ is a max $(2k-2j)$-representative family for $\mathcal{X}_j^{(i)}$.*

Observe that any solution constructed by Algorithm 1 is always a valid matching. Therefore, if there is no matching of size k, Algorithm 1 always returns a No. On the other hand, if given a Yes instance, we now show that Algorithm 1 always finds a 3D-Matching of size k with weight at least W.

▶ **Lemma 3.2.** *Let $(U_1, U_2, U_3, \mathcal{F}, w : \mathcal{F} \to \mathbb{N}, k, W)$ be a* Yes*-instance of* Weigted Exact 3D-Matching*. Then, Algorithm 1 successfully computes a 3D-Matching of size k with weight at least W.*

Proof. Let $(U_1, U_2, U_3, \mathcal{F}, w : \mathcal{F} \to \mathbb{Z}^+, k, W)$ be a Yes-instance of Weighted Exact 3D-Matching. Let $\mathcal{M} = \{S_1, S_2, \ldots, S_k\}$, be a k-sized 3D-Matching in $\mathcal{F}$ and $w(\mathcal{M}) \geq W$. Suppose $\lambda(\mathcal{M}) = i$. Without loss of generality, let $S_k[3] = c_i$. Finally, let $\mathcal{M}' = \mathcal{M} \setminus S_k$. Recall that $\mathcal{Q}_j^{(i)}$ is the set of all 3D-Matchings of size j with maximum last index at most i, and $\mathcal{X}_j^{(i)}$ contains the projections of these matchings on their first two coordinates. Therefore, $\mathcal{M}' \in \mathcal{Q}_{k-1}^{(i-1)}$ and $\mathcal{M}'_{12} \in \mathcal{X}_{k-1}^{(i-1)}$. Note that $w'(\mathcal{M}'_{12}) \geq w(\mathcal{M}')$.

By Lemma 3.1, we have that $\mathcal{R}_{k-1}^{(i-1)}$ is a max 2-representative family for $\mathcal{X}_{k-1}^{(i-1)}$. Let $Y = \{S_k[1], S_k[2]\}$. Since $\mathcal{R}_{k-1}^{(i-1)} \subseteq^2_{maxrep} \mathcal{X}_{k-1}^{(i-1)}$, we have that $\mathcal{R}_{k-1}^{(i-1)}$ contains a 2-representative $Z^\star$, for $\mathcal{M}'_{12}$ with respect to Y. So we have $w'(Z^\star) \geq w'(\mathcal{M}'_{12})$. Let $\mathcal{M}^\star$ be the matching associated with $Z^\star$. Therefore $w(\mathcal{M}^\star) = w'(Z^\star)$. It is easy to see that $\mathcal{M}^\star \cap S_k = \emptyset$ and $\lambda(\mathcal{M}^\star \cup S_k) = i$. Therefore, in Step 2 of Algorithm 1, we have that $\mathcal{L}^{(i)}$ contains the matching $\mathcal{M}^\star \cup S_k$ which is then classified in the set $\mathcal{L}_k^{(i)}$. Consider the weight of $\mathcal{M}^\star \cup S_k$.

$$\begin{aligned} w(\mathcal{M}^\star \cup S_k) &= w(\mathcal{M}^\star) + w(S_k) \\ &\geq w(\mathcal{M}') + w(S_k) && \text{(Since } w(\mathcal{M}^\star) = w'(Z^\star) \geq w'(\mathcal{M}'_{12}) \geq w(\mathcal{M}')\text{)} \\ &\geq W && \text{(Since } w(\mathcal{M}') + w(S_k) = w(\mathcal{M}) \geq W\text{)} \end{aligned}$$

Thus, the algorithm will return a matching of size k and weight at least W in Step 9. ◀

▶ **Lemma 3.3** ($\star$). *The running time of Algorithm 1 is bounded by* $O(2.851^{2k})$.

We remark that in case of unweigted 3D-Matching, if there exists a 3D-Matching of size at least k then there exists a 3D-Matching of size k, and so we can use Algorithm 1 to solve 3D-Matching. The extension of these ideas to the Weighted Exact r-Dimensional Matching is discussed in the full version of the paper. The final result is the following.

▶ **Theorem 3.4.** Weighted Exact r-Dimensional Matching *and* r-Dimensional Matching *can be solved deterministically in time* $\mathcal{O}^*(2.851^{(r-1)k})$.

4 Iterative Expansion and Representative Sets

In this section, we provide a faster algorithm for 3D-Matching using iterative expansion. The idea is to first solve the following improvement problem: given a 3D-Matching of size j as input, can we find a 3D-Matching of size $(j+1)$? Once we have an algorithm for solving the improvement question, we can use it as a subroutine k times to find a matching of size k, by starting with a trivial matching of size one. Therefore, for the rest of this section, we focus on the improvement problem:

3D-Matching Improvement	**Parameter:** k
Input: A universe $U := U_1 \uplus U_2 \uplus U_3$, a family $\mathcal{F} \subseteq U_1 \times U_2 \times U_3$, and $\mathcal{K} \subseteq \mathcal{F}$, a 3D-Matching of size k.	
Question: Does $\mathcal{F}$ have a 3D-Matching of size $(k+1)$?	

Let $\mathcal{M}$ be a 3D-Matching given by $\{S_1, \ldots, S_t\}$. Generalizing the notation in the previous section, we let $\mathcal{M}_{pq}$ denote the subset of $(U_p \cup U_q)$ obtained by projecting the elements of $\mathcal{M}$ on the p and q co-ordinates, that is, $\mathcal{M}_{pq} := \bigcup_{S \in \mathcal{M}} \{S[p], S[q]\}$. On other hand, let

$$\gamma_{pq} := \{(\mathcal{M}, \mathcal{M}_{pq}) \mid \mathcal{M} \text{ is a matching in } \mathcal{F}\}.$$

The definition of $\gamma^{\dagger}_{pq}$ is also as expected: For a set $S \subseteq U_p \cup U_q$, we define $\gamma^{\dagger}_{pq}(S)$ as the smallest matching $\mathcal{M}$ (with respect to $\leq$) for which $(\mathcal{M}, S) \in \gamma_{pq}$. Finally, for $r \in \{1,2,3\}$, let $\mathcal{M}_r := \{S_i[r] \mid i \in [t]\} \subseteq U_r$. Let $U_r = \{c_1, c_2, \ldots, c_n\}$. We define the *maximum last index of* $\mathcal{M}$ *with respect to* r, denoted by $\lambda_r(\mathcal{M})$, as the largest index i for which $c_i \in \mathcal{M}_r$. For empty matching $\emptyset$, $\lambda_r(\mathcal{M}) = 0$. To use the method of iterative expansion to our advantage, we invoke the following result from [3], which states that if there is *some* matching of size $(k+1)$, then there is also one that has a large intersection with the given matching $\mathcal{K}$.

▶ **Lemma 4.1** (Lemma 3.4 and Theorem 3.6, [3]). *Let* $(U_1, U_2, U_3, \mathcal{F}, \mathcal{K})$ *be a* Yes*-instance of* 3D-Matching Improvement*, that is,* $\mathcal{F}$ *admits a matching* $\mathcal{M}$ *of size* $(k+1)$*. Then, there is also a matching* $\mathcal{M}^\star$ *of size* $(k+1)$ *such that there exist two indices* $p, q \in \{1,2,3\}$*, for which* $|\mathcal{M}^\star_{pq} \cap \mathcal{K}_{pq}| \geq (4/3)k$.

The improved algorithm for solving the 3D-Matching Improvement (see Algorithm 2) is along the lines of Algorithm 1, in that the dynamic programming procedure is very similar. The difference lies in how we compute the representative families (see lines 17-19, Algorithm 2), and this is also what brings about the improved running time.

We begin by considering the given matching, $\mathcal{K}$. By Lemma 4.1, we know that there is always a solution that has a large common intersection with $\mathcal{K}$ (if one solution exists).

In particular, if we denote this solution by $\mathcal{M}^\star$ then Lemma 4.1 indicates that there is a choice of coordinates $p, q \in \{1,2,3\}$ for which the number of elements in $\mathcal{K}_{pq} \cap \mathcal{M}^\star_{pq}$ is at least $(4/3)k$. So our first step involves guessing the coordinates p and q (Steps 1-2 in Algorithm 2). Once we have fixed a choice of p and q, we further guess the intersection $\mathcal{K}_{pq} \cap \mathcal{M}^\star_{pq}$. We do this by iterating over all subsets of $\mathcal{K}_{pq}$ (see Line 12, Algorithm 2). Let U be a fixed guess of the elements in the intersection. In $\mathcal{P}^{(i)}_{j,U}$ we store (as before) the projection of the matchings from $\mathcal{L}^{(i)}_j$ on U_p and U_q — however we are now only interested in matchings that satisfy $\mathcal{M}_{pq} \cap \mathcal{K}_{pq} = U$. In the next step, we remove the elements of U from every set in $\mathcal{P}^{(i)}_{j,U}$ to obtain $\mathcal{D}^{(i)}_{j,U}$. Now, we compute a $(2k/3 - 2j + |U| + 2)$-representative family of $\mathcal{D}^{(i)}_{j,U}$. The intuitive explanation is the following. Since $\mathcal{M}_{pq}$ already has $(4k/3)$ elements in common with $\mathcal{K}_{pq}$, we could isolate these elements, set them aside, and compute representative families only for the "rest". Due to this restrictive computation of representative families, the amount of time we spend on this step improves considerably.

Towards correctness, we define $\mathcal{Q}^{(i)}[\![pq]\!]$, which is analogous to the notion of $\mathcal{Q}^{(i)}$ in the previous section. The only difference is that the maximum last index value is considered along the r coordinate, where $r \in \{1,2,3\}$ is such that $r \neq p, q$. Further, it will be useful to define $\mathcal{Q}^{(i)}_{j,U}[\![pq]\!]$, which further partitions the collection $\mathcal{Q}^{(i)}[\![pq]\!]$ based on the matching size and the intersection with $\mathcal{K}_{pq}$, i.e, $\mathcal{Q}^{(i)}_{j,U}[\![pq]\!] := \{\mathcal{M} \mid \mathcal{M} \in \mathcal{Q}^{(i)}, |\mathcal{M}| = j, \mathcal{M}_{pq} \cap \mathcal{K}_{pq} = U\}$ Finally let $\mathcal{X}^{(i)}_{j,U}[\![pq]\!] := \{\mathcal{M}_{pq} \mid \mathcal{M} \in \mathcal{Q}^{(i)}_{j,U}[\![pq]\!]\}$.

In this setting, our first claim is that if we have a YES instance of 3D-MATCHING IMPROVEMENT, then for a correct guess of the coordinates p and q, at least one relevant partial solution is preserved in $\mathcal{L}^{(i)}$ at every stage of the algorithm.

▶ **Lemma 4.2.** *For all $0 \leq i \leq n$, for all $U \subseteq \mathcal{K}_{pq}$, and for all $0 \leq j \leq k$ such that $2j \leq (2k/3) + |U| + 2$, we have that $\mathcal{R}^{(i)}_{j,U}[\![pq]\!]$ is a $\alpha(j,U)$-representative family for $\mathcal{X}^{(i)}_{j,U}[\![pq]\!]$, where $\alpha(j,U) = (2k/3) - (2j - |U|) + 2$.*

Proof. Towards a proof, we need to show that for all $Y \subseteq U_p \cup U_q$ such that $|Y| \leq \alpha(j,U)$, if there exists $Z \in \mathcal{X}^{(i)}_{j,U}[\![pq]\!]$ such that $Y \cap Z = \emptyset$, then there also exists $Z^\star \in \mathcal{R}^{(i)}_{j,U}[\![pq]\!]$ such that $Z^\star \cap Y = \emptyset$. Note that since Y is assumed to be disjoint from Z, and $U \subseteq Z$ (since $Z \in \mathcal{X}^{(i)}_{j,U}[\![pq]\!]$), we have that $Y \cap U = \emptyset$. The proof of the lemma is by induction on i. The base case is when $i = 0$.

$$\mathcal{R}^{(0)}_{j,U}[\![pq]\!] = \mathcal{X}^{(0)}_{j,U}[\![pq]\!] = \begin{cases} \{\emptyset\} & \text{if } j = 0 \text{ and } U = \emptyset, \\ \emptyset & \text{Otherwise.} \end{cases}$$

Hence $\mathcal{R}^{(0)}_{j,U}[\![pq]\!]$ is $\alpha(j,U)$-representative family for $\mathcal{X}^{(0)}_{j,U}[\![pq]\!]$ for all $0 \leq j \leq k$ and $U \subseteq \mathcal{K}_{pq}$. The induction hypothesis states that $\mathcal{R}^{(i)}_{j,U}[\![pq]\!]$ is an $\alpha(j,U)$-representative family for $\mathcal{X}^{(i)}_{j,U}[\![pq]\!]$ for all $U \subseteq \mathcal{K}_{pq}$ and for all $0 \leq j \leq k$ such that $2j \leq (2k/3) + |U| + 2$. For the induction step, we will show that $\mathcal{R}^{(i+1)}_{j,U}[\![pq]\!]$ is an $\alpha(j,U)$-representative family for $\mathcal{X}^{(i+1)}_{j,U}[\![pq]\!]$ for all $U \subseteq \mathcal{K}_{pq}$ and for all $0 \leq j \leq k$ such that $2j \leq (2k/3) + |U| + 2$. As with the proof of Lemma 3.1, we note that the corner case when $j = 0$ is trivial. To this end, let $U \subseteq \mathcal{K}_{pq}$ and j be fixed. Let $Y \subseteq U_p \cup U_q$ such that $|Y| \leq \alpha(j,U)$. Further, suppose there exists a set $Z \in \mathcal{X}^{(i+1)}_{j,U}[\![pq]\!]$ such that $Z \cap Y = \emptyset$. Let $\mathcal{M}^Z$ be the matching associated with Z. Since $\mathcal{M}^Z$ is derived from $\mathcal{X}^{(i+1)}_{j,U}[\![pq]\!]$, note that $\lambda_r(\mathcal{M}^Z) \leq i + 1$. We distinguish two cases, depending on whether $\mathcal{M}^Z$ contains a set with c_{i+1} as the r-coordinate.

Algorithm 2: An Expansion Algorithm using Representative Families

Input: $(U_1, U_2, U_3, \mathcal{F}, \mathcal{K})$, where $\mathcal{F} \subseteq U_1 \times U_2 \times U_3$, $\mathcal{K} \subseteq \mathcal{F}$ is a 3D-Matching of size k
Output: A 3D-Matching of size $(k+1)$ if it exists, No otherwise.

for r **in** $\{1,2,3\}$ **do**
  $p \leftarrow \max\{\{1,2,3\} \setminus \{r\}\}$ and $q \leftarrow \min\{\{1,2,3\} \setminus \{r\}\}$
  $\mathcal{R}^{(0)}_{0,\emptyset}[\![pq]\!] \leftarrow \{\emptyset\}$, $\mathcal{L}^{(0)} \leftarrow \{\emptyset\}$
  $\mathcal{R}^{(0)}_{j,U}[\![pq]\!] \leftarrow \emptyset$, if $j \neq 0$ or $U \neq \emptyset$, for all $0 \leq j \leq k$ and $U \subseteq \mathcal{K}_{pq}$
  for $i \in \{1,2,\ldots,n\}$ **do**
    $\mathcal{L}^{(i)} \leftarrow \{\mathcal{M} \cup S \mid \mathcal{M} \in \mathcal{L}^{(i-1)}, S \in \mathcal{F}, \mathcal{M} \cap S = \emptyset, S[r] = i\} \cup \mathcal{L}^{(i-1)}$
    for $j \in \{0,1,2,\ldots,k+1\}$ **do**
      $\mathcal{L}^{(i)}_j \leftarrow \{\mathcal{M} \mid \mathcal{M} \in \mathcal{L}^{(i)}, |\mathcal{M}| = j\}$
      if $\mathcal{L}^{(i)}_{k+1} \neq \emptyset$ **then**
        return $\mathcal{M}$, where $\mathcal{M} \in \mathcal{L}^{(i)}_{k+1}$.
      $\gamma_{pq} \leftarrow \{(\mathcal{M}, \mathcal{M}_{pq}) \mid \mathcal{M}$ is a matching in $\mathcal{L}^{(i)}_j\}$;
      for $U \subseteq \mathcal{K}_{pq}$ **do**
        if $2j > (2/3)k + |U| + 2$ **then**
          Continue;
        $\mathcal{P}^{(i)}_{j,U}[\![pq]\!] \leftarrow \{\mathcal{M}_{pq} \mid \mathcal{M} \in \mathcal{L}^{(i)}_j$ and $\mathcal{M}_{pq} \cap \mathcal{K}_{pq} = U\}$
        $\mathcal{D}^{(i)}_{j,U}[\![pq]\!] \leftarrow \{X \setminus U \mid X \in \mathcal{P}^{(i)}_{j,U}[\![pq]\!]\}$
        Let $\mathcal{T}^{(i)}_{j,U}[\![pq]\!] \subseteq^{\alpha(j,U)}_{rep} \mathcal{D}^{(i)}_{j,U}[\![pq]\!]$, where $\alpha(j,U) = (2k/3) - (2j - |U|) + 2$
        Let $\mathcal{R}^{(i)}_{j,U}[\![pq]\!] \leftarrow \{X \cup U \mid X \in \mathcal{T}^{(i)}_{j,U}[\![pq]\!]\}$
      $\mathcal{L}^{(i)}_j \leftarrow \bigcup_{U \subseteq \mathcal{K}_{pq}} \{\gamma^{\dagger}_{pq}(\mathcal{M}_{pq}) \mid \mathcal{M}_{pq} \in \mathcal{R}^{(i)}_{j,U}[\![pq]\!]\}$
    $\mathcal{L}^{(i)} \leftarrow \bigcup_{j=1}^{k} \mathcal{L}^{(i)}_j$
return No;

Case 1. $\mathcal{M}^Z$ contains a set with c_{i+1} as the r-coordinate.

Let $S \in \mathcal{M}^Z$ be such that $c_{i+1} \in S$. Define the smaller matching $\mathcal{M}^{Z \setminus S} := \mathcal{M}^Z \setminus S$. Notice that $|\mathcal{M}^{Z \setminus S}| = j - 1$ and $\lambda_r(\mathcal{M}^{Z \setminus S}) \leq i$. Let $U_S := U \cap S$ and $U' = U \setminus U_S$. Therefore, $\mathcal{M}^{Z \setminus S} \in \mathcal{Q}^{(i)}_{j-1,U'}[\![pq]\!]$ and $\mathcal{M}^{Z \setminus S}_{pq} \in \mathcal{X}^{(i)}_{j-1,U'}[\![pq]\!]$. Let $A = \mathcal{M}^{Z \setminus S}_{pq}$ and $Y^S = Y \cup (\{S[p], S[q]\} \setminus U_S)$. It is easy to check that $A \cap Y^S = \emptyset$. We claim that $|Y^S| \leq \alpha(j-1, U')$:

$$|Y^S| = |Y| + 2 - |U_S| \leq (2k/3) - 2j + |U| + 2 + 2 - |U_S| = \alpha(j, U')$$

We also claim that $2(j-1) \leq (2k/3) + |U'| + 2$. Suppose not, then $2j > (2k/3) + |U'| + 4 > (2k/3) + |U| + 2$ (Since $|U| = |U'| + |U_S| \leq |U'| + 2$). This contradicts the assumption that $2j \leq (2k/3) + |U| + 2$. Hence, we are now in a situation where $A \in \mathcal{X}^{(i)}_{j-1,U'}[\![pq]\!]$, $|Y^S| \leq \alpha(j-1, U')$, $A \cap Y^S = \emptyset$ and $2(j-1) \leq (2k/3) + |U'| + 2$. By the induction hypothesis, we have that $\mathcal{R}^{(i)}_{j-1,U'}[\![pq]\!]$ contains a $\alpha(j-1, U')$-representative of A with respect to Y^S. Let us denote this representative by B. Note that $B \cap Y^S = \emptyset$ by definition. Also note that $B \cap U_S = \emptyset$ because $B \cap \mathcal{K}_{pq} = U'$. Let $\mathcal{M}^B$ be the matching associated with B. Since $B \in \mathcal{R}^{(i)}_{j-1,U'}[\![pq]\!]$, we have that $\mathcal{M}^B \in \mathcal{L}^{(i)}_{j-1}$. Since the p,q-coordinates of S are disjoint from B and the r-coordinate of S is c_{i+1}, we have that $\mathcal{M}^B \cap S = \emptyset$. Thus, the matching $\mathcal{M}^B \cup S \in \mathcal{L}^{(i+1)}_j$. Let us denote this matching by $\mathcal{M}^{B|S}$. It is easy to check that

$\mathcal{M}_{pq}^{B|S} \cap \mathcal{K}_{pq} = U$, and correspondingly, $\mathcal{M}_{pq}^{B|S} \in \mathcal{P}_{j,U}^{(i+1)}[\![pq]\!]$. Let $W := \mathcal{M}_{pq}^{B|S} \setminus U$. Note that $W \in \mathcal{D}_{j,U}^{(i+1)}[\![pq]\!]$ and $W \cap Y = \emptyset$. Since $\mathcal{T}_{j,U}^{(i+1)}[\![pq]\!]$ is an $\alpha(j,U)$-representative family for $\mathcal{D}_{j,U}^{(i+1)}[\![pq]\!]$, we have that $\mathcal{T}_{j,U}^{(i+1)}[\![pq]\!]$ contains an $\alpha(j,U)$-representative $W^\star$ for W with respect to Y. We now define $Z^\star$ to be $W^\star \cup U$. Note that $Z^\star \in \mathcal{R}_{j,U}^{(i+1)}[\![pq]\!]$. Since $W^\star$ is disjoint from Y by definition, and we know that $U \cap Y = \emptyset$, we conclude that $Z^\star$ is the desired representative.

Case 2. $\mathcal{M}^Z$ contains no set with c_{i+1} as the r-coordinate.

Recall that $Z \in \mathcal{X}_{j,U}^{(i+1)}[\![pq]\!]$, and therefore, $Z \cap \mathcal{K}_{pq} = U$. Since we additionally have that $\lambda(\mathcal{M}^Z) \leq i$, $\mathcal{M}^Z \in \mathcal{Q}_{j,U}^{(i)}[\![pq]\!]$. Consequently, $\mathcal{M}_{pq}^Z \in \mathcal{X}_{j,U}^{(i)}[\![pq]\!]$. By induction hypothesis there exists a $\alpha(j,|U|)$-representative, B, for $\mathcal{M}_{pq}^Z$ with respect to Y, in $\mathcal{R}_{j,U}^{(i)}[\![pq]\!]$ Note that $B \cap Y = \emptyset$. Thus the matching associated with B, say $\mathcal{M}^B$, belongs to $\mathcal{L}_j^{(i)}$. Further, since $\mathcal{L}_j^{(i+1)} \supseteq \mathcal{L}_j^{(i)}$, we have that $\mathcal{M}^B \in \mathcal{L}_j^{(i+1)}$. As before, this implies that $\mathcal{M}_{pq}^B \in \mathcal{P}_{j,U}^{(i+1)}[\![pq]\!]$. Note that this may be equivalently stated as $B \in \mathcal{P}_{j,U}^{(i+1)}[\![pq]\!]$. The rest of the argument is identical to the last part of the previous case, leading to a representative of Z as desired. ◀

We now establish the correctness of Algorithm 2, and then establish the running time.

▶ **Lemma 4.3.** *Let $(U_1, U_2, U_3, \mathcal{F}, \mathcal{K})$ be a* Yes*-instance of* 3D-Matching Improvement. *Then, Algorithm 2 successfully computes a 3D-Matching of size $(k+1)$.*

Proof. Since $(U_1, U_2, U_3, \mathcal{F}, \mathcal{K})$ be a Yes-instance of 3D-Matching Improvement, by Lemma 4.1, we have that there exists a matching $\mathcal{M}$ such that $|\mathcal{M}| = (k+1)$ and there exists $p, q \in \{1,2,3\}$ such that $|\mathcal{M}_{pq} \cap \mathcal{K}_{pq}| \geq (4k/3)$. Let $r \in \{1,2,3\}$ such that $r \neq p, q$. Let $\mathcal{M} = \{S_1, S_2, \ldots, S_k, S_{k+1}\}$. Suppose $\lambda_r(\mathcal{M}) = i$. Without loss of generality, let $S_{k+1}[r] = c_i$. Finally, let $\mathcal{M}'$ denote the matching $\mathcal{M} \setminus \{S_{k+1}\}$. Let $U = \mathcal{M}_{pq} \cap \mathcal{K}_{pq}$, $U_S = S_{k+1} \cap \mathcal{K}_{pq}$ and $U' = U \setminus U_S$. Note that $\mathcal{M}'_{pq} \in \mathcal{X}_{k,U'}^{(i-1)}$ and $2k \leq (2k/3) + |U'| + 2$ (since $|U'| \geq (4k/3) - 2$). Now consider the value $\alpha(k, U')$.

$$\alpha(k,U') = (2k/3) - 2k + |U'| + 2 \geq (2k/3) - 2k + (4k/3 - |U_S|) + 2 = 2 - |U_S|$$

By Lemma 4.2, we have that $\mathcal{R}_{k,U'}^{(i-1)}$ is a $\alpha(k,U')$-representative family for $\mathcal{X}_{k,U'}^{(i-1)}$, where $\alpha(k,U') \geq 2 - |U_S|$. Now consider $Y := \{S_k[p], S_k[q]\} \setminus U_S$. Note that $|Y| = 2 - |U_S|$. Since $\mathcal{M}'_{pq} \in \mathcal{X}_{k,U'}^{(i-1)}$ is clearly disjoint from Y, by the definition of a representative family, there exists some set, say $X \in \mathcal{R}_{j,U'}^{(i-1)}$, which is also disjoint from Y. Let $\mathcal{M}^\star$ be the matching associated with X, that is, let $\mathcal{M}^\star := \gamma_{pq}^\dagger(X)$. It is easily checked that $\mathcal{M}^\star \cup S_{k+1}$ is included in $\mathcal{L}_{k+1}^{(i)}$ and is therefore returned as a matching of size $(k+1)$ in Step 9 of Algorithm 2. ◀

▶ **Lemma 4.4 ($\star$).** *The running time of Algorithm 2 is bounded by $\mathcal{O}^\star(2.0034^{3k})$.*

Putting together Lemmas 4.2, 4.3, and 4.4, we have the following theorem.

▶ **Theorem 4.5.** *The* 3D-Matching *problem can be solved in time $\mathcal{O}^\star(2.0034^{3k})$.*

5 Conclusions

We have demonstrated that when combined with techniques like iterative expansion, representative families can be used to a great advantage inside dynamic programming routines.

We were able to significantly improve on the state of the art for the of r-Dimensional Matching problem (even with weights), and we further improved this running time for the special case of the 3D-Matching problem. We already know that representative families can be applied to a variety of other packing problems. The naive approach, however, yields only incremental improvements in running times. The more dramatic improvements obtained for the matching problems considered in this work were due to the additional structure in the input: the partitioned universe was exploited in the dynamic programming.

It will be very interesting to see what other techniques can be combined with this method to obtain improved algorithms for other packing questions, such as the classic Set Packing problem, or the problem of packing paths of length two from an underlying graph.

References

1 Alon, Yuster, and Zwick. Color-coding. *JACM: Journal of the ACM*, 42, 1995.
2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, abs/1007.1161, 2010.
3 Jianer Chen, Yang Liu, Songjian Lu, Sing-Hoi Sze, and Fenghui Zhang. Iterative expansion and color coding: An improved algorithm for 3D-matching. *ACM Transactions on Algorithms*, 8(1):6, 2012.
4 Shenshi Chen and Zhixiang Chen. Faster deterministic algorithms for packing, matching and t-dominating set problems. *CoRR*, abs/1306.3602, 2013.
5 R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.
6 R. G. Downey, M. R. Fellows, and M. Koblitz. Techniques for exponential parameterized reductions in vertex set problems. (Unpublished, reported in [5], §8.3).
7 Michael R. Fellows, Christian Knauer, Naomi Nishimura, Prabhakar Ragde, Frances A. Rosamond, Ulrike Stege, Dimitrios M. Thilikos, and Sue Whitesides. Faster fixed-parameter tractable algorithms for matching and packing problems. *Algorithmica*, 52(2):167–176, 2008.
8 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. *CoRR*, abs/1304.4626, 2013.
9 M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, San Francisco, 1979.
10 I. Koutis. A faster parameterized algorithm for set packing. *Information Processing Letters*, 94:7–9, 2005.
11 I. Koutis and R. Williams. Limits and applications of group algebras for parameterized problems. In *Proceedings of Automata, Languages and Programming, 36th International Colloquium, ICALP*, volume 5555 of *LNCS*, pages 653–664. Springer, 2009.
12 Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Proceedings of Automata, Languages and Programming, 35th International Colloquium, ICALP*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008.
13 Y. Liu, S. Lu, J. Chen, and S.-H. Sze. Greedy localization and color-coding: improved matching and packing algorithms. In *International Workshop on Parameterized and Exact Computation IWPEC*, volume 4169 of *LNCS*, pages 84–95. Springer, 2006.
14 Dániel Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci*, 410(44):4471–4479, 2009.
15 B. Monien. How to find long paths efficiently. *Ann. Discrete Math.*, 25:239–254, 1985.

Distributed and Parallel Algorithms for Set Cover Problems with Small Neighborhood Covers *

Archita Agarwal[1], Venkatesan T. Chakaravarthy[1], Anamitra R. Choudhury[2], Sambuddha Roy[1], and Yogish Sabharwal[1]

1 **IBM Research Lab, New Delhi, India**
{archiaga,vechakra,sambuddha,ysabharwal}@in.ibm.com
2 **IIT Delhi, New Delhi, India**
anamitra@cse.iitd.ac.in

Abstract

In this paper, we study a class of set cover problems that satisfy a special property which we call the *small neighborhood cover* property. This class encompasses several well-studied problems including vertex cover, interval cover, bag interval cover and tree cover. We design unified distributed and parallel algorithms that can handle any set cover problem falling under the above framework and yield constant factor approximations. These algorithms run in polylogarithmic communication rounds in the distributed setting and are in NC, in the parallel setting.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation algorithms, set cover problem, tree cover

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.249

1 Introduction

In the classical set cover problem, we are given a set system $\langle E, \mathcal{S} \rangle$, where E is a *universe* consisting of m *elements* and $\mathcal{S}$ is a collection of n subsets of E. Each set $S \in \mathcal{S}$ has cost $w(S)$ associated with it. The goal is to select a collection of sets $\mathcal{R} \subseteq \mathcal{S}$ having the minimum aggregate cost such that every element is included in at least one of the sets found in $\mathcal{R}$.

There are two well-known classes of approximation algorithms for the set cover problem [17]. The first class of algorithms have an approximation ratio of $O(\log \Delta)$, where Δ is the maximum cardinality of the sets in $\mathcal{S}$. The second class of algorithms have an approximation ratio of f, where f is the *frequency parameter* which is the maximum number of sets of $\mathcal{S}$ that any element belongs to. The above approximation ratios are nearly optimal [6, 16, 7]. In general the parameters Δ and f can be arbitrary and so the above algorithms do not yield constant factor approximations. The goal of this paper is to develop parallel/distributed constant factor approximation algorithms for certain special cases of the problem.

In the parallel setting, we shall use the NC model of computation and its randomized version RNC. Under this model, Rajagopalan and Vazirani [15] presented a randomized parallel $O(\log m)$-approximation algorithm for the general set cover problem. Under the same model, Khuller et al. [10] presented a $(f+\epsilon)$-approximation algorithm for any constant frequency parameter f and $\epsilon > 0$.

* Full version of the paper available as Arxiv preprint.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 249–261

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

In the distributed setting, we shall adopt a natural communication model which has also been used in prior work. In this model, there is a processor for every element and there is a communication link between any two elements e_1 and e_2, if and only if both e_1 and e_2 belong to some common set $S \in \mathcal{S}$. We shall view the element itself as the processor. Each element has a unique ID and knows all the sets to which it belongs. We shall assume the standard synchronous, message passing model. The algorithm proceeds in multiple communication rounds, where in each round an element can send a message to each of its neighbors in the communication network. We allow each element to perform a polynomial amount of processing in each round and the messages to be of polynomial size. We are interested in two performance measures: (i) the approximation ratio achieved by the algorithm; and (ii) the number of communication rounds. Ideally a distributed algorithm should have polylogarithmic communication rounds. Under the above distributed model, Kuhn et al. [12] and Koufogiannakis and Young [11] presented distributed algorithms for the general set cover problem with approximation ratios of $O(\log \Delta)$ and f, respectively; both the algorithms run in polylogarithmic communication rounds.

There are special cases of the set cover problem wherein both Δ and f are arbitrary, which nevertheless admit constant factor approximation algorithms. In this paper, we study one such class of problems satisfying a criteria that we call the *small neighborhood cover* property (SNC-property). This class encompasses several well-studied problems such as vertex cover, interval cover and tree cover. Furthermore, the class subsumes set cover problems with a constant frequency parameter f. Our results generalize the known constant factor approximation algorithm for the latter class.

Our goal is to design unified distributed and parallel algorithms that can handle any set cover problem falling under the above framework. In order to provide an intuition of the SNC-property, we next present an informal (and slightly imprecise) description of the property. We then illustrate the concept using some example problems and intuitively show why these problems fall under the framework. The body of the paper will present the precise definition of SNC set systems.

SNC Property. Fix an integer constant $\tau \geq 1$. We say that two elements are neighbors, if some $S \in \mathcal{S}$ contains both of them. The neighborhood of an element is defined to be the set of all its neighbors (including itself). We say that an element $e \in E$ is a τ-SNC element, if there exist at most τ sets that cover the neighborhood of e. The given set system is said to have the τ-SNC property, if for any subset $X \subseteq E$, the set system restricted[1] to X contains a τ-SNC element. The requirement that every restriction has a τ-SNC element will be useful in solving the problem iteratively.

Example Problems. We next present some example τ-SNC set cover problems.

Vertex Cover: Given a graph G, we can construct a set system by taking the edges as the elements and the vertices as sets. In this setup, an element belongs to only two sets and hence, the set systems defined by the vertex cover problem satisfy the 2-SNC property. In general, set cover problems having a constant frequency parameter f would induce τ-SNC set systems with $\tau = f$.

Interval Cover: In this problem, we are given a timeline divided into some m discrete timeslots $1, 2, \ldots, m$. The input includes a set of intervals $\mathcal{I}$, where each interval $I \in \mathcal{I}$ is specified by a range $[s(I), e(I)]$, where $s(I)$ and $e(I)$ are the starting and ending points of I.

[1] the restricted set system is $\langle X, \mathcal{S}' \rangle$, where $\mathcal{S}' = \{S \cap X \ : \ S \in \mathcal{S}\}$

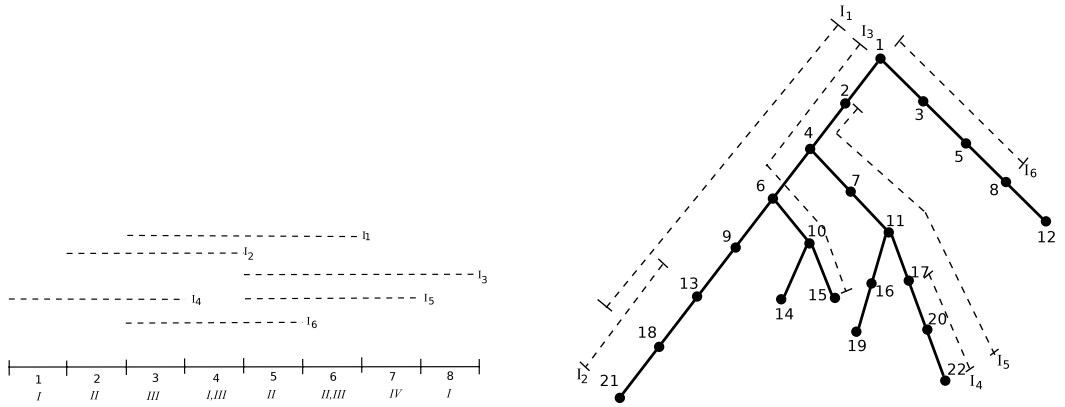


Figure 1 Illustration for example problems.

Each interval I also has an associated cost $w(I)$. We say that an interval I covers a timeslot $1 \leq e \leq m$, if $e \in [s(I), e(I)]$. The goal is to find a collection of intervals having minimum aggregate cost such that every timeslot t is covered by at least one interval in the collection. We can view the problem as a set cover instance by taking the timeslots to be the elements and taking each interval $I \in \mathcal{I}$ as a set consisting of the timeslots covered by I. See the picture on the left in Figure 1 for an illustration (ignore the Roman numerals). Consider any timeslot e and let $\mathcal{Q} \subseteq \mathcal{I}$ be the set of intervals covering e. Among the intervals in $\mathcal{Q}$, the interval I_l with the minimum starting point and the interval I_r having the maximum ending point can cover the neighborhood of e (resolving ties arbitrarily). For example, for timeslot 3, $I_l = I_4$ and $I_r = I_1$. Hence, the set systems defined by the interval cover problem satisfy the 2-SNC property.

Tree Cover Problem: In the tree cover problem, we are given a *rooted* tree $T = (V, H)$. The input includes a set of intervals $\mathcal{I}$, where each interval is specified as a pair of nodes $\langle u, v \rangle$ such that u is an ancestor of v. The interval I can be visualized as the path from u to v. The interval is said to cover an edge $e \in H$, if e is found along the above path. Each interval I has a cost $w(I)$ associated with it. The goal is to find a collection of intervals of minimum cost covering all the edges. We can view the problem as a set cover instance by taking the edges to the elements and taking the intervals as sets. It is not difficult to see that the tree cover problem generalizes the interval cover problem. See the picture on the right in Figure 1 for an illustration. Consider any leaf edge e. Let $\mathcal{Q}$ be a set of intervals covering the edge e. Among the intervals in $\mathcal{Q}$, let $\widehat{I}$ be the interval extending the most towards to the root. Note that $\widehat{I}$ covers the neighborhood of e. For example, in the figure, for the leaf edge $\langle 20, 22 \rangle$, the interval I_5 will serve as $\widehat{I}$. Thus, any leaf edge satisfies the 1-SNC property. It is not difficult to see that any restriction will also contain an element satisfying the 1-SNC property. Hence, the set systems defined by the tree cover problem satisfy the 1-SNC property.

Bag Interval Cover Problem: This problem generalizes both vertex cover and interval cover problems. The input consists of a timeline divided into discrete timeslots $\{1, 2, \ldots, T\}$. We have a set of n intervals $\mathcal{I}$. Each interval $I \in \mathcal{I}$ has a starting timeslot $s(I)$, an ending timeslot $e(I)$ and a weight $w(I)$. Timeslots are grouped into m *bags* $B_1, B_2, \ldots, B_m$; a timeslot may belong to more than one bag. The interval I is said to cover a bag B_i, if it spans at least one timeslot from the bag B_i. The goal is to find a collection of intervals having minimum aggregate cost such that each bag is covered by some interval in the collection.

The *girth* of the system is defined to be the maximum cardinality of any bag and it is denoted g; Viewed as a set cover problem, each bag will correspond to an element and each interval will correspond to a set. See the picture on the left in Figure 1 for an illustration. The bag number are shown in Roman numerals. For instance, Bag I consists of timeslots $\{1, 4, 8\}$. The girth of the system is 3.

Consider any element (bag) B containing timeslots $\{e_1, e_2, \ldots, e_r\}$ (with $r \leq g$). For each timeslot e_i, among the intervals spanning e_i select the intervals having the minimum staring point and the maximum ending point. This set of $2r$ intervals can cover the neighborhood of B. Thus any element satisfies the $2g$-SNC property. Hence, the set systems defined by the bag interval cover problem satisfies the $2g$-SNC property.

Layer Decomposition. An important concept that will determine the running time of our algorithms is that of layer decomposition. We present an intuitive description of layer decomposition. The formal definition will be presented in the body of the paper.

Consider a set system $\langle E, \mathcal{S} \rangle$ satisfying the τ-SNC property for some constant τ. Let Z_1 be the set of all τ-SNC elements in the given set system. Let Z_2 be the set of τ-SNC elements in the set system obtained by restricting to $E - Z_1$. Proceeding this way, for $k \geq 2$, let Z_k be the set of τ-SNC elements in the set system obtained by restricting to $E - (Z_1 \cup Z_2 \cup \cdots Z_{k-1})$. We continue the process until no more elements are left. Let L be the number of iterations taken by this process. The sequence $Z_1, Z_2, \ldots, Z_L$ is called the *layer decomposition* of the set system $\langle E, \mathcal{S} \rangle$. Each set Z_k is called a *layer*. The number of layers L is called the *decomposition length*.

In this paper, we will only focus on set cover problems having logarithmic decomposition length and derive distributed/parallel algorithms with polylogarithmic rounds/running-time for such problems. We note that there are set cover problems that induce τ-SNC systems with a constant τ, but having arbitrary decomposition length. One example is provided by the priority interval cover problem studied by Chakrabarty et al. [5] and Chakaravarthy et al. [4]. This problem is a generalization of the interval cover problem. In this case, the set systems satisfy the τ-SNC property with $\tau = 2$, but the decomposition length would equal the number of priorities, which can be arbitrary. The details are deferred to the full version.

We next study the decomposition length for our example problems. In the case of vertex cover, interval cover and bag interval cover problems, we saw that all the elements satisfy the τ-SNC property in the given system $\langle E, \mathcal{S} \rangle$ itself. Hence, the decomposition length of these set systems is one. In the tree cover problem, recall that all the leaf edges in the given tree T are 1-SNC elements. Thus, all the leaf edges will belong to the first layer Z_1. Once these leaf edges are removed, the leaf edges in the remaining tree will belong to the second layer Z_2. Proceeding this way, we will get a layer decomposition in which the number of layers will be the same as the depth of the tree; later, we describe how to reduce the decomposition length to be $O(\log m)$.

Our Results. In this paper, we introduce the concept of τ-SNC property. We note that all the example problems considered earlier can be solved optimally or within constant factors using the primal-dual paradigm. All these algorithms have certain common ingredients; these are abstracted by τ-SNC framework. We present three algorithms for the set cover problem on τ-SNC set systems.

- A simple sequential τ-approximation algorithm.
- A distributed τ-approximation algorithm for τ-SNC set systems of logarithmic decomposition length. The algorithm is randomized and uses $O(\log^2 m)$ communication rounds.

- A parallel $(1 + 8\tau^2)$-approximation algorithm for τ-SNC set systems of logarithmic decomposition length. The algorithm can be implemented in NC.

Our algorithms have the following salient features:

- They provide unified constant factor approximations for set cover problems falling under the τ-SNC framework with logarithmic decomposition length, in both distributed and parallel settings.
- A surprising and interesting characteristic of these algorithms is that they are model independent. Meaning, they only require the set system as input and do not need the underlying model defining the set system. For instance, in the tree cover problem, the algorithms do no need the structure of the tree as input. At a technical level, we show that the layer decomposition can be constructed by considering only the local neighborhood information; this fact is crucial in a distributed setting.

Regarding the example problems, we saw that in case of the vertex cover, interval cover and bag-interval cover problems, the decomposition length is one. Thus our parallel and distributed algorithms will apply to these problems. The case of tree cover problem is more interesting. As we observed earlier, the set systems arising from the tree cover problem are 1-SNC set systems, however the the decomposition length is the same as the depth of the tree, which could be as large as $\Omega(m)$ (where m is the number of edges). Hence our parallel and distributed algorithms cannot be applied to this case. However, we shall show that it is possible to reduce the decomposition length to $O(\log m)$, if we settle for a slightly higher SNC parameter of $\tau = 2$:

- We prove that the set systems defined by the tree cover problem satisfy the 2-SNC property with decomposition length $O(\log m)$.

In other words, the tree cover problem instances induce a 1-SNC set systems of arbitrary decomposition length, as well as 2-SNC set systems of decomposition length $O(\log m)$. Using the above fact, we can apply our parallel and distributed algorithms and obtain constant factor apporoximations.

It is easy to see that for any constant f, set systems with frequency parameter f satisfy the τ-SNC property, with $\tau = f$. Dinur et al. [6] proved that for any $f \geq 3$, it is NP-hard to approximate the set cover problem within a factor of $(f - 1 - \epsilon)$, for any $\epsilon > 0$. Thus, the approximation ratio of the sequential and distributed algorithms are nearly optimal. In the parallel setting, we present an algorithm with an approximation ratio of $(1 + 8\tau^2)$. Improving the approximation ratio is an interesting open problem.

While this is the first paper to consider the general τ-SNC framework, the specific example problems have been studied in the sequential, parallel and distributed settings. Improved algorithms are known in specific cases. We next present a brief survey of such prior work and provide a comparison to our results.

Comparison to Prior Work on Example Problems. For the vertex cover problem, sequential 2-approximation algorithms are well known [17]. In the parallel setting, Khuller et al. [10] presented a parallel NC algorithm having approximation ratio of $2+\epsilon$, for any $\epsilon > 0$ (see also [8]). Koufogiannakis and Young [11] presented the first parallel algorithm with approximation ratio of 2. Their algorithm is randomized and runs in RNC. The above algorithms can also be implemented in the distributed setting (see also [9]).

The interval cover problem can be solved optimally in the sequential setting via dynamic programming. Bertossi [3] presented an optimal parallel (NC) algorithm, which can also handle the more general case of circular arc covering. However, their algorithm requires the

underlying model (i.e., the timeline and intervals) explicitly as input. We are not familiar with prior work on the problem in the distributed setting.

Chakrabarty et al. [5] study the tree cover problem and its generalizations under the sequential setting. In this setting, the problem can be solved optimally via dynamic programming or the primal-dual paradigm. Furthermore, the constraint matrices defined by the problem are totally unimodular (see [5]). We are not familiar with any prior work on parallel/distributed algorithms for this problem. For this problem $\tau = 2$ and so, our sequential/distributed algorithms provide an approximation ratio of 2. The parallel algorithm has an approximation factor of 33. However, we note that one of the reasons for the high ratio is that the algorithm is oblivious to the underlying model. When the model (i.e., the tree and the paths) is given explicitly as part of the input, we can improve the approximation ratio to 17; a discussion of this improvement is deferred to the full version of the paper.

To the best of our knowledge, the bag interval cover problem has not been considered before. However, the notion of bag constraints has been considered in the related context of interval packing problems (see [1, 2]). Covering integer programs (CIP) generalize the set cover problem. These are well studied in both sequential and distributed settings (see [11, 5], and references therein).

Proof Techniques. All the algorithms in the paper utilize the primal-dual paradigm. The sequential algorithm is fairly straightforward and it is similar to that of the primal-dual algorithm f-approximation algorithm for the set cover problem. The latter algorithm works by constructing a maximal feasible solution to the dual which would automatically yield an f-approximate integral primal solution. Our problem requires two additional ingredients. The first is that an arbitraty maximal dual solution would not suffice. Instead, the solution needs be constructed in accordance with the layered decomposition. Secondly, a maximal dual solution would not automatically yield a τ-approximate integral primal solution. A reverse delete phase is also needed. In this context, we present a polynomial time algorithm for computing the layer decomposition of the given set system, which can also be implemented in both parallel and distributed settings.

In the distributed setting, the only issue is that the above steps need to be performed within polylogarithmic number of rounds. We address the issue by grouping the elements based on the Linial-Saks decomposition [13] of the communication network.

The parallel algorithm is more involved and forms the main technical component of the paper. For a general set system, Khuller et al. [10] (see also [8]) present a parallel procedure for computing nearly maximal dual solution with maximality parameter of $(1-\epsilon)$, using the idea of raising several dual variables simultaneously. However, the parallel running of the procedure is $O(f \log(1/\epsilon) \log m)$, where f is the frequency parameter. In our problems, the parameter f could be arbitrary and the above running time is not satisfactory. We present a procedure that produces a near maximal solution with maximality parameter $1/8$. While the maximality parameter is worse compared to prior work, the running time of our procedure is independent of f. This procedure could be of independent interest. The procedure is similar in spirit to that of Khuller et al., but the analysis for bounding the number of iteration takes a different approach.

As mentioned earlier, our setting requires an additional reverse delete phase, whose parallelization poses interesting technical issues. Our procedure executes the phase by processing the layer decomposition in a zig-zag manner. In iteration i, the procedure processes layer i and performs the reverse delete for the particular layer. However, this involves revisiting the older layers $1, 2, \ldots, i-1$. Each step involves computing the maximal independent set of a

suitable graph, for which we utilize the parallel algorithm due to Luby [14]. The overall number of steps would be $O(L^2)$ (where L is the decomposition length) and the approximation ratio is $8\tau^2$ (as against the ratio τ achieved by the sequential/distributed algorithms).

Organization. Due to lack of space, the paper presents only the parallel algorithm. The sequential and distributed algorithms are discussed the full version. Similarly, algorithms for computing layered decomposition and logarithmic length decompositions for the tree cover problem are deferred to the full version.

2 Preliminaries

In this section, we present the formal definition of the τ-SNC property and related concepts. We also present algorithms for computing the layer decomposition for a given τ-SNC set system.

τ-SNC Element: Fix an integer constant $\tau \geq 1$. Consider a subset of elements $X \subseteq E$ and an element $e \in X$. Let $\mathcal{Q} \subseteq \mathcal{S}$ be the collection of all sets that contain e. The element e is said to be a *τ-SNC element within X*, if for any $\mathcal{P} \subseteq \mathcal{Q}$, there exist at most r sets $S_1, S_2, \ldots, S_r \in \mathcal{P}$ (with $r \leq \tau$) such that every element in $e \in X$ covered by $\mathcal{P}$ is also covered by one of the τ sets: $\bigcup_{S \in \mathcal{P}} S \cap X = \bigcup_{i=1}^{r} S_i \cap X$. Note that the τ sets must be selected from the collection $\mathcal{P}$. The property is trivially true if $|\mathcal{P}| \leq \tau$, but it becomes interesting if $|\mathcal{P}| \geq \tau + 1$.

τ-SNC Set System: The given set system $\langle E, \mathcal{S} \rangle$ is said to be a *τ-SNC set system* if for every subset of elements $X \subseteq E$, there exists an element $e \in X$ which is a τ-SNC element within X. The set system is said to be a *total τ-SNC set system*, if for every subset $X \subseteq E$, every $e \in X$ is a τ-SNC element within X. The following property is easy to verify.

▶ **Proposition 1.** *If an element $e \in X$ is a τ-SNC element within X, then for any $Y \subseteq X$ such that $e \in Y$, e is also a τ-SNC element within Y.*

However, the converse of the above statement may not be true. Namely, an element e may be a τ-SNC element within a set X, but it may not be a τ-SNC element within a superset $Y \supset X$. To see this, suppose $\mathcal{P}$ is a collection of sets such that every $S \in \mathcal{P}$ contains e. The collection $\mathcal{P}$ may cover an element $x \in Y - X$, which may not be covered by some τ sets of $\mathcal{P}$ that cover the neighborhood of e within X.

Layer Decomposition: Consider a τ-SNC set system $\langle E, \mathcal{S} \rangle$. The notion of layer decomposition is defined via an iterative process, as described in the introduction. Let Z_1 be the set of τ-SNC elements within E. For $k \geq 2$, let Z_k be the set of τ-SNC elements within $E - (Z_1 \cup Z_2 \cup \cdots \cup Z_{k-1})$ We terminate the process when there are no elements left. Let L be the number of iterations taken by the process. The sequence $Z_1, Z_2, \ldots, Z_L$ is called the *layer decomposition* of the given set system. Each set Z_i is called a layer and L is called the *decomposition length* We consider Z_1 to be the left-most layer and Z_L as the right-most layer.

Computing Layer Decompositions: As part of our algorithms, we will need a procedure for computing the layer decomposition of a given τ-SNC set system. The following lemma provides such a procedure. We defer the proof to the full version of the paper.

▶ **Lemma 2.** *There exists a procedure for computing the layer decomposition of a given τ-SNC set system. In the sequential setting, it can be implemented in polynomial time. In the distributed setting, it can be implemented in $O(L)$ communication rounds. In the parallel setting, the algorithm takes L iterations each of which can be implemented in NC.*

Remark: Notice that any τ_1-SNC set system is also a τ_2-SNC set system for any $\tau_2 \geq \tau_1$. The decomposition length of the system will depend on the choice of τ. The procedure stated in the lemma will produce the layer decomposition corresponding to the value of τ provided as input to the procedure.

3 Parallel Algorithm for τ-SNC Set Systems

In this section, we present a parallel algorithm for the set cover problem on τ-SNC set systems with logarithmic decomposition length. The approximation ratio of the algorithm is $(1+8\tau^2)$. The algorithm uses the primal-dual paradigm. The primal and the dual for the input set system $\langle E, \mathcal{S}\rangle$ are given below.

$$\begin{aligned} \min \quad & \sum_{S\in\mathcal{S}} x(S)\cdot w(S) \\ \sum_{S\in\mathcal{S}\,:\,e\in S} & x(S) \geq 1 \quad (\forall e\in E) \end{aligned} \qquad\qquad \begin{aligned} \max \quad & \sum_{e\in E} \alpha(e) \\ \sum_{e\in S} & \alpha(e) \leq w(S) \quad (\forall S\in\mathcal{S}) \end{aligned}$$

The primal LP includes a variable $x(S)$ for each set $S \in \mathcal{S}$ and a constraint for each element $e \in E$. The dual includes a variable $\alpha(e)$ for each element $e \in E$ (corresponding to the primal constraint) and a constraint for each set $S \in \mathcal{S}$ (corresponding to the primal variable). The primal and the dual would also include the non-negativity constraints $x(S) \geq 0$ and $\alpha(e) \geq 0$. The algorithm would proceed in two phases, a forward phase and a reverse-delete phase. A pseudocode for the algorithm can be found in full version.

3.1 Forward Phase

Consider a pair of solutions $\langle \mathcal{A}, \alpha\rangle$, where $\mathcal{A} \subseteq \mathcal{S}$ is a feasible cover and α is a dual feasible solution. For a constant $\lambda \in [0, 1]$, we say that the above pair is λ-*maximal*, if for any $S \in \mathcal{A}$, the corresponding dual constraint is approximately tight:

$$\sum_{e\in S} \alpha(e) \geq \lambda \cdot w(S) \tag{1}$$

In the forward phase, we shall construct a $(1/8)$-maximal solution. The procedure runs in $O(L \cdot [\log m + \log \frac{w_{\max}}{w_{\min}}])$ iterations, where each iteration can be implemented in NC, where L is the decomposition length. As we shall see, via a standard preprocessing trick, we can ensure that $w_{\max}/w_{\min}$ is bounded by $O(m)$. The process would increase the approximation ratio by an additive factor of one. Thus when L is logarithmic, the procedure runs in NC. Furthermore, our procedure would satisfy certain additional properties to be specified later.

Remark: While we shall describe our algorithm for the specific scenario of τ-SNC set systems, it can handle arbitrary set systems and produce $(1/8)$-maximal solutions in $O(\log m + \log(w_{\max}/w_{\min}))$ iterations. The problem of finding such approximately maximal solutions in parallel for general set systems is of independent interest. Khuller et al.[10] (see also [8]) presented procedure for computing $(1-\epsilon)$-maximal solutions, for any $\epsilon > 0$. Their algorithm takes $O(f \log(1/\epsilon) \log(m))$ iterations, where f is the frequency parameter. For

the specific case of $f = 2$ (the vertex cover scenario), a parallel procedure for producing 1-maximal solutions is implicit in the work of Koufogiannakis and Young [11]. Their procedure runs in $O(\log m)$ iterations. While our procedure has inferior value on the parameter λ, the number of iteration is independent of the frequency parameter f. The procedure could be independent interest. The procedure is similar to that of Khuller et al. [10], but the goes via a different analysis for bounding the number of iterations.

We now discuss the forward phase. Using the procedure given in Lemma 2, compute the layer decomposition $Z_1, Z_2, \ldots, Z_L$, where L is the decomposition length. Initialize $\mathcal{A} = \emptyset$ and set $\alpha(e) = 0$, for all elements $e \in E$. The forward phase works in L epochs processing the layers from left to right. For $1 \leq k \leq L$, the goal of epoch k is to ensure that $\mathcal{A}$ covers all the elements in Z_k.

Consider an epoch k. While the goal of the previous $k-1$ epochs would have been to ensure coverage for $Z_1, Z_2, \ldots, Z_{k-1}$, the collection $\mathcal{A}$ might already be covering some elements from Z_k (unintentionally). Let $R_k \subseteq Z_k$ be the set of elements found in Z_k which are not covered by $\mathcal{A}$. The purpose of epoch k is to ensure coverage for all the elements in R_k. The epoch k works in multiple iterations. Consider an iteration $j \geq 1$. A set $S \in \mathcal{S}$ is said to *participate* in iteration j, if it is not already included in $\mathcal{A}$. Similarly, an element $e \in R_k$ is said to *participate* in iteration j, if it is not all already covered by $\mathcal{A}$. For each participating set S, compute: (i) Current degree $d_j(S)$, which is the number of participating elements found in S; (ii) Current LHS value of dual constraint of S: $h_j(S) = \sum_{e \in S} \alpha(e)$; (iii) Current difference between LHS and RHS of the dual constraint of S: $c_j(S) = w(S) - \sum_{e \in S} \alpha(S)$; (iv) Current *penalty* for S: $p_j(S) = c_j(S)/d_j(S)$ (intuitively, if S is included in S, $d_j(S)$ elements will be newly covered and this is the cost/penalty each such element pays). For each participating element e, compute the minimum penalty offered by each set covering e: $q_j(e) = \min_{S\,:\,e \in S} p_j(S)$. Increase (or raise) the dual variable $\alpha(e)$ by $q_j(e)$. This would raise the value of the LHS of the dual constraints. For every participating set S, check if its dual constraint is approximately tight: $\sum_{e \in S} \alpha(e) \geq w(S)/8$. If the above condition is true, then add S to $\mathcal{A}$. This completes the description of the iteration j. The above process is continued until all the elements in R_k are covered by $\mathcal{A}$. This completes epoch k and we proceed to epoch $k+1$.

Notice that any dual variable $\alpha(e)$ is raised only to an extent of its minimum penalty $q_j(e)$. This ensures that all the dual constraints will remain satisfied at the end of each iteration. The above procedure can be implemented in both distributed and parallel settings. In the distributed setting, each participating element (or the corresponding node in the network) can raise its dual variable $\alpha(e)$ independently using information obtained from its neighbors. Thus, each iteration can be implemented in a single round. In the parallel setting, in each iteration, the dual variables can be raised in parallel.

The above procedure returns a pair of solutions $\mathcal{A}$ and α. It is easy to see that $\mathcal{A}$ is a feasible solution for the given set cover instance. Furthermore, only sets satisfying the bound (1) are added to the collection $\mathcal{A}$. Hence, the pair satisfies the desired approximate primal slackness property.

Let us next analyze the number of iterations taken by the algorithm. The number of epochs is L. Fix any epoch k. For any iteration j, define the minimum penalty value $p_j^{\min} = \min_S p_j(S)$ (where the minimum is taken over all sets participating in iteration j). We now establish a bound on the number of iterations taken by the any epoch k, by tracking minimum penalty value. For a set S participating in successive iterations j and $j+1$, its penalty may decrease (because both the values $\delta(S)$ and $c(S)$ may decrease across iterations). Nevertheless, the lemma below shows that the minimum penalty will increase

by a factor of at least (3/2) across successive iterations. The proof is deferred to the full version.

▶ **Lemma 3.** *For any iteration j, $p^{\min}_{j+1} \geq (3/2)p^{\min}_j$.*

We shall derive a bound on the number of iteration by making some observation on the maximum and minimum values possible for $d_j(S)$ and $c_j(S)$. The $d_j(S)$ values can vary between 1 and m. The maximum value possible for $c_j(S)$ is $w_{\max}$; the minimum value possible is $(7/8)w_{\min}$ (because sets with smaller $c_j(S)$ would have got added to $\mathcal{A}$). Therefore, epoch k will take at most $O(\log m + \log \frac{w_{\max}}{w_{\min}})$ iterations. Hence, the overall forward phase algorithm runs in $O(L \cdot [\log m + \log \frac{w_{\max}}{w_{\min}}])$ iterations.

We next record some useful properties satisfied by the pair of solution $\langle \mathcal{A}, \alpha \rangle$ output by the forward phase. These properties will be useful during the reverse-delete phase. Partition the collection $\mathcal{A}$ into $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_L$, where $\mathcal{A}_k$ is the collection of sets added to $\mathcal{A}$ in the epoch k of the forward phase. For $1 \leq k \leq L$, let F_k be the set of elements freshly covered by $\mathcal{A}_k$ (meaning, the elements covered by $\mathcal{A}_k$ which are not covered by $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_{k-1}$). We say that $\mathcal{A}_k$ is *responsible* for the elements in F_k. Intuitively, in epoch k, the main task of the algorithm was to ensure coverage for $R_k \subseteq Z_k$ and the sets in $\mathcal{A}_k$ were selected for this purpose. But some elements belonging to $Z_{k+1}, Z_{k+2}, \ldots, Z_k$ might also be covered by $\mathcal{A}_k$. The set F_k consists of R_k and the above elements.

▶ **Proposition 4.** (i) *F_k contains elements only from layers $Z_k, Z_{k+1}, \ldots, Z_L$, for $1 \leq k \leq L$.* (ii) *For $1 \leq k \leq L$, the collection $\mathcal{A}_k$ does not cover any element from $R_{k+1}, R_{k+2}, \ldots, R_L$.* (iii) *The elements found in $R_1, R_2, \ldots, R_L$ are the only elements whose dual variables could potentially have been raised in the forward phase.*

3.2 Reverse Delete Phase

The forward phase produces a pair of solutions $\langle \mathcal{A}, \alpha \rangle$. In the reverse delete phase, we prune the collection $\mathcal{A}$ and obtain a solution $\mathcal{B} \subseteq \mathcal{A}$ such that the solution $\mathcal{B}$ satisfies the approximate complementary slackness property: for any $e \in E$, if $\alpha(e) > 0$ then

$$|\{S \in \mathcal{B} \ : \ S \text{ covers } e\}| \leq \tau^2. \tag{2}$$

Furthermore, we will not alter the dual variables during the reverse-delete phase. Hence, the final pair of solutions $\mathcal{B}$ and α satisfy both the primal and dual approximate complementary slackness properties, namely bounds (1) and (2). The weak duality theorem implies that the solution $\mathcal{B}$ is an $(8\tau^2)$-approximate solution.

We now describe the reverse-delete phase that would satisfy the bound (2). By the third part of Proposition 4, it suffices if we consider elements in $R_1, R_2, \ldots R_L$. The reverse delete procedure is also iterative and works in L epochs, but it will consider the layers in the reverse direction, namely, the iterations are from $k = L$ to 1. Initialize $\mathcal{B} = \emptyset$. At the end of epoch k, we will ensure two properties: (i) all the elements in $F_L, F_{L-1}, \ldots, F_k$ are covered by $\mathcal{B}$; (ii) all the elements in $R_L, R_{L-1}, \ldots, R_k$ obey the slackness property (2).

Assume by induction that we have satisfied the above two properties in iteration $L, L-1, \ldots, k+1$ and consider epoch k. Our plan is to ensure coverage of F_k by adding sets from $\mathcal{A}_k$ to $\mathcal{B}$ (recall that $\mathcal{A}_k$ is responsible for F_k). An important issue here is that the sets added to $\mathcal{B}$ in the previous iterations $L, L-1, \ldots, k+1$ will be from $\mathcal{A}_L, \mathcal{A}_{L-1}, \ldots, \mathcal{A}_{k+1}$, which are not responsible for covering the elements in F_k; nevertheless, some of these sets might still be covering the elements in $R_k \subseteq F_k$ (this is an unintended side-effect of the forward phase). While ensuring slackness property (2) for the elements in R_k, we have to take the

above phenomenon into account and may have to delete sets from $\mathcal{B}$. In doing so, we should not affect the coverage of the elements in $F_L, F_{L-1}, \ldots, F_{k+1}$. The procedure given by the lemma below helps us in achieving the above objectives; the lemma is proved in Section 3.3.

▶ **Lemma 5.** *Let $A \subseteq E$ be a set of elements belonging to layers $Z_k, Z_{k+1}, \ldots, Z_L$, for some given k. Let $\mathcal{X} \subseteq \mathcal{S}$ be a cover for A. There exists a parallel procedure that takes $\mathcal{X}$ and A as input, and outputs a collection $\mathcal{Y} \subseteq \mathcal{X}$ such that: (i) $\mathcal{Y}$ is a cover for A; (ii) for any element in $e \in A$ belonging to layer Z_k, at most τ^2 sets from $\mathcal{Y}$ cover e. The algorithm takes at most L iterations, where the dominant operation in each iteration is computing a maximal independent set (MIS) in an arbitrary graph.*

We are now ready to discuss epoch k. Let $\mathcal{X} = \mathcal{B} \cup \mathcal{A}_k$. Let $A = F_L \cup F_{L-1} \cup \cdots \cup F_k$. Notice that the requirements of the Lemma 5 are satisfied by A and $\mathcal{X}$ (because by induction, $\mathcal{B}$ covers $F_L, F_{L-1}, \ldots, F_{k+1}$ and $\mathcal{A}_k$ covers F_k). Invoke the procedure given by the lemma and obtain a set $\mathcal{Y}$.

We claim that $\mathcal{Y}$ satisfies two properties: (i) $\mathcal{Y}$ is a cover for $F_L, F_{L-1}, \ldots, F_k$; (ii) for any element e in $R_L, R_{L-1}, \ldots, R_k$ at most τ^2 sets from $\mathcal{Y}$ cover e. The first property is ensured by the lemma itself. Moreover, the lemma guarantees that the second property is true for any element $e \in R_k$. So, consider an element e belonging to one of the sets $R_L, R_{L-1}, \ldots, R_{k+1}$. The lemma ensures that $\mathcal{Y} \subseteq \mathcal{X} = \mathcal{B} \cup \mathcal{A}_k$ and hence, the sets e must come from $\mathcal{B}$ or $\mathcal{A}_k$. Proposition 4 implies that $\mathcal{A}_k$ does not contain any set covering e. Therefore, all the sets covering e must come from $\mathcal{B}$; by the induction hypothesis, there are at most τ^2 such sets. We have shown that $\mathcal{B}$ satisfies the induction hypothesis. We set $\mathcal{B} = \mathcal{Y}$ and proceed to the next epoch $k - 1$.

We see that the overall algorithm produces a $8\tau^2$-approximate solution. Let us now analyze the running time. We can preprocess the sets so that $w_{\max}/w_{\min}$ is bounded by m, while incurring an increase approximation ratio by an additive factor of one (see [15]). Computing the layer decomposition will take $O(L)$ iterations and the forward phase will take $O(L \log m)$ iterations, where each iteration can be implemented in NC. The reverse delete phase consists of L^2 iteration, where each iteration mainly involves computing MIS, which can be computed in NC [14]. Thus, when L is logarithmic in m, the overall algorithm runs in NC and produces an $(1 + 8\tau^2)$-approximate solution.

3.3 Proof of Lemma 5

We initialize $\mathcal{Y} = \emptyset$. Partition the set A according to the layers: for $k \leq j \leq L$, let $A_j = A \cap Z_j$. We process the sequence $A_k, A_{k+1}, \ldots, A_L$ iteratively – in each iteration j, we will add some appropriate sets from $\mathcal{X}$ to $\mathcal{Y}$ so as to ensure coverage for all elements in A_j.

Consider any element $e \in A$. Let $A_j \subseteq Z_j$ be the partition to which e belongs. Let $\mathcal{P}(e) \subseteq \mathcal{X}$ be the collection of all sets found in $\mathcal{X}$ which contain e. By the properties of layered decompositions, e is a τ-SNC element within $Z_j \cup Z_{j+1} \cup \cdots \cup Z_L$. Hence, there exist sets $S_1, S_2, \ldots, S_r \in \mathcal{P}(e)$ (with $r \leq \tau$) such that any element $e \in Z_j \cup Z_{j+1} \cup \cdots Z_L$ covered by $\mathcal{P}(e)$ is also covered by one of $S_1, S_2, \ldots, S_r$. We call these r sets as the *petals* of e.

For $j = k$ to L, iteration j is described next. Of the elements in A_j, some of the elements would already be covered by $\mathcal{Y}$. Let the set of remaining uncovered elements be $\widetilde{A}_j$. Construct a graph G_j with $\widetilde{A}_j$ as the vertex set; add an edge between two vertices $e_1, e_2 \in \widetilde{A}_j$, if some set $S \in \mathcal{X}$ includes both of them. Find an MIS B_j within the graph G_j. We call the elements in B_j as *anchors*. For each anchor $e \in B_j$ add its petals to the collection $\mathcal{Y}$. Proceed to the next iteration.

We now prove that the collection $\mathcal{Y}$ constructed by the above process satisfies the properties stated in the lemma. First, consider the coverage property. For $k \leq j \leq L$, let us argue that $\mathcal{Y}$ covers A_j. In the beginning of iteration j, $\mathcal{Y}$ would have already covered some elements from A_j. So, we need to bother only about the remaining elements $\widetilde{A}_j$. Consider any element $e \in \widetilde{A}_j$. If e was selected as part of the MIS B_j, then e is covered by its petals. Otherwise, there must exist some element $a \in B_j$ such that e and a share an edge in G_j. This means that some set $S \in \mathcal{X}$ contains both e and a. Therefore one of the petals of a would cover e. Since we added all the petals of a to $\mathcal{Y}$, $\mathcal{Y}$ would cover e.

Consider the second part of the lemma. We shall first argue that any two anchors are independent: namely, for any two anchors, a_1 and a_2, no set $S \in \mathcal{X}$ contains both of them. By contradiction, suppose some set $S \in \mathcal{X}$ contains both a_1 and a_2. Consider two cases: (i) the two elements belong to the same layer; (ii) they belong to different layers. The first case will contradict the fact that B_j is an MIS, where j is the layer to which both the anchors belong. For the second case, suppose $a_1 \in A_{j_1}$ and $a_2 \in A_{j_2}$ with $j_1 \leq j_2$. Our assumption is that the set S contains both a_1 and a_2. This would mean that a_2 will belong to one of the petals of a_1. Hence, in the beginning of the iteration j_2, the collection $\mathcal{Y}$ would have already covered a_2. This contradicts the fact that a_2 is an anchor.

We return to the second part of the lemma. Consider any element $e \in A_k$. We analyze two cases: (i) e is an anchor; (ii) e is not an anchor. In the first case, since the anchors are independent, the petals of no other anchor can include e. So, the only sets in $\mathcal{Y}$ which include e are the petals of e itself; the number of such petals is at most τ. Now, consider the second case. Let C be the set of all anchors a such that at least one petal of a includes e. We claim that $|C| \leq \tau$. By contradiction, suppose $|C| \geq \tau + 1$. Take any $\tau + 1$ anchors $a_1, a_2, \ldots, a_{\tau+1}$ found in C. The element e belongs to the layer Z_k. So, it will be a τ-SNC element within $Z_k \cup Z_{k+1} \cup \cdots Z_L$. Hence, the petals of e will cover all the anchors $a_1, a_2, \ldots, a_{\tau+1}$. But, the number of petals of e is at most τ. Hence, by the pigeon hole principle, two of these anchors must be covered by the same petal of e. This contradicts our previous claim that the anchors are independent. Therefore, $|C| \leq \tau$. The element may belong to more than one petal of an anchor. Each anchor $a_i \in C$ has at most τ petals. It follows that at most τ^2 petals of the anchors can cover e. This proves the second part of the claim.

References

1 A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

2 P. Berman and B. DasGupta. Improvements in throughout maximization for real-time scheduling. In *STOC*, 2000.

3 A. Bertossi and S. Moretti. Parallel algorithms on circular-arc graphs. *Information Processing Letters*, 33(6):275–281, 1990.

4 V. Chakaravarthy, A. Kumar, S. Roy, and Y. Sabharwal. Resource allocation for covering time varying demands. In *ESA*, 2011.

5 D. Chakrabarty, E. Grant, and J. Könemann. On column-restricted and priority covering integer programs. In *IPCO*, 2010.

6 I. Dinur, V. Guruswami, S. Khot, and O. Regev. A new multilayered PCP and the hardness of hypergraph vertex cover. *SIAM Journal of Computing*, 34(5):1129–1146, 2005.

7 U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

8 R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53(1):55–84, 2004.

9 F. Grandoni, J. Könemann, and A. Panconesi. Distributed weighted vertex cover via maximal matchings. *ACM Transactions on Algorithms*, 5(1), 2008.

10 S. Khuller, U. Vishkin, and N. E. Young. A primal-dual parallel approximation technique applied to weighted set and vertex covers. *Journal of Algorithms*, 17(2):280–289, 1994.

11 C. Koufogiannakis and N. Young. Distributed algorithms for covering, packing and maximum weighted matching. *Distributed Computing*, 24(1):45–63, 2011.

12 F. Kuhn, T. Moscibroda, and R. Wattenhofer. The price of being near-sighted. In *SODA*, pages 980–989, 2006.

13 N. Linial and M. Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993.

14 M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal of Computing*, 15(4):1036–1053, 1986.

15 S. Rajagopalan and V. Vazirani. Primal-dual rnc approximation algorithms for set cover and covering integer programs. *SIAM Journal of Computing*, 28(2):525–540, 1998.

16 R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *STOC*, 1997.

17 D. Williamson and D. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

Replica Placement via Capacitated Vertex Cover*

Sonika Arora[1], Venkatesan T. Chakaravarthy[2], Neelima Gupta[1], Koyel Mukherjee[3], and Yogish Sabharwal[2]

1 **Department of Computer Science, University of Delhi**
sonika.ta@gmail.com, ngupta@cs.du.ac.in
2 **IBM India Research Lab, New Delhi, India**
{vechakra,ysabharwal}@in.ibm.com
3 **Department of Computer Science, University of Maryland, College Park**
koyelm@cs.umd.edu

Abstract

In this paper, we study the replica placement problem on trees and present a constant factor approximation algorithm (with an additional additive constant factor). This improves the best known previous algorithm having an approximation ratio dependent on the maximum degree of the tree. Our techniques also extend to the partial cover version. Our algorithms are based on the LP rounding technique. The core component of our algorithm exploits a connection between the natural LP solutions of the replica placement problem and the capacitated vertex cover problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation Algorithms, LP Rounding

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.263

1 Introduction

The Replica placement problem in tree networks is a well-studied problem [4, 12, 2, 1] and several variants of it have been examined in the literature. In this paper, we study an important version called replica placement with distances and present a first constant factor approximation algorithm (with an additional additive constant factor) for this problem. Our techniques also extend to the partial cover version of the problem yielding a similar result.

Replica Placement Problem. The input consists of a *rooted* tree (or tree network) $G = (V, E)$. Each leaf node represents a *client* and let A be the set of all clients. Let $|A| = m$. The input specifies a *request* $r(a)$ for each client $a \in A$. The input also includes a *capacity* W. For each edge (u, v) in the tree, the input specifies a distance $d(u, v)$. For a node u and a client a such that u is an ancestor of a, let $d(u, a)$ be the distance from u to a. Each client a is associated with a quantity $d_{\max}(a)$, the maximum distance its request can travel. We assume that the capacity W and the requests $r(\cdot)$ are integral. Furthermore, we assume that $r(a) \leq W$ for all clients $a \in A$ and that W is polynomially bounded in the number of nodes.

A feasible solution selects a subset of nodes and places replicas on them in order to service the requests of the clients. The solution must assign the requests of the clients to the replicas. The request of a client a can be assigned to a node u, only if u is an ancestor of a and $d(a, u) \leq d_{\max}(a)$. Furthermore, the total request assigned to any replica must not exceed

* Full version of the paper is available as an arxiv preprint.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 263–274

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

the capacity W. We allow the solution to open replicas at the leaf nodes corresponding to the clients. Our goal is to minimize the number of replicas placed.

In the partial cover version of the replica placement problem, the input also includes an integer $K \leq m$, called the *partiality parameter* and it suffices if some K clients are serviced. Namely, a feasible solution selects a subset of nodes for placing replicas and a set of K clients so that the replicas can service the set of K clients. The full cover version corresponds to the case where K is the number of clients.

Prior Work. The replica placement problem finds applications in internet and video on demand service delivery (see [7, 9, 4]). We refer to [12] for additional applications. The above problem and its variants have been well-studied in the existing literature [4, 12, 2, 8, 1], from both practical and algorithmic perspectives. From an algorithmic perspective, the full cover version and its special cases have been addressed in prior work and approximation algorithms have been developed.

Benoit et al. [2, 1] studied the full cover version of the replica placement problem. They showed that it is NP-hard to approximate the problem within a factor of 3/2 even when there are no distance constraints (i.e., $d_{\max}(a) = \infty$, for all clients a) and the given network is a binary tree. They obtained the above result by showing that the above problem generalizes the bin-packing problem. Benoit et al. [1] presented a 2-approximation for the replica placement problem without distances. For the case with distances, they designed a greedy algorithm with an approximation ratio of $(1 + \Delta)$, where Δ is the maximum number of children of any node. Regarding the replica placement problem, we are not aware of any prior work on the partial cover versions.

Our Results and Discussion. The main goal of this paper is to study the full cover and the partial cover versions of the replica placement problem (with distances). As discussed earlier, the best known algorithm for the full cover version [1] has an approximation ratio of $(1 + \Delta)$, where Δ is the maximum number of children of any node in the tree. Clearly, the parameter Δ could be of the order of the number of nodes in the tree. The main contribution of the paper is a constant factor approximation algorithm (with an additional additive constant factor) for the partial cover version.

> **Main Result:** We present a constant factor approximation algorithm (with an additional additive constant factor) for the partial cover version of the replica placement problem (with distances).

The algorithms of Benoit et al. [2, 1] for different versions of the replica placement problem are primarily based on dynamic programming and the greedy method. It is not clear how to use such techniques to derive constant factor approximation algorithms that handle the dual aspects of distance constraints and partial covering. We should mention that the constant factor obtained in the paper is fairly large. The constant factor can be substantially improved via a more careful book keeping. However, for the sake of exposition and brevity, we defer such an analysis to the full version of the paper.

Our algorithms are based on the LP rounding technique. Our main technical contribution is to show a connection between the natural LP solutions of the replica placement problem and the capacitated vertex cover problem. We will exploit this connection to derive approximation algorithms for our problems. We believe our techniques can be applied to other special cases of the capacitated set cover problem as well. The above connection and our proof techniques are outlined below.

Relationship to Capacitated Set Cover and Proof Techniques. The capacitated set cover problem generalizes the classical set cover problem. In this problem, we are given a set system consisting of a universe of elements U and a collection $\mathcal{S}$ of subsets of the universe. Each set $S \in \mathcal{S}$ in the collection has associated capacity $w(S)$. A feasible solution selects a collection of sets $\mathcal{R} \subseteq \mathcal{S}$ and assigns each element $a \in U$ to some set $S \in \mathcal{R}$ such that $a \in S$. Furthermore, the solution must satisfy the property that for any selected set S, the number of elements assigned to it does not exceed $w(S)$.

Note that in the above problem definition we do not associate requests with the elements, because in such generalization, it is NP-hard even to test whether a solution exists and so, the problem is inapproximable [3]. In our problems, such an issue does not arise, since we allow replicas to be placed at the client nodes. Clearly, the unit request case of the full cover version of the replica placement is a special case of the capacitated set cover problem.

The capacitated vertex cover problem is the special case where each element a appears in exactly two sets. Equivalently, we are given a multi-graph wherein each vertex has a capacity. The solution must select a vertex cover and assign each edge to one of its end points in the cover such that the capacity is not violated at any vertex.

The capacitated set cover problem can be approximated within a factor of $O(\log n)$ using the work of Wolsey [11]. The problem is NP-hard to approximate within a factor of $\Omega(\log n)$ [5]. Chuzhoy and Naor [3] and Gandhi et al. [6] presented algorithms for the capacitated vertex cover problem with an approximation ratio of 3 and 2, respectively. However, their algorithms can handle only the case of simple graph (no parallel edges). Saha and Khuller [10] presented a 34-approximation algorithm for the more general case of multi-graphs. Their technique also extends to hypergraphs and yields an $O(f)$-approximation algorithm for the capacitated set cover problem, where the frequency parameter f is the maximum number of sets an element appears in. All these algorithms are based on rounding of a natural LP formulation of the problem.

Our algorithm for the partial cover version of the replica placement problem also goes via considering a natural LP formulation. The core component of our algorithm takes any LP solution and carefully transforms it into a solution that can be construed as an LP solution of a suitable instance of the capacitated vertex cover problem on multi-graphs. This allows us to utilize the rounding procedure of Saha and Khuller [10] and obtain a constant factor approximation algorithm for our problem.

We highlight some of the challenges involved in the above transformation. Firstly, in our problem, a client can be assigned to an arbitrary number of nodes, whereas in the capacitated vertex cover scenario, an edge can be assigned to only two nodes. Thus the transformation lets us move from a set system having arbitrary frequency parameter to a setting where the parameter is two. Secondly, the transformation converts a solution for a partial cover setting to a full cover setting. In this context, we should note that Saha and Khuller [10] present an algorithm for partial cover version of the capacitated set cover problem. However, they obtain the result via a simple reduction from the partial cover version to the full cover version via creation of a dummy set. We do not know of any such simple reduction for our setting. Thirdly, we note that a client would be considered serviced and counted as part of the partiality parameter K, only when the whole of its request is serviced. Despite the above challenges, there are certain aspects of the problem that let us perform the transformation. The first is that our problem definition allows a solution to place a replica on a client node itself. Secondly, the set systems arising in our context enjoy some special properties because of the structure of the tree.

2 Overview of the Main Result

The goal of this section is to establish the main result of the paper.

▶ **Theorem 1.** *There exists a polynomial time constant factor approximation algorithm (with an additional additive constant factor) for the partial cover version of the replica placement problem.*

The algorithm goes via LP rounding. Here, we shall present an intuitive and detailed overview of the algorithm and prove the above result. In doing so, we will highlight the different components of the algorithm. Discussion on the individual components are deferred to the subsequent sections.

LP Formulation: We consider a natural LP formulation. We say that a solution *opens* node u, if it places a replica on it. We say that a client a is *attachable* to a node u, if u is an ancestor of a and $d_{\max}(a) \leq d(u,a)$. Let $\texttt{Att}(a)$ denote the set of all nodes to which the client a can be attached and let $\texttt{Att}(u)$ denote the set of all clients that can be attached to a node u.

For each node $u \in V$, we introduce a variable $y(u)$ that specifies the extent to which u is open. For each client $a \in A$ and each node $u \in \texttt{Att}(a)$, we introduce a variable $x(a,u)$ that specifies the extent to which a is assigned to u.

$$\min \quad \sum_{u \in V} y(u)$$

$$\sum_{a \in \texttt{Att}(u)} x(a,u) \cdot r(a) \leq y(u) \cdot W \qquad (\forall u \in V) \tag{1}$$

$$\sum_{a \in A} \sum_{u \in \texttt{Att}(a)} x(a,u) \geq K \tag{2}$$

$$x(a,u) \leq y(u) \quad (\forall a \in A, u \in \texttt{Att}(a)) \tag{3}$$

$$\sum_{u \in \texttt{Att}(a)} x(a,u) \leq 1 \qquad (\forall a \in A) \tag{4}$$

$$y(u) \leq 1 \qquad (\forall u \in V) \tag{5}$$

Further, we add non-negativity constraints for all the variables. Constraint (1) (called the *capacity constraint*) enforces that at any node the total request assigned does not exceed the capacity W. Constraint (2) ensures that at least K clients are served. Constraint (3) ensures that a client can be assigned to a node only to an extent the node is open; without this constraint, it can be shown that the LP has an unbounded integrality gap. Constraint (4) enforces that every client is serviced to an extent of at most one. Constraint (5) requires that a node can be opened to an extent of at most one. Consider an LP solution $\sigma = \langle x, y \rangle$. The cost of an LP solution is given by the objective function: $\texttt{cost}(\sigma) = \sum_{u \in V} y(u)$.

LP Rounding: It is not difficult to show that the above LP has an unbounded integrality gap if we restrict ourselves to multiplicative approximation ratios. However, we shall show that the above issue can be overcome by allowing additive errors and prove the following result. We note that in the following lemma and all the subsequent lemmas, the transformations claimed can be implemented in polynomial time.

▶ **Lemma 2.** *Any LP solution σ_{in} can be transformed into an integral solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq c_1 \cdot \texttt{cost}(\sigma_{in}) + c_2$, where c_1 and c_2 are constants.*

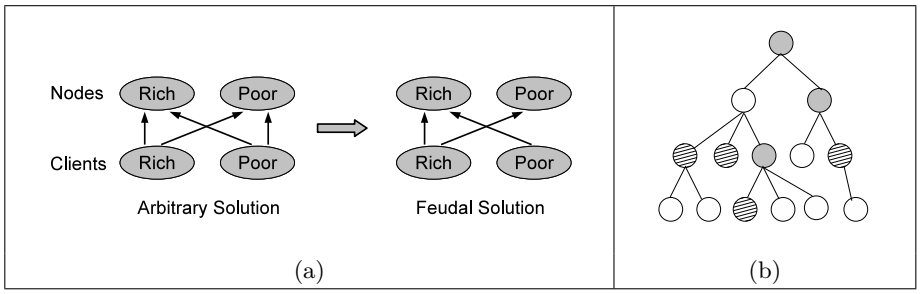


Figure 1 (a) Transformation from arbitrary solution to feudal solution; (b) Hierarchical solution.

The above lemma implies the main result (Theorem 1). We next present an overview of the proof of the lemma. Consider an LP solution $\sigma = \langle x, y \rangle$. We say that a node u is *fully-open*, if $y(u) = 1$; it is said to be *fully-closed*, if $y(u) = 0$. The node is said to be *partially open*, if $0 < y(u) < 1$. We shall make a similar classification of the clients. Let a be a client and consider the extent to which it is served under σ, namely, define $\rho(a) = \sum_{u \in \texttt{Att}(a)} x(a, u)$. The client a is said to be: (i) *fully-serviced*, if $\rho(a) = 1$; (ii) *partially serviced*, if $0 < \rho(a) < 1$; (iii) *unserviced*, if $\rho(a) = 0$. The client a is said to be *serviced*, if $\rho(a) > 0$. A node u is said to service a client a, if $x(a, u) > 0$. The solution σ is said to be *integrally open*, if every node is either fully-open or fully-closed. Similarly, it is said to be *integrally serviced*, if every client is either fully-serviced or unserviced.

Our LP rounding procedure works in four stages and it goes via the important notion of dual assigned solutions. A solution is said to be *dual assigned*, if every client is serviced by at most two nodes. The four stages are as below:

1. Converts any LP solution σ_{in} into an integrally serviced solution σ_1.
2. Converts σ_1 into an integrally serviced, dual assigned solution σ_2.
3. Converts σ_2 into an integrally open, integrally serviced solution σ_3.
4. Converts σ_3 into an integral solution σ_{out}.

Intuitively, the first stage identifies the K clients to be serviced and helps us move from the partial cover setting into a full cover setting. In an arbitrary LP solution, a client may be serviced by any number of nodes. The scenario corresponds to the capacitated set cover setting. On the other hand, a dual assigned solution corresponds to the capacitated vertex cover setting. Thus, the second stage helps us move from the capacitated set cover setting to the capacitated vertex cover setting. These two stages form the main technical component of the paper. As far as the third stage is concerned, we invoke a rounding algorithm for the capacitated vertex cover problem, due to Saha and Khuller [10] and obtain an integrally open, integrally serviced solution. The only task remaining is to make the assignments (i.e., $x(\cdot, \cdot)$) integral, which is performed by the last stage.

Stage 1: Integrally Serviced Solutions. We first identify groups of nodes that are approximately fully-open. Let $\sigma = \langle x, y \rangle$ be an LP solution. We say that a node u is *rich* if $y(u) \geq 1/3$. The node u is said to be *poor* if $0 < y(u) < 1/3$. We also use a similar terminology for the clients. A client a is said to be *rich* if the extent to which it is serviced is at least $1/3$, i.e., $\rho(a) \geq 1/3$. The client is said to be *poor* if $0 < \rho(a) < 1/3$. As it turns out, it is easy to handle two types of solutions: those having only rich nodes and those having only rich clients. For instance, in the first case, we can simply open all the rich nodes and obtain an integrally open solution (at a three factor loss in approximation). So, the main challenge lies in handling solutions wherein both poor nodes and poor clients are present.

Our algorithm would address the issue by applying a sequence of transformations to the given solution. The assignments from clients to nodes can be classified into four types, based on whether a client is rich or poor, and whether a node is rich or poor. Our first transformation removes one of these assignments, namely the assignments from poor clients to poor nodes. Towards that goal we next define the notion of *feudal solutions*. A solution σ is said to be *feudal*, if poor nodes service only rich clients. See Figure 1(a). The first transformation is stated below.

▶ **Lemma 3.** *Any LP solution σ_{in} can be converted into a feudal solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 1 + 2\texttt{cost}(\sigma_{in})$.*

The natural next step would be to remove the assignments from the rich clients to the poor nodes, so that the poor nodes are eliminated altogether. However, we do not know how to achieve the above task in a direct manner. Instead, our next transformation will make progress towards that goal. Let $\sigma = \langle x, y \rangle$ be an LP solution. We say that a node u is *terminal*, if u is not fully-closed and all its descendants are fully-closed. The solution is said to be *hierarchical*, if every poor node is terminal and all the other nodes are either fully-open or fully-closed. See Figure 1(b). In the figure, the colored nodes are fully-open, the hatched nodes are terminal (poor) nodes and the white nodes are fully-closed. A direct implication of the hierarchical property is that any client will be serviced by at most one poor node. Thus, the set of clients serviced by any two poor nodes will be disjoint. Our next transformation is stated below.

▶ **Lemma 4.** *Any feudal solution σ_{in} can be converted into a hierarchical solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 11 \cdot \texttt{cost}(\sigma_{in})$.*

Our next transformation would convert hierarchical solutions into integrally serviced solutions, thereby taking us from the realm of partial covering to the realm of full covering. In doing so, the procedure would preserve the hierarchical property.

▶ **Lemma 5.** *Any hierarchical solution σ_{in} can be converted into a hierarchical integrally serviced solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 6 + 12 \cdot \texttt{cost}(\sigma_{in})$.*

The main idea behind the above lemma is as follows. Ignoring the fully-closed nodes, the input solution σ_{in} contains only fully-open nodes and terminal poor nodes. We consider each fully open node u and perform a shifting procedure on the clients serviced by u. Pick any two such clients a and b with $r(a) \leq r(b)$. We can decrease $x_{in}(a, u)$ by δ and increase $x_{in}(b, u)$ by δ (for a suitable δ). Via this method, we can construct a solution σ' such that the clients fall into two types: (i) fully-serviced clients, which will be serviced only by fully-open nodes; (ii) clients serviced by poor terminal nodes. In order to get a fully-serviced solution, we need to focus only on the second type of clients. We get an integrally serviced solution by exploiting the fact that the set of clients serviced by terminal nodes are disjoint.

Stage 2: Integrally Serviced Dual Assigned Solutions. Our next task is to obtain integrally serviced, dual assigned solutions. Let $\sigma_{in} = \langle x_{in}, y_{in} \rangle$ be a hierarchical integrally serviced solution, as output by Lemma 4. In σ_{in}, all the clients are fully-serviced or unserviced. Let Q be the set of fully-serviced clients. The nodes are of only three types: fully-open, fully-closed and poor. Let U be the set of all fully-open nodes and let P be the set of poor nodes. Any client $a \in Q$ may be serviced by multiple nodes from U, but it can be serviced by at most one node from P. We shall focus on U and Q and apply a "cycle cancellation" procedure to adjust the assignments of the clients to the nodes. We will not modify the extents to

which the nodes are open. This way we shall identify a subset of clients Q' and obtain a new solution $\sigma' = \langle x', y_{in} \rangle$ such that any client $a \in Q'$ is serviced by at most one from U. Thus, under σ', all the clients in Q' are dual-assigned (these will be serviced by at most one fully-open node and at most one poor node). We will ensure that the number of clients left out (namely $|Q| - |Q'|$) is at most $|U|$. We then fully open all the clients in $Q - Q'$ and obtain a dual-assigned solution σ_{out}. The cost of the solution σ_{out} will be at most $\texttt{cost}(\sigma_{in}) + |U|$, which is at most $2 \cdot \texttt{cost}(\sigma_{in})$. We do not know how to perform the above cycle cancellation operation over partially open nodes and so, we focus on the fully-open nodes. Via the above strategy, we will establish:

▶ **Lemma 6.** *Any hierarchical integrally serviced solution σ_{in} can be converted into an integrally serviced, dual assigned solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot \texttt{cost}(\sigma_{in})$.*

Stage 3: Integrally Open, Integrally Serviced Solutions. Let σ_{in} be any integrally serviced, dual assigned solution. Our next task is to convert it into an integrally open, integrally serviced solution. Under σ_{in}, each client is either fully-serviced or unserviced. Let Q be the set of fully-serviced clients. Each client $a \in Q$ is serviced by at most two nodes. For the ease of exposition, assume that each client $a \in Q$ is serviced by exactly two nodes. We can construct a multi-graph by taking the nodes to be the vertices. Each client a serviced by two nodes u and v can be represented by a set of $r(a)$ parallel edges between u and v. The LP solution σ_{in} can be construed as an LP solution to the capacitated vertex cover problem on the above graph. Saha and Khuller [10] present a 34-approximation LP rounding procedure for the above problem. Using their procedure we can get an integrally open, integrally serviced solution.

▶ **Lemma 7.** *Any integrally serviced, dual assigned solution σ_{in} can be converted into an integrally open, integrally serviced solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 34 \cdot \texttt{cost}(\sigma_{in})$.*

Stage 4: Integral Solution. The final stage is to convert an integrally open, integrally serviced solution σ_{in} into an integral solution σ_{out}. The only issue with σ_{in} is that the request $r(a)$ of a client a may be split and assigned to multiple nodes. On the other hand, our problem definition requires that the request must be wholly assigned to a single node. We can address the issue by using a cycle cancellation procedure similar to that of Stage 2.

▶ **Lemma 8.** *Any integrally open, integrally serviced solution σ_{in} can be converted into an integral solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot \texttt{cost}(\sigma_{in})$.*

Overall Algorithm: The overall algorithm is obtained by combining the procedures given by Lemmas 3 – 8. In this paper, we shall only prove Lemmas 3 , 4 and 6 and defer the other ones to the full version of the paper.

3 Feudal Solutions: Proof of Lemma 3

Let $\sigma_{in} = \langle x_{in}, y_{in} \rangle$ be the given solution. Create a pool $\mathcal{P}$ consisting of all poor nodes. The procedure is iterative and it works in multiple phases. Each phase would modify the current solution into a new solution, which is fed as input to the subsequent phase. To start with, we take the input solution to be the current solution. In each phase, we shall remove one or more nodes from the pool. For each such node u, we will ensure that u is either rich or it services only rich clients (or both). Each phase proceeds as follows.

Let $\pi_{old} = \langle x, y \rangle$ be the current solution. With respect to π_{old}, let H be the rich clients and L be the poor clients. For each node $u \in \mathcal{P}$, compute the ratio:

$$\lambda(u) = \frac{1}{y_{old}(u)} \sum_{a \in \mathtt{Att}(u) \cap L} x_{old}(a, u).$$

Informally, the numerator signifies the contribution towards the partiality parameter K (i.e., "gain") given by the poor clients, and the denominator signifies the cost of the node. Higher the ratio, the node is better. Let u^* be the node in $\mathcal{P}$ having the maximum ratio $\lambda(\cdot)$; we call the node as the *leader* of the phase. The goal of the current phase is to convert u^* into a rich node.

Excluding u^*, arrange all the other nodes in $\mathcal{P}$ in an arbitrary order, say $u_1, u_2, \ldots, u_\ell$. Let s be the least index such that

$$y_{old}(u^*) + \sum_{j=1}^{s} y_{old}(u_j) \geq 1/3.$$

If no such index exists (meaning $y_{old}(u^*) + \sum_{j=1}^{\ell} y_{old}(u_j) < 1/3$), set $s = \ell$. We call the nodes $u_1, u_2, \ldots, u_s$ as the *slaves* of u^*.

Let $\delta = \sum_{j=1}^{s} y_{old}(u_j)$. Construct a new solution $\pi_{new} = \langle x_{new}, y_{new} \rangle$ as follows. First we eliminate all the assignments of poor clients onto the slaves. Namely, for each slave u_j and all clients $a \in \mathtt{Att}(u_j) \cap L$, set $x_{new}(a, u_j) = 0$. This would reduce the partiality value (i.e., LHS of constraint (2)) and hence, the constraint would be violated. To compensate, we increase the assignments of the poor clients to the leader as follows. For each client $a \in \mathtt{Att}(u^*) \cap L$, set

$$x_{new}(a, u^*) = x_{old}(a, u^*) \frac{(y_{old}(u^*) + \delta)}{y_{old}(u^*)}.$$

Using the fact that $\lambda(u_j) \leq \lambda(u^*)$, we can show that the loss in the partiality value is compensated by the above gain. The above increase in the assignments at node u^* may lead to violation of constraints (1) or (3) at node u^*. To overcome the issue, we increase the extent to which u^* is open. Set

$$y_{new}(u^*) = y_{old}(u^*) + \delta.$$

All the other values of $y_{new}(\cdot)$ and $x_{new}(\cdot, \cdot)$ are retained as in π_{old}.

▶ **Lemma 9.** *π_{new} is a feasible solution.*

The above claim is proved via induction by assuming that π_{old} is feasible and then showing that π_{new} is also feasible. The fact that the clients are classified into rich and poor based on the threshold of 1/3 is used to show that no client gets serviced to an extent of more than one. The proof is deferred to the full version of the paper.

At the end of the phase we remove the leader and the slaves from the pool. If the pool is non-empty, we proceed to the next phase and pick a new leader. The solution constructed π_{new} is taken as the current solution. On the other hand, the pool may be empty (this will happen when all the nodes were picked as slaves, i.e., $s = \ell$). In this case, we refer to the leader as the *loner*. We open loner fully and terminate the procedure.

Let us now analyze the solution output by the process. Let $\sigma_{in} = \langle x_{in}, y_{in} \rangle$ be the solution input to the procedure and let $\sigma_{out} = \langle x_{out}, y_{out} \rangle$ be the solution output by the procedure. Let R and P be the set of all nodes which are rich and poor with respect to σ_{in},

respectively. The extent to which the nodes in R are open was left undisturbed and so, these remain rich with respect to σ_{out}. The nodes in P were added to the pool and all of them got removed as either leaders or as slaves. We ensured that all the leaders are rich when they left the pool. Regarding the slaves, we ensured that all the poor clients got their assignments to these nodes eliminated. It follows that the output solution is feudal.

Let us analyze the cost of the solution σ_{out}. We did not modify the extent to which the nodes in R and the slaves are open. So, we need to consider only the leaders. Except the loner, consider any leader u. Let $\delta(u)$ be the increase in its extent of opening; $\delta(u)$ is the sum of extents to which the slaves of u are open under σ_{in}. A node can serve as a slave for at most one node. Therefore,

$$\texttt{cost}(\sigma_{out}) \leq \texttt{cost}(\sigma_{in}) + \sum_{u \in S} y_{in}(u),$$

where S is the set of all slaves. The sum in the RHS is at most $\texttt{cost}(\sigma_{in})$. Hence, we get that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot \texttt{cost}(\sigma_{in})$. We need to add an extra cost of one for opening the loner. This completes the proof of Lemma 3.

4 Hierarchical Solutions: Proof of Lemma 4

The lemma is proved in two stages. The first stage converts the input feudal solution into a sandwich solution, defined next. A solution σ is said to be a *sandwich solution* if for any two poor nodes u_1 and u_2 such that u_1 is an ancestor of u_2, there exists a rich node v in between the path connecting u_1 and u_2 in the given tree. We first prove the following lemma.

▶ **Lemma 10.** *Any feudal solution σ_{in} can be converted into a sandwich solution σ_{out} such that $\texttt{cost}(\sigma_{out}) \leq 5 \cdot \texttt{cost}(\sigma_{in})$.*

The sandwich property can be equivalently restated as follows. For a node u, let v be the closest ancestor which is not fully-closed; we call v as the *least non-trivial ancestor* of u. We note that the above definition does not apply to the root node and any node u whose path to the root (including root) consists only of fully-closed nodes. A solution satisfies the sandwich property if for any poor node u having a least non-trivial ancestor v, the node v is rich.

Let $\sigma_{in} = \langle x_{in}, y_{in} \rangle$ be the given solution. We process the nodes from leaf level to the root level; namely, a node will be processed once all its children are processed. The processing of each node takes the current solution and produces a new solution.

The processing of a node u^* is as follows. First consider some easy cases: (i) u^* is fully-closed; (ii) u^* is rich; (iii) u^* does not have a least non-trivial ancestor; (iv) u^* has a rich least non-trivial ancestor v^*. In these cases we do nothing.

The case left remaining is where both u^* and its least non-trivial ancestor v^* are poor. Let p^* be the parent of u^* (this could be the same as v^*). Let $\pi_{old} = \langle x_{old}, y_{old} \rangle$ be the current solution and we will construct a new solution $\pi_{new} = \langle x_{new}, y_{new} \rangle$. We say that a client a is *critical* at u^* if u^* services a and u^* is the highest node that a can be attached (i.e., $d(a, p^*) > d_{\max}(a)$ and so, $a \notin \texttt{Att}(p^*)$). We consider two cases based on whether there exists a critical client at u^*. If there is such no such client, we perform the following *merge operation*. The idea is to close u^* fully and shift its extent of opening and all its client assignments to the parent p^*. Compute the new solution by setting $y_{new}(u^*) = 0$ and $y_{new}(p^*) = y_{old}(p^*) + y_{old}(u^*)$. For each client a serviced by u^*, set $x_{new}(a, p^*) = x_{old}(a, p^*) + x_{old}(a, u^*)$ and set $x_{new}(a, u^*) = 0$. All other entries of the functions y_{new} and x_{new} are retained as in π_{old}. It can be verified that π_{new} is feasible.

Consider the more interesting case where some client a^* is critical at u^*. In this case, we cannot perform the above merge operation. We simply open the node u^* and set $y_{new}(u^*) = 1$. We next perform a *pull procedure* as follows. The procedure is iterative. Let $\pi_{old} = \langle x_{old}, y_{old} \rangle$ be the current solution. Consider any client a such that a is serviced by some ancestor v of u^*. We will pull the assignment of a from v and reassign it to u^*. In doing so, we must ensure that the capacity constraint at u^* does not get violated. Let the current capacity utilization at u^* be $\texttt{cap}(u^*) = \sum_{a' \in \texttt{Att}(u^*)} r(a') x_{old}(a', u^*)$. Compute

$$\delta = \min \left\{ x(a, v), \frac{W - \texttt{cap}(u^*)}{r(a)} \right\}.$$

Set $x_{new}(a, v) = x_{new}(a, v) - \delta$ and $x_{new}(a, u^*) = x_{old}(a, u^*) + \delta$. The above operation is performed iteratively until we can find no such client a or the capacity utilization at u^* becomes W. If the procedure terminates under the first criterion, we call u^* a *critical node*; if it terminated under the second criterion, we call u^* a *complete node*. This completes the description of the procedure.

Let $\sigma_{in} = \langle x_{in}, y_{in} \rangle$ be the input solution and $\sigma' = \langle x', y' \rangle$ be the solution output by the procedure. It is easy to see that σ' satisfies the sandwich property. We compare the cost of σ' and σ_{in}. The merge operation preserves the cost and so, we need to bother only about the critical and complete nodes. Let n_C be the number of critical nodes and n_W the number of complete nodes. We have that

$$\texttt{cost}(\sigma') \leq \texttt{cost}(\sigma_{in}) + n_C + n_W.$$

For a client a, let $\theta_{in}(a) = \sum_u x_{in}(a, u)$ and $\theta'(a) = \sum_u x'(a, u)$ be the extent to which a is serviced under σ_{in} and σ', respectively. The above procedure does not change the extent of service for any client and so $\theta_{in}(a) = \theta'(a)$. Thus, the total volume of service extended by the two solutions is the same:

$$\sum_{a \in A} r(a) \theta_{in}(a) = \sum_{a \in A} r(a) \theta'(a).$$

Clearly, the quantity $n_W \cdot W$ is at most the RHS, because the n_W complete nodes have capacity utilization W. Moreover, $\texttt{cost}(\sigma_{in})$ is at least LHS$/W$ since a node has only capacity W. It follows that $n_W \leq \texttt{cost}(\sigma_{in})$.

We now consider the quantity n_C. Let $u_1, u_2, \ldots, u_s$ be the critical nodes, where $s = n_C$. For each $1 \leq j \leq s$, u_j has a critical client a_j under σ'. Under σ_{in}, either the poor node u_j services a_j or some poor node merged with u_j services it. Either way a_j is serviced by some poor node under σ_{in}. Since σ_{in} is a feudal solution, $\theta_{in}(a_j) \geq 1/3$. Let p_j be the path connecting u_j and leaf node a_j in the tree. Let $\mu(a_j)$ the sum of extents to which the nodes found along the above path are open in σ_{in}, i.e., $\sum_{u \in p_j} y_{old}(u)$. By the constraint (3), we have that $\mu(a_j) \geq \theta_{in}(a_j)$ and hence, $\mu(a_j) \geq 1/3$. The pulling procedure ensures that for any a_i and a_j, the paths p_i and p_j are disjoint. Therefore,

$$\texttt{cost}(\sigma_{in}) \geq \sum_{j=1}^{s} \mu(a_j) \geq s/3.$$

This shows that $n_C \leq 3\texttt{cost}(\sigma_{in})$. Hence, we get that $\texttt{cost}(\sigma') \leq 5 \cdot \texttt{cost}(\sigma_{in})$.

We now transform the sandwich solution σ' into a hierarchical solution σ_{out}. With respect to $\sigma' = \langle x', y' \rangle$, let R be the set of rich nodes and let P be the set of poor nodes. Of the poor nodes, let P_1 be the set of terminal nodes and let P_2 be the other nodes. The sandwich property

implies that $|P_2| \leq |R|$. We fully open all the nodes in R and P_2, and obtain a hierarchical solution σ_{out}. We see that $\texttt{cost}(\sigma') \geq |R|/3$ and that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot |R| + \texttt{cost}(\sigma')$. Therefore, $\texttt{cost}(\sigma_{out}) \leq 7 \cdot \texttt{cost}(\sigma')$. This shows that $\texttt{cost}(\sigma_{out}) \leq 35 \cdot \texttt{cost}(\sigma_{in})$ However, by combining the analysis of the two stages, we can derive an improved bound: $\texttt{cost}(\sigma_{out}) \leq 11 \cdot \texttt{cost}(\sigma_{in})$ (details are deferred to the full version of the paper).

5 Proof of Lemma 6

Let σ_{in} be the input hierarchical integrally serviced solution. In σ_{in}, all the clients are fully-serviced or unserviced, and the nodes are of only three type: fully-open, fully-closed and poor. Each client can be serviced by at most one poor node. We will adjust the assignments of the clients to the fully-open nodes and obtain a new solution wherein most of the clients are serviced by at most one fully-open node. The following client-server setup is useful for this purpose.

Consider a bipartite graph with a set of servers U on one side and a set of clients Q on the other side. For a client a, all its neighboring servers are said to be *accessible* to a. Each client has an integral requirement $q(a)$ and each server has an integral capacity W. An edge-assignment g over a subset of clients $Q' \subseteq Q$ is a mapping that takes as input a client $a \in Q'$ and node u accessible to it and outputs a value $g(a,u) \in [0, q(a)]$ (intuitively, it assigns $g(a,u)$ part of the request $q(a)$ to u). We require that for any client a, $\sum_u g(a,u) = q(a)$ and for any node u, $\sum_a g(a,u) \leq W$. We allow $q(a)$ and $g(a,u)$ to be non-integral. We say that an edge-assignment is a *single policy assignment*, if for any client $a \in Q$, all its request is assigned to a single accessible server. The following lemma shows how to convert any assignment into a single policy assignment with only a minimal loss.

▶ **Lemma 11.** *There exists a polynomial time procedure that takes any edge-assignment g over Q as input and outputs a subset of clients Q' and an edge-assignment g' over Q' such that: (i) g' is a single policy edge-assignment over Q'; (ii) the number of ignored clients is at most U, i.e., $|Q| - |Q'| \leq |U|$.*

Proof. Let $G = (V, E)$ be the weighted bipartite graph where $V = U \cup Q$ and E is the set of edges between clients and their neighboring servers. The weight on an edge $(a,u) \in E$ is defined to be the assignment $g(a,u)$. Let G^+ be the graph induced on G by considering only the edges with strictly positive weights, i.e., corresponding to which the assignments g are non-trivial ($g(a,u) > 0$).

We first show that it is possible to eliminate cycles from the induced graph G^+. The main idea is to employ cycle cancelling. Consider any cycle, C, in G^+. Since G is a bipartite graph, the cycle is of even length. We partition the edges of the cycle C into two sets C_{even} and C_{odd} by considering alternate edges in each set. We now modify the weights as follows. Let δ be the weight of the minimum weight edge in the cycle. For every edge, (a,u) in C_{even}, we increase $g(a,u)$ by an amount $\delta/q(a)$ and for every edge (a,u) in C_{odd}, we decrease $g(a,u)$ by an amount $\delta/q(a)$. This breaks the cycle. We apply this procedure repeatedly until all the cycles are eliminated.

We next analyze the forest G^+. Let $Q' = \{a \in Q \cap G^+ \ : \ degree(a) = 1 \text{ in } G^+\}$. Note that all these clients are already fully assigned. Now consider the forest $\widetilde{G^+}$ induced by the vertices of $U \cup (Q \setminus Q')$ on G^+. Consider any tree H in the forest. Pick an arbitrary vertex $r \in U \cap H$ and make it the root of the tree. For any vertex v of H, let $\ell(v)$ denote the level (distance from the root r) of the vertex v in the tree. Since G^+ is a bipartite graph, we note that the vertices at even level (odd level resp.) belong to $U \cap H$ ($Q \cap H$ resp.). Note that

all the leaves in the tree are in U. We now construct a one-to-one mapping from $Q \cap H$ to $U \cap H$. This is obtained by mapping every element of Q in the tree to any one of its children arbitrarily. This shows that $|Q \cap H| \leq |U \cap H|$. We repeat the above procedure for every tree in the forest $\widetilde{G^+}$. The one-to-one mapping gives us that $|Q| - |Q'| \leq |U|$. This completes the proof of the lemma. ◀

We can now prove Lemma 6 using the above client-server setup. Consider the input solution $\sigma_{in} = \langle x_{in}, y_{in} \rangle$. Let U be the set of all fully-open nodes in σ_{in} and let Q be the set of fully-serviced clients. For each $a \in Q$, let $q(a)$ be the portion of the request of a serviced by the nodes in U; let $q(a) = \sum_{u \in U \cap \texttt{Att}(a)} r(a) \cdot x_{in}(a, u)$. Construct an edge-assignment g: for each client $a \in Q$ and $u \in U \cap \texttt{Att}(a)$, let $g(a, u) = x_{in}(a, u)r(a)$. Invoke Lemma 11 and obtain a set of clients $Q' \subseteq Q$ and an edge-assignment g' over Q'. Using g' we can construct a new solution $\sigma_{out} = \langle x_{out}, y_{out} \rangle$ such that all the clients in Q' are serviced by at most one fully-open node. All these clients are dual assigned. As far as the clients in $Q - Q'$ are concerned, we simply fully open the leaf nodes corresponding to these clients. The new solution σ_{out} will have cost at most $\texttt{cost}(\sigma_{in}) + |Q| - |Q'|$. The above lemma guarantees that $|Q| - |Q'| \leq |U|$. Since $\texttt{cost}(\sigma_{in}) \geq |U|$, we have that $\texttt{cost}(\sigma_{out}) \leq 2 \cdot \texttt{cost}(\sigma_{in})$.

References

1. A. Benoit, H. Larchevêque, and P. Renaud-Goud. Optimal algorithms and approximation algorithms for replica placement with distance constraints in tree networks. In *IPDPS*, pages 1022–1033, 2012.
2. A. Benoit, V. Rehn-Sonigo, and Y. Robert. Replica placement and access policies in tree networks. *IEEE Trans. on Parallel and Dist. Systems*, 19:1614–1627, 2008.
3. J. Chuzhoy and J. Naor. Covering problems with hard capacities. *SIAM Journal Computing*, 36(2):498–515, 2006.
4. I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electronic content. *Computer Networks*, 40:205–218, 2002.
5. U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.
6. R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *Journal of Computer and System Sciences*, 72(1), 2006.
7. K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Trans. on Parallel and Distributed Systems*, 12:628–637, 2001.
8. M.J. Kao and C.S. Liao. Capacitated domination problem. *Algorithmica*, pages 1–27, 2009.
9. Y.F. Lin, P. Liu, and J.J. Wu. Optimal placement of replicas in data grid environments with locality assurance. In *ICPADS*, 2006.
10. B. Saha and S. Khuller. Set cover revisited: Hypergraph cover with hard capacities. In *ICALP*, 2012.
11. L. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.
12. J.J. Wu, Y.F. Lin, and P. Liu. Optimal replica placement in hierarchical data grids with locality assurance. *J. of Parallel and Dist. Computing*, 68:1517–1538, 2008.

Knapsack Cover Subject to a Matroid Constraint

Venkatesan T. Chakaravarthy[1], Anamitra Roy Choudhury[2], Sivaramakrishnan R. Natarajan[3], and Sambuddha Roy[4]

1 **IBM Research, New Delhi, India,** vechakra@in.ibm.com
2 **IBM Research, New Delhi, India,** anamchou@in.ibm.com
3 **University of Washington, Seattle, USA,** srk2siva@gmail.com
4 **IBM Research, New Delhi, India,** sambuddha@in.ibm.com

Abstract

We consider the Knapsack Covering problem subject to a matroid constraint. In this problem, we are given an *universe* U of n items where item i has attributes: a *cost* $c(i)$ and a *size* $s(i)$. We also have a demand D. We are also given a matroid $\mathcal{M} = (U, \mathcal{I})$ on the set U. A feasible solution S to the problem is one such that (i) the cumulative *size* of the items chosen is at least D, and (ii) the set S is *independent* in the matroid $\mathcal{M}$ (i.e. $S \in \mathcal{I}$). The objective is to *minimize* the total cost of the items selected, $\sum_{i \in S} c(i)$. Our main result proves a 2-factor approximation for this problem.

The problem described above falls in the realm of *mixed packing covering* problems. We also consider packing extensions of certain other covering problems and prove that in such cases it is not possible to derive any constant factor approximations.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation Algorithms, LP rounding, Matroid Constraints, Knapsack problems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.275

1 Introduction

In this paper, we consider the problem of computing the *minimum cost* Knapsack cover subject to *matroid* constraints. The Knapsack Cover problem (shortened as KC) is a minimization problem that takes as input, an universe U of n elements. Each element i is equipped with a cost $c(i)$ and a size $s(i)$. There is also a demand D; a feasible solution S is a collection of items such that the cumulative size is at least D. The objective in this problem is to find the feasible solution of minimum total cost. Thus, the KC problem is the *covering* version of the (more usual) knapsack packing problem.

The main problem considered in this paper is the KC problem subject to certain constraints that are called *matroid* constraints. In this scenario, in addition to the input for the KC problem as mentioned above, we are given a *matroid* $\mathcal{M} = (U, \mathcal{I})$ where $\mathcal{I} \subseteq 2^U$ is the family of *independent* sets (we give the formal definition of a matroid in Section 5). A set S is considered feasible iff (i) the cumulative size of S is at least D, and (ii) S is *independent* i.e. $S \in \mathcal{I}$.

We will denote the Knapsack Cover problem subject to a Matroid constraint as the KCM problem.

The KC problem naturally arises in various applications where we have a certain demand to fulfill and a certain number of options, varying in *profit* (i.e. *size*) and cost. For instance, consider a workplace where we need a certain number of developers for a certain project; and the project manager wants to outsource this to various teams/companies where each team

can provide a certain number of developers at a specific cost. This is precisely the knapsack cover problem discussed above.

Given this context, consider the following variants. Any single company can either provide 5 developers at a cost of 100 units or 6 developers at a cost of 115 units, etc. In the framework of knapsack cover problems, we may model these different options as distinct items in the knapsack, but with a *partition matroid* constraint on these distinct items. In this setting this means that a feasible solution may pick at most one of the options from a single company; this is precisely what we want. Another (admittedly less realistic) scenario is where different companies have mutual incompatibilities; thus two companies may not be employed for the same project at the same time. Again, this corresponds to matroid constraints in addition to the demand fulfillment constraints. Thus these are all instances of the KCM problem.

The main result of this paper is the following:

▶ **Theorem 1.** *There is a* PTAS *for the* KCM *problem; specifically, given any* $\epsilon > 0$*, there is an algorithm that runs in time* $n^{\mathcal{O}(1/\epsilon)}$ *and outputs a* $(1+\epsilon)$*-factor approximate solution.*

Given that the KCM problem is a natural generalization of the KC problem, it is instructive to compare solution approaches for the KC problem. For the knapsack cover problem, a FPTAS is known via dynamic programming with parameter scaling. The natural LP for the KC problem has an unbounded integrality gap. Despite this, authors [4, 3] have shown how to achieve constant integrality gap via augmenting the natural LP with so-called "flow-cover inequalities". Carr et al. [4] first used such LP based relaxations and LP rounding to provide a 2-factor approximation for the KC problem (among other capacitated covering problems). Carnes and Shmoys [3] showed an elegant primal dual algorithm for the same LP to also derive a 2-factor approximation.

Note that while the vanilla version of the knapsack covering problem is a *covering* problem, in the KCM problem, we have both covering *and* packing constraints. Typically, such *mixed* packing covering problems are harder to analyse as compared to pure packing or pure covering problems. This is partially because for such problems, even checking the feasibility of the constraint set can be a NP-hard problem. However, for the KCM problem, the feasibility problem is indeed in polynomial time, as we show in Section 7. To the best of our knowledge the KCM problem has not been considered earlier in literature.

2 Our Contribution & Techniques

We prove the following results:

- Given the knapsack cover problem with a single matroid constraint, we show a PTAS.
- Given knapsack cover with multiple matroid constraints, the feasibility problem is NP-hard. Given this, we show a *bicriteria* approximation guarantee: we exhibit an algorithm that outputs a solution of value at most that of the optimal solution OPT that is *nearly-feasible*. The formal statement is given as Theorem 5 in Section 8.

Given the mixed packing/covering nature of the KCM problem, it is difficult to apply primal dual schemas; since the dual objective function in this context have both positive and negative coefficients. Previous literature has indeed considered primal dual schemas with dual objective functions having coefficients of either sign (for instance, see [9]). However in such cases, the primal dual algorithms give only *approximately* feasible solutions. One other possibility to consider are combinatorial algorithms. For instance there is a simple minded *greedy* algorithm for the knapsack cover KC problem, based on the *cost-effectiveness* $c(i)/s(i)$ of an element i. The algorithm proceeds as follows: it guesses the *costliest* element (say, of

cost C^*)in the optimal cover OPT, and removes all the elements of cost larger than C^*. The algorithm considers the rest of the elements in increasing order of their cost-effectiveness, i.e. their $c(i)/s(i)$ values, and continues selecting elements while the covering requirement is not met. This gives a 2-factor approximation to the KC problem. However, there are certain issues in adapting this greedy approach to the KCM problem as we illustrate below.

For the KCM problem, if we pick up elements according to non-decreasing $c(i)/s(i)$ values, it may happen that we cannot pick any more elements without violating the matroid constraints and yet the cumulative size of the elements picked falls short of the requisite demand D. For instance, this happens in the following scenario. We are given as input, $U = \{1, 2, 3, 4\}$, with $D = 4$ and a cardinality matroid constraint over the whole universe with $k = 2$. The items ordered by their $c(\cdot)/s(\cdot)$ values are $\{\frac{1-\epsilon}{1}, \frac{2}{2}, \frac{2}{2}, \frac{3+\epsilon}{3}\}$. In this instance, OPT would pick either $\{1, 4\}$ or $\{2, 3\}$. The greedy solution would yield $\{1, 2\}$ of total cost $(3 - \epsilon)$ but satisfying only 3 units of the demand.

There exists another ordering natural for the problem. Picking elements in decreasing order of their $s(i)$'s while being feasible for the matroid constraints is the quickest way to satisfy the knapsack covering constraint; but this may cause the cost to blow up. For the above instance, this gives the (non-optimal) solution $\{3, 4\}$ of cost $(5 + \epsilon)$.

One natural idea then, is to proceed along an *amalgam* of the two orderings. Thus, one could start out with the knapsack greedy ordering, and then when some matroid constraint becomes tight, start *swapping* or *exchanging* items, while ensuring an improvement in $\sum s(i)x_i$ towards the target demand of D. We are not able to make such ideas work as of now.

On the other hand, in order to attempt LP rounding for the KCM problem, we have to surmount the obstacle of high integrality gap. For the KC problem, [3, 4] include the exponentially many *knapsack flow-cover* inequalities to overcome the unbounded gap. In the KCM setting, since we already have exponentially many matroid constraints, it would be preferable not to add another collection of exponentially many constraints.

As mentioned above, the natural LP for the problem has an unbounded integrality gap that it inherits from the LP for the KC problem. Nevertheless, we adopt a LP based approach, where we use properties of the *basic feasible solutions* of the LP. In the case of cardinality matroids, we also show a *cycle cancelling* approach to derive the desired result.

3 Other Results

For the special case of the KCM problem where the matroid is a *partition matroid*, we are able to show a FPTAS; this uses the dynamic programming approach along with parameter scaling. For space considerations, we defer the proof to the full version.

We consider certain other covering problems and augment such problems with matroid constraints. We demonstrate that the above cases are the exception rather than the rule.

As a sample, we consider the problem of *interval covering* with a partition matroid constraint. In this problem, we are given a collection of intervals $\mathcal{I} = \{I_1, I_2, \cdots, I_n\}$ over a time range $\mathcal{T} = \{1, 2, \cdots, T\}$. We are also given a partition $\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_m$ of the intervals in $\mathcal{I}$. A feasible solution S is a collection of intervals such that (1) *every* timeslot in $\mathcal{T}$ is *covered* by some interval in S and (2) there is at most one interval chosen from each $\mathcal{P}_j$ (for $1 \leqslant j \leqslant m$). The objective is to choose the minimum number of intervals. We prove that testing feasibility in this case is NP-hard (and so, no constant factor approximation algorithm is possible for this problem). This is shown via a reduction from the Vertex Cover problem; details are provided in Section 9. On the other hand, the problem considered with a *cardinality matroid* (where, a feasible solution can pick at most a certain number of intervals

from $\mathcal{I}$) has a dynamic programming solution that solves it in polynomial time.

It is more likely than not that augmenting a covering problem with a matroid constraint might render the feasibility problem NP-hard. As another instance, consider *Vertex Cover*. If we were to add a cardinality constraint, then the feasibility problem is an instance of *unweighted* Vertex Cover, and is NP-hard.

4 Related Work

In this paper, we consider the problem of knapsack covering subject to matroid constraints. This naturally leads us to the question of considering more general objectives; thus we might want to minimize an arbitrary *submodular* function subject to one knapsack covering constraint and a matroid constraint. However, note that in this case, strong lower bounds exist, even if we ignore the matroid constraint. Svitkina and Fleischer [14] prove the following: they show that given a submodular function $f(S)$ over a universe of size n, and a cardinality constraint $S \geqslant k$, it is NP-hard to get a factor better than $\Omega(\sqrt{\frac{n}{\log n}})$ for approximating the problem. They call this the SML (Submodular Minimization with cardinality Lower bounds) problem. Their lower bound is even stronger: it holds even for the special case of *monotone* submodular functions; and they derive lower bounds for bicriteria algorithms.

In the recent past, there has been a surge of work in the area of *submodular* maximization under various constraints. There have been several papers considering the problem of maximizing a monotone submodular function subject to matroid constraints, culminating in the breakthrough result of Vondrák [15](also see [2]). Vondrák [15] shows the optimal $(1 - 1/e)$-factor approximation for the problem via a *continuous greedy* process. Also, see the paper by Filmus and Ward [7] who give an elegant *non-oblivious* local search technique achieving the same approximation factor. In the space of non-monotone submodular functions, a recent result of Buchbinder et al. [1] (also see [6]) gives the optimal 1/2-factor approximation for the problem of unconstrained submodular maximization. The interested reader is referred to a presentation by Vondrák [16] (see slide 45) for an overview of the results in the area of submodular maximization subject to various (knapsack, matroid) constraints and their combinations.

Another line of work considers the problem of *minimizing* a submodular function subject to matroid constraints. Clearly, it does not make sense to minimize a monotone submodular function subject to such packing constraints. Thus, research has focused on the problem of minimizing a *symmetric* submodular function subject to matroid constraints. Originating from work by Shaddin [5], Goemans and Soto [8] prove a strong result that shows that symmetric submodular functions can be minimized subject to such constraints in polynomial time! In fact, their result extends to a wider class of constraints called "hereditary" constraints.

Thus, given the matroid constraints in the KCM problem, it is natural to ask about the relation between our problem and the existing literature. Here, we reiterate that most of the problems considered in literature are mostly purely packing problems: for instance, submodular maximization subject to matroid constraints or *symmetric* submodular minimization subject to matroid constraints. Our problem does not belong to the above frameworks because of the mixed packing covering nature of our constraints.

4.1 Organization

We present the relevant definitions in Section 5. We prove our result for the case of a cardinality matroid in Section 6 (see Theorem 2). We build on the case of cardinality

matroids to give the proof of Theorem 1 in Section 7. We state and prove the bicriteria factor approximation for knapsack cover subject to multiple matroid constraints in Section 8. We conclude with discussions and open problems in Section 9.

5 Preliminaries

Sets: In this paper, we will use the following notation: given *disjoint* sets A and B we will use $A + B$ to serve as shorthand for $A \cup B$. Vice versa, when we write $A + B$ it will hold implicitly that the sets A and B are disjoint.

We will use the letter U for the universe; the universe will typically contain n elements. Given a set A, let $\chi(A)$ denote its *characteristic vector*: this is a vector such that $\{\chi(A)\}_i = 1$ if $i \in A$ and 0 otherwise.

Also given an element-wise function f with domain U, we will extend it to *subsets* in the natural way: $f(S) = \sum_{i \in S} f(i)$ for $S \subseteq U$.

Monotone: A set function f is called *monotone* if $f(S) \leqslant f(T)$ whenever $S \subseteq T$.

Submodular: A set function $f : 2^U \to \mathbb{R}^+$ over a universe U is called *submodular* if the following holds for any two sets $A, B \subseteq U$:

$$f(A) + f(B) \geqslant f(A \cup B) + f(A \cap B)$$

Matroid: A matroid is a pair $\mathcal{M} = (U, \mathcal{I})$ where $\mathcal{I} \subseteq 2^U$, and

1. (Hereditary Property) $\forall B \in \mathcal{I}, A \subset B \implies A \in \mathcal{I}$.
2. (Extension Property) $\forall A, B \in \mathcal{I} : |A| < |B| \implies \exists x \in B \setminus A : A + x \in \mathcal{I}$

Matroids are generalizations of vector spaces in linear algebra and are ubiquitous in combinatorial optimization because of their connection with greedy algorithms. Typically the sets in $\mathcal{I}$ are called *independent* sets, this being an abstraction of linear independence in linear algebra. The maximal independent sets in a matroid are called the *bases* (again preserving the terminology from linear algebra). An important fact for matroids is that all bases have equal cardinality – this is an outcome of the Extension Property of matroids.

Any matroid is equipped with a *rank function* $r : 2^U \to \mathbb{R}^+$. The rank of a subset S is defined to be the size of the *largest* independent set contained in the subset S. By the Extension Property, this is well-defined. The rank function of any matroid is well-known to be a *monotone* submodular function. See the excellent text by Schrijver [13] for details.

Two commonly encountered matroids are the (i) *Cardinality Matroid:* Given a universe U and $r \in \mathbb{N}$, the *cardinality matroid* is the matroid $\mathcal{M} = (U, \mathcal{I})$, where a set A is *independent* (i.e. belongs to $\mathcal{I}$) iff $|A| \leqslant r$. (ii) *Partition Matroid:* Given a universe U and a partition of U as $U_1, \cdots, U_r$ and non-negative integers $r_1, \cdots, r_t$, the *partition matroid* is $\mathcal{M} = (U, \mathcal{I})$, where a set A belongs to $\mathcal{I}$ iff $|A \cap U_i| \leqslant r_i$ for all $i = 1, 2, \cdots, t$.

Knapsack Cover with Matroid Constraints (KCM): We are given n items of sizes $s(1), s(2), \cdots, s(n)$, and with costs $c(1), c(2), \cdots, c(n)$. We are also given a cumulative demand D and a matroid $\mathcal{M}$. A feasible solution is a subset F such that the cumulative size of the subset F is at least D, and so that the set F is *independent* in the matroid $\mathcal{M}$. The objective is to produce a feasible solution of *minimum* cumulative cost.

Knapsack Cover with Cardinality Matroid (KCCard): In this variant of KCM, we are given a number k and a specific subset $A \subseteq \{1, 2, \cdots, n\}$. A feasible solution is a subset F such that the cumulative size of F is at least D and *no more* than k elements are chosen from the subset A.

6 KCM with Cardinality Matroids

In this section, we will consider the Knapsack Cover problem subject to a cardinality matroid constraint. Recall that in this scenario, we are given a subset A of the universe U and we may pick at most k elements from A in a feasible solution.

We use $x(A)$ as a shorthand for $\sum_{i \in A} x_i$. The LP is as follows:

$$\text{LP}_1 \quad \begin{aligned} \min \quad & \sum_i c_i \cdot x_i \\ s.t. \quad & \sum_i s_i \cdot x_i \geqslant D \\ & x(A) \leqslant k \\ \forall i \quad & 0 \leqslant x_i \leqslant 1 \end{aligned}$$

We will call the constraints $x_i \geqslant 0$ and $x_i \leqslant 1$ as *trivial*; the constraints $\sum s_i \cdot x_i \geqslant D$ or $x(A) \leqslant k$ will be called the *non-trivial constraints*. In order that we may produce feasible solutions to the above LP, it is necessary to check that the feasibility problem is solvable in polynomial time. This is easy to do for the specific case of a cardinality matroid. In fact, we will prove the result in Lemma 3 for LPs corresponding to *arbitrary* matroid constraints.

▶ **Theorem 2.** *There is a* PTAS *for the* KCCard *problem.*

Proof. Let us consider a BFS solution to the above LP. It is easy to see that since there are *two* non-trivial constraints on the $x_i's$, in a BFS solution, at most 2 variables will be set fractionally, and the other variables will be set integrally. Also note that the only way that a BFS solution may have *two* fractional variables is if *both* the constraints $\sum s_i \cdot x_i \geqslant D$ and $x(A) \leqslant k$ are *tight*.

Renaming variables, let the fractional variables be x_1 and x_2. Since the other variables are integral, the following equalities hold: $s_1 x_1 + s_2 x_2 = D'$ and $x_1 + x_2 = k'$ where k' is an integer. Clearly because of the constraints $0 \leqslant x_i \leqslant 1$ we have that $0 \leqslant k' \leqslant 2$. If $k' = 0$, then $x_1 = x_2 = 0$, contrary to their being fractional. Likewise if $k' = 2$, then $x_1 = x_2 = 1$, again a contradiction. Thus, the only case is that $k' = 1$. Thus D' is a convex combination of s_1 and s_2. Without loss of generality, let $s_1 \geqslant D' \geqslant s_2$. Since the constraint $x(A) \leqslant k$ is a packing constraint, we will not be able to pick *both* of x_1 and x_2. We will simply pick x_1 (this makes the constraint $\sum s_i x_i \geqslant D$ feasible), and set x_2 to be 0.

Thus, in this process we have raised at most 1 fractional variable.

We now use the idea (in the manner of [12]) of *pruning*. Let $c_{\max}$ be the item of *highest cost* in OPT. Although we do not know this item, we can *guess* this item by running through the n possibilities for such an item. Using this guess, we can remove items i from the LP that have $c_i > c_{\max}$. Thus, in the above process, when we raise x_1 from its current fractional value to 1, we increase the cost of our solution by at most $c_{\max}$. Thereby the total cost of the solution generated is $\text{OPT} + c_{\max} \leqslant 2\text{OPT}$.

In order to achieve a $(1+\epsilon)$-factor approximation for any $\epsilon > 0$, we may guess *all* the items in OPT of cost at least $\epsilon \cdot \text{OPT}$. Note that there can be at most $1/\epsilon$ such items. Let S denote this set of items, of total cost $C(S)$. Remove the items of S from the input instance to derive a *modified* instance. Thus, the modified instance has its parameters D and k appropriately reduced. Note that if S is indeed the set of items in OPT of cost at least $\epsilon \cdot \text{OPT}$, then the optimal cost of the modified instance is at most $\text{OPT} - C(S)$.

Since we remove all the items of S from the LP, for every item i still remaining in the LP, it holds that $c_i \leqslant \epsilon \cdot \text{OPT}$. As before, the total cost of our solution is at most $(\text{OPT} - C(S)) + \epsilon \cdot \text{OPT} + C(S) = (1+\epsilon)\text{OPT}$.

Thus the algorithm considers all possible sets S of size $1/\epsilon$, and for each choice of S, solves the LP for the modified instance. The total run-time of the algorithm is $n^{\mathcal{O}(1/\epsilon)}$. ◀

Alternative Proof

The proof given above considers the BFS of the LP and proves certain properties of the BFS solution; in this sense, it is akin to *iterative rounding* (see [11]). However, one can provide another proof of the fact that any optimal solution to LP_1 can be modified to an optimal solution that contains at most *two* fractional variables. This proof goes via *cycle cancelling*.

In the normal usage of cycle cancelling, the "weight"(i.e. the LP values) is *shifted* between two variables. However, in the current scenario, we will need to shift the weight between *three* variables. This is the primary reason why we have *two* fractional variables.

In fact, the (more combinatorial styled) cycle cancelling technique applied to LP_1 shows that *any* feasible solution can be modified in this manner.

Suppose we are given a solution x to LP_1, and wlog rename the variables so that the fractional variables are $x_1, x_2, \cdots, x_m$. Given these fractional variables, we will attempt to change only the first 3 variables, by amounts δ_1, δ_2, and δ_3 (and leave the other variables - fractional or integral - unchanged). Thus the values of the variables x_1, x_2, x_3 will become $(x_1 + \delta_1), (x_2 + \delta_2), (x_3 + \delta_3)$. It is required that this operation does not violate the knapsack cover constraint or the cardinality constraint. We also want that in this process, the objective function does not degrade. These conditions translate to the following inequalities for $\delta_1, \delta_2, \delta_3$:

$$\begin{aligned}
&\delta_1 + \delta_2 + \delta_3 = 0 \\
&s_1\delta_1 + s_2\delta_2 + s_3\delta_3 \geqslant 0 \\
&c_1\delta_1 + c_2\delta_2 + c_3\delta_3 \leqslant 0
\end{aligned}$$

Since the system is homogeneous, the $(0, 0, 0)$ solution is always feasible. However, in order to perform cycle cancelling, we want a *non-zero* δ_i vector.

We may eliminate the variable δ_3 from the above system of inequations to get:

$$\begin{aligned}
&(s_1 - s_3)\delta_1 + (s_2 - s_3)\delta_2 \geqslant 0 \\
&(c_1 - c_3)\delta_1 + (c_2 - c_3)\delta_2 \leqslant 0
\end{aligned}$$

The pertinent question is whether this system of inequalities always has a non-zero solution in (δ_1, δ_2). The two inequalities correspond to two halfplanes in $\mathbb{R}^2$. The halfplanes are intersecting - for instance, the point $(0, 0)$ lies on both of them. But the intersection of two halfplanes is an unbounded region, so has a non-zero vector in it.

For instance, if $s_1 = s_3$ and $c_1 = c_3$, we would need to set $\delta_2 = 0$, but we may set $\delta_1 = 1$ (and consequently, δ_3 is set to -1).

While this method works for the cardinality matroid (where we have a single constraint corresponding to the matroid constraint), we do not know how to make this work for an arbitrary matroid.

7 KCM with arbitrary Matroids

In this section, we extend the result in Theorem 2 to arbitrary matroids. Let us recall the problem: the universe U consists of n items, each item with attributes costs $c(i)$ and sizes $s(i)$. There is a minimum coverage demand D. We are also given a matroid $\mathcal{M} = (U, \mathcal{I})$. A *feasible* solution S to the KCM is one that satisfies the knapsack covering constraints and

Guess $c_{\max}$, the costliest element in OPT.
for each guess of $c = c_{\max}$ **do**
  $A \leftarrow \{i : c(i) > c_{\max}\}$
  Augment LP_2 with constraints $x_i = 0, \forall i \in A$
  Solve LP_2; let the two fractional variables be x_1 and x_2
  If $s_1 \geqslant s_2$, raise x_1 to 1, $x_2 = 0$;
  else $x_1 = 0$, $x_2 = 1$.
  Let this solution be denoted by S_c.
end for
Output the subset S_c with the minimum cost

Figure 1 Main Algorithm.

such that $S \in \mathcal{I}$. Let $r(\cdot)$ denote the *rank* function of the matroid M. Then the LP for the KCM problem stands as follows:

$$\mathrm{LP}_2 : \begin{array}{ll} \min & \sum_i c(i) \cdot x_i \\ s.t. & \sum_i s(i) \cdot x_i \geqslant D \\ \forall S & x(S) \leqslant r(S) \\ \forall i & 0 \leqslant x_i \leqslant 1 \end{array}$$

Firstly, we show that we can solve the feasibility problem in polynomial time.

▶ **Lemma 3.** *The feasibility problem for* LP_2 *is solvable in polynomial time.*

Proof. Note that the feasibility problem may be converted into the following problem:

$$\mathrm{LP}_3 : \begin{array}{ll} \max & \sum_i s(i) \cdot x_i \\ \forall S & x(S) \leqslant r(S) \\ \forall i & 0 \leqslant x_i \leqslant 1 \end{array}$$

But this is precisely the maximum independent set question in a matroid and is well known to be solvable in polynomial time [13]. ◀

We are now ready to prove Theorem 1.

Proof (of Theorem 1). Let x^* denote a BFS solution to LP_2. Let there be ℓ fractional variables in the solution x^*. We will call any constraint that is not of the form $x_i \geqslant 0$ or $x_i \leqslant 1$ as "nontrivial". We will consider the non-trivial constraints that are satisfied with equality by the solution x^*. Given that there are ℓ fractional (and hence $n - \ell$ integral) variables, we have that precisely ℓ non-trivial constraints are tight. Moreover, this collection of tight constraints are *linearly independent*. We will also assume the following *normal* form for the tight constraints provided by a BFS. If a variable x_i is integral (either 0 or 1) in the solution x^*, we will consider the corresponding equation $x_i = 0$ or $x_i = 1$ as being tight. Thus, by virtue of linear independence of the tight constraints in a BFS, any non-trivial *tight* constraint has to contain at least *one* fractional variable.

There are two cases: either the constraint $\sum_i s(i)x_i \geqslant D$ is tight or is not. Let us consider the situation where this constraint is *not* tight. Thus the ℓ tight constraints are all of the form $x(S) \leqslant r(S)$ for some subset S. It is a well known property (for instance, see [11]) that the sets corresponding to these tight constraints may be assumed to form a *chain*.

We record this as a claim and for completeness, we prove this here. This proposition uses the fact that the rank function of a matroid is *submodular*.

▶ **Claim 4.** The *linearly independent* tight constraints $x(S) = r(S)$ can be assumed to form a *chain*. Thus, if there are ℓ linearly independent tight constraints, then the corresponding sets may be relabeled $S_1, S_2, \cdots, S_\ell$ such that $S_i \subset S_{i+1}$ for all $1 \leqslant i \leqslant (\ell - 1)$.

Proof. Consider any two tight constraints corresponding to sets S_1 and S_2. Thus, $x(S_1) = r(S_1)$ and $x(S_2) = r(S_2)$. Consider the following chain of inequalities:

$$\begin{aligned} r(S_1) + r(S_2) &\overset{tight}{=} x(S_1) + x(S_2) = x(S_1 \cap S_2) + x(S_1 \cup S_2) \\ &\leqslant r(S_1 \cap S_2) + r(S_1 \cup S_2) \overset{submodular}{\leqslant} r(S_1) + r(S_2) \end{aligned}$$

Thus equalities hold throughout the chain, and so $x(S_1 \cap S_2) = r(S_1 \cap S_2)$ and $x(S_1 \cup S_2) = r(S_1 \cup S_2)$. This means that $S_1 \cup S_2$ and $S_1 \cap S_2$ also are tight sets. Note that $\chi(S_1) + \chi(S_2) = \chi(S_1 \cap S_2) + \chi(S_1 \cup S_2)$. So, if S_1 and S_2 are such that $S_1 \setminus S_2 \neq \emptyset$ and $S_2 \setminus S_1 \neq \emptyset$, then we can replace one of the sets S_1 or S_2 in our system of linearly independent equations by $S_1 \cap S_2$ and $S_1 \cup S_2$. Repeating this process ensures that a maximal collection of *linearly independent* tight constraints form a *chain*. ◀

The sets S_i may not be equal since that would violate the linear independence of the corresponding constraints. Also, an earlier observation implies that $S_{i+1} \setminus S_i$ has to contain at least *one* fractional variable (and, bottoming out, this holds true for $S_1 \setminus \emptyset = S_1$ too). However since $x(S_{i+1}) - x(S_i)$ is an integer, there has to be at least 2 fractional variables in $S_{i+1} \setminus S_i$. But since the sets $S_{i+1} \setminus S_i$ are disjoint (for $1 \leqslant i \leqslant (\ell - 1)$ we thereby collect at least 2ℓ fractional variables. Since we started the argument with ℓ fractional variables, this implies that $\ell \geqslant 2\ell$, which is impossible since $\ell \geqslant 1$.

Thus, the constraint $\sum s(i)x_i \geqslant D$ has to be tight. But we can again decompose the tight constraints of the form $x(S) \leqslant r(S)$ as a chain of $(\ell - 1)$ constraints. A similar argument like the one above gives that there are at least $2(\ell - 1)$ fractional variables. Thus $\ell \geqslant 2(\ell - 1)$, and we get that $\ell \leqslant 2$. Since $\sum s(i)x_i \geqslant D$ is tight, this implies that there is precisely one tight constraint of the form $x(S) \leqslant r(S)$. Now we are back to the cardinality case, and we can mimic the proof of Theorem 2, and we thereby prove that LP_3 has at most 2 fractional variables.

Let the fractional variables be x_1 and x_2 (modulo renaming of variables). The tight constraint $\sum s(i)x_i \geqslant D$ simplifies to $s_1x_1 + s_2x_2 = D'$. Also we have that $x_1 + x_2 = k'$ for some *integral* k'. Since $0 < x_i < 1$ for all $i = 1, 2$, the only possibility is that $k' = 1$. This implies that D' is a convex combination of s_1 and s_2, and thus, one of these quantities is at least as large as D'; wlog, let $s_1 \geqslant D'$. Thus, we can raise the variable x_1 to 1 and *reduce* the variable x_2 to 0. This change in the variables x_1 and x_2 may potentially make the solution infeasible for the LP. To this end, let us consider the constraints in which the variables x_1 *or* x_2 appear. First, note that the constraint $\sum s(i)x_i \geqslant D$ is kept feasible, because of the choice of the fractional variable to raise. Now consider any matroid constraint $x(S) \leqslant r(S)$. If S contains both of items x_1 and x_2, then feasibility for this constraint is maintained (since $x_1 + x_2$ is not changed). Suppose that S contains only the item x_1. Then the constraint $x(S) \leqslant r(S)$ could not have been tight in the BFS solution x^*. This is because $r(S)$ is an integer whereas $x(S)$ contains just a single fractional variable and cannot be an integer. Thus, raising x_1 to 1 does not violate feasibility for this constraint. If S contains only the item x_2, the argument is simpler: the value of x_2 is lowered, and so feasibility is maintained for the packing constraint $x(S) \leqslant r(S)$.

Finally, given an $\epsilon > 0$, we prune items of the input by guessing the elements of *high-cost* (i.e. elements of cost $> \epsilon \cdot \mathrm{OPT}$) and modifying the input instance as in Theorem 2; this gives us the $(1+\epsilon)$-factor guarantee. ◀

8 Multiple Matroids

In this section, we consider the knapsack cover problem subject to *multiple matroid* constraints; call this the KCMM problem. In this problem, in addition to the knapsack cover constraint $\sum s(i)x_i \geqslant D$, we have $t \in \mathbb{N}$ matroid constraints. Given the matroids $\mathcal{M}_i$ for $1 \leqslant i \leqslant t$, let r_i denote the rank function for matroid $\mathcal{M}_i$. The IP for this problem is as follows:

$$\mathrm{IP}_4: \quad \begin{array}{ll} \min & \sum_i c(i) \cdot x_i \\ s.t. & \sum_i s(i) \cdot x_i \geqslant D \\ \forall S, t & x(S) \leqslant r_t(S) \\ \forall i & x_i \in \{0,1\} \end{array}$$

The feasibility problem for this IP is the *Matroid Intersection* problem. For $t \geqslant 3$, this is NP-hard: for instance, the Hamiltonian Circuit problem is a special case of the intersection of 3 matroids.

This leads to the hardness of approximation of the KCMM problem: for $t \geqslant 3$, it is impossible to achieve any factor approximation for the KCMM problem. We show that this obstacle can be overcome by considering *bicriteria* approximation algorithms. The main result of this section is that the KCMM problem allows good bicriteria approximations.

▶ **Theorem 5.** *There is a bicriteria approximation algorithm for the* KCMM *problem that outputs a solution S that satisfies the following properties: (i) $c(S) \leqslant c(\mathrm{OPT})$, (ii) S satisfies all the matroid constraints and (iii) $s(S) \geqslant \theta(\frac{D}{t})$.*

Proof. Consider the auxiliary IP parametrized by a quantity β. This IP is obtained by interchanging the roles of the objective and the knapsack cover constraint in IP_4.

$$\mathrm{IP}_5: \quad \begin{array}{ll} \max & \sum_i s(i) \cdot x_i \\ s.t. & \sum_i c(i) \cdot x_i \leqslant \beta \\ \forall S, t & x(S) \leqslant r_t(S) \\ \forall i & x_i \in \{0,1\} \end{array}$$

Thus we have converted the mixed packing covering problem IP_4 into a purely packing problem IP_5. The objective in the problem IP_5 is a submodular function (in fact a linear function) and the problem is to maximize this function subject to 1 knapsack (packing) constraint and t matroid constraints. There are efficient algorithms [10] that show $\mathcal{O}(t)$-factor approximations for this problem; denote this procedure by P.

We run the procedure P for various values of β in decreasing order; for each value of β, P gives an approximate solution to IP_5. We stop when the objective value of the solution returned by P is $D/\theta(t)$. Let this solution be S. We return S as the solution to IP_4 and the corresponding value of β is the objective value.

If there is a feasible solution F to IP_4 of objective value β', then F is a feasible solution to IP_5 (with $\beta = \beta'$) of objective value at least D. This is because F is a *feasible* solution to IP_4; so $\sum_{i \in F} s(i) \geqslant D$. So, procedure P on IP_5 with $\beta = \beta'$ produces a solution F' that has value $\sum_{i \in F} s(i) \geqslant D/\theta(t)$.

This proves the result.

◀

9 Discussion & Open Problems

In this paper we considered a mixed packing-covering problem called the KCM problem. We showed that it admits a PTAS. We note that the same result applies if we replace the matroid constraints by so-called *polymatroid* constraints (where the constraints are $x(S) \leqslant f(S)$ for an arbitrary *submodular* function f instead of the rank function).

However, we do not expect (even) constant factor approximations for most covering problems with matroid constraints. This is because the *feasibility problem* for such a covering problem with a matroid constraint may be NP-hard.

As an instance, let us consider the interval cover problem mentioned in Section 3. We will reduce the Vertex Cover problem to an interval cover instance, with a partition matroid. Let the input instance be $G = (V, E)$ and a number k; the decision problem is to find if G has a vertex cover of size at most k. The interval cover instance is constructed as follows. For every vertex $v \in V$, there is a *long* interval I_v; for different vertices v and v' the corresponding intervals are disjoint. Given an edge $e = (u, v)$, there are *two short* intervals corresponding to the edge $I_{e,u}$ and $I_{e,v}$. The span of $I_{e,u}$ for an edge e incident on the vertex u is contained within the interval I_u. The various intervals $I_{e,u}$ for different e's incident on u are all disjoint (and contained in I_u). The partition matroid constraints are as follows: out of all the intervals I_u ($u \in V$), we are allowed to pick at most k; and out of the intervals $I_{e,u}$ and $I_{e,v}$ (for $e = (u, v)$) we are allowed to pick at most 1. Suppose there is a vertex cover $S \subset V$ of size at k in G. Then our feasible solution for the interval cover problem will be constructed as follows: pick interval I_u for $u \in S$. Since S is a vertex cover, every edge e has one of its endpoints in S. Thus at least one of the short intervals $I_{e,u}$ or $I_{e,v}$ is not required, since the overlapping long interval I_u (or I_v) is picked. Thus the other short interval corresponding to e may be picked, while preserving the partition matroid constraint for $I_{e,u}$ and $I_{e,v}$. In the other direction, consider a feasible solution F for the interval cover instance. For any edge e, at most one of the short intervals $I_{e,u}$, $I_{e,v}$ may be in the feasible solution. Thus for the short interval that is absent, say, $I_{e,u}$, the long interval I_u has to be present in the solution. This means that the set of u's such that the interval I_u belongs to the solution F forms a vertex cover. Since feasibility stipulates at most k of the long intervals be selected, this means that we are able to extract a vertex cover of size at most k.

Despite feasibility being the principal obstacle for covering problems with matroid constraints, some open problems do remain. The principal open question is the following. In Section 8, we considered the case of t matroid constraints along with a knapsack cover constraint. However, note that for $t = 2$, the feasibility problem is in polynomial time (this is the Matroid Intersection problem for two matroids). So, a constant factor approximation algorithm is not ruled out for $t = 2$.

Our solution to the KCM problem involves solving multiple LP's. The knapsack cover (KC) problem has efficient greedy algorithms. Therefore, a natural question is whether the KCM problem has efficient greedy algorithms.

Our results do not rule out a FPTAS for the KCM problem. In fact, we are able to show FPTASes for the case of the KCM problem with a *partition matroid*. Is there a FPTAS for the KCM problem for arbitrary matroids?

Acknowledgements. We gratefully acknowledge discussions with Amit Kumar, Rajiv Gandhi, Guy Kortsarz and Yogish Sabharwal. We also gratefully acknowledge modifications suggested by the anonymous reviewers. The submission version contained only a 2-factor approximation for the KCM problem; a reviewer noted that a slight modification

of the same technique also yields a PTAS (that is now included as Theorem 1). The same reviewer also observed that the results essentially carry over to polymatroid constraints.

References

1 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *FOCS*, pages 649–658, 2012.

2 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

3 Tim Carnes and David B. Shmoys. Primal-dual schema for capacitated covering problems. In *IPCO*, pages 288–302, 2008.

4 Robert D. Carr, Lisa Fleischer, Vitus J. Leung, and Cynthia A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *SODA*, pages 106–115, 2000.

5 Shaddin Dughmi. Submodular functions: Extensions, distributions, and algorithms. a survey. *CoRR*, abs/0912.0322, 2009.

6 Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM J. Comput.*, 40(4):1133–1153, 2011.

7 Yuval Filmus and Justin Ward. A tight combinatorial algorithm for submodular maximization subject to a matroid constraint. In *FOCS*, pages 659–668, 2012.

8 Michel X. Goemans and José A. Soto. Symmetric submodular function minimization under hereditary family constraints. *CoRR*, abs/1007.2140, 2010.

9 Fabrizio Grandoni, Jochen Könemann, Alessandro Panconesi, and Mauro Sozio. A primal-dual bicriteria distributed algorithm for capacitated vertex cover. *SIAM J. Comput.*, 38(3):825–840, 2008.

10 Anupam Gupta, Viswanath Nagarajan, and R. Ravi. Thresholded covering algorithms for robust and max-min optimization. In *ICALP (1)*, pages 262–274, 2010.

11 Lap-Chi Lau, R. Ravi, and Mohit Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

12 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *FOCS*, pages 217–224, 1987.

13 A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

14 Zoya Svitkina and Lisa Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM J. Comput.*, 40(6):1715–1737, 2011.

15 Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*, pages 67–74, 2008.

16 Jan Vondrák. Submodular functions and their applications, Plenary talk at SODA. 2013.

Jumping Automata for Uniform Strategies

Bastien Maubert and Sophie Pinchinat

Université de Rennes 1, IRISA, Rennes, France

Abstract

The concept of uniform strategies has recently been proposed as a relevant notion in game theory for computer science. It relies on properties involving sets of plays in two-player turn-based arenas equipped with a binary relation between plays. Among the two notions of *fully-uniform* and *strictly-uniform* strategies, we focus on the latter, less explored. We present a language that extends CTL^* with a quantifier $\boxminus$ over all related plays, which enables to express a rich class of uniformity constraints on strategies. We show that the existence of a uniform strategy is equivalent to the language non-emptiness of a *jumping tree automaton*. While the existence of a uniform strategy is undecidable for *rational* binary relations, restricting to *recognizable* relations yields a 2Exptime-complete complexity, and still captures a class of two-player imperfect-information games with epistemic temporal objectives. This result relies on a translation from jumping tree automata with recognizable relations to two-way tree automata.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases Games, Imperfect information, Uniform strategies, Jumping automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.287

1 Introduction

Infinite-duration game models have been intensively studied for their applications in computer science [2] and logic [10]. First, infinite-duration games provide a natural abstraction of computing systems' non-terminating interaction [1] (think of a communication protocol between a printer and its users, or control systems). Second, infinite-duration games naturally occur as a tool to handle logical systems for the specification of non-terminating behaviors, such as for the propositional μ-calculus [8], leading to a powerful theory of automata, logics and infinite games [10] and to the development of algorithms for the automatic verification ("model-checking") and synthesis of hardware and software systems. In all cases, solving games aims at computing a strategy (of some distinguished player) whose outcomes fulfill ω-regular conditions meant to describe some desirable property.

In essence, ω-regular conditions are evaluated on individual plays, independently of other plays that result from the strategy. However, turning to imperfect-information games raises the need to deal with *sets* of plays, as the strategic decision has to be the same in indistinguishable situations [19]. This typical property of strategies in imperfect-information games is in general dealt aside from the ω-regular winning conditions. However, this splitting is a real issue when considering properties of strategies that mix, *e.g.* knowledge and time.

In an attempt to study this problem in depth, [14] introduced a general notion of *uniform strategies* and showed that it captures a variety of settings from the literature. Uniformity properties of strategies are expressed in a logic that combines standard temporal modalities with two new quantifiers, $\boxminus$ and $\boxbox$, that universally quantify over "related" plays according to some binary relation between plays. The difference between the two quantifiers is in their range. While the *strict* quantifier $\boxminus$ only quantifies over related plays that follow the strategy, the *full* quantifier $\boxbox$ ranges over all related plays in the arena.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 287–298

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

These quantifiers generalize the knowledge operator K of epistemic temporal logics [11]: classically, the semantics of K is a universal quantification over histories related to the actual one by some observational equivalence relation that captures the capabilities of the agent – perfect/imperfect recall, synchronous/asynchronous, ... [11]. In contrast, the setting of [14] allows for arbitrary binary relations for the semantics of the quantifiers, as long as they are *rational*, *i.e.* recognized by *finite state transducers* [6, 3]. Noticeably, most equivalence relations used in epistemic temporal logics are recognized by very simple transducers. Additionally, rational relations need not be equivalences, and can capture relations used in belief revision, and for modelling plausibility, with K45 or KD45 axiomatization [9].

Restricting to formulas that use only one kind of quantifier, the strict one or the full one, yields the notions of *strictly-uniform strategies* and *fully-uniform strategies* respectively. Deciding the existence of a fully-uniform strategy has already been investigated [15]: the problem is k-Exptime-complete for logical specifications that involve up to k nested $\boxminus$ quantifiers – 2Exptime-complete if $k \leq 2$.

In this work, we focus on the even more involved case of strict-uniformity which, as opposed to full-uniformity, does not allow for bottom-up constructions, as one cannot evaluate inner-most formulas before knowing the whole searched strategy. Actually, this intricate feature yields undecidability of the *strictly-uniform strategy problem* (i.e. the existence of a strictly uniform strategy) when the whole class of rational relations is considered. More precisely, we first show that this undecidability result holds even if we restrict to the subclass of regular[1] equivalence relations.

In an effort to better understand the difficulty of this problem, we propose an automata-based approach inspired by [21] for solving LTL games. We introduce and study *jumping tree automata (JTA)*, a class of tree automata which generalizes standard alternating tree automata. JTA are equipped with a binary relation between branches of trees and, in addition to normal behaviour of alternating automata, they allow for jumps between related nodes of the input tree. Intuitively, the jumps of JTA "implement" the meaning of the $\boxminus$ operator in the logic $\boxminus\mathsf{CTL}^*$, that we also define in this contribution as an extension of the logic $\boxminus\mathsf{LTL}$ introduced in [14]. We show that JTA capture the full logic $\boxminus\mathsf{CTL}^*$, and that from a uniformity property a JTA can be built that accepts the tree unfoldings of strictly-uniform strategies.

Although the language emptiness problem for JTA is unsurprisingly undecidable when considering arbitrary rational relations over branches of trees, we identify a decidable case when the class of binary relations between branches is confined to the well-known family of *recognizable* relations; basically, such relations only challenge a bounded amount of information in each branch. Decidability of JTA emptiness in this case is shown by an effective transformation of JTA with recognizable relations into equivalent two-way tree automata, and decidability of the strictly-uniform strategy problem for recognizable relations follows. More precisely, we establish that the emptiness problem for JTA with recognizable relations is Exptime-complete, and that the strictly-uniform strategy problem for this class of relations is 2Exptime-complete.

This latter result sheds light on phenomena in games of imperfect information. In such games, only *observation-based* strategies are allowed, enforcing players to play identically along plays that share the same sequence of observations. A weaker form of this requirement is the *knowledge-based*[2] strategies, *i.e.* players play identically along plays that yield the

[1] captured by synchronous transducers

[2] This vocabulary is highly misleading, and we now rather say *information-set-based* instead.

same information set. Although being rational, the observation-based equivalence relation is not recognizable, unlike the information-set-based equivalence relation. Interestingly, in two-player games with ω-regular winning objectives, the existence of an observation-based strategy implies the existence of an information-set-based one, hence looking for the latter is enough; but this does no longer hold for more players. Our results (on undecidability/decidability) distinguishing arbitrary rational relations from recognizable ones gives a new insight on the frontier between imperfect-information games with two players and games with more players.

Additionally, our automata-theoretic approach based on jumping tree automata yields an effective method to solve two-player imperfect-information games with epistemic temporal logic specifications of winning conditions. By interpreting the $\boxminus$ quantifier of the logic $\boxminus\mathsf{CTL}^*$ as the information-set-based knowledge operator, we can in a unified formalism express that, *e.g.* a strategy is information-set-based on the one hand, and fulfills some branching-time epistemic temporal property, on the other hand. Actually, we may even extend the setting to deal with several $\boxminus_i$ modalities, referring to different binary relations in a single specification: in particular, our decidability result for recognizable relations would still hold, with no additional complexity cost. It would then be possible to seek for a single player's information-set-based strategy with a fairly rich epistemic condition involving the knowledge of other players. Also, because recognizable relations are closed under intersection, such an extension would enable to reason on multi-player imperfect-information arenas about the existence of a coalition strategy (involving two meta players: the coalition and the anti-coalition) with distributed knowledge, as long as this knowledge can be bounded. Due to lack of space, we do not present this significant generalization in the paper.

The rest of the paper is organized as follows. In Section 2, we remind some notions concerning trees and game arenas. We present the language $\boxminus\mathsf{CTL}^*$ to specify uniform strategies in Section 3, and we prove in Section 4 that the strictly-uniform strategy problem is undecidable for regular equivalence relations. In Section 5 we define jumping tree automata, we show that they capture $\boxminus\mathsf{CTL}^*$ and we present a reduction of the strictly-uniform strategy problem to their non-emptiness. Finally, we prove in Section 6 that JTA with recognizable relations can be turned into equivalent two-way tree automata, yielding the decidability of the strictly-uniform strategy problem for recognizable relations.

2 Preliminaries

2.1 Basic notions on trees

For the rest of the paper, let $k \in \mathbb{N}$ be a natural number, and let $[k]$ denote the set $\{1, \dots, k\}$. An *infinite tree* is a nonempty set $t \subseteq [k]^*$ such that if $x \cdot i \in t$, then $x \in t$, and if $x \in t$, there exists $i \in [k]$ such that $x \cdot i \in t$. The elements of t are called *nodes*, and the empty word ϵ is the *root* of the tree. If $x \cdot i \in t$, we say that $x \cdot i$ is a *child* of x, and that x is the *parent* of $x \cdot i$. We will note $x \cdot \uparrow$ the parent of a node x: $(x \cdot i) \cdot \uparrow := x$. Note that $\epsilon \cdot \uparrow$ is undefined as the root has no parent. Given a node x of a tree t, we let $Paths(x)$ be the set of infinite paths $\pi = x_0 x_1 \dots$ in t such that $x_0 = x$ and for all i, x_{i+1} is a child of x_i. Also, for a path $\pi = x_0 x_1 x_2 \dots$, π^i denotes $x_i x_{i+1} x_{i+2} \dots$

Given a finite alphabet Σ, a Σ-*labelled tree* is a pair $T = (t, \ell)$, where t is a tree and $\ell : t \to \Sigma$ is a *labelling*. For a node $x = i_1 i_2 \dots i_n$ in t, $n \geq 0$, we define its *node word* $w(x)$ made of the sequence of labels from the root to this node: $w(x) := \ell(\epsilon)\ell(i_1)\ell(i_1 i_2) \dots \ell(i_1 \dots i_n)$.

2.2 Two-player turn-based game arenas

We consider two-player turn-based games played on finite graphs with vertices labelled with propositions. For the rest of the paper, we let AP be an infinite countable set of *atomic propositions*.

An *arena* is a finite structure $\mathcal{G} = (V, E, v_0, \nu)$ where $V = V_1 \uplus V_2$ is the finite set of *positions*, partitioned between positions of Player 1 (V_1) and those of Player 2 (V_2), $E \subseteq V \times V$ is the set of *edges*, $v_0 \in V$ is the *initial position* and $\nu : V \to 2^{AP}$ is a *valuation function*, mapping each position to a *finite* set of atomic propositions. We will assume that for each position v there is an atomic proposition p_v such that $p_v \in \nu(v')$ if, and only if, $v = v'$. For $v \in V$, we write $E(v) = \{v' \mid (v, v') \in E\}$ for the set of successors of v, and we assume that for all $v \in V$, $E(v)$ is nonempty. $Plays_*$ and $Plays_\omega$ are, respectively, the set of finite and infinite *plays*. For an infinite play $\pi = v_0 v_1 \dots$ and $i \in \mathbb{N}$, $\pi[i] := v_i$ and $\pi[0, i] := v_0 \dots v_i$.

A *strategy* for Player 1 is a partial function $\sigma : Plays_* \to V$ that maps a finite play ending in $v \in V_1$ to some successor of v. We say that a play $\pi \in Plays_\omega$ is *induced by* σ if for all $i \geq 0$ such that $\pi[i] \in V_1$, $\pi[i+1] = \sigma(\pi[0, i])$, and the *outcome of* σ, noted $\mathrm{Out}(\sigma) \subseteq Plays_\omega$, is the set of all infinite plays that are induced by σ. In the following, it is convenient to see a strategy σ as the tree T_σ, obtained by unfolding the arena and pruning moves that do not follow the strategy. Formally, a *strategy tree* $T = (t, \ell)$ is a 2^{AP}-labelled tree such that $\ell(\epsilon) = \nu(v_0)$ and for every $x \in t$, letting v be the unique position such that $p_v \in \ell(x)$:

- if $v \in V_1$ then x has a unique child x', and $\ell(x') = \nu(v')$ for some $v' \in E(v)$
- if $v \in V_2$ then x has exactly one child x' for each $v' \in E(v)$, and $\ell(x') = \nu(v')$.

3 Uniform strategies

A uniform strategy is a strategy subject to a uniformity constraint, such as being *observation-based* (see Example 2). In the purpose of automated synthesis of uniform strategies, we introduce a logical formalism $\boxminus\mathsf{CTL}^*$, in the spirit of [14], but extending the temporal features from LTL to CTL^* [7] (enabling us to capture, *e.g.* module-checking [13]). The logic $\boxminus\mathsf{CTL}^*$ is interpreted over infinite trees, and we use it to reason on strategies seen as trees. It contains a quantifier $\boxminus$ that universally quantifies over "related" finite plays, or equivalently over nodes in the strategy tree related by some binary relation $\rightsquigarrow$.

The set of well-formed $\boxminus\mathsf{CTL}^*$ state formulas is given by the following grammar:

$$\begin{array}{lll} \text{State formulas:} & \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid A\psi \mid \boxminus\varphi & \text{where } p \in AP \\ \text{Path formulas:} & \psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi\mathbf{U}\psi & \end{array}$$

Classically, we define the Boolean conjunction for both types of formula $\varphi \wedge \varphi$ and $\psi \wedge \psi$ with their expected meaning. Also, for each path formula ψ, we write $E\psi$ for the state formula $\neg A \neg\psi$, $\mathbf{F}\psi$ for $\top\mathbf{U}\psi$, $\mathbf{G}\psi$ for $\neg\mathbf{F}\neg\psi$ and for each state formula φ, we write $\diamondminus\varphi$ for $\neg\boxminus\neg\varphi$.

Given a binary relation $\rightsquigarrow$ over $(2^{AP})^*$, a 2^{AP}-labelled tree [3] $T = (t, \ell)$ and two nodes $x, y \in t$, we will abuse notations by writing $x \rightsquigarrow y$ for $w(x) \rightsquigarrow w(y)$ (that is their node words are related by $\rightsquigarrow$). We define the semantics of $\boxminus\mathsf{CTL}^*$ as follows, where $x \in t$ is a node and π is an infinite path:

[3] In fact, by 2^{AP}, we mean $2^{AP'}$, where AP' is some finite subset of AP.

$x \models p$ if $p \in \ell(x)$ $\quad$ $x \models \neg\varphi$ if $x \not\models \varphi$
$x \models \varphi_1 \vee \varphi_2$ if $w \models \varphi_1$ or $w \models \varphi_2$ $\quad$ $x \models A\psi$ if for all $\pi \in Paths(x)$, $\pi \models \psi$
$x \models \boxminus\varphi$ if for all $y \in t$ such that $x \rightsquigarrow y$, $y \models \varphi$
$\pi \models \varphi$ if $x_0 \models \varphi$, where $\pi = x_0x_1\ldots$ $\quad$ $\pi \models \neg\psi$ if $\pi \not\models \psi$
$\pi \models \psi_1 \vee \psi_2$ if $\pi \models \psi_1$ or $\pi \models \psi_2$ $\quad$ $\pi \models \mathbf{X}\psi$ if $\pi^1 \models \psi$
$\pi \models \psi_1 \mathbf{U} \psi_2$ if there exists $i \geq 0$ such that $\pi^i \models \psi_2$ and for all $0 \leq j < i$, $\pi^j \models \psi_1$

We extend the semantics to trees by writing $T \models \varphi$ whenever $\epsilon \models \varphi$. In particular, $T \models A\psi$ if every branch of T satisfies ψ.

▶ **Definition 1.** Let $\mathcal{G}$ be an arena, $\rightsquigarrow$ be a relation over $(2^{AP})^*$ and φ be a $\boxminus$CTL* formula. A strategy σ is $(\rightsquigarrow, \varphi)$-*uniform* if the strategy tree of σ satisfies φ, *i.e.* $T_\sigma \models \varphi$.

▶ **Example 2.** Consider the case of observation-based strategies in two-player games with imperfect information. Such games are played on graphs with edges labelled by *actions* and proceed as follows. In position v, Player 1 chooses an action a, and Player 2 chooses a new position reachable from v via an a-edge. The imperfect information is caused by the inability of Player 1 to see the very position chosen by Player 2, but only its *observation*; note that the same observation may be shared by several positions. In such games, strategies for Player 1 are required to assign the same action to observationally equivalent situations.

Such a requirement can be specified as a uniformity constraint in our framework. To do so, we convert actions into positions: choosing action a in node v is simulated by moving to the newly created position (v, a), and we enrich the arena labelling by atomic propositions p_o for observations, and by p_a in all positions of the form (v, a). Noting $\rightsquigarrow$ the observational equivalence relation, and defining $\psi_{Obs} := \mathbf{G} \bigwedge_{a \in Act} (\mathbf{X}p_a \rightarrow \boxminus E\mathbf{X}p_a)$, we have: a strategy is observation-based if, and only if, it is a $(\rightsquigarrow, A\psi_{Obs})$-uniform strategy. Indeed, a strategy tree that verifies $A\psi_{Obs}$ represents a strategy that, whenever it recommends action a after some finite play, it does so in all $\rightsquigarrow$-related (*i.e.* observationally equivalent) finite plays.

Beyond the example above, the approach clearly enables to represent various kinds of observation-based strategies by tuning relation $\rightsquigarrow$ according to the desired player's observational abilities and/or memory resources (perfect-recall, memoryless, synchronous, asynchronous, etc). Note that in this framework, a formula like $A\psi_{Obs} \wedge \varphi$ naturally combines on observation-based constraint with an arbitrary epistemic temporal winning condition φ. Also we refer the interested reader to [14] for more examples of uniform strategies.

4 A first undecidability result

In order to address automated synthesis of uniform strategies one has to make assumptions on the relation used in the semantics of the language $\boxminus$CTL*, and in particular how it can be finitely represented. We therefore consider the expressive class of *rational relations*.

4.1 Rational relations

We briefly recall the notions of rational relations and transducers. A *finite state transducer* over the alphabet Σ is a tuple $\mathcal{M} = (Q, q_0, Q_F, \Delta)$, where Q is a finite set of states, $q_0 \in Q$ is a distinguished *initial state*, $Q_F \subseteq Q$ is a set of *accepting states*, and $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Sigma \cup \{\epsilon\}) \times Q$ is a finite set of *transitions*. A transducer is *synchronous* if $\Delta \subseteq Q \times \Sigma \times \Sigma \times Q$.

Intuitively, a transducer reads a finite word $w \in \Sigma^*$ on its input tape, writes a finite word $w' \in \Sigma^*$ on its output tape, and accepts (w, w') if it reaches an accepting state. The relation $[\mathcal{M}]$ *recognized by* $\mathcal{M}$ is the set of pairs (w, w') accepted by $\mathcal{M}$.

It is well known that transducers recognize *rational* relations [3], and that synchronous transducers recognize *regular* relations [12].

For example, the observational equivalence relation for a player with asynchronous perfect-recall is rational, and accepted by a fairly simple transducer with one state per observation. For synchronous perfect-recall, the relation is regular and is accepted by an even simpler synchronous transducer with only one state.

4.2 The problem SUS and its undecidability

Back to our central topic on uniform strategies, we consider the following decision problem:

$$\mathrm{SUS} := \left\{ (\mathcal{G}, \mathcal{M}, \varphi) \;\middle|\; \begin{array}{l} \mathcal{G} \text{ is a finite arena, } \mathcal{M} \text{ is a transducer, } \varphi \in \boxminus\mathsf{CTL}^* \\ \text{and there exists a } ([\mathcal{M}], \varphi)\text{-uniform strategy in } \mathcal{G}. \end{array} \right\}$$

It can be proven, by reduction of the Post Correspondence Problem, that SUS is undecidable, but we propose here the stronger result.

▶ **Theorem 3.** SUS *is undecidable for rational relations, even if we restrict to regular equivalence relations.*

The rest of the section is dedicated to the proof of Theorem 3. We reduce the *distributed strategy problem for three-player imperfect-information games with safety objective*, as addressed by [16, 4]. We present the problem as stated in [4], in which two players with imperfect information (Player A and Player B) play against nature (the third player). Formally, let Act_A (resp. Act_B) be a finite set of available actions for Player A (resp. Player B), and Γ_A (resp. Γ_B) be a finite set of observations for Player A (resp. Player B). We write $Act = Act_A \times Act_B$.

A finite *safety imperfect-information game* is a tuple $\mathcal{G}^{imp} = (V, E, v_0, \gamma_A, \gamma_B)$ where $E \subseteq V \times Act \times V$ is a set of transitions, $\gamma_X : V \to \Gamma_X$ is an observation function ($X \in \{A, B\}$), and with a designated subset $Bad \subseteq V$ of "bad" positions that Player A and Player B should avoid. In each round, Player X chooses an action $c_X \in Act_X$, which gives an action profile $x = (c_A, c_B)$, and nature chooses a next position in $E(v, x) = \{v' \mid (v, x, v') \in E\}$. We suppose that all actions are allowed in every position: for all $v \in V$, $a \in Act_A$ and $b \in Act_B$, we have $E(v, (a, b)) \neq \emptyset$. The observation functions are extended to finite plays $\rho = v_0 x_0 v_1 \dots x_{n-1} v_n$ by letting $\gamma_X(\rho) = \gamma_X(v_0)\gamma_X(v_1)\dots\gamma_X(v_n)$. Note that actions are not observed.

A strategy for Player X is a partial mapping $\sigma_X : (V \cdot Act)^* \cdot V \to Act_X$ that assigns an action to each finite play. It must be observation-based: for any finite plays ρ and ρ' such that $\gamma_X(\rho) = \gamma_X(\rho')$, $\sigma_X(\rho) = \sigma_X(\rho')$. A *distributed strategy* is a pair (σ_A, σ_B) of strategies for Player A and Player B. The *outcome* of a distributed strategy is the set of infinite plays that are both induced by σ_A and σ_B, and a distributed strategy is winning if no play in the outcome ever visits a position in Bad.

It is well known [16, 4] that the following problem is undecidable : *given a safety imperfect-information game, does there exist a winning distributed strategy?*

We reduce this problem to SUS. Let $\mathcal{G}^{imp} = (V, E, v_0, \gamma_A, \gamma_B)$ be an imperfect-information arena with observations Γ_A and Γ_B and bad states Bad. We build a game arena $\mathcal{G} = (V', E', v_0', \nu)$ in which Player 1 plays for both Player A and Player B, and Player 2 plays for nature; Figure 1 shows how each transition in $\mathcal{G}^{imp}$ is transformed into a widget in $\mathcal{G}$.

The set of positions $V' = V_1^A \uplus V_1^B \uplus V_2$ is split into three: in positions of $V_1^A = V$, Player 1 simulates moves of Player A, in positions of $V_1^B = V \times Act_A$, Player 1 simulates moves of Player B, and in positions of $V_2 = V \times Act_A \times Act_B$, Player 2 simulates moves

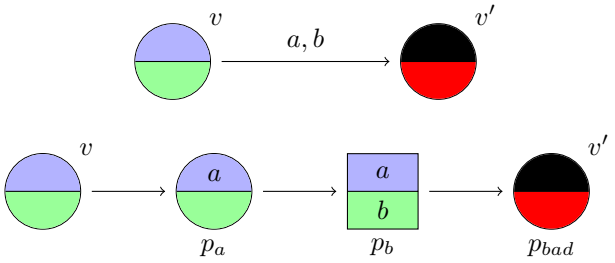


Figure 1 Coding in $\mathcal{G}$ a transition $(v,(a,b),v')$ of $\mathcal{G}^{imp}$, with $v' \in Bad$. Colors represent the observations of Player A and Player B.

of nature. Hence for all v, v', a, b, we have $(v,(v,a)) \in E'$, $((v,a),(v,a,b)) \in E'$, and if $(v,(a,b),v') \in E$ then $((v,a,b),v') \in E'$. For each action $c \in Act_X$, p_c labels positions in which the last move was Player 1 simulating the choice of action c by Player X. In addition, "bad" positions are marked with proposition p_{bad}. As it is assumed in our definition of game arenas that each position v is identified by an atomic proposition p_v, we keep it silent in the following formal definition of the labelling. We consider the set of atomic propositions $\{p_c \mid c \in Act_A \cup Act_B\} \cup \{p_{bad}\}$, and we label the arena as follows: if $v \in Bad$, $\nu(v) = \{p_{bad}\}$, $\nu(v,a) = \{p_a, p_{bad}\}$ and $\nu(v,a,b) = \{p_b, p_{bad}\}$; otherwise, $\nu(v) = \emptyset$, $\nu(v,a) = \{p_a\}$ and $\nu(v,a,b) = \{p_b\}$. Finally, we set $v'_0 = v_0$.

Clearly, since Player 1 plays for the coalition $\{A,B\}$, we expect each branch of her strategy to satisfy the following path formula:

$$\psi_{Safe} := \mathbf{G}\neg p_{bad}. \tag{1}$$

For a finite play $\rho = v_0(v_0,a_0)(v_0,a_0,b_0)v_1\ldots$, we note $\gamma_X(\rho) = \gamma_X(v_0)\gamma_X(v_1)\ldots$

We want to enforce that when Player 1 simulates a move of Player X, her choice is only based on Player X's observation. To do so, we define the symmetric and transitive relation $\rightsquigarrow$ over V'^* that relates two sequences of positions if they end in positions belonging to the same player (A or B), and are observationally equivalent for this player:[4]

$$\rightsquigarrow := \left\{ (\rho,\rho') \;\middle|\; \begin{array}{l} \text{last}(\rho) \in V_1^1 \text{ and last}(\rho') \in V_1^A \text{ and } \gamma_A(\rho) = \gamma_A(\rho'), \text{ or} \\ \text{last}(\rho) \in V_1^B \text{ and last}(\rho') \in V_1^B \text{ and } \gamma_B(\rho) = \gamma_B(\rho') \end{array} \right\}. \tag{2}$$

Note that for the sake of clarity we defined $\rightsquigarrow$ on V^* instead of $(2^{AP})^*$, but by considering the propositions $p_v \in AP$ ($v \in V$) and working with the alphabet $\Sigma = \{\{p_v\} \cup \nu(v) \mid v \in V\}$, one can easily rephrase relation $\rightsquigarrow$ of Equation (2) as a binary relation over Σ^*.

The following path formula states that whenever Player 1 simulates a move of Player X, she chooses the same action in all plays observationally equivalent for Player X:

$$\psi_{Obs} := \mathbf{G} \bigwedge_{c \in Act_A \cup Act_B} \mathbf{X}p_c \rightarrow \boxminus E\mathbf{X}p_c. \tag{3}$$

Combining Equations (1) and (3) we get the following reduction.

▶ **Lemma 4.** *There is a winning distributed strategy in $\mathcal{G}^{imp}$ if and only if there is a $(\rightsquigarrow, A(\psi_{Obs} \wedge \psi_{Safe}))$-uniform strategy in $\mathcal{G}$.*

[4] For a finite word w, last(w) classically denotes its last letter.

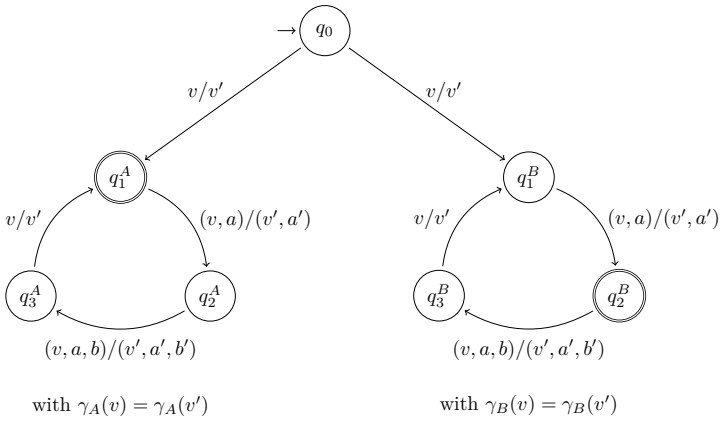


Figure 2 The synchronous transducer $\mathcal{M}_{A,B}$.

We show that $\rightsquigarrow$ is regular.

Consider the synchronous transducer $\mathcal{M}_{A,B}$ of Figure 2. State q_0 is the initial state (ingoing arrow), q_1^A and q_2^B are final states (doubly circled). Transducer $\mathcal{M}_{A,B}$ works as follows: before reading a word w, the transducer guesses whether we are interested in Player A or Player B's observation. In the first case it goes to the left, reads w and writes a word w' observationally equivalent for Player A (remember that actions are not observed). The pair (w, w') is accepted if w (and w') indeed ends in a position of Player A. The second case is symmetric. So $\mathcal{M}_{A,B}$ recognizes $\rightsquigarrow$, hence $\rightsquigarrow$ is regular. Note that $\rightsquigarrow$ is not reflexive, but its reflexive $\sim$ closure is also regular (plug in $\mathcal{M}_{A,B}$ the synchronous transducer for the identity relation). Lemma 4 would also hold for $\sim$, which concludes the proof of Theorem 3.

5 Intermezzo: jumping tree automata

We remind the notions of alternating tree automata and two-way tree automata, and we define *jumping tree automata (JTA)*. For an introduction to the theory of automata on infinite trees see [20].

For a set X, $\mathbb{B}^+(X)$ is the set of positive boolean formulas over X, *i.e.* formulas built with elements of X as atomic propositions and using only connectives $\vee$ and $\wedge$. We also allow for formulas $\top$ and $\bot$, and $\wedge$ has precedence over $\vee$. Elements of $\mathbb{B}^+(X)$ are denoted by $\alpha, \beta \ldots$ Let $D \subseteq [k] \cup \{\epsilon, \uparrow, \diamondsuit, \boxminus\}$ be a set of *directions*. A D-automaton is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, C)$ where Σ is a finite alphabet, Q a finite set of states, $q_0 \in Q$ an initial state, C an accepting condition, and $\delta : Q \times \Sigma \to \mathbb{B}^+(D \times Q)$ a transition function. If D contains $\diamondsuit$ or $\boxminus$ then we additionally require the automaton to be equipped with a binary relation $\rightsquigarrow$ over Σ^*.

We note $D_A = [k]$, $D_\uparrow = D_A \cup \{\epsilon, \uparrow\}$ and $D_\rightsquigarrow = D_A \cup \{\diamondsuit, \boxminus\}$. D_A-automata are *alternating tree automata*, $D_\uparrow$-automata are *two-way alternating tree automata*, and $D_\rightsquigarrow$-automata are *jumping alternating tree automata (JTA)*.

In this work we are not interested in identifying children of a node in a tree, but only in existentially or universally quantifying over them. Therefore, for the sake of readability, we use the abstract directions $D_A = \{\Diamond, \Box\}$ instead of $[k]$, as in alternating automata on graphs [5, 17]. We use notation $[\Diamond, q]$ as a macro for $[1, q] \vee \ldots \vee [k, q]$, and similarly $[\Box, q]$ for $[1, q] \wedge \ldots \wedge [k, q]$. All that we establish on this version of jumping tree automata can be easily adapted to the setting where directions are made explicit.

Acceptance of an input tree $T = (t, \ell)$ in a designated node $x_0 \in t$ by a D-automaton $\mathcal{A}$ is classically defined on a two-player game between Eve (the proponent) and Adam (the opponent): Let $T = (t, \ell)$ be a Σ-labelled tree, $x_0 \in t$, and $\mathcal{A} = (\Sigma, Q, \delta, q_0, C)$ be a D-automaton. We define the game $\mathcal{G}^{x_0}_{\mathcal{A},T} = (V, E, v_0)$: The set of positions is $V = t \times Q \times \mathbb{B}^+(D \times Q)$, the initial position is $(x_0, q_0, \delta(q_0, \ell(x_0)))$, and a position (x, q, α) belongs to Eve if α is of the form $\alpha_1 \vee \alpha_2$, $[\Diamond, q']$ or $[\diamondsuit, q']$; otherwise it belongs to Adam.

Moves in $\mathcal{G}^{x_0}_{\mathcal{A},T}$ are defined by Rules (4a)-(4e).

$$(x, q, \alpha_1 \dagger \alpha_2) \to (x, q, \alpha_i) \qquad \text{where } \dagger \in \{\vee, \wedge\} \text{ and } i \in \{1, 2\} \tag{4a}$$

$$(x, q, [\bigcirc, q']) \to (y, q', \delta(q', \ell(y))) \text{ where } \bigcirc \in \{\Diamond, \Box\} \text{ and } y \text{ is a child of } x \tag{4b}$$

$$(x, q, [\epsilon, q']) \to (x, q', \delta(q', \ell(x))) \tag{4c}$$

$$(x, q, [\uparrow, q']) \to (y, q', \delta(q', \ell(y))) \qquad \text{where } y \text{ is } x\text{'s parent} \tag{4d}$$

$$(x, q, [\ominus, q']) \to (y, q', \delta(q', \ell(y))) \qquad \text{where } \ominus \in \{\diamondsuit, \boxminus\} \text{ and } x \rightsquigarrow y \tag{4e}$$

Positions of the form $(x, q, \top)$ and $(x, q, \bot)$ are deadlocks, winning for Eve and Adam respectively. Positions of the form $(\epsilon, q, [\uparrow, q'])$ are also deadlocks as the root of a tree has no parent; they are winning for Adam.

Most of the time the starting node x_0 will be the root ϵ of the tree, and in this case we simply write $\mathcal{G}_{\mathcal{A},T}$ instead of $\mathcal{G}^{\epsilon}_{\mathcal{A},T}$. The winning condition of $\mathcal{G}^{x_0}_{\mathcal{A},T}$ results from the acceptance condition C of $\mathcal{A}$. In this work, we consider *parity condition*: C is a mapping that assigns to each state of the automaton a natural number called its *colour*, and an infinite play is winning for Eve if the least colour seen infinitely often during the play is even. A tree T is *accepted* by $\mathcal{A}$ if Eve has a winning strategy in $\mathcal{G}_{\mathcal{A},T}$, and we denote by $\mathcal{L}(\mathcal{A})$ the set of trees accepted by $\mathcal{A}$.

We first prove that the class of JTA is closed by complementation, by a construction inspired from classical alternating automata. To this aim we classically define the *dualization* $\widetilde{\alpha}$ of a formula $\alpha \in \mathbb{B}^+(D_{\rightsquigarrow} \times Q)$ by induction as follows: $\widetilde{\top} = \bot$, $\widetilde{\bot} = \top$, $\widetilde{\alpha \vee \beta} = \widetilde{\alpha} \wedge \widetilde{\beta}$, $\widetilde{\alpha \wedge \beta} = \widetilde{\alpha} \vee \widetilde{\beta}$, $\widetilde{[\Diamond, q]} = [\Box, q]$, $\widetilde{[\Box, q]} = [\Diamond, q]$, and, as expected, $\widetilde{[\diamondsuit, q]} = [\boxminus, q]$ and $\widetilde{[\boxminus, q]} = [\diamondsuit, q]$.

▶ **Definition 5.** Let $\mathcal{A} = (\Sigma, Q, \delta, q_0, C)$ be a jumping tree automaton. We define *the complement* of $\mathcal{A}$ by $\widetilde{\mathcal{A}} = (\Sigma, Q, \widetilde{\delta}, q_0, \widetilde{C})$, where $\widetilde{C}(q) = C(q) + 1$, and $\widetilde{\delta}(q, a) = \widetilde{\delta(q, a)}$.

▶ **Lemma 6.** *Eve wins $\mathcal{G}^{x_0}_{\widetilde{\mathcal{A}},T}$ if, and only if, Eve loses $\mathcal{G}^{x_0}_{\mathcal{A},T}$.*

Proof. The arenas of both games are isomorphic, and if a position belongs to Eve in $\mathcal{G}^{x_0}_{\mathcal{A},T}$ then its counterpart in $\mathcal{G}^{x_0}_{\widetilde{\mathcal{A}},T}$ belongs to Adam, and vice versa. Also, a play is winning for a player in one game if and only if its counterpart in the other game is winning for the opponent. From this we have that a winning strategy for a player in one game gives a winning strategy for its opponent in the other, and because parity games are determined [23], the result follows. ◀

We now establish that JTA capture $\boxminus\mathsf{CTL}^*$.

▶ **Theorem 7.** *Let φ be an $\boxminus\mathsf{CTL}^*$ formula and $\rightsquigarrow$ a binary relation over $(2^{AP})^*$. There exists a jumping tree automaton $\mathcal{A}_\varphi$ of size exponential in $|\varphi|$ and equipped with $\rightsquigarrow$ such that $T \in \mathcal{L}(\mathcal{A}_\varphi)$ if, and only if, $T \models \varphi$.*

The following expresses that JTA are adequate machines for the decision problem SUS.

▶ **Theorem 8.** *Let $\mathcal{G}$ be a finite arena, $\rightsquigarrow$ be a binary relation and $\varphi \in \boxminus\mathsf{CTL}^*$. There is a jumping tree automaton $\mathcal{A}$ equipped with $\rightsquigarrow$ such that σ is a $(\rightsquigarrow, \varphi)$-uniform strategy if, and only if, $T_\sigma \in \mathcal{L}(\mathcal{A})$. Moreover, $\mathcal{A}$ can be chosen of size $|\mathcal{A}| = O(|\mathcal{G}| \times 2^{|\varphi|})$.*

Proof. One simply builds from $\mathcal{G}$ a nondeterministic tree automaton $\mathcal{A}_\mathcal{G}$ that accepts the set of strategy trees for Player 1 in $\mathcal{G}$. $\mathcal{A}_\mathcal{G}$ is of size linear in $|\mathcal{G}|$. Then, by Theorem 7, one can build a JTA $\mathcal{A}_\varphi$ equipped with $\rightsquigarrow$ that accepts the models of φ. This automaton is of size $|\mathcal{A}_\varphi| = O(2^{|\varphi|})$. The product $\mathcal{A} = \mathcal{A}_\mathcal{G} \times \mathcal{A}_\varphi$ is thus a JTA that accepts precisely the strategy trees that verify φ. ◀

The following is a direct consequence of Theorems 3 and 8.

▶ **Corollary 9.** *The emptiness problem for jumping tree automata with regular equivalence relations is undecidable.*

6 The special case of recognizable relations

We first show that JTA with recognizable relations (Definition 10) can be effectively transformed into equivalent two-way tree automata of linear size (Theorem 12). We then get two central corollaries: first, the emptiness problem for JTA with recognizable relations is decidable (Corollary 13), and second, the problem SUS becomes decidable when restricted to recognizable relations (Corollary 14) and it is 2ExpTime-complete.

▶ **Definition 10.** A relation $\rightsquigarrow \subseteq \Sigma^* \times \Sigma^*$ is *recognizable* if there is a family of regular languages $\mathcal{L}_1, \mathcal{L}'_1, \ldots, \mathcal{L}_n, \mathcal{L}'_n \subseteq \Sigma^*$ such that $\rightsquigarrow = \bigcup_{i=1}^{n} \mathcal{L}_i \times \mathcal{L}'_i$.

We rather use another, easily shown equivalent, characterization. Let $\overline{w} \in \Sigma^*$ denote the mirror image of a word $w \in \Sigma^*$ (*i.e.* $\overline{\epsilon} = \epsilon$ and $\overline{aw} = \overline{w}a$).

▶ **Proposition 11.** *A relation $\rightsquigarrow$ over Σ^* is recognizable if the language $\mathcal{L}_\rightsquigarrow := \{\overline{u}\#v \mid u \rightsquigarrow v\}$ is regular (accepted by a finite state automaton), with fresh symbol $\# \notin \Sigma$.*

Given a recognizable relation $\rightsquigarrow$, we write $\mathcal{B}_\rightsquigarrow = (\Sigma \cup \{\#\}, Q_\rightsquigarrow, \delta_\rightsquigarrow, s_0, F_\rightsquigarrow)$ for the canonical deterministic finite state automaton of $\mathcal{L}_\rightsquigarrow$, with standard interpretation of the components of $\mathcal{B}_\rightsquigarrow$. We will abuse vocabulary by saying that "$\mathcal{B}_\rightsquigarrow$ *recognizes relation* $\rightsquigarrow$".

A typical example of recognizable relation relates to *information-set-based* strategies in two-player imperfect-information games, where finite plays are related whenever they share the same *information set* according to the player under imperfect information, *i.e.* the set of states the player considers possible after "observing" the course of the game. As in finite arenas information sets are finitely many, a mere powerset construction provides the deterministic finite state automaton that recognizes the information-set-based relation.

A central result of our contribution is an alternative characterization of JTA with recognizable relations.

▶ **Theorem 12.** *Let $\mathcal{A}$ be a jumping tree automaton with a recognizable relation $\rightsquigarrow$. Then, there is a two-way tree automaton $\widehat{\mathcal{A}}$ of size $O(|\mathcal{A}| \times |\mathcal{B}_\rightsquigarrow|)$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\widehat{\mathcal{A}})$.*

We sketch the proof: when JTA $\mathcal{A}$ goes down along a branch of a tree, $\widehat{\mathcal{A}}$ behaves likewise. The critical points are the jump instructions of $\mathcal{A}$, say in a node x of the tree. At this point, $\widehat{\mathcal{A}}$ stops behaving like $\mathcal{A}$ and enters a *jump mode* that simulates this jump: $\widehat{\mathcal{A}}$ triggers automaton $\mathcal{B}_\rightsquigarrow$ and goes up to the root while running $\mathcal{B}_\rightsquigarrow$ on the reversed branch. When reaching the root, $\mathcal{B}_\rightsquigarrow$ has read $\overline{w(x)}$, the mirror of the node word of x, and $\widehat{\mathcal{A}}$ feeds $\mathcal{B}_\rightsquigarrow$ with

the # symbol. Then $\widehat{\mathcal{A}}$ goes down along some or all (depending on the jump: respectively existential or universal) branch(es) of the tree while still running $\mathcal{B}_{\rightsquigarrow}$. Each time $\mathcal{B}_{\rightsquigarrow}$ reaches a final state in a node y, it has read $\overline{w(x)}\#w(y)$, which by Proposition 11 means that $x \rightsquigarrow y$; the jump mode can then be closed. In this case, $\widehat{\mathcal{A}}$ resumes the simulation of $\mathcal{A}$.

From Theorem 12, we immediately infer two significant corollaries.

▶ **Corollary 13.** *The non-emptiness problem for jumping tree automata with recognizable relations is decidable in time exponential in the size of the jumping automaton and of the word automaton recognizing the relation.*

Proof. This is a direct consequence of Theorem 12 along with the EXPTIME complexity of the non-emptiness problem for two-way alternating tree automata [22]. ◀

Also, combining Theorem 8 with Corollary 13 gives a decidable subproblem of SUS.

▶ **Corollary 14.** *The problem*

$$\mathrm{SUS}_{rec} := \left\{ (\mathcal{G}, \mathcal{B}_{\rightsquigarrow}, \varphi) \;\middle|\; \begin{array}{c} \mathcal{G} \text{ is a finite arena, } \rightsquigarrow \text{ is a recognizable relation, } \varphi \in \boxminus\mathsf{CTL}^* \\ \text{and there exists a } (\rightsquigarrow, \varphi)\text{-uniform strategy in } \mathcal{G}. \end{array} \right\}$$

is 2EXPTIME*-complete, and the decision procedure synthesizes a solution strategy (if any).*

Proof. Let $\mathcal{G}$ be a finite arena, $\rightsquigarrow$ be a recognizable relation, and $\varphi \in \boxminus\mathsf{CTL}^*$. By Theorem 8, there is a JTA $\mathcal{A}$ equipped with relation $\rightsquigarrow$ of size $O(|\mathcal{G}| \times 2^{|\varphi|})$ whose language is the tree unfoldings of $(\rightsquigarrow, \varphi)$-uniform strategies. By Corollary 13, emptiness of $\mathcal{L}(\mathcal{A})$ can be decided in time $2^{O(|\mathcal{G}| \times 2^{|\varphi|} \times |\mathcal{B}_{\rightsquigarrow}|)}$, hence $(\mathcal{G}, \mathcal{B}_{\rightsquigarrow}, \varphi) \in \mathrm{SUS}_{rec}$ can be decided in 2EXPTIME. Furthermore, a strategy (if any) is synthesized when checking the non-emptiness of the JTA $\mathcal{A}$: we can make $\mathcal{A}$ an equivalent two-way tree automaton, then an equivalent non-deterministic tree automaton and apply classic algorithms to exhibit a regular tree (if any).

The 2EXPTIME-hardness of SUS_{rec} comes from the 2EXPTIME-completeness of solving LTL games [18]: given an arena $\mathcal{G}$ and an LTL formula ψ, we consider the instance $(\mathcal{G}, \mathcal{B}_{\rightsquigarrow}, A\psi)$ of SUS_{rec}, where $\rightsquigarrow$ is *e.g.* the full relation. Clearly, as there is no $\boxminus$ quantifier in ψ, a strategy in $\mathcal{G}$ which satisfies $A\psi$ is a $(\rightsquigarrow, A\psi)$-uniform strategy in $\mathcal{G}$, and vice versa. ◀

Future work: We want to study the links between jumping tree automata and the logic $\boxminus\mathsf{L}_\mu$, *i.e.* extending the full μ-calculus with the $\boxminus$ quantifier. We conjecture that jumping automata capture $\boxminus\mathsf{L}_\mu$, but the other direction deserves further investigations. Also, in order to handle richer uniformity constraints, we seek for a class of relations for which jumping tree automata languages would exceed the line of ω-regulararity, and still enjoy a decidable emptiness problem. Finally, the notion of uniform strategy generalizes to the case of concurrent game structures, and investigating what results are preserved is yet another interesting perspective, as well as possible extensions to strategic logics, with modalities quantifying over uniform strategies. We foresee that questions of strategy context may become even trickier than usual when rational relations are involved.

References

1 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):672–713, 2002.

2 K.R. Apt and E. Grädel. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, 2011.

3 Jean Berstel. *Transductions and context-free languages*, volume 4. Teubner Stuttgart, 1979.

4 Dietmar Berwanger and Lukasz Kaiser. Information tracking in games on graphs. *Journal of Logic, Language and Information*, 19(4):395–412, 2010.

5 Mikolaj Bojanczyk. Two-way alternating automata and finite models. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 833–844. Springer, 2002.

6 Samuel Eilenberg. *Automata, languages, and machines*, volume 1. Access Online via Elsevier, 1974.

7 E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

8 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, pages 368–377. IEEE Computer Society, 1991.

9 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about knowledge*, volume 4. MIT press Cambridge, 1995.

10 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

11 Joseph Y. Halpern and Moshe Y. Vardi. The complexity of reasoning about knowledge and time. 1. Lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237, 1989.

12 Bakhadyr Khoussainov and Sasha Rubin. Automatic structures: overview and future directions. *Journal of Automata, Languages and Combinatorics*, 8(2):287–301, 2003.

13 Orna Kupferman and Moshe Y. Vardi. Module checking revisited. In Orna Grumberg, editor, *CAV*, volume 1254 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 1997.

14 Bastien Maubert and Sophie Pinchinat. A general notion of uniform strategies. *To appear in International Game Theory Review*, 2013.

15 Bastien Maubert, Sophie Pinchinat, and Laura Bozzelli. The complexity of synthesizing uniform strategies. In Proceedings 1st International Workshop on *Strategic Reasoning,* Rome, Italy, March 16-17, 2013, volume 112 of *Electronic Proceedings in Theoretical Computer Science*, pages 115–122. Open Publishing Association, 2013.

16 Gary Peterson, John H. Reif, and Salman Azhar. Lower bounds for multiplayer noncooperative games of incomplete information. *Computers & Mathematics with Applications*, 41(7):957–992, 2001.

17 Nir Piterman and Moshe Y. Vardi. Global model-checking of infinite-state systems. In Rajeev Alur and Doron Peled, editors, *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 387–400. Springer, 2004.

18 A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. 16th Int. Coll. on Automata, Languages and Programming, ICALP'89, Stresa, Italy, LNCS 372*, pages 652–671. Springer-Verlag, July 1989.

19 John H. Reif. The complexity of two-player games of incomplete information. *Journal of computer and system sciences*, 29(2):274–301, 1984.

20 Wolfgang Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 133–192. 1990.

21 Moshe Y. Vardi. Verification of concurrent programs: The automata-theoretic framework. *Annals of Pure and Applied Logic*, 51(1):79–98, 1991.

22 Moshe Y. Vardi. Reasoning about the past with two-way automata. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998.

23 Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, June 1998.

Emptiness Of Alternating Tree Automata Using Games With Imperfect Information*

Nathanaël Fijalkow[1,2], Sophie Pinchinat[3], and Olivier Serre[1]

1 **LIAFA, Université Paris Diderot – Paris 7 & CNRS, France**
2 **University of Warsaw, Poland**
3 **IRISA (INRIA & Université de Rennes 1), France**

Abstract

We consider the emptiness problem for alternating tree automata, with two acceptance semantics: classical (all branches are accepted) and qualitative (almost all branches are accepted). For the classical semantics, the usual technique to tackle this problem relies on a Simulation Theorem which constructs an equivalent non-deterministic automaton from the original alternating one, and then checks emptiness by a reduction to a two-player perfect information game. However, for the qualitative semantics, no simulation of alternation by means of non-determinism is known.

We give an alternative technique to decide the emptiness problem of alternating tree automata, that does not rely on a Simulation Theorem. Indeed, we directly reduce the emptiness problem to solving an *imperfect information* two-player parity game. Our new approach can successfully be applied to both semantics, and yields decidability results with optimal complexity; for the qualitative semantics, the key ingredient in the proof is a positionality result for stochastic games played over infinite graphs.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Alternating Automata, Emptiness checking, Two-player games, Imperfect Information Games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.299

1 Introduction

Tree automata [24, 14] are a powerful tool to handle sets of infinite trees which are widely needed in verification, since they provide a natural representation of branching-time system executions. It is well known that by equipping tree automata with the parity condition, one captures all ω-regular tree languages [16]. Additionally, tree automata may use *alternation* [8], which makes their complementation a simple task. In particular, combining alternation with the parity condition yields the automata-theoretic counterpart of the propositional μ-calculus, where the translation from one to the other can be done in linear time [1, 16]. Hence, the model-checking and the satisfiability/validity of logical formulas amount to respectively verifying membership and non-emptiness/universality on their corresponding tree automata.

The membership problem for alternating tree automata has a fairly simple algorithm: one compiles the input tree and the automaton into a polynomial size perfect information parity game and solves it. On the contrary, the usual roadmap to check emptiness of an alternating tree automaton is more involved. First one builds an equivalent non-deterministic automaton thanks to the Simulation Theorem [21], and then one checks emptiness of this latter automaton

* The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 259454 (GALE) and n° 239850 (SOSNA).

by solving an associated two-player perfect information game. This yields an exponential time algorithm, which is optimal as the emptiness problem is EXPTIME-complete.

The first contribution of this paper is an alternative technique to solve the emptiness problem of alternating tree automata, by directly reducing it to two-player games with *imperfect information*.

This builds on a long tradition initiated by Reif in [23], that advocates the use of games with imperfect information to solve algorithmic problems for automata; in his seminal paper, Reif introduced the notion of *blindfold games* and used them to check the universality of non-deterministic automata over finite words. This approach has been later extended in [12, 25] and combined with antichains representations, to check universality and inclusion of non-deterministic automata, as well as emptiness for alternating automata. This was backed with solid experimental results (see *e.g.* the tool Alaska [26]), where the emptiness of alternating Büchi word automata was considered, building on the Miyano-Hayashi construction [20]. To the best of our knowledge, antichains approaches have not yet been extended to alternating parity tree automata. However, solving the emptiness problem for alternating parity tree automata through games of imperfect information has been considered by Puchala in his PhD [22], where he provides a reduction of the emptiness problem for alternating parity automata to solving a *three-player* game with imperfect information, but no algorithm to solve the latter.

We first illustrate our technique in the classical case of alternating parity tree automata, reducing the emptiness problem to two-player parity games with imperfect information. This does not lead to a gain in complexity due to intrinsic hardness, but unravels the two key ingredients: the first one is the positional determinacy of parity games, to prove the correctness of the reduction, and the second is the determinisation property of ω-word automata, to solve the obtained two-player imperfect information game. We compare this with the classical approach for alternating parity tree automata: the Simulation Theorem [21] also combines the above two key ingredients.

Our technique is of interest for at least two reasons: (1) it pushes the algorithmic difficulty to the game solving part, for which antichains representations have recently been developed [4], hence could lead to efficient algorithms, and (2) a "Simulation Theorem"-free technique is required for classes of tree automata for which no (effective) Simulation Theorem exists.

The second contribution illustrates this latter situation. Indeed, we consider an alternation extension of the class of qualitative tree automata as introduced in [6]: rather than requiring all branches to be accepting (classical semantics), the qualitative semantics requires *almost all* branches to be accepting. We apply our technique to check emptiness of an alternating qualitative Büchi tree automaton. Furthermore, we observe that the emptiness problem becomes undecidable for the co-Büchi condition, implying that there is no simulation theorem for alternating qualitative tree co-Büchi automata. For our technique to go through, the key ingredient is a positionality result for stochastic Büchi games over *infinite* arenas. To the best of our knowledge, very few positionality results are known in the literature that combine both stochastic aspects and infiniteness of the game arena; notable exceptions are [5, 18].

The paper is organised as follows. Section 2 gives the key definitions of both perfect/imperfect information games and classical/qualitative alternating tree automata. Section 3 introduces our technique by revisiting the emptiness problem for alternating parity tree automata and compares with the usual approach. Our main contribution is developed in Section 4: we prove that the emptiness problem for alternating qualitative Büchi tree automata is EXPTIME-complete. This is divided into two subsections. The main technical contribution is given in Section 4.1, where we establish a positionality result for stochastic games played

on infinite finite out-degree chronological arenas. Finally, we use this positionality result in Section 4.2 to apply our technique to qualitative alternating Büchi tree automata.

2 Definitions

Let X be a (possibly infinite) alphabet. We denote by X^* (*resp.* X^ω) the set of finite (*resp.* infinite) words over X and we let ε be the empty word. Let Σ be a *finite* alphabet. An *infinite* Σ*-labelled binary tree* (or simply a *tree* when Σ is clear from the context) is a map $t : \{0,1\}^* \to \Sigma$. In this setting, we shall refer to an element $n \in \{0,1\}^*$ as a *node* and to ε as the *root*. For a node n, we call $t(n)$ the *label* of n in t. A (tree) *language* is a set of infinite Σ-labelled binary trees.

A *graph* is a pair $G = (V, E)$ where V is a set of *vertices* and $E \subseteq V \times V$ is a set of *edges*. For every vertex v we let $E(v) = \{w \mid (v,w) \in E\}$. A *dead-end* is a vertex v such that $E(v) = \emptyset$; in the rest of the paper, we only consider graphs that have no dead-end, hence this is implicit from now on. The *size* of a graph is defined to be $|V| + |E|$.

For a finite set S, a probability distribution on S is a function $\delta : S \to [0,1]$ such that $\sum_{s \in S} \delta(s) = 1$. We denote the set of probability distributions on S by $\mathcal{D}(S)$ and we write $\mathrm{Supp}(\delta) = \{s \in S \mid \delta(s) > 0\}$ for the support set of δ.

2.1 Perfect Information Stochastic Games

A (turn-based) *stochastic arena* is a tuple $\mathcal{G} = (G, V_\mathrm{E}, V_\mathrm{A}, V_R, \delta, v_0)$ where $G = (V, E)$ is a graph, $V = V_\mathrm{E} \uplus V_\mathrm{A} \uplus V_R$ is a partition of the vertices among two players, Éloïse and Abélard, and an extra player Random, $\delta : V_R \to \mathcal{D}(V)$ is a map such that $\mathrm{Supp}(\delta(v)) = E(v)$ for all $v \in V_R$, and $v_0 \in V$ is an *initial* vertex. In a vertex $v \in V_\mathrm{E}$ (*resp.* $v \in V_\mathrm{A}$) Éloïse (*resp.* Abélard) chooses a successor vertex from $E(v)$ and in a random vertex $v \in V_R$, a successor vertex is chosen according to the probability distribution $\delta(v)$.

A (pure) *strategy*[1] for Éloïse is a function $\varphi_\mathrm{E} : V^* \cdot V_\mathrm{E} \to V$ such that for every $\lambda \cdot v \in V^* \cdot V_\mathrm{E}$ one has $\varphi_\mathrm{E}(\lambda \cdot v) \in E(v)$. Strategies of Abélard are defined likewise, and usually denoted φ_A.

Fix a strategy φ_E for Éloïse and a strategy φ_A for Abélard. This induces a random walk on G. Indeed, define $\mathrm{Plays}^{\varphi_\mathrm{E},\varphi_\mathrm{A}}$ to be the set of all possible *plays* when the game starts on v_0 and when Éloïse and Abélard chooses their moves accordingly to φ_E and φ_A. Formally, an infinite play $v_0 v_1 v_2 \cdots \in V^\omega$ belongs to $\mathrm{Plays}^{\varphi_\mathrm{E},\varphi_\mathrm{A}}$ if for every $i \geq 0$ one has $v_i \in V_\mathrm{E} \Rightarrow v_{i+1} = \varphi_\mathrm{E}(v_0 \cdots v_i)$, $v_i \in V_\mathrm{A} \Rightarrow v_{i+1} = \varphi_\mathrm{A}(v_0 \cdots v_i)$ and $v_i \in V_R \Rightarrow v_{i+1} \in E(v_i)$.

Define a partial play as a prefix of a play in $\mathrm{Plays}^{\varphi_\mathrm{E},\varphi_\mathrm{A}}$: with any partial play λ, the *cone* for λ is the set $\mathit{cone}(\lambda) = \lambda \cdot V^\omega \cap \mathrm{Plays}^{\varphi_\mathrm{E},\varphi_\mathrm{A}}$ of all infinite plays with prefix λ. Denote by Cones the set of all possible cones and let $\mathcal{F}$ be the Borel σ-field generated by Cones considered as a set of basic open sets (*i.e.* $\mathcal{F}$ is the smallest set containing Cones and closed under complementation, countable union): then $(\mathrm{Plays}^{\varphi_\mathrm{E},\varphi_\mathrm{A}}, \mathcal{F})$ is a σ-algebra.

A pair of strategies $(\varphi_\mathrm{E}, \varphi_\mathrm{A})$ induces a probability space over $(\mathrm{Plays}^{\varphi_\mathrm{E},\varphi_\mathrm{A}}, \mathcal{F})$. Indeed one can define a measure $\mu^{\varphi_\mathrm{E},\varphi_\mathrm{A}} : \mathrm{Cones} \to [0,1]$ on cones (this task is easy as a cone is uniquely defined by a finite partial play) and then uniquely extends it to a probability measure on $\mathcal{F}$ using the Carathéodory Unique Extension Theorem. For this, one defines $\mu^{\varphi_\mathrm{E},\varphi_\mathrm{A}}$ inductively on cones:

[1] We do not consider randomised strategies as pure strategies are the right model here.

- $\mu^{\varphi_E,\varphi_A}(cone(v)) = 1$ if $v = v_0$, and $\mu^{\varphi_E,\varphi_A}(v) = 0$ otherwise.
- For every partial play λ ending in $V_E \cup V_A$, let $\mu^{\varphi_E,\varphi_A}(cone(\lambda \cdot v)) = \mu^{\varphi_E,\varphi_A}(cone(\lambda))$ if $v = \varphi_E(\lambda)$ or $v = \varphi_A(\lambda)$, and $\mu^{\varphi_E,\varphi_A}(cone(\lambda \cdot v)) = 0$ otherwise;
- For every partial play λ ending in $v \in V_R$, let $\mu^{\varphi_E,\varphi_A}(cone(\lambda \cdot v')) = \mu^{\varphi_E,\varphi_A}(cone(\lambda)) \cdot \delta(v)(v')$.

Denote by $\Pr^{\varphi_E,\varphi_A}$ the unique extension of $\mu^{\varphi_E,\varphi_A}$ to a probability measure on $\mathcal{F}$. Then $(\text{Plays}^{\varphi_E,\varphi_A}, \mathcal{F}, \Pr^{\varphi_E,\varphi_A})$ is a probability space.

A *winning condition* is a subset[2] $\Omega \subseteq V^\omega$ and a (two-player perfect information) *stochastic game* is a pair $\mathbb{G} = (\mathcal{G}, \Omega)$. A game is *deterministic* whenever $V_R = \emptyset$ (and in this case we omit both V_R and δ in the notations).

A strategy φ_E for Éloïse is *surely winning* if $\text{Plays}^{\varphi_E,\varphi_A} \subseteq \Omega$ for every strategy φ_A of Abélard; it is *almost-surely winning* if $\Pr^{\varphi_E,\varphi_A}(\Omega) = 1$ for every strategy φ_A of Abélard. Similar notions for Abélard are defined dually.

A *reachability game* is one with a winning condition of the form V^*FV^ω, *i.e.* winning plays are those that eventually visit a vertex in F (we refer to vertices in F as final ones). We consider the *parity* winning condition: a *colouring* function ρ is a mapping $\rho : V \to \text{Col} \subset \mathbb{N}$ where Col is a *finite* set of colours; the *parity condition* associated with ρ is the set $\Omega_\rho = \{v_0 v_1 \cdots \in V^\omega \mid \liminf(\rho(v_i))_{i \geq 0} \text{ is even}\}$. A *parity game* is a game equipped with a parity winning condition and we shall denote it $\mathbb{G} = (\mathcal{G}, \rho)$ (*i.e.* writing ρ instead of Ω_ρ). *Büchi games* are those where $\text{Col} = \{0, 1\}$ and we refer to vertices v such that $\rho(v) = 0$ as *Büchi vertices*. The dual is *co-Büchi*, for $\text{Col} = \{1, 2\}$.

A positional strategy φ is one that does not require memory, *i.e.* such that for any two partial plays of the form $\lambda \cdot v$ and $\lambda' \cdot v$, one has $\varphi(\lambda \cdot v) = \varphi(\lambda' \cdot v)$, equivalently φ only depends on the current vertex. It is well-known that positional strategies suffice to surely win in deterministic parity games (see *e.g.* [27]).

▶ **Theorem 1** (Positional determinacy). *Let $\mathbb{G}$ be a deterministic parity game. Then either Éloïse or Abélard has a positional surely winning strategy.*

Working with deterministic parity games we only consider positional strategies and see them as maps from V_E (or V_A depending on the player) to V.

For stochastic games the following result is well-known (see *e.g.* [15] for a slightly more general result).

▶ **Theorem 2.** *Let $\mathbb{G}$ be a stochastic parity game played on a* finite *arena. If Éloïse almost-surely wins then she can do so using a positional strategy.*

To the best of our knowledge, no extension of this result is known when dropping the assumption that the arena is finite. We give such an extension (for Büchi games on so-called chronological arenas of finite out-degree) in Theorem 7.

2.2 Alternating Parity Tree Automata

An *alternating parity tree automaton* is a tuple $\mathcal{A} = (Q_\exists, Q_\forall, \Sigma, \Delta, q_{\text{in}}, \rho)$, where $Q_\exists$ is a set of existential states and $Q_\forall$ is a set of universal states such that $Q_\exists$ and $Q_\forall$ are disjoint (we let $Q = Q_\exists \uplus Q_\forall$), $q_{\text{in}} \in Q$ is an initial state, Σ is a labelling finite alphabet, $\Delta \subseteq Q \times \Sigma \times Q \times Q$ is a (finite) transition relation and $\rho : Q \to \mathbb{N}$ is a colouring function. We additionally assume

[2] Formally one needs to require that Ω is measurable for all $\Pr^{\varphi_E,\varphi_A}$, which will be trivially true for all cases considered here as Ω will always be Borel.

without loss of generality that for all $(q, \sigma) \in Q \times \Sigma$ there is at least one $(q, \sigma, q_0, q_1) \in \Delta$. A *non-deterministic* parity tree automaton is an alternating automaton in which all states are existential (hence, we omit $Q_\forall$ in this case).

In the following, we use tree automata as acceptors for tree languages, and acceptance is defined by means of a perfect information parity game. We will define two semantics for acceptance: *classical* and *qualitative*.

Fix an alternating parity tree automaton $\mathcal{A} = (Q_\exists, Q_\forall, \Sigma, \Delta, q_{\text{in}}, \rho)$ and a Σ-labelled tree t. From $\mathcal{A}$ and t, we define two games: $\mathbb{G}_{\mathcal{A},t}$ and $\mathbb{G}^{=1}_{\mathcal{A},t}$.

Intuitively, a play in those two games consists in moving a pebble along a branch of t in a top-down manner: the pebble is attached to a state and in a node n with state q Éloïse (if $q \in Q_\exists$) or Abélard (if $q \in Q_\forall$) picks a transition $(q, t(n), q_0, q_1) \in \Delta$, and then *Abélard* (in $\mathbb{G}_{\mathcal{A},t}$) or *Random* (in $\mathbb{G}^{=1}_{\mathcal{A},t}$) chooses to move down the pebble either to $n \cdot 0$ (and update the state to q_0) or to $n \cdot 1$ (and update the state to q_1).

Formally, one let $G = (V_\exists \uplus V_\forall \uplus V_\Delta, E)$ with $V_\exists = \{0,1\}^* \times Q_\exists$, $V_\forall = \{0,1\}^* \times Q_\forall$ and $V_\Delta = \{(n, q, q_0, q_1) \mid n \in \{0,1\}^* \text{ and } (q, t(n), q_0, q_1) \in \Delta\}$ and

$$\begin{aligned} E \quad = \quad & \{((n,q),(n,q,q_0,q_1)) \mid (n,q,q_0,q_1) \in V_\Delta)\} \quad \cup \\ & \{((n,q,q_0,q_1),(n \cdot x, q_x)) \mid x \in \{0,1\} \text{ and } (n,q,q_0,q_1) \in V_\Delta)\} \end{aligned}$$

Then let $\mathcal{G}_{\mathcal{A},t} = (G, V_{\text{E}}, V_{\text{A}}, (\varepsilon, q_{\text{in}}))$ be the deterministic arena defined by letting $V_{\text{E}} = V_\exists$ and $V_{\text{A}} = V_\forall \cup V_\Delta$ and let $\mathcal{G}^{=1}_{\mathcal{A},t} = (G, V_{\text{E}}, V_{\text{A}}, V_R, \delta, (\varepsilon, q_{\text{in}}))$ be the stochastic arena defined by letting $V_{\text{E}} = V_\exists$, $V_{\text{A}} = V_\forall$, $V_R = V_\Delta$ and $\delta((n,q,q_0,q_1))$ be the distribution $(n \cdot 0, q_0) \mapsto \frac{1}{2}$ and $(n \cdot 1, q_1) \mapsto \frac{1}{2}$.

Extend ρ on V by letting $\rho((n,q)) = \rho((n,q,q_0,q_1)) = \rho(q)$. Finally one let $\mathbb{G}_{\mathcal{A},t} = (\mathcal{G}_{\mathcal{A},t}, \rho)$ and $\mathbb{G}^{=1}_{\mathcal{A},t} = (\mathcal{G}^{=1}_{\mathcal{A},t}, \rho)$.

A tree t is *accepted* (*resp. qualitatively accepted*) by $\mathcal{A}$ if Éloïse has a surely (*resp.* almost-surely) winning strategy in the game $\mathbb{G}_{\mathcal{A},t}$ (*resp.* $\mathbb{G}^{=1}_{\mathcal{A},t}$). Finally, we define the set $L(\mathcal{A})$ as the set of trees accepted by $\mathcal{A}$ and the set $L^{=1}(\mathcal{A})$ as the set of trees qualitatively accepted by $\mathcal{A}$.

The languages of the form $L(\mathcal{A})$, *regular tree languages*, have many remarkable properties and characterisations (see *e.g.* [16]). The languages of the form $L^{=1}(\mathcal{A})$ when $\mathcal{A}$ is non-deterministic, *qualitative tree languages*, have been introduced in [6] and also enjoy many good properties: for instance they are closed under union and intersection, and their emptiness can be tested in *polynomial* time.

▶ **Example 3.** A typical language $L^{=1}(\mathcal{A})$ with $\mathcal{A}$ being non-deterministic is the set of trees that have almost all their branches containing infinitely many a's. An example of a language $L^{=1}(\mathcal{A}_\mathcal{B})$ with $\mathcal{A}_\mathcal{B}$ being alternating is the set of trees such that all subtrees belongs to some $L^{=1}(\mathcal{B})$ where $\mathcal{B}$ is non-deterministic.

▶ Remark. There are several definitions of alternating tree automata, and another popular one is by not distinguishing between existential and universal states but replacing the transition relation by a map $\delta : Q \times \Sigma \to \mathcal{B}^+(Q \times \{0,1\})$ where $\mathcal{B}^+(X)$ denotes the positive Boolean formulas over X (see *e.g.* [19]). Our model is easily seen to be equi-expressive with that one.

▶ Remark. Any *positional* strategy for Éloïse in $\mathbb{G}_{\mathcal{A},t}$ or $\mathbb{G}^{=1}_{\mathcal{A},t}$ can be described as a function $\varphi : \{0,1\}^* \times Q_\exists \to Q \times Q$ that satisfies the following property: $\forall n \in \{0,1\}^*$, if $\varphi(n,q) = (q_0, q_1)$ then $(q, t(n), q_0, q_1) \in \Delta$. Equivalently, in a curryfied form, φ is a map $\{0,1\}^* \to (Q_\exists \to Q \times Q)$. Hence, if one let $\mathcal{T}$ be the set of functions from $Q_\exists$ into $Q \times Q$, Éloïse's positional strategies are in bijection with $\mathcal{T}$-labelled binary trees.

2.3 Imperfect Information Stochastic Parity Games

In the following we introduce a quite restrictive class of games with imperfect information which is essentially a stochastic version of the model in [10].

An *arena of imperfect information* is a tuple $\mathcal{G} = (S, s_0, A, T, \sim)$ where S is a finite set of states, $s_0 \in S$ is an initial state, A is the finite alphabet of Éloïse's actions, $T \subseteq S \times A \times \mathcal{D}(S)$ is a (finite) stochastic transition relation and $\sim$ is an equivalence relation over S. We additionally require that for all $(s, a) \in S \times A$ there is at least one $\delta \in \mathcal{D}(S)$ such that $(s, a, \delta) \in T$. If for all probability distributions in T the support is a singleton, we say that $\mathcal{G}$ is an imperfect information *deterministic* arena and we see T as a subset of $S \times A \times S$. An *imperfect information stochastic parity game* is a pair $\mathbb{G} = (\mathcal{G}, \rho)$ where $\mathcal{G}$ is an arena of imperfect information with set of states S and $\rho : S \to \mathbb{N}$ is a colouring function defining a parity condition $\Omega_\rho \subseteq S^\omega$. The game is *deterministic* if its arena is deterministic. A *play* starts from the initial state s_0 and proceeds as follows: Éloïse plays an action $a_0 \in A$, then Abélard resolves the non-determinism by choosing a distribution δ_0 such that $(s_0, a_0, \delta_0) \in T$ and finally a new state is randomly chosen according to δ_0. Then Éloïse plays a new action, Abélard resolves the non-determinism and a new state is randomly chosen and so on forever. Hence a play is an infinite word $s_0 a_0 \delta_0 s_1 a_1 \delta_1 s_2 \cdots \in (S \times A \times \mathcal{D}(S))^\omega$ and is won by Éloïse if $s_0 s_1 s_2 \cdots \in \Omega_\rho$. A *partial play* is a prefix of a play.

Imperfect information is modeled thanks to the equivalence relation $\sim$ with the meaning that Éloïse cannot distinguish two states that are $\sim$-equivalent which is important when defining strategies for Éloïse. Intuitively, she should not play differently in two indistinguishable plays, where the indistinguishability of Éloïse is based on *perfect recall* [13], that is: Éloïse cannot distinguish two plays $s_0 a_0 \delta_0 s_1 a_1 \delta_1 \cdots s_\ell$ and $s'_0 a'_0 \delta'_0 s'_1 a'_1 \delta'_1 \cdots s'_\ell$ with $s_i \sim s'_i$ for all $i \leq \ell$ and $a_i = a'_i$ for all $i < \ell$. Hence, a strategy for Éloïse is a function $\varphi : (S_{/\sim} \times A)^* \cdot (S_{/\sim}) \to A$ assigning an action to every set of indistinguishable plays (here $S_{/\sim}$ denotes the set of equivalence classes of $\sim$ in S, and for every $s \in S$, we shall write $[s]_\sim$ for its $\sim$-equivalence class). Éloïse *respects a strategy* φ during a play $\lambda = s_0 a_0 \delta_0 s_1 a_1 \delta_1 \cdots$ if $a_{i+1} = \varphi([s_0]_\sim a_0 [s_1]_\sim \cdots [s_i]_\sim)$, for all $i \geq 0$. A strategy φ for Éloïse is *surely winning* if Éloïse wins all plays consistent with φ and it is *almost-surely winning* if Éloïse wins almost all plays[3] consistent with φ.

▶ Remark. Our model of imperfect information games belongs is quite restrictive compared to general models developed in [17, 3, 9, 7], as here Abélard is perfectly informed. However, our model turns out to be expressive enough for our purpose.

▶ Remark. It is important to note that Éloïse may not observe the colour of the current state in general, as we do not require that $s \sim s' \Rightarrow \rho(s) = \rho(s')$. In particular, this has to be taken into account when eventually solving the game.

3 Revisiting the Emptiness using Imperfect Information Game

In this section, we introduce our technique by revisiting emptiness checking of regular tree languages when described by an alternating parity tree automaton. The standard technique [21] first removes alternation and then reduces emptiness to decide the winner in a finite perfect information game; our technique goes directly to decide the winner in a game but

[3] To formally define what it means to win almost all plays one needs to define Abélard's strategies and explain how a pair of strategies for Éloïse and Abélard induces a probabilistic space over plays consistent with those strategies. This is essentially identical to what we did in Section 2.1 for perfect information games.

as a drawback imperfect information is needed. Our construction is reminiscent of the notion of *blindfold games* introduced by Reif in [23], and later thoroughly extended [12, 25, 26].

We first give our construction and then compare it to the classical one, arguing that rather than being novel our technique is indeed a change of perspective. Later in Section 4 we use this new perspective to design an emptiness test for alternating qualitative tree automata for which the classical perspective was unsuccessful.

Fix an alternating parity tree automaton $\mathcal{A} = (Q_\exists, Q_\forall, \Sigma, \Delta, q_{\text{in}}, \rho)$. Our goal is to check whether $L(\mathcal{A}) = \emptyset$ and for this we design an imperfect information *deterministic* parity game. We first present the game and show how it permits to decide emptiness; then we compare our construction with the standard approach.

3.1 An Imperfect Information Emptiness Game

We define an imperfect information deterministic parity game $\mathbb{G}_\mathcal{A}$ that intuitively works as follows. Éloïse describes both a tree t and a positional strategy φ_t for her in the game $\mathbb{G}_{\mathcal{A},t}$; the strategy φ_t is described as a $\mathcal{T}$-labeled tree (where $\mathcal{T}$ is the set of functions from $Q_\exists$ into $Q \times Q$, see Remark 2.2). As the plays are of ω-length, she actually does not fully describe t and φ_t but only a branch: this branch is chosen by Abélard, who also takes care of computing the sequence of states along it (either by updating an existential state accordingly to φ_t or, when the state is universal, by choosing an arbitrary valid transition of the automaton). In this game, Éloïse observes the directions, but not the actual control state of the automaton (indeed, otherwise she could easily "cheat").

Formally, we let $\mathcal{G}_\mathcal{A} = (S, s_{\text{in}}, A, T, \sim)$ where $S = (Q \times \{0,1\}) \cup \{(q_{\text{in}}, \varepsilon)\}$ and $s_{\text{in}} = (q_{\text{in}}, \varepsilon)$; $A \subseteq \Sigma \times \mathcal{T}$ is the set of pairs (a, τ) such that for all $q \in Q_\exists$ we have that $(q, a, q_0, q_1) \in \Delta$ where $\tau(q) = (q_0, q_1)$, $(q, i) \sim (q', i)$ for all $q, q' \in Q$ and $i \in \{0,1\}$, and

$$T = \{((q,i),(a,\tau),(q_0,0)),((q,i),(a,\tau),(q_1,1)) \mid q \in Q_\exists \text{ and } \tau(q) = (q_0,q_1)\}$$
$$\cup\ \{((q,i),(a,\tau),(q_0,0)),((q,i),(a,\tau),(q_1,1)) \mid q \in Q_\forall \text{ and } (q,a,q_0,q_1) \in \Delta\}$$

Finally we let $\mathbb{G}_\mathcal{A} = (\mathcal{G}_\mathcal{A}, \rho_\mathcal{A})$ be the imperfect information *deterministic* parity game obtained by letting $\rho_\mathcal{A}(q,i) = \rho(q)$ for any $(q,i) \in S$.

▶ **Lemma 4.** *Éloïse has a surely winning strategy in $\mathbb{G}_\mathcal{A}$ iff $L(\mathcal{A}) \neq \emptyset$.*

▶ Remark. From the proof of Lemma 4, one can also conclude that if $L(\mathcal{A}) \neq \emptyset$ then $L(\mathcal{A})$ contains a regular tree (*i.e.* the unfolding of a finite graph). Indeed, this is a direct consequence of the fact that if Éloïse has a surely winning strategy in $\mathbb{G}_\mathcal{A}$, then she has one that uses finite memory [10].

Lemma 4 provides a reduction of the emptiness problem to deciding the existence of a surely winning strategy in an imperfect information deterministic game. We prove a converse result.

▶ **Lemma 5.** *For any imperfect information deterministic parity game $\mathbb{G}$ one can construct an alternating parity tree automaton $\mathcal{A}_\mathbb{G}$ such that Éloïse surely wins in $\mathbb{G}$ iff $L(\mathcal{A}_\mathbb{G}) \neq \emptyset$. Moreover in $\mathcal{A}_\mathbb{G}$ all states are universal.*

3.2 Comparison with the Standard Approach

The usual roadmap to check emptiness of an alternating tree automaton is as follows. First one builds an equivalent non-deterministic automaton thanks to Simulation Theorem (see below) and then one checks emptiness of this latter automaton by solving an associated

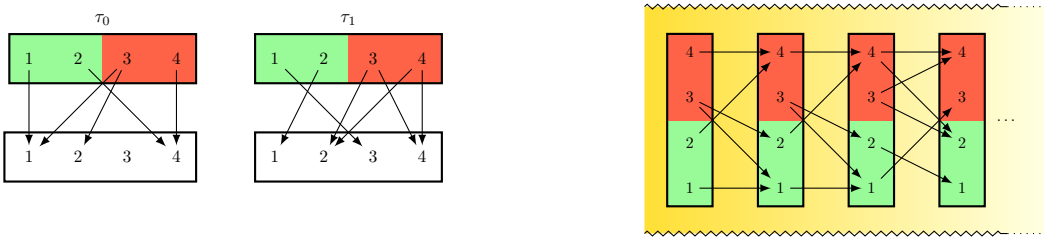


Figure 1 **(a)** Semi-tiles τ_0 and τ_1 such that (τ_0, τ_1) represents the tile $\{(1,1,3),(2,4,1),(3,1,2),(3,2,4),(4,4,2),(4,4,4)\}$ with $Q_\exists = \{1,2\}$ and $Q_\forall = \{3,4\}$. **(b)** Representation of the sequence of semi-tiles $\tau_0\tau_0\tau_1\cdots$.

perfect information game. It is a well-known result that alternating and non-deterministic automata are equi-expressive [21].

▶ **Theorem 6** (Simulation Theorem). *Let $\mathcal{A}$ be an alternating parity tree automaton with n states and using k colours. Then one can effectively construct a non-deterministic parity tree automaton $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$. The automaton $\mathcal{B}$ has $2^{\mathcal{O}(nk\log(nk))}$ states and it uses $\mathcal{O}(nk)$ colours.*

Proof. We do not give a complete proof of this classical result [21] but we rather exhibit the crucial arguments here to later revisit the emptiness problem for alternating parity tree automata.

Fix an alternating parity tree automaton $\mathcal{A} = (Q_\exists, Q_\forall, \Sigma, \Delta, q_{\text{in}}, \rho)$. For any letter $\sigma \in \Sigma$, call a *σ-tile* any subset $\tau \subseteq Q \times Q \times Q$ such that

- for all $q, q_0, q_1 \in Q$, if $(q, q_0, q_1) \in \tau$ then $(q, \sigma, q_0, q_1) \in \Delta$;
- for all $q \in Q_\exists$ there exists a *unique* $(q_0, q_1) \in Q^2$ such that $(q, q_0, q_1) \in \tau$;
- for all $q \in Q_\forall$ and for all $(q, \sigma, q_0, q_1) \in \Delta$, $(q, q_0, q_1) \in \tau$.

Hence, one should think of a σ-tile as a description of the local value of a *positional* strategy for Éloïse in a node labeled by σ from a tree t in the game $\mathbb{G}_{\mathcal{A},t}$ (the case of $q \in Q_\forall$ is here to leave all options of Abélard). In the following it is convenient to think of a tile τ as a pair (τ_0, τ_1) of *semi-tiles* where $\tau_0, \tau_1 \subseteq Q \times Q$ and $(q, q_0) \in \tau_0$ (*resp* $(q, q_1) \in \tau_1$) if and only if there exists $p \in Q$ such that $(q, q_0, p) \in \tau$ (*resp.* $(q, p, q_1) \in \tau$). See Figure 1a for an example.

Now an equivalent non-deterministic automaton $\mathcal{A}'$ is obtained by choosing as control states the set S of all possible semi-tiles augmented with an extra dummy initial state s_{in}. The transition relation Δ' consists of all those elements of the form $(s, \sigma, \tau_0, \tau_1)$ where s is any state and (τ_0, τ_1) is a σ-tile. Acceptance for $\mathcal{A}'$ is then defined by means of a game $\mathbb{G}_{\mathcal{A}',t}$ as previously except that the winning condition is more involved than a parity condition. A play is an element $\lambda = v_0 v_1 \cdots \in ((\{0,1\}^* \times S) \cdot (\{0,1\}^* \times S \times S \times S))^\omega$ to which we can associate a sequence of semi-tiles $\pi(\lambda) = s_1 s_2 \cdots$ where $v_{2i} = (n_i, s_i)$ for all integer $i \geq 1$ (we ignore the dummy initial state). The sequence $\pi(\lambda)$ can be seen as a set of infinite paths in an infinite ribbon obtained by gluing together the semi-tiles $s_1, s_2, \ldots$ (see Figure 1b). An infinite sequence $q_1 q_2 \cdots$ is an infinite path in $\pi(\lambda)$ if and only if for all $i \geq 1$ one has $(q_i, q_{i+1}) \in s_i$; it is *good* if $\liminf(\rho(q_i))_{i\geq 1}$ is even; and $\pi(\lambda)$ is *good* if all plays in it are good. Finally we define those winning plays for Éloïse as those plays λ such that $\pi(\lambda)$ is good.

Then one can easily remark that the set of all λ such that $\pi(\lambda)$ is good is an ω-regular language over S, hence is accepted by a deterministic parity ω-word automaton $\mathcal{C}$. Considering a "synchronised" product of $\mathcal{C}$ together with $\mathcal{A}'$ leads $\mathcal{B}$. The desired complexity is achieved by carefully constructing $\mathcal{C}$. ◀

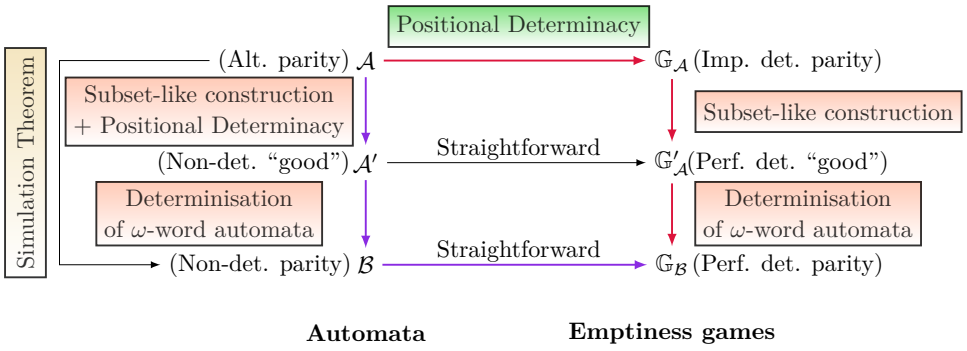


Figure 2 Roadmaps to emptiness checking: the classical one (purple) *vs* ours (in red).

Consider now a *non-deterministic* parity tree automaton $\mathcal{K} = (Q, \Sigma, \Delta, q_{\text{in}}, \rho)$. A perfect information emptiness game for $\mathcal{K}$ is built as follows. We let $G_{\mathcal{K}} = (V_{\text{E}} \uplus V_{\text{A}}, E)$ where $V_{\text{E}} = Q$, $V_{\text{A}} = \Delta$ and $E = \{(q, (q, \sigma, q_0, q_1)), ((q, \sigma, q_0, q_1), q_0), ((q, \sigma, q_0, q_1), q_1) \mid (q, \sigma, q_0, q_1) \in \Delta\}$. We define $\mathbb{G}_{\mathcal{K}} = (\mathcal{G}_{\mathcal{K}}, \rho)$ with $\mathcal{G}_{\mathcal{K}} = (G_{\mathcal{K}}, V_{\text{E}}, V_{\text{A}}, q_{\text{in}})$ and where we extend ρ by letting $\rho((q, \sigma, q_0, q_1)) = \rho(q)$. Then one easily has that $L(\mathcal{K}) \neq \emptyset$ if and only if Éloïse surely wins in $\mathbb{G}_{\mathcal{K}}$. Indeed, strategies for Éloïse in $\mathbb{G}_{\mathcal{K}}$ are in bijection with pairs made of a tree t and a strategy for Éloïse in $\mathbb{G}_{\mathcal{K},t}$, and this bijection preserves the fact that a strategy is surely winning.

Now think of adapting this construction to the automaton $\mathcal{A}'$ from the proof of the Simulation Theorem and recall that acceptance for $\mathcal{A}'$ was defined thanks to a game as the classical one except that the winning condition was more involved. Then, the same construction provides a game $\mathbb{G}_{\mathcal{A}'}$ where Éloïse's vertices are semi-tiles and Abélard's vertices are tuple made of a semi-tile, a letter in Σ and a tile, and whose winning condition for a play $\lambda = v_0 v_1 \cdots \in ((\Sigma \times S) \cdot (\Sigma \times S \times S \times S))^\omega$ is that $\pi(\lambda) = s_1 s_2 \cdots$ is good (in the previous sense) where $v_{2i} = (\sigma_i, s_i)$ for all integer $i \geq 1$.

Now think back to our reduction from $\mathcal{A}$ to $\mathbb{G}_{\mathcal{A}}$: it makes crucial use of positional determinacy while determinisation of automata on infinite word is implicitly needed when deciding whether Éloïse surely wins in $\mathbb{G}_{\mathcal{A}}$. Indeed, one first applies the subset construction of [10], getting an intermediate perfect information game isomorphic to $\mathbb{G}_{\mathcal{A}'}$ and then, since Éloïse does not observe the colour, one has to embed in the previous subset/tile construction a deterministic parity automaton over infinite words that checks that all plays consistent with the observations fit the parity condition. As this latter automaton is essentially the automaton $\mathcal{C}$ one gets a perfect information parity game isomorphic to $\mathbb{G}_{\mathcal{B}}$. Figure 2 summarises the previous discussion.

4 Checking Emptiness Using an Imperfect Information Game: The Case of $L^{=1}(\mathcal{A})$ for Büchi Condition

We are now coming to the central contributions of this paper. We design an emptiness test for alternating *qualitative* Büchi tree automata adapting the approach developed in Section 3. Recall that it relies on two key arguments: a positionality result and a decidability result for games. Hence, we start by proving a positionality result (Theorem 7), and then explain how to obtain a (decidable) emptiness game (Theorem 10).

4.1 A Positionality Result for Chronological Games

In this subsection, we prove a positionality result for a subclass of infinite arenas, satisfying the following two conditions:

- (finite out-degree) the underlying graph has finite degree, *i.e* for every $v \in V$ there are finitely many outgoing edges from v, and
- (chronological) there exists a function $\mathrm{rank} : V \to \mathbb{N}$ such that $\mathrm{rank}^{-1}(0) = \{v_0\}$ and for $(v, v') \in E$, $\mathrm{rank}(v') = \mathrm{rank}(v) + 1$.

Note that both assumptions hold for $\mathbb{G}_{\mathcal{A}}^{=1}$. Also, observe that for any k, the set $\mathrm{rank}^{-1}(k)$ is finite, and the set V of vertices is countable.

▶ **Theorem 7.** *Let $\mathbb{G}$ be a Büchi game whose arena has finite out-degree and is chronological. Then if Éloïse wins almost-surely, then she has a positional winning strategy.*

We will use the following result for finite arenas. We state it here in a rather weak form (with the chronological assumption), as it can be easily proved by a backward induction, whereas the proof for general finite arenas is more involved. We refer to [11] for the original proof, and to [18] for a nice survey.

▶ **Lemma 8.** *Let $\mathbb{G}$ be a stochastic reachability game played on a* chronological *and* finite *arena. Let W be the set of vertices from which Éloïse has a strategy ensuring to win with probability at least $\frac{1}{2}$. Then there exists a positional strategy which ensures to win with probability at least $\frac{1}{2}$ from every vertex in W.*

Note that the important point here is that the constructed strategy is *uniform*, *i.e.* the same strategy is winning from every vertex.

We first sketch the proof. The main idea is to note that if Éloïse can ensure to reach F with probability 1 from some initial vertex, then there exists a bound k such that she can ensure to reach F with probability at least $\frac{1}{2}$ within k steps against *any* strategy of Abélard. This allows to "slice" the arena into infinitely many disjoint finite arenas: in each slice Éloïse plays to reach F with probability at least half. Since each slice forms a finite subarena, optimal positional strategies exist by Lemma 8. The resulting strategy consists in playing in turns the above positional strategies; since each slice gives a probability to reach F of at least half before proceeding to the next, the probability to reach F infinitely often is 1.

Proof. We assume that Éloïse almost-surely wins $\mathbb{G}$. Without loss of generality, we can assume that she wins almost-surely from everywhere, by restricting the arena to vertices reachable by a fixed almost-surely winning strategy.

In the next statement and later on, by a strategy in $\mathbb{G}$ from a vertex v we mean a strategy in the game obtained from $\mathbb{G}$ by changing the initial vertex of the arena $\mathcal{G}$ to be v.

▶ **Lemma 9.** *Let φ_{E} be an almost-surely winning strategy for Éloïse in $\mathbb{G}$ from v. There exists an integer k such that for all strategies φ_{A} of Abélard, we have $\Pr^{\varphi_{\mathrm{E}},\varphi_{\mathrm{A}}}(V^{\leq k}FV^{\omega}) \geq \frac{1}{2}$.*

Proof. Toward a contradiction, assume that such a k does not exist. Hence, for each k there exists a strategy $\varphi_{\mathrm{A},k}$ such that $\Pr^{\varphi_{\mathrm{E}},\varphi_{\mathrm{A},k}}(V^{\leq k}FV^{\omega}) < \frac{1}{2}$. Moreover we can assume that the $\varphi_{\mathrm{A},k}$ are positional strategies: indeed, one can pick for $\varphi_{\mathrm{A},k}$ an optimal strategy for Abélard in the finite reachability game obtained by restricting $\mathbb{G}$ to the vertices of rank at most k. As this game is finite, by Lemma 8, an optimal strategy can always be chosen to be positional.

From the sequence $(\varphi_{\mathrm{A},k})_{k\geq 0}$ we can extract a strategy $\varphi_{\mathrm{A},\infty}$ that is consistent, for any $k \geq 0$, with infinitely many $\varphi_{\mathrm{A},h}$ on its k first moves. For this, fix an enumeration $v_1, v_2, \cdots$ of the vertices in V. We will define by induction on i the following objects:

- I_i an infinite set of vertices such that $I_0 \supseteq I_1 \supseteq I_2 \supseteq I_3 \supseteq \cdots$, and
- $\varphi_{\mathrm{A},\infty}$ such that at step i, $\varphi_{\mathrm{A},\infty}$ is defined on $v_1, \ldots, v_i$ and is consistent on these vertices with all those strategies $\varphi_{\mathrm{A},h}$ with $h \in I_i$.

Let $I_0 = \mathbb{N}$ be the set of all positive integers. At step i let us consider the values of $\varphi_h(v_i)$ for all $h \in I_{i-1}$: as $\mathcal{G}$ has finite out-degree, there is one v'_i such that for infinitely many $h \in I_{i-1}$, $\varphi_{\mathrm{A},h}(v_i) = v'_i$. We define $\varphi_{\mathrm{A},\infty}(v_i) = v'_i$ and we let $I_i = \{h \in I_{i-1} \mid \varphi_{\mathrm{A},h}(v_i) = v'_i\}$.

Now for $k \geq 0$ let i be such that $\{v_j \mid j \leq i\}$ contains all vertices of rank at most k: then as required $\varphi_{\mathrm{A},\infty}$ is consistent with infinitely many $\varphi_{\mathrm{A},h}$ — all the $\varphi_{\mathrm{A},h}$ with $h \in I_i$ — on its k first moves. In particular, for all k, there is some $h \geq k$ such that

$$\Pr^{\varphi_{\mathrm{E}},\varphi_{\mathrm{A},\infty}}(V^{\leq k}FV^\omega) = \Pr^{\varphi_{\mathrm{E}},\varphi_{\mathrm{A},h}}(V^{\leq k}FV^\omega) \leq \Pr^{\varphi_{\mathrm{E}},\varphi_{\mathrm{A},h}}(V^{\leq h}FV^\omega) < \frac{1}{2}$$

As $V^*FV^\omega = \bigcup_{k\geq 0} V^{\leq k}FV^\omega$ and as the sequence $(V^{\leq k}FV^\omega)_{k\geq 0}$ is increasing for set inclusion, one concludes that $\Pr^{\varphi_{\mathrm{E}},\varphi_{\mathrm{A},\infty}}(V^*FV^\omega) = \lim_{k\to\infty} \Pr^{\varphi_{\mathrm{E}},\varphi_{\mathrm{A},\infty}}(V^{\leq k}FV^\omega) \leq \frac{1}{2} < 1$ which leads a contradiction with φ_{E} being almost-surely winning. ◀

Let $k < k'$, we define $\mathbb{G}_{[k,k']}$ the reachability game induced by $\mathbb{G}$ restricted to vertices of rank in $[k, k']$. Since $\mathcal{G}$ has finite out-degree, there are finitely many vertices of rank in $[k, k']$, hence $\mathbb{G}_{[k,k']}$ is finite.

We define inductively a sequence of ranks $(k_i)_{i\geq 1}$ together with a sequence of strategies $(\varphi_{\mathrm{E},[k_i,k_{i+1}[})_{i\geq 0}$ such that for all $i \geq 0$, $\varphi_{\mathrm{E},[k_i,k_{i+1}[}$ is a positional strategy, defined on all vertices of rank $[k_i, k_{i+1}[$, such that from all vertices of rank k_i, for all strategies φ_{A}, we have $\Pr^{\varphi_{\mathrm{E},[k_i,k_{i+1}[},\varphi_{\mathrm{A}}}(V^{\leq \ell}FV^\omega) \geq \frac{1}{2}$, where $\ell = k_{i+1} - k_i$.

Set $k_0 = 0$. Assume the first i ranks and strategies are defined. For each vertex of rank k_i, Lemma 9 shows the existence of a bound; since there are finitely many such vertices, we can consider the maximum of those bounds, and denote it by k_{i+1}. By construction, from all vertices of rank k_i, for all strategies φ_{A}, we have $\Pr^{\varphi_{\mathrm{E}},\varphi_{\mathrm{A}}}(V^{\leq \ell}FV^\omega) \geq \frac{1}{2}$, where $\ell = k_{i+1} - k_i$. In other words, Éloïse wins the chronological and finite reachability game $\mathbb{G}_{[k_i,k_{i+1}]}$ with probability at least $\frac{1}{2}$, so, thanks to Theorem 2, there exists a uniform positional strategy ensuring to reach F with probability at least $\frac{1}{2}$, denote it $\varphi_{\mathrm{E},[k_i,k_{i+1}[}$. This concludes the inductive construction.

Now define $\varphi_{\mathrm{E},\infty}$ as the disjoint union of the strategies $\varphi_{\mathrm{E},[k_i,k_{i+1}[}$. This is a positional strategy; we argue that it is almost-surely winning. Indeed, since $\varphi_{\mathrm{E},\infty}$ ensures that going through any slice, a vertex in F will be visited with probability half, the Borel-Cantelli Lemma implies that infinitely many vertices in F will be visited with probability one. ◀

4.2 The Reduction

Fix an alternating Büchi tree automaton $\mathcal{A} = (Q_\exists, Q_\forall, \Sigma, \Delta, q_{\mathrm{in}}, \rho)$. In order to check whether $L^{=1}(\mathcal{A}) = \emptyset$, we design an imperfect information *stochastic* Büchi game $\mathbb{G}^{=1}_{\mathcal{A}}$, in a way similar to the one to decide whether $L(\mathcal{A}) = \emptyset$ taking advantage the positionality result established in Theorem 7. In the game, Éloïse describes both a tree t and a positional strategy φ_t for her in the game $\mathbb{G}^{=1}_{\mathcal{A},t}$; the strategy φ_t is described as a $\mathcal{T}$-labeled tree (where $\mathcal{T}$ is the set of functions from $Q_\exists$ into $Q \times Q$, see Remark 2.2. As the plays are of ω-length, she actually does not fully describe t and φ_t but only a branch: this branch is chosen by Random, and Abélard takes care of computing the sequence of states along it (either by updating an existential state accordingly to φ_t or, when the state is universal, by choosing an arbitrary valid transition of the automaton). Éloïse observes the directions. Formally, we let $\mathcal{G}^{=1}_{\mathcal{A}} = (S, s_{\mathrm{in}}, A, T, \sim)$ where $S = (Q \times \{0, 1\}) \cup \{(q_{\mathrm{in}}, \varepsilon)\}$ and $s_{\mathrm{in}} = (q_{\mathrm{in}}, \varepsilon)$; $A \subseteq \Sigma \times \mathcal{T}$ is the set of pairs (a, τ) such

that for all $q \in Q_\exists$, $(q, a, q_0, q_1) \in \Delta$ where $\tau(q) = (q_0, q_1)$ and $T = \{((q,i),(a,\tau),d_{q_0,q_1}) \mid q \in Q_\exists \text{ and } \tau(q) = (q_0,q_1)\} \cup \{((q,i),(a,\tau),d_{q_0,q_1}) \mid q \in Q_\forall \text{ and } (q,a,q_0,q_1) \in \Delta\}$ where d_{q_0,q_1} is the probability distribution $(q_0,0) \mapsto 1/2$ and $(q_1,1) \mapsto 1/2$, and $(q,i) \sim (q',i)$ for all $q, q' \in Q$ and $i \in \{0,1\}$. Define $\mathbb{G}^{=1}_{\mathcal{A}} = (\mathcal{G}_\mathcal{A}, \rho_\mathcal{A})$ with $\rho_\mathcal{A}(q,i) = \rho(q)$ for any $(q,i) \in S$.

With a proof similar to the one of Lemma 4, we have the following result.

▶ **Theorem 10.** *Éloïse almost-surely wins in $\mathbb{G}^{=1}_{\mathcal{A}}$ iff $L^{=1}(\mathcal{A}) \neq \emptyset$.*

From [9, 7], one can decide almost-sure winning in imperfect information Büchi games in EXPTIME, hence the same holds for checking emptiness of languages of the form $L^{=1}(\mathcal{A})$. One can also reduce the emptiness problem for probabilistic ω-words automaton with the almost-sure semantics (in the sense of [2]) to check emptiness of languages of the form $L^{=1}(\mathcal{A})$ hence, it implies lower bounds as well as undecidability results.

▶ **Theorem 11.** *(1) Deciding whether $L^{=1}(\mathcal{A}) = \emptyset$ for a given alternating Büchi tree automaton $\mathcal{A}$ is an EXPTIME-complete problem. (2) Deciding whether $L^{=1}(\mathcal{A}) = \emptyset$ for a given alternating co-Büchi tree automaton $\mathcal{A}$ is an undecidable problem.*

▶ Remark. As one can decide whether $L^{=1}(\mathcal{A}) = \emptyset$ for non-deterministic tree automata [6], Theorem 11 implies that there is no effective simulation theorem for co-Büchi alternating qualitative tree automata.

References

1 A. Arnold and D. Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001.

2 C. Baier, M. Größer, and N. Bertrand. Probabilistic ω-automata. *Journal of the ACM*, 59(1):1, 2012.

3 N. Bertrand, B. Genest, and H. Gimbert. Qualitative determinacy and decidability of stochastic games with signals. In *Proc of LICS'09*, pages 319–328. IEEE, 2009.

4 D. Berwanger, K. Chatterjee, M. De Wulf, L. Doyen, and T.A Henzinger. Alpaga: A tool for solving parity games with imperfect information. In *Proc of TACAS'09*, LNCS. Springer, 2009.

5 T. Brázdil, A. Kučera, and P. Novotný. Determinacy in stochastic games with unbounded payoff functions. In *Proc of MEMICS'13*, pages 94–105, 2013.

6 A. Carayol, A. Haddad, and O. Serre. Qualitative tree languages. In *Proc of LICS'11*, pages 13–22. IEEE, 2011.

7 A. Carayol, C. Löding, and O. Serre. Pure strategies in imperfect information stochastic games. Submitted.

8 A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

9 K. Chatterjee and L. Doyen. Partial-observation stochastic games: How to win when belief fails. In *Proc of LICS'12*, pages 175–184. IEEE, 2012.

10 K. Chatterjee, L. Doyen, T.A. Henzinger, and J.-F. Raskin. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science*, 3(3), 2007.

11 A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

12 L. Doyen and J.-F. Raskin. Antichain algorithms for finite automata. In *Proc of TACAS'10*, volume 6015 of *LNCS*, pages 2–22. Springer, 2010.

13 R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

14 J. Flum, E. Grädel, and T. Wilke. *Logic and Automata: History and Perspectives*. Amsterdam University Press, 2007.

15 H. Gimbert and W. Zielonka. Perfect information stochastic priority games. In *Proc of ICALP'07*, LNCS, pages 850–861. Springer, 2007.

16 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

17 V. Gripon and O. Serre. Qualitative concurrent stochastic games with imperfect information. In *Proc of ICALP'09*, volume 5556 of *LNCS*, pages 200–211. Springer, 2009.

18 A. Kučera. *Turn-Based Stochastic Games*. Lectures in Game Theory for Computer Scientists. Cambridge University Press, 2011.

19 O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

20 S. Miyano and T. Hayashi. Alternating finite automata on omega-words. *Theoretical Computer Science*, 32:321–330, 1984.

21 D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata. *Theoretical Computer Science*, 141(1&2):69–107, 1995.

22 Bernd Puchala. *Synthesis of Winning Strategies for Interaction under Partial Information*. PhD thesis, RWTH Aachen University, 2013.

23 J.H. Reif. Universal games of incomplete information. In *Proc of STOC'79*, pages 288–308. ACM, 1979.

24 W. Thomas. Languages, automata, and logic. In *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.

25 M. De Wulf, L. Doyen, T.A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proc of CAV'06*, volume 4144 of *LNCS*, pages 17–30. Springer, 2006.

26 M. De Wulf, L. Doyen, N. Maquet, and J.-F. Raskin. Alaska. In *Proc of ATVA'08*, volume 5311 of *LNCS*, pages 240–245. Springer, 2008.

27 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.

Saturation of Concurrent Collapsible Pushdown Systems

Matthew Hague

Royal Holloway University of London, UK / LIGM, Marne-la-Vallée, France
matthew.hague@rhul.ac.uk

Abstract

Multi-stack pushdown systems are a well-studied model of concurrent computation using threads with first-order procedure calls. While, in general, reachability is undecidable, there are numerous restrictions on stack behaviour that lead to decidability. To model higher-order procedures calls, a generalisation of pushdown stacks called collapsible pushdown stacks are required. Reachability problems for multi-stack collapsible pushdown systems have been little studied. Here, we study ordered, phase-bounded and scope-bounded multi-stack collapsible pushdown systems using saturation techniques, showing decidability of control state reachability and giving a regular representation of all configurations that can reach a given control state.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Concurrency, Automata, Higher-Order, Verification, Model-Checking

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.313

1 Introduction

Pushdown systems augment a finite-state machine with a stack and accurately model first-order recursion. Such systems then are ideal for the analysis of sequential first-order programs and several successful tools, such as Moped [27] and SLAM [3], exist for their analysis. However, the domination of multi- and many-core machines means that programmers must be prepared to work in concurrent environments, with several interacting execution threads.

Unfortunately, the analysis of concurrent pushdown systems is well-known to be undecidable. However, most concurrent programs don't interact pathologically and many restrictions on interaction have been discovered that give decidability (e.g. [5, 6, 28, 15, 16]).

One particularly successful approach is *context-bounding*. This underapproximates a concurrent system by bounding the number of context switches that may occur [26]. It is based on the observation that most real-world bugs require only a small number of thread interactions [25]. Additionally, a number of more relaxed restrictions on stack behaviour have been introduced. In particular phase-bounded [31], scope-bounded [32], and ordered [7] (corrected in [2]) systems. There are also generic frameworks – that bound the tree- [22] or split-width [10] of the interactions between communication and storage – that give decidability for all communication architectures that can be defined within them.

Languages such as C++, Haskell, Javascript, Python, or Scala increasingly embrace higher-order procedure calls, which present a challenge to verification. A popular approach to modelling higher-order languages for verification is that of (higher-order recursion) schemes [11, 23, 17]. Collapsible pushdown systems (CPDS) are an extension of pushdown systems [14] with a "stack-of-stacks" structure. The "collapse" operation allows a CPDS to retrieve information about the context in which a stack character was created. These features give CPDS equivalent modelling power to schemes [14].

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 313–325

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

These two formalisms have good model-checking properties. E.g, it is decidable whether a μ-calculus formula holds on the execution graph of a scheme [23] (or CPDS [14]). Although, the complexity of such analyses is high, it has been shown by Kobayashi [16] (and Broadbent *et al.* for CPDS [8]) that they can be performed in practice on real code examples.

However concurrency for these models has been little studied. Work by Seth considers phase-bounding for CPDS without collapse [29] by reduction to a finite state parity game. Recent work by Kobayashi and Igarashi studies context-bounded recursion schemes [19].

Here, we study global reachability problems for ordered, phase-bounded, and scope-bounded CPDS. We use *saturation* methods, which have been successfully implemented by e.g. Moped [27] for pushdown systems and C-SHORe [8] for CPDS. Saturation was first applied to model-checking by Bouajjani *et al.* [4] and Finkel *et al.* [12]. We presented a saturation technique for CPDS in ICALP 2012 [9]. Here, we present the following advances.

1. Global reachability for ordered CPDSs (§5). This is based on Atig's algorithm [1] for ordered PDSs and requires a non-trivial generalisation of his notion of *extended* PDSs (§3). For this we introduce the notion of *transition automata* that encapsulate the behaviour of the saturation algorithm. In the full article we show how to use the same machinery to solve the global reachability problem for phase-bounded CPDSs.
2. Global reachability for scope-bounded CPDSs (§6). This is a backwards analysis based upon La Torre and Napoli's forwards analysis for scope-bounded PDSs, requiring new insights to complete the proofs.

Because the naive encoding of a single second-order stack has an undecidable MSO theory (we show this folklore result in the full paper) it remains a challenging open problem to generalise the generic frameworks above ([22, 10]) to CPDSs, since these frameworks rely on MSO decidability over graph representations of the storage and communication structure.

A full version of this paper with all definitions and proofs is available [13].

2 Preliminaries

Before defining CPDSs, we define $2\uparrow_0(x) = x$ and $2\uparrow_{i+1}(x) = 2^{2\uparrow_i(x)}$.

2.1 Collapsible Pushdown Systems (CPDS)

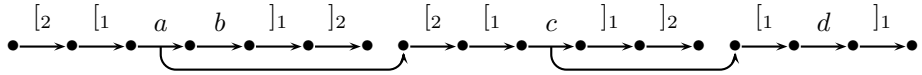
For a readable introduction to CPDS we defer to a survey by Ong [24]. Here, we can only briefly describe higher-order collapsible stacks and their operations. We use a notion of collapsible stacks called *annotated stacks* (which we refer to as collapsible stacks). These were introduced in ICALP 2012, and are essentially equivalent to the classical model [9].

Higher-Order Collapsible Stacks. An order-1 stack is a stack of symbols from a stack alphabet Σ, an order-n stack is a stack of order-$(n-1)$ stacks. A collapsible stack of order n is an order-n stack in which the stack symbols are annotated with collapsible stacks which may be of any order $\leq n$. Note, often in examples we will omit annotations for clarity. We fix the maximal order to n, and use k to range between n and 1. We simultaneously define for all $1 \leq k \leq n$, the set Stacks_k^n of order-k stacks whose symbols are annotated by stacks of order at most n. Note, we use subscripts to indicate the order of a stack. Furthermore, the definition below uses a least fixed-point. This ensures that all stacks are finite. An order-k stack is a collapsible stack in Stacks_k^n.

▶ **Definition 2.1** (Collapsible Stacks)**.** The family of sets $(\text{Stacks}_k^n)_{1\leq k\leq n}$ is the smallest family (for point-wise inclusion) such that:

1. for all $2 \leq k \leq n$, Stacks_k^n is the set of all (possibly empty) sequences $[w_1 \dots w_\ell]_k$ with $w_1, \dots, w_\ell \in \text{Stacks}_{k-1}^n$.
2. Stacks_1^n is all sequences $[a_1{}^{w_1} \dots a_\ell{}^{w_\ell}]_1$ with $\ell \geq 0$ and for all $1 \leq i \leq \ell$, a_i is a stack symbol in Σ and w_i is a collapsible stack in $\bigcup_{1 \leq k \leq n} \text{Stacks}_k^n$.

An order-n stack can be represented naturally as an edge-labelled tree over the alphabet $\{[_{n-1}, \dots, [_1,]_1, \dots,]_{n-1}\} \uplus \Sigma$, with Σ-labelled edges having a second target to the tree representing the annotation. We do not use $[_n$ or $]_n$ since they would appear uniquely at the beginning and end of the stack. An example order-3 stack is given below, with only a few annotations shown (on a and c). The annotations are order-3 and order-2 respectively.



Given an order-n stack $w = [w_1 \dots w_\ell]_n$, we define $top_{n+1}(w) = w$ and

$$\begin{array}{rcll} top_n([w_1 \dots w_\ell]_n) & = & w_1 & \text{when } \ell > 0 \\ top_n([]_n) & = & []_{n-1} & \text{otherwise} \\ top_k([w_1 \dots w_\ell]_n) & = & top_k(w_1) & \text{when } k < n \text{ and } \ell > 0 \end{array}$$

noting that $top_k(w)$ is undefined if $top_{k'}(w) = []_{k'-1}$ for any $k' > k$.

We write $u :_k v$ – where u is order-$(k-1)$ – to denote the stack obtained by placing u on top of the top_k stack of v. That is, if $v = [v_1 \dots v_\ell]_k$ then $u :_k v = [uv_1 \dots v_\ell]_k$, and if $v = [v_1 \dots v_\ell]_{k'}$ with $k' > k$, $u :_k v = [(u :_k v_1)\, v_2 \dots v_\ell]_{k'}$. This composition associates to the right. E.g., the stack $[[[a^w b]_1]_2]_3$ above can be written $u :_3 v$ where u is the order-2 stack $[[a^w b]_1]_2$ and v is the empty order-3 stack $[]_3$. Then $u :_3 u :_3 v$ is $[[[a^w b]_1]_2 [[a^w b]_1]_2]_3$.

Operations on Order-n Collapsible Stacks. The following operations can be performed on an order-n stack where *noop* is the null operation $noop(w) = w$.

$$\mathcal{O}_n \quad = \quad \{noop, pop_1\} \cup \left\{ rew_a, push_a^k, copy_k, pop_k \mid a \in \Sigma \wedge 2 \leq k \leq n \right\}$$

We define each $o \in \mathcal{O}_n$ for an order-n stack w. Annotations are created by $push_a^k$, which pushes a character onto w and annotates it with $top_{k+1}(pop_k(w))$. This, in essence, attaches a closure to a new character.

1. We set $pop_k(u :_k v) = v$.
2. We set $copy_k(u :_k v) = u :_k u :_k v$.
3. We set $collapse_k\left(a^{u'} :_1 u :_{(k+1)} v\right) = u' :_{(k+1)} v$ when u is order-k and $1 \leq k < n$; and $collapse_n(a^u :_1 v) = u$ when u is order-n.
4. We set $push_b^k(w) = b^u :_1 w$ where $u = top_{k+1}(pop_k(w))$.
5. We set $rew_b(a^u :_1 v) = b^u :_1 v$.

For example, beginning with $[[a]_1[b]_1]_2$ and applying $push_c^2$ we obtain $[[c^{[[b]_1]_2} a]_1 [b]_1]_2$. In this setting, the order-2 context information for the new character c is $[[b]_1]_2$. We can then apply $copy_2; collapse_2$ to get $[[c^{[[b]_1]_2} a]_1 [c^{[[b]_1]_2} a]_1 [b]_1]_2$ then $[[b]_1]_2$. That is, $collapse_k$ replaces the current top_{k+1} stack with the annotation attached to c.

Collapsible Pushdown Systems. We are now ready to define collapsible PDS.

▶ **Definition 2.2** (Collapsible Pushdown Systems)**.** An order-n *collapsible pushdown system (n-CPDS)* is a tuple $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R})$ where $\mathcal{P}$ is a finite set of control states, Σ is a finite stack alphabet, and $\mathcal{R} \subseteq (\mathcal{P} \times \Sigma \times \mathcal{O}_n \times \mathcal{P})$ is a set of rules.

We write *configurations* of a CPDS as a pair $\langle p, w\rangle \in \mathcal{P} \times \text{Stacks}_n^n$. We have a transition $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ via a rule (p, a, o, p') when $top_1(w) = a$ and $w' = o(w)$.

Consuming and Generating Rules. We distinguish two kinds of rule or operation: a rule (p, a, o, p') or operation o is *consuming* if $o = pop_k$ or $o = collapse_k$ for some k. Otherwise, it is *generating*. We write $\mathcal{R}_{\mathcal{G}_n}^{\mathcal{P},\Sigma}$ for the set of generating rules of the form (p, a, o, p') such that $p, p' \in \mathcal{P}$ and $a \in \Sigma$, and $o \in \mathcal{O}_n$. We simply write $\mathcal{R}_{\mathcal{G}_n}$ when no confusion may arise.

2.2 Saturation for CPDS

Our algorithms for concurrent CPDSs build upon the saturation technique for CPDSs [9]. In essence, we represent sets of configurations C using a $\mathcal{P}$-stack automaton A reading stacks. We define such automata and their languages $\mathcal{L}(A)$ below. Saturation adds new transitions to A – depending on rules of the CPDS and existing transitions in A – to obtain A' representing configurations with a path to a configuration in C. I.e., given a CPDS $\mathcal{C}$ with control states $\mathcal{P}$ and a $\mathcal{P}$-stack automaton A_0, we compute $Pre_{\mathcal{C}}^*(A_0)$ which is the smallest set s.t. $Pre_{\mathcal{C}}^*(A_0) \supseteq \mathcal{L}(A_0)$ and $Pre_{\mathcal{C}}^*(A_0) \supseteq \{\langle p, w\rangle \mid \exists \langle p, w\rangle \longrightarrow \langle p', w'\rangle \text{ s.t. } \langle p', w'\rangle \in Pre_{\mathcal{C}}^*(A_0)\}$.

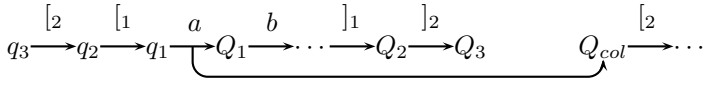
Stack Automata. Sets of stacks are represented using order-n stack automata. These are alternating automata with a nested structure that mimics the nesting in a higher-order collapsible stack. We recall the definition below.

▶ **Definition 2.3** (Order-n Stack Automata)**.** An *order-n stack automaton* is a tuple $A = (\mathbb{Q}_n, \ldots, \mathbb{Q}_1, \Sigma, \Delta_n, \ldots, \Delta_1, \mathcal{F}_n, \ldots, \mathcal{F}_1)$ where Σ is a finite stack alphabet, $\mathbb{Q}_n, \ldots, \mathbb{Q}_1$ are disjoint, and

1. for all $2 \le k \le n$, we have $\mathbb{Q}_k$ is a finite set of states, $\mathcal{F}_k \subseteq \mathbb{Q}_k$ is a set of accepting states, and $\Delta_k \subseteq \mathbb{Q}_k \times \mathbb{Q}_{k-1} \times 2^{\mathbb{Q}_k}$ is a transition relation such that for all q and Q there is *at most one* q' with $(q, q', Q) \in \Delta_k$, and
2. $\mathbb{Q}_1$ is a finite set of states, $\mathcal{F}_1 \subseteq \mathbb{Q}_1$ is a set of accepting states, and the transition relation is $\Delta_1 \subseteq \bigcup_{2 \le k \le n} \left(\mathbb{Q}_1 \times \Sigma \times 2^{\mathbb{Q}_k} \times 2^{\mathbb{Q}_1}\right)$.

States in $\mathbb{Q}_k$ recognise order-k stacks. Stacks are read from "top to bottom". A stack $u :_k v$ is accepted from q if there is a transition $(q, q', Q) \in \Delta_k$, written $q \xrightarrow{q'} Q$, such that u is accepted from $q' \in \mathbb{Q}_{(k-1)}$ and v is accepted from each state in Q. At order-1, a stack $a^u :_1 v$ is accepted from q if there is a transition (q, a, Q_{col}, Q) where u is accepted from all states in Q_{col} and v is accepted from all states in Q. An empty order-k stack is accepted by any state in $\mathcal{F}_k$. We write $w \in \mathcal{L}_q(A)$ to denote the set of all stacks w accepted from q. Note that a transition to the empty set is distinct from having no transition.

We show a part run using $q_3 \xrightarrow{q_2} Q_3 \in \Delta_3$, $q_2 \xrightarrow{q_1} Q_2 \in \Delta_2$, $q_1 \xrightarrow[Q_{col}]{a} Q_1 \in \Delta_1$.



Long-form Transitions. We will often use a *long-form* notation (defined below) that captures nested sequences of transitions. E.g. we can write $q_3 \xrightarrow[Q_{col}]{a} (Q_1, Q_2, Q_3)$ to represent the use of $q_3 \xrightarrow{q_2} Q_3$, $q_2 \xrightarrow{q_1} Q_2$, and $q_1 \xrightarrow[Q_{col}]{a} Q_1$ for the first three transitions of the run above. Note that this latter long-form transition starts at the very beginning of the stack and reads its top_1

character. Formally, for a sequence of transitions $q \xrightarrow{q_{k-1}} Q_k, q_{k-1} \xrightarrow{q_{k-2}} Q_{k-1}, \ldots, q_1 \xrightarrow[Q_{col}]{a} Q_1$ in Δ_k to Δ_1 respectively, we write $q \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_k)$.

$\mathcal{P}$-Stack Automata. We define $\mathcal{P}$-automata [4] for CPDSs. Given control states $\mathcal{P}$, an *order-n $\mathcal{P}$-stack automaton* is an order-n stack automaton such that for each $p \in \mathcal{P}$ there exists a state $q_p \in \mathbb{Q}_n$. We set $\mathcal{L}(A) = \{\langle p, w\rangle \mid w \in \mathcal{L}_{q_p}(A)\}$.

The Saturation Algorithm. We recall the saturation algorithm. For a detailed explanation of the saturation function complete with examples, we refer the reader to our ICALP paper [9]. Here we present an abstracted view of the algorithm, relegating details that are not directly relevant to the remainder of the main article to the full version.

The saturation algorithm iterates a *saturation function* Π that adds new transitions to a given automaton. Beginning with A_0 representing a target set of configurations, we iterate $A_{i+1} = \Pi(A_i)$ until $A_{i+1} = A_i$. Once this occurs, we have that $\mathcal{L}(A_i) = Pre^*_{\mathcal{C}}(A_0)$.

We define Π in terms of a family of auxiliary saturation functions Π_r (defined in the full article) which return a set of long-form transitions to be added by saturation. When r is consuming, $\Pi_r(A)$ returns the set of long-form transitions to be added to A due to the rule r. When r is generating Π_r also takes as an argument a long-form transition t of A. Thus $\Pi_r(t, A)$ returns the set of long-form transitions that should be added to A as a result of the rule r combined with the transition t (and possibly other transitions of A).

For example, if $r = (p, a, rew_b, p')$ and $t = q_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$ is a transition of A, then $\Pi_r(t, A)$ contains only the long-form transition $t' = q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$. The idea is if $\langle p', b^u :_1 w\rangle$ is accepted by A via a run whose first (sequence of) transition(s) is t, then by adding t' we will be able to accept $\langle p, a^u :_1 w\rangle$ via a run beginning with t' instead of t. We have $\langle p, a^u :_1 w\rangle \in Pre^*_{\mathcal{C}}(A)$ since it can reach $\langle p', b^u :_1 w\rangle$ via the rule r.

▶ **Definition 2.4** (The Saturation Function Π)**.** For a CPDS with rules $\mathcal{R}$, and given an order-n stack automaton A_i we define $A_{i+1} = \Pi(A_i)$. The state-sets of A_{i+1} are defined implicitly by the transitions which are those in A_i plus, for each $r = (p, a, o, p') \in \mathcal{R}$, when

1. o is consuming and $t \in \Pi_r(A_i)$, then add t to A_{i+1},
2. o is generating, t is in A_i, and $t' \in \Pi_r(t, A)$, then add t' to A_{i+1}.

In ICALP 2012 we showed that saturation adds up to $\mathcal{O}(2 \uparrow_n (f(|\mathcal{P}|)))$ transitions, for some polynomial f, and that this can be reduced to $\mathcal{O}(2 \uparrow_{n-1} (f(|\mathcal{P}|)))$ (which is optimal) by restricting all Q_n to have size 1 when A_0 is "non-alternating at order-n". Since this property holds of all A_0 used here, we use the optimal algorithm for complexity arguments.

3 Extended Collapsible Pushdown Systems

To analyse concurrent systems, we extend CPDS following Atig [1]. Atig's extended PDSs allow words from arbitrary languages to be pushed on the stack. Our notion of extended CPDSs allows sequences of *generating operations* from a language $\mathcal{L}_g$ to be applied, rather than a single operation per rule. We can specify $\mathcal{L}_g$ by any system (e.g. a Turing machine).

▶ **Definition 3.1** (Extended CPDSs)**.** An order-n *extended CPDS (n-ECPDS)* is a tuple $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R})$ where $\mathcal{P}$ is a finite set of control states, Σ is a finite stack alphabet, and $\mathcal{R} \subseteq (\mathcal{P} \times \Sigma \times \mathcal{O}_n \times \mathcal{P}) \cup \left(\mathcal{P} \times \Sigma \times 2^{\left(\mathcal{R}^{\mathcal{P},\Sigma}_{\mathcal{G}_n}\right)^*} \times \mathcal{P}\right)$ is a set of rules.

As before, we have a transition $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ of an n-ECPDS via a rule (p, a, o, p') with $top_1(w) = a$ and $w' = o(w)$. Additionally, we have a transition $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ when we have a rule $(p, a, \mathcal{L}_g, p')$, a sequence $(p, a, o_1, p_1)\,(p_1, a_2, o_2, p_2)\ldots(p_{\ell-1}, a_\ell, o_\ell, p') \in \mathcal{L}_g$ and $w' = o_\ell(\cdots o_1(w))$. That is, a single extended rule may apply a sequence of stack updates in one step. A run of an ECPDS is a sequence $\langle p_0, w_0\rangle \longrightarrow \langle p_1, w_1\rangle \longrightarrow \cdots$.

3.1 Reachability Analysis

We adapt saturation for ECPDSs. In Atig's algorithm, an essential property is the decidability of $\mathcal{L}_g \cap \mathcal{L}(A)$ for some order-1 $\mathcal{P}$-stack automaton A and a language $\mathcal{L}_g$ appearing in a rule of the extended PDS. We need analogous machinery in our setting. For this, we first define a class of finite automata called *transition* automata, written $\mathcal{T}$. The states of these automata will be long-form transitions of a stack automaton $t = q \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$. Transitions $t \xrightarrow{r} t'$ are labelled by rules. We write $t \xrightarrow{\overrightarrow{r}}_* t'$ to denote a run over $\overrightarrow{r} \in (\mathcal{R}_{\mathcal{G}_n})^*$.

During the saturation algorithm we will build from A_i a transition automaton $\mathcal{T}$. Then, for each rule $(p, a, \mathcal{L}_g, p')$ we add to A_{i+1} a new long-form transition t if there is a word $\overrightarrow{r} \in \mathcal{L}_g$ such that $t \xrightarrow{\overrightarrow{r}}_* t'$ is a run of $\mathcal{T}$ and t' is already a transition of A_i.

For example, consider $(p, a, \mathcal{L}_g, p')$ where $\mathcal{L}_g = \{(p, a, rew_b, p')\}$. A transition

$$\left(q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)\right) \xrightarrow{(p,a,rew_b,p')} \left(q_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)\right)$$

will correspond to the fact that the presence of $q_{p'} \xrightarrow[Q_{col}]{b} (Q_1, \ldots, Q_n)$ in A_i causes $q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ to be added by Π. A run $t_1 \xrightarrow{r_1} t_2 \xrightarrow{r_2} t_3$ comes into play when e.g. $\mathcal{L}_g = \{r_1 r_2\}$. If the rule were split into two ordinary rules with intermediate control states, Π would first add t_2 derived from t_3, and then from t_2 derive t_1. In the case of extended CPDSs, the intermediate transition t_2 is not added to A_{i+1}, but its effect is still present in the addition of t_1. Below, we repeat the above intuition more formally. Fix a n-ECPDS $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R})$.

Transition Automata. We build a transition automaton from a given $\mathcal{P}$-stack automaton A. Let A have order-n to order-1 state-sets $Q_n, \ldots, Q_1$ and alphabet Σ, let T_A be the set of all $q \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$ with $q \in Q_n$, for all k, $Q_k \subseteq \mathbb{Q}_k$, and for some k, $Q_{col} \subseteq \mathbb{Q}_k$.

▶ **Definition 3.2** (Transition Automata)**.** Given an order-n $\mathcal{P}$-stack automaton A with alphabet Σ, and $t, t' \in T_A$, we define the transition automaton $\mathcal{T}^A_{t,t'} = \left(T_A, \mathcal{R}^{\mathcal{P},\Sigma}_{\mathcal{G}_n}, \delta, t, t'\right)$ such that $\delta \subseteq T_A \times \mathcal{R}^{\mathcal{P},\Sigma}_{\mathcal{G}_n} \times T_A$ is the smallest set such that $t_1 \xrightarrow{r} t_2 \in \delta$ if $t_1 \in \Pi_r(t_2, A)$.

We define $\mathcal{L}\left(\mathcal{T}^A_{t,t'}\right) = \left\{\overrightarrow{r} \;\middle|\; t \xrightarrow{\overrightarrow{r}}_* t'\right\}$.

Extended Saturation Function. We now extend the saturation function following the intuition explained above. For $t = q_p \xrightarrow[Q_{col}]{a} (Q_1, \ldots, Q_n)$, let $top_1(t) = a$ and $control(t) = p$.

▶ **Definition 3.3** (Extended Saturation Function Π)**.** The extended Π is Π from Definition 2.4 plus for each extended rule $(p, a, \mathcal{L}_g, p') \in \mathcal{R}$ and t, t', we add t to A_{i+1} whenever

1. $control(t) = p$ and $top_1(t) = a$,
2. t' is a transition of A_i with $control(t') = p'$, and
3. $\mathcal{L}_g \cap \mathcal{L}\left(\mathcal{T}^{A_i}_{t,t'}\right) \neq \emptyset$.

▶ **Theorem 3.4** (Global Reachability of ECPDS). *Given an ECPDS $\mathcal{C}$ and a $\mathcal{P}$-stack automaton A_0, the fixed point A of the extended saturation procedure accepts $Pre^*_{\mathcal{C}}(A_0)$.*

In order for the saturation algorithm to be effective, we need to be able to decide $\mathcal{L}_g \cap \mathcal{L}\left(\mathcal{T}^{A_i}_{t,t'}\right) \neq \emptyset$. We argue in the full paper that number of transitions added by extended saturation has the same upper bound as the unextended case.

4 Multi-Stack CPDSs

We define a general model of concurrent collapsible pushdown systems, which we later restrict. In the sequel, assume a bottom-of-stack symbol $\perp$ and define the "empty" stacks $\perp_0 = \perp$ and $\perp_{k+1} = [\perp_k]_{k+1}$. As standard, we assume that $\perp$ is neither pushed onto, nor popped from, the stack (though may be copied by $copy_k$).

▶ **Definition 4.1** (Multi-Stack Collapsible Pushdown Systems). An order-n *multi-stack collapsible pushdown system (n-MCPDS)* is a tuple $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_m)$ where $\mathcal{P}$ is a finite set of control states, Σ is a finite stack alphabet, and for each $1 \leq i \leq m$ we have a set of rules $\mathcal{R}_i \subseteq \mathcal{P} \times \Sigma \times \mathcal{O}_n \times \mathcal{P}$.

A configuration of $\mathcal{C}$ is a tuple $\langle p, w_1, \ldots, w_m \rangle$. There is a transition $\langle p, w_1, \ldots, w_m \rangle \longrightarrow \langle p', w_1, \ldots, w_{i-1}, w'_i, w_{i+1}, \ldots, w_m \rangle$ via $(p, a, o, p') \in \mathcal{R}_i$ when $a = top_1(w_i)$ and $w'_i = o(w_i)$.

We also need MCPD*Automata*, which are MCPDSs defining languages over an input alphabet Γ. For this, we add labelling input characters to the rules. Thus, a rule (p, a, γ, o, p') reads a character $\gamma \in \Gamma$. This is defined formally in the full paper.

We are interested in two problems for a given n-MCPDS $\mathcal{C}$.

▶ **Definition 4.2** (Control State Reachability Problem). Given control states $p_{\text{in}}, p_{\text{out}}$ of $\mathcal{C}$, decide if there is for some $w_1, \ldots, w_m$ a run $\langle p_{\text{in}}, \perp_n, \ldots, \perp_n \rangle \longrightarrow \cdots \longrightarrow \langle p_{\text{out}}, w_1, \ldots, w_m \rangle$.

▶ **Definition 4.3** (Global Control State Reachability Problem). Given a control state p_{out} of $\mathcal{C}$, construct a representation of the set of configurations $\langle p, w_1, \ldots, w_m \rangle$ such that there exists for some $w'_1, \ldots, w'_m$ a run $\langle p, w_1, \ldots, w_m \rangle \longrightarrow \cdots \longrightarrow \langle p_{\text{out}}, w'_1, \ldots, w'_m \rangle$.

We represent sets of configurations as follows. In the full paper we show it forms an effective boolean algebra, membership is linear time, and emptiness is in PSPACE.

▶ **Definition 4.4** (Regular Set of Configurations). A regular set R of configurations of a multi-stack CPDS $\mathcal{C}$ is definable via a finite set χ of tuples $(p, A_1, \ldots, A_m)$ where p is a control state of $\mathcal{C}$ and A_i is a stack automaton with designated initial state q_i for each i. We have $\langle p, w_1, \ldots, w_m \rangle \in R$ iff there is some $(p, A_1, \ldots, A_m) \in \chi$ such that $w_i \in \mathcal{L}_{q_i}(A_i)$ for each i.

Finally, we often partition runs of an MCPDS $\sigma = \sigma_1 \ldots \sigma_\ell$ where each σ_i is a sequence of configurations of the MCPDS. A transition from c to c' occurs in segment σ_i if c' is a configuration in σ_i. Thus, transitions from σ_i to σ_{i+1} are said to belong to σ_{i+1}.

5 Ordered CPDS

We generalise *ordered multi-stack pushdown systems* [7]. Intuitively, we can only remove characters from stack i whenever all stacks $j < i$ are empty.

▶ **Definition 5.1** (Ordered CPDS). An order-n *ordered CPDS* (n-OCPDS) is an n-MCPDS $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_m)$ such that a transition from $\langle p, w_1, \ldots, w_m\rangle$ using the rule r on stack i is permitted iff, when r is consuming, for all $1 \leq j < i$ we have $w_j = \bot_n$.

▶ **Theorem 5.2** (Decidability of Reachability Problems). *For n-OCPDSs the control state reachability problem and the global control state reachability problem are decidable.*

We outline the proofs below. In the full paper we show control state reachability uses $\mathcal{O}\big(2 \uparrow_{m(n-1)} (\ell)\big)$ time, where ℓ is polynomial in the size of the OCPDS, and we have at most $\mathcal{O}(2 \uparrow_{mn} (\ell))$ tuples in the solution to the global problem. First observe that reachability can be reduced to reaching $\langle p_{\text{out}}, \bot_n, \ldots, \bot_n\rangle$ by clearing the stacks at the end of the run.

Control State Reachability. Using our notion of ECPDS, we may adapt Atig's inductive algorithm for ordered PDSs [1] for the control state reachability problem. The induction is over the number of stacks. W.l.o.g. we assume that all rules $(p, \bot, o, p')$ of $\mathcal{C}$ have $o = push_a^n$.

In the base case, we have an n-OCPDS with a single stack, for which the global reachability problem is known to be decidable (e.g. [4]).

In the inductive case, we have an n-OCPDS $\mathcal{C}$ with m stacks. By induction, we can decide the reachability problem for n-OCPDSs with fewer than m stacks. We first show how to reduce the problem to reachability analysis of an extended CPDS, and then finally we show how to decide $\mathcal{L}_g \cap \mathcal{L}\Big(\mathcal{T}^{A_i}_{t,t'}\Big) \neq \emptyset$ using an n-OCPDS with $(m-1)$ stacks.

Consider the mth stack of $\mathcal{C}$. A run of $\mathcal{C}$ can be split into $\sigma_1\tau_1\sigma_2\tau_2\ldots\sigma_\ell\tau_\ell$. During the subruns σ_i, the first $(m-1)$ stacks are non-empty, and during τ_i, the first $(m-1)$ stacks are empty. Moreover, during each σ_i, only generating operations may occur on stack m.

We build an extended CPDS that directly models the mth stack during the τ_i segments where the first $(m-1)$ stacks are empty, and uses rules of the form $(p, a, \mathcal{L}_g, p')$ to encapsulate the behaviour of the σ_i sections where the first $(m-1)$ stacks are non-empty. The $\mathcal{L}_g$ attached to such a rule is the sequence of updates applied to the mth stack during σ_i.

We begin by defining, from the OCPDS $\mathcal{C}$ with m stacks, an OCPDA $\mathcal{C}^L$ with $(m-1)$ stacks. This OCPDA will be used to define the $\mathcal{L}_g$ described above. $\mathcal{C}^L$ simulates a segment σ_i. Since all updates to stack m in σ_i are generating, $\mathcal{C}^L$ need only track its top character, hence only keeps $(m-1)$ stacks. The top character of stack m is kept in the control state, and the operations that would have occurred on stack m are output.

▶ **Definition 5.3** ($\mathcal{C}^L$). Given an n-OCPDS $\mathcal{C} = (\mathcal{P}, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_m)$, we define $\mathcal{C}^L$ to be an n-OCPDA with $(m-1)$ stacks $\big(\mathcal{P} \times \Sigma, \Sigma, \mathcal{R}'_1 \cup \mathcal{R}', \mathcal{R}'_2, \ldots, \mathcal{R}'_{m-1}\big)$ over input alphabet $\mathcal{R}_{\mathcal{G}_n}$ where for all i

$$\mathcal{R}'_i = \{((p,a), b, (p,a,noop,p'), o, (p',a)) \mid a \in \Sigma \wedge (p,b,o,p') \in \mathcal{R}_i\}, \text{ and}$$

$$\begin{aligned}\mathcal{R}' \;=\; & \{((p,a), b, r, noop, (p',c)) \mid b \in \Sigma \wedge r = (p,a,rew_c,p') \in \mathcal{R}_m\} \cup \\ & \{((p,a), b, r, noop, (p',a)) \mid b \in \Sigma \wedge r = (p,a,copy_k,p') \in \mathcal{R}_m\} \cup \\ & \big\{((p,a), b, r, noop, (p',c)) \mid b \in \Sigma \wedge r = \big(p,a,push_c^k,p'\big) \in \mathcal{R}_m\big\} \cup \\ & \{((p,a), b, r, noop, (p',a)) \mid b \in \Sigma \wedge r = (p,a,noop,p') \in \mathcal{R}_m\}\ .\end{aligned}$$

We define the language $\mathcal{L}^{b,i}_{p,a,p'}\big(\mathcal{C}^L\big)$ to be the set of words $\gamma_1\ldots\gamma_\ell$ such that there exists a run of $\mathcal{C}^L$ over input $\gamma_1\ldots\gamma_\ell$ from $\langle(p,a), w_1, \ldots, w_{m-1}\rangle$ to $\langle(p',c), \bot_n, \ldots, \bot_n\rangle$ for some c, where $w_i = push_b^n(\bot_n)$ and $w_j = \bot_n$ for all $j \neq i$. This language describes the effect on stack m of a run σ_j from p to p'. (Note, by assumption, all σ_j start with some $push_b^n$.)

We now define the extended CPDS $\mathcal{C}^R$ that simulates $\mathcal{C}$ by keeping track of stack m in its stack and using extended rules based on $\mathcal{C}^L$ to simulate parts of the run where the first

$(m-1)$ stacks are not all empty. Note, since all rules operating on $\perp$ (i.e. $(p, \perp, o, p')$) have $o = push_b^n$, rules from $\mathcal{R}_1, \ldots, \mathcal{R}_{m-1}$ may only fire during (or at the start of) the segments where the first $(m-1)$ stacks are non-empty (and thus appear in $\mathcal{R}_{\mathcal{L}_g}$ below).

▶ **Definition 5.4** ($\mathcal{C}^R$). Given an n-OCPDS $\mathcal{C} = (\mathcal{P} \times \Sigma, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_m)$ with m stacks, we define $\mathcal{C}^R$ to be an n-ECPDS such that $\mathcal{C}^R = (\mathcal{P}, \Sigma, \mathcal{R}')$ where $\mathcal{R}' = \mathcal{R}_m \cup \mathcal{R}_{\mathcal{L}_g}$ and

$$\mathcal{R}_{\mathcal{L}_g} = \left\{ (p, a, \mathcal{L}^{b,i}_{p_1,a,p_2}(\mathcal{C}^L), p_2) \mid a \in \Sigma \wedge (p, \perp, push_b^n, p_1) \in \mathcal{R}_i \wedge 1 \leq i < m \right\}$$

▶ **Lemma 5.5** ($\mathcal{C}^R$ simulates $\mathcal{C}$). *Given an n-OCPDS $\mathcal{C}$ and control states p_{in}, p_{out}, we have $\langle p_{in}, w\rangle \in Pre^*_{\mathcal{C}^R}(A)$, where A is the $\mathcal{P}$-stack automaton accepting only the configuration $\langle p_{out}, \perp_n\rangle$ iff $\langle p_{in}, \perp_n, \ldots, \perp_n, w\rangle \longrightarrow \cdots \longrightarrow \langle p_{out}, \perp_n, \ldots, \perp_n\rangle$.*

Lemma 5.5 only gives an effective decision procedure if we can decide $\mathcal{L}_g \cap \mathcal{L}\left(\mathcal{T}^{A_i}_{t,t'}\right) \neq \emptyset$ for all rules $(p, a, \mathcal{L}_g, p')$ appearing in $\mathcal{C}^R$. For this, we use a standard product construction between the $\mathcal{C}^L$ associated with $\mathcal{L}_g$, and $\mathcal{T}^{A_i}_{t,t'}$. This gives an ordered CPDS with $(m-1)$ stacks, for which, by induction over the number of stacks, reachability (and emptiness) is decidable. Note, the initial transition of the construction sets up the initial stacks of $\mathcal{C}^L$.

▶ **Definition 5.6** ($\mathcal{C}_\emptyset$). Given the non-emptiness problem $\mathcal{L}^{b,i}_{p_1,a,p_2}(\mathcal{C}^L) \cap \mathcal{L}\left(\mathcal{T}^{A_i}_{t,t'}\right) \neq \emptyset$, where $top_1(t) = a$, $\mathcal{C}^L = (\mathcal{P} \times \Sigma, \Sigma, \mathcal{R}_1, \ldots, \mathcal{R}_{m-1})$ and $\mathcal{T}^{A_i}_{t,t'} = (T_{A_i}, \mathcal{R}_{\mathcal{G}_n}, \delta, t, t')$, we define an n-OCPDS $\mathcal{C}_\emptyset = \left(\mathcal{P}^\emptyset, \Sigma, \mathcal{R}^\emptyset_1, \ldots, \mathcal{R}^\emptyset_i \cup \mathcal{R}_{I/O}, \ldots, \mathcal{R}^\emptyset_{m-1}\right)$ where, for all $1 \leq i \leq (m-1)$,

$$\begin{aligned}
\mathcal{P}^\emptyset &= \{p_1, p_2\} \uplus \{(p, t_1) \mid t_1 \in T_{A_i} \wedge control(t_1) = p\} \ , \\
\mathcal{R}_{I/O} &= \{(p_1, \perp, push_b^n, (p_1, t))\} \cup \{((p_2, t), \perp, noop, p_2) \mid t \in T_{A_i}\} \ , \text{ and} \\
\mathcal{R}^\emptyset_i &= \{((p, t_1), c, o, (p', t_2)) \mid ((p, top_1(t_1)), c, r, o, (p', top_1(t_2))) \in \mathcal{R}_i \wedge (t_1, r, t_2) \in \Delta\}
\end{aligned}$$

▶ **Lemma 5.7** (Language Emptiness for OCPDS). *We have $\mathcal{L}^{b,i}_{p_1,a,p_2}(\mathcal{C}^L) \cap \mathcal{L}\left(\mathcal{T}^{A_i}_{t,t'}\right) \neq \emptyset$ iff, in $\mathcal{C}_\emptyset$ from Definition 5.6, we have that $\langle p_2, \perp_n, \ldots \perp_n\rangle$ is reachable from $\langle p_1, \perp_n, \ldots, \perp_n\rangle$.*

Global Reachability. We sketch a solution to the global reachability problem, giving a full proof in the full paper. From Lemma 5.5 ($\mathcal{C}^R$ simulates $\mathcal{C}$) we gain a representation $A_m = Pre^*_{\mathcal{C}^R}(A)$ of the set of configurations $\langle p, \perp_n, \ldots, \perp_n, w_m\rangle$ that have a run to $\langle p_{\text{out}}, \perp_n, \ldots, \perp_n\rangle$. Now take any $\langle p, \perp_n, \ldots, \perp_n, w_{m-1}, w_m\rangle$ that reaches $\langle p_{\text{out}}, \perp_n, \ldots, \perp_n\rangle$. The run must pass some $\langle p', \perp_n, \ldots, \perp_n, w'_m\rangle$ with $\langle p', w'_m\rangle$ accepted by A_m. From the product construction above, one can (though not immediately) extract a tuple (p, A_{m-1}, A'_m) such that w_{m-1} is accepted by A_{m-1} and w_m is accepted by A'_m. We repeat this reasoning down to stack 1 and obtain a tuple of the form $(p, A_1, \ldots, A_m)$. We can only obtain a finite set of tuples in this manner, giving a solution to the global reachability problem.

6 Scope-Bounded CPDS

Recently, scope-bounded multi-pushdown systems were introduced [32] and their reachability problem was shown to be decidable. Furthermore, reachability for scope- and phase-bounding was shown to be incomparable [32]. Here we consider scope-bounded CPDS.

A run $\sigma = \sigma_1 \ldots \sigma_\ell$ of an MCPDS is *context-partitionable* when, for each σ_i, if a transition in σ_i is via $r \in \mathcal{R}_j$ on stack j, then all transitions of σ_i are via rules in $\mathcal{R}_j$ on stack j. A *round* is a context-partitioned run $\sigma_1 \ldots \sigma_m$, where during σ_i only $\mathcal{R}_i$ is used. A *round-partitionable* run can be partitioned $\sigma_1 \ldots \sigma_\ell$ where each σ_i is a round. A run of an SBCPDS is such

that any character or stack removed from a stack must have been created at most ζ rounds earlier. For this, we define pop- and collapse-rounds for stacks. That is, we mark each stack and character with the round in which it was created. When we copy a stack via $copy_k$, the pop-round of the new copy of the stack is the current round. However, all stacks and characters within the copy of u keep the same pop- and collapse-round as in the original u.

E.g. take $[u]_2$ where $u = [ab]_1$, u and a have pop-round 2, and b has pop-round 1. Suppose in round 3 we use $copy_2$ to obtain $[uu]_2$. The new copy of u has pop-round 3 (the current round), but the a and b appearing in the copy of u still have pop-rounds 2 and 1 respectively. If the scope-bound is 2, the latest each a and the original u could be popped is in round 4, but the new u may be popped in round 5.

We will write ${}_{\mathfrak{p}}w$ for a stack w with pop-round $\mathfrak{p}$ and ${}_{\mathfrak{p},\mathfrak{c}}a$ for a character with pop-round $\mathfrak{p}$ and collapse-round $\mathfrak{c}$. Pop- and collapse-rounds will be sometimes omitted for clarity. Note, the outermost stack will always have pop-round 0. In particular, for all $u :_k v$ in the definition below, the pop-round of v is 0.

▶ **Definition 6.1** (Pop- and Collapse-Round)**.** Given a round-partitioned run $\sigma_1 \dots \sigma_\ell$ we define inductively the pop- and collapse-rounds. The pop- and collapse-round of each stack and character in the first configuration of σ_1 is 0. Take a transition $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ with $\langle p', w'\rangle$ in σ_z via a rule (p, a, o, p'). If $o = noop$ then $w = w'$, otherwise when

1. $o = copy_k$ and $w = {}_{\mathfrak{p}}u :_k v$, then $w' = {}_z u :_k ({}_{\mathfrak{p}}u :_k v)$ where ${}_z u = {}_z[{}_{\mathfrak{p}_1}u_1 \dots {}_{\mathfrak{p}_\ell}u_\ell]_{k-1}$ when ${}_{\mathfrak{p}}u = {}_{\mathfrak{p}}[{}_{\mathfrak{p}_1}u_1 \dots {}_{\mathfrak{p}_\ell}u_\ell]_{k-1}$.
2. $o = push_b^k$, then $w' = {}_{z,\mathfrak{c}}b^{({}_{\mathfrak{p}'}u)} :_1 w$ where ${}_{\mathfrak{p}'}u = top_{k+1}(pop_k(w))$ and $\mathfrak{c}$ is the pop-round of $top_k(w)$. (Note, when $k = n$, we know $\mathfrak{p}' = 0$ since the top_{n+1} stack is outermost.)
3. $o = pop_k$, when $w = u :_k v$ then $w' = v$.
4. We set $collapse_k\left(a^{({}_{\mathfrak{p}}u')} :_1 u :_{(k+1)} v\right) = {}_{\mathfrak{p}}u' :_{(k+1)} v$ when u is order-k and $1 \le k < n$; and $collapse_n\left(a^{({}_{0}u)} :_1 v\right) = {}_0u$ when u is order-n.
5. $o = rew_b$ and $w = {}_{\mathfrak{p},\mathfrak{c}}a^{({}_{\mathfrak{p}'}u)} :_1 v$, then $w' = {}_{\mathfrak{p},\mathfrak{c}}b^{({}_{\mathfrak{p}'}u)} :_1 v$.

▶ **Definition 6.2** (Scope-Bounded CPDS)**.** A ζ-scope-bounded n-CPDS (n-SBCPDS) $\mathcal{C}$ is an order-n MCPDS whose runs are all runs of $\mathcal{C}$ that are round-partitionable, that is $\sigma_1 \dots \sigma_\ell$, such that for all z, if a transition in σ_z from $\langle p, w\rangle$ to $\langle p', w'\rangle$ is

1. a pop_k transition with $1 < k \le n$ and $w = {}_{\mathfrak{p}}u :_k v$, then $z - \zeta \le \mathfrak{p}$,
2. a pop_1 transition with $w = {}_{\mathfrak{p},\mathfrak{c}}a^u :_1 v$, then $z - \zeta \le \mathfrak{p}$, or
3. a $collapse_k$ transition with $w = {}_{\mathfrak{p},\mathfrak{c}}a^u :_1 v$, then $z - \zeta \le \mathfrak{c}$.

La Torre and Napoli's decidability proof for the order-1 case already uses the saturation method [32]. However, while La Torre and Napoli use a forwards-reachability analysis, we must use a backwards analysis. This is because the forwards-reachable set of configurations is in general not regular. We thus perform a backwards analysis for CPDS, resulting in a similar approach. However, the proofs of correctness of the algorithm are quite different.

▶ **Theorem 6.3** (Decidability of Reachability Problems)**.** *For n-OCPDSs the control state reachability problem and the global control state reachability problem are decidable.*

In the full paper we show our non-global algorithm requires $\mathcal{O}(2 \uparrow_{n-1} (\ell))$ space, where ℓ is polynomial in ζ and the size of the SBCPDS, and we have at most $\mathcal{O}(2 \uparrow_n (\ell))$ tuples in the global reachability solution. La Torre and Parlato give an alternative control state reachability algorithm at order-1 using *thread interfaces*, which allows sequentialisation [21] and should generalise order-n, but, does not solve the global reachability problem.

Control State Reachability. Fix initial and target control states p_{in} and p_{out}. The algorithm first builds a *reachability graph*, which is a finite graph with a certain kind of path iff p_{out} can be reached from p_{in}. To build the graph, we define layered stack automata. These have states q_p^i for each $1 \leq i \leq \zeta$ which represent the stack contents i rounds later. Thus, a layer automaton tracks the stack across ζ rounds, which allows analysis of scope-bounded CPDSs.

▶ **Definition 6.4** (ζ-Layered Stack Automata). A *ζ-layered stack automaton* is a stack automaton A such that $\mathbb{Q}_n = \left\{ q_p^i \mid p \in \mathcal{P} \wedge 1 \leq i \leq \zeta \right\}$.

A state q_p^i is of layer i. A state q' labelling $q \xrightarrow{q'} Q$ has the same layer as q. We require that there is no $q \xrightarrow{q'} Q$ with $q'' \in Q$ where q is of layer i and q'' is of layer $j < i$. Similarly, there is no $q \xrightarrow[Q_{col}]{a} Q$ with $q' \in Q \cup Q_{col}$ where q is of layer i and q' is of layer $j < i$.

Next, we define several operations from which the reachability graph is constructed. The $\texttt{Predecessor}_j$ operation connects stack j between two rounds. We define for stack j

$$\texttt{Predecessor}_j(A, q_p, q_{p'}) = \texttt{Saturate}_j\big(\texttt{EnvMove}\big(\texttt{Shift}(A), q_{p_1}^1, q_{p_2}^2\big)\big)$$

where definitions of $\texttt{Shift}$, $\texttt{EnvMove}$ and $\texttt{Saturate}_j$ are given in the full paper. $\texttt{Shift}$ moves transitions in layer i to layer $(i+1)$. E.g. $q_p^1 \xrightarrow{q} \left\{q_{p'}^2\right\}$ would become $q_p^2 \xrightarrow{q} \left\{q_{p'}^3\right\}$. Moreover, transitions involving states in layer ζ are removed. This is because the stack elements in layer ζ will "go out of scope". $\texttt{EnvMove}$ adds a new transition (analogously to a (p_1, a, rew_a, p_2) rule) corresponding to the control state change from p_1 to p_2 effected by the runs over the other stacks between the current round and the next (hence layers 1 and 2 in the definition above). $\texttt{Saturate}_j$ gets by saturation all configurations of stack j that can reach via $\mathcal{R}_j$ the stacks accepted from the layer-1 states of its argument (i.e. saturation using initial states $\left\{q_p^1 \mid p \in \mathcal{P}\right\}$, which accept stacks from the next round).

The current layer automaton represents a stack across up to ζ rounds. The predecessor operation adds another round on to the front of this representation. A key new insight in our proofs is that if a transition goes to a layer i state, then it represents part of a run where the stack read by the transition is removed in i rounds time. Thus, if we add a transition at layer 0 (were it to exist) that depends on a transition of layer ζ, then the push or copy operation would have a corresponding pop $(\zeta + 1)$ scopes away. Scope-bounding forbids this.

The Reachability Graph. The reachability graph $\mathcal{G}_{\mathcal{C}}^{p_{\text{out}}} = (\mathcal{V}, \mathcal{E})$ has vertices $\mathcal{V}$ and edges $\mathcal{E}$. Firstly, $\mathcal{V}$ contains some *initial* vertices $(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)$ where $p_m = p_{\text{out}}$, and for all $1 \leq i \leq m$ we have that A_i is the layer automaton $\texttt{Saturate}_i(A)$ where for all w, A accepts $\langle p_i, w\rangle$ from $q_{p_i}^1$. Furthermore, we require that there is some w such that $\langle p_{i-1}, w\rangle$ is accepted by A_i from $q_{p_i}^1$. That is, there is a run from $\langle p_{i-1}, w\rangle$ to p_i. Intuitively, initial vertices model the final round of a run to p_{out} with context switches at $p_0, \ldots, p_m$.

The complete set $\mathcal{V}$ is the set of all tuples $(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)$ where there is some w such that $\langle p_{i-1}, w\rangle$ is accepted by A_i from state $q_{p_{i-1}}^1$. To ensure finiteness, we can bound A_i to at most N states. The value of N is $\mathcal{O}(2 \uparrow_{n-1} (\ell))$ where ℓ is polynomial in ζ and the size of $\mathcal{C}$. We give a full definition of N and proof in the full paper.

We have an edge from a vertex $(p_0, A_1, \ldots, A_m, p_m)$ to $(p_0', A_1', \ldots, A_m', p_m')$ whenever $p_m = p_0'$ and for all i we have $A_i = \texttt{Predecessor}_i\left(A_i', q_{p_i}, q_{p_{i-1}'}\right)$. An edge means the two rounds can be concatenated into a run since the control states and stack contents match up.

▶ **Lemma 6.5** (Simulation by $\mathcal{G}_{\mathcal{C}}^{p_{\text{out}}}$). *Given a scope-bounded CPDS $\mathcal{C}$ and control states p_{in}, p_{out}, there is a run of $\mathcal{C}$ from $\langle p_{in}, w_1, \ldots, w_m\rangle$ to $\langle p_{out}, w_1', \ldots, w_m'\rangle$ for some $w_1', \ldots, w_m'$*

iff there is a path in $\mathcal{G}_{\mathcal{C}}^{p_{out}}$ *to a vertex* $(p_0, A_1, \ldots, A_m, p_m)$ *with* $p_0 = p_{in}$ *from an initial vertex where for all* i *we have* $\langle p_{i-1}, w_i \rangle$ *accepted from* $q^1_{p_i}$ *of* A_i.

Global Reachability. The $(p_0, A_1, p_1, \ldots, p_{m-1}, A_m, p_m)$ in $\mathcal{G}_{\mathcal{C}}^{p_{\text{out}}}$ reachable from an initial vertex are finite in number. We know by Lemma 6.5 that there is such a vertex accepting all $\langle p_{i-1}, w_i \rangle$ iff $\langle p_0, w_1, \ldots, w_m \rangle$ can reach the target control state. Let χ be the set of tuples $(p_0, A_1, \ldots, A_m)$ for each reachable vertex as above, where A_i is restricted to the initial state $q^1_{p_{i-1}}$. This is a regular solution to the global control state reachability problem.

7 Conclusion

We have shown decidability of global reachability for ordered and scope-bounded collapsible pushdown systems (and phase-bounded in the full article). This leads to a challenge to find a general framework capturing these systems. Furthermore, we have only shown upper-bound results. Although, in the case of phase-bounded systems, our upper-bound matches that of Seth for CPDSs without collapse [29], we do not know if it is optimal. Obtaining matching lower-bounds is thus an interesting though non-obvious problem. Recently, a more relaxed notion of scope-bounding has been studied [20]. It would be interesting to see if we can extend our results to this notion. We are also interested in developing and implementing algorithms that may perform well in practice.

Acknowledgments. Many thanks for initial discussions with Arnaud Carayol and to the referees for their helpful remarks. This work was supported by Fond. Sci. Math. Paris; AMIS [ANR 2010 JCJC 0203 01 AMIS]; FREC [ANR 2010 BLAN 0202 02 FREC]; VAPF (Région IdF); and the Engineering and Physical Sciences Research Council [EP/K009907/1].

References

1 M. F. Atig. Model-checking of ordered multi-pushdown automata. In *LMCS*, 8(3), 2012.
2 M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2etime-complete. In *DLT*, 2008.
3 T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In *POPL*, 2002.
4 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, 1997.
5 A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. *SIGPLAN Not.*, 38(1):62–73, 2003.
6 A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. *CONCUR*, 2005.
7 L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.
8 C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. C-SHORe: A collapsible approach to verifying higher-order programs. To appear in *ICFP*, 2013.
9 C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *ICALP*, 2012.
10 A. Cyriac, P. Gastin, and K. N. Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR*, 2012.
11 W. Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982.

12 A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *INFINITY*, 1997.

13 M. Hague. Saturation of concurrent collapsible pushdown systems, 2013. arXiv:1310.2631 [cs.FL].

14 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, 2008.

15 A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In *FOSSACS*, 2010.

16 V. Kahlon. Reasoning about threads with bounded lock chains. In *CONCUR*, 2011.

17 T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *ICALP*, 2005.

18 N. Kobayashi. Higher-order model checking: From theory to practice. In *LICS*, 2011.

19 N. Kobayashi and A. Igarashi. Model-Checking Higher-Order Programs with Recursive Types In *ESOP*, 2013.

20 S. La Torre and M. Napoli. A temporal logic for multi-threaded programs. In *IFIP TCS*, 2012.

21 S. La Torre and G. Parlato. Scope-bounded multistack pushdown systems: Fixed-point, sequentialization, and tree-width. In *FSTTCS*, 2012.

22 P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL*, 2011.

23 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, 2006.

24 L. Ong. Recursion schemes, collapsible pushdown automata and higher-order model checking. In *LATA*, 2013.

25 S. Qadeer. The case for context-bounded verification of concurrent programs. In *SPIN*, 2008.

26 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, 2005.

27 S. Schwoon. *Model-checking Pushdown Systems*. PhD thesis, Technical University of Munich, 2002.

28 K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV*, 2006.

29 A. Seth. Games on higher order multi-stack pushdown systems. In *RP*, 2009.

30 A. Seth. Global reachability in bounded phase multi-stack pushdown systems. In *CAV*, 2010.

31 S. L. Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, 2007.

32 S. L. Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR*, 2011.

Decidability Results on the Existence of Lookahead Delegators for NFA

Christof Löding and Stefan Repke

Lehrstuhl für Informatik 7, RWTH Aachen, Aachen, Germany

Abstract

In this paper, we study lookahead delegators for nondeterministic finite automata (NFA), which are functions that deterministically choose transitions by additionally using a bounded lookahead on the input word. Of course, the delegator has to lead to an accepting state for each word that is accepted by the NFA. In the special case where no lookahead is allowed, a delegator coincides with a deterministic transition function that preserves the language.

Typical decision problems are to decide whether a delegator with a given fixed lookahead exists, or whether a delegator with some bounded lookahead exists for a given NFA. In a paper of Ravikumar and Santean from 2007, the complexity and decidability of these questions have been tackled, mainly for the case of unambiguous NFA. In this paper, we revisit the subject and provide results for the case of general NFA. First, we correct a complexity result from the above paper by showing that the existence of delegators with fixed lookahead can be decided in time polynomial in the number of states. We use two player games on graphs as a tool to obtain the result. As second contribution, we show that the problem becomes PSPACE-complete if the bound on the lookahead is a part of the input. The third result provides a bound on the maximal required amount of lookahead. We use this to show that the (previously open) problem of deciding the existence of a bounded lookahead delegator is also PSPACE-complete.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Automata, Lookahead Delegators, Safety Games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.327

1 Introduction

We revisit questions on the decidability of so called lookahead delegators for nondeterministic finite automata (NFA) that have been studied in [9]. A lookahead delegator for an NFA is a function that deterministically chooses one of the possible transitions for the next input symbol, based on a bounded lookahead in the input. The run that is constructed in this way by the delegator for an input word should be accepting if the input word belongs to the language accepted by the NFA.

Motivated from the composition of e-services that are modeled by finite automata, lookahead delegators have been studied in [4] in a slightly different context. In the setting of [4], instead of a single NFA, a tuple of deterministic finite automata (DFA) or NFAs is given, and the delegator has to decide for each input symbol by which subset of the automata this symbol has to be processed, such that in the end, all automata finish with a successful run. Given a bound k on the allowed lookahead, [4] presents an algorithm to compute a k-lookahead delegator (if one exists) for a tuple of DFAs. The algorithm is exponential in k and the number of DFAs in the list.

By taking the product of the tuple of the given automata, in which a transition nondeterministically chooses a subset of the automata that move on the next input symbol, the setting can be reduced to the question on lookahead delegators for a single given NFA. This

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 327–338

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

version of the problem has been analyzed in [9]. There are two main questions concerning the existence of lookahead delegators. For a given bound k on the lookahead, the decidability of the existence of a k-lookahead delegator is clear, because there are only finitely many possible candidates for a given NFA. So, for a given (or a fixed) k, the main question is the complexity of deciding the existence of a k-lookahead delegator (and computing one if it exists). If the bound is not given, the question is whether there exists some k such that a given NFA has a k-lookahead delegator.

In this paper, we will reconsider the three versions of the decision problem that were defined in [9, Section 4]:

a) k-DELEGATOR for a fixed number $k \in \mathbb{N}$: decide for a given NFA $\mathcal{A}$ whether $\mathcal{A}$ has a k-lookahead delegator.
b) DELEGATOR: decide for a given NFA $\mathcal{A}$ and $k \in \mathbb{N}$ whether $\mathcal{A}$ has a k-lookahead delegator.
c) BOUNDED-DELEGATOR: decide for a given NFA $\mathcal{A}$ whether $\mathcal{A}$ has a bounded lookahead delegator.

These problems have been solved for the restricted case of unambiguous NFAs in [9, Theorems 2 to 4]: A polynomial time algorithm is given for k-DELEGATOR, DELEGATOR is shown to be in co-NP, and BOUNDED-DELEGATOR is shown to be in PSPACE.

We study these problems here for the case of general NFAs. Our main contributions are as follows. We provide an algorithm that decides k-DELEGATOR in time polynomial in the number of states of a given NFA (and computes a k-lookahead delegator, if one exists) where k and the input alphabet are fixed. This generalizes [9, Theorem 2] from unambiguous to arbitrary NFAs.[1] As a main tool, we use two person games on graphs. This formulation of the problem yields a simple algorithm and correctness proof.

We furthermore show that the more general problem DELEGATOR, where the allowed lookahead k is a part of the input, is PSPACE-complete. The algorithm that runs in polynomial space is different from the one based on games for a fixed k, which is exponential in k and thus doubly exponential in the binary representation of k.

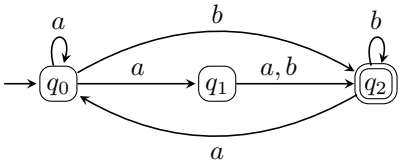
Finally, we prove that if an NFA $\mathcal{A}$ has a k-lookahead delegator for some k, then it also has a K-lookahead delegator for a bound K that is singly exponential in the size of $\mathcal{A}$. This shows the decidability of the problem BOUNDED-DELEGATOR which was left open in [9] and in [4]. In combination with the result for DELEGATOR, we also obtain PSPACE-completeness of this problem.

The remainder of the paper is structured as follows. Section 2 introduces basic terminology and results for later use. Then, in Sections 3, 4 and 5, we present solutions to the three problems k-DELEGATOR, DELEGATOR, and BOUNDED-DELEGATOR, respectively.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, \ldots\}$ be the set of non-negative integers. By $|S|$, we denote the cardinality of a set S. An *alphabet* Σ is a finite set of symbols. For $i \in \mathbb{N}$, Σ^i denotes the set of all sequences of Σ-symbols of length i. An element $w \in \Sigma^i$ is called a *word* and $|w| = i$ is its length. The *empty word*, denoted by ε, is the unique word of length $|\varepsilon| = 0$. Let $\Sigma^{\leq k} = \bigcup_{i=0}^{k} \Sigma^i$ for $k \in \mathbb{N}$ and $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$. A subset $L \subseteq \Sigma^*$ is called a *language*. For languages $L_1, L_2 \subseteq \Sigma^*$, let $L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$ be their concatenation.

[1] We note here that this corrects an error of [9, Theorem 5], which states that the problem is PSPACE-hard for arbitrary NFAs. The proof uses a reduction from an inclusion problem of the form $L(\mathcal{A}) \subseteq L_0$ for a fixed language L_0 and an NFA $\mathcal{A}$. However, this problem is not PSPACE-hard.



(a) NFA $\mathcal{A}$ with accepting states $F = \{q_2\}$

w	$(aw)^{-1}L_0$	$w^{-1}L_0$	$w^{-1}L_1$
ε	$L_0 \cup L_1$	L_0	L_1
a	$L_0 \cup L_1 \cup L_2$	$L_0 \cup L_1$	L_2
aa	$L_0 \cup L_1 \cup L_2$	$\underline{L_0 \cup L_1 \cup L_2}$	L_0
ab	L_2	$\underline{L_2}$	$\underline{L_2}$
b	L_2	$\underline{L_2}$	$\underline{L_2}$

(b) Left quotients of $\mathcal{A}$ (correct choices of the successor according to Lemma 3 are underlined)

Figure 1 NFA and its left quotients.

Automata. A *nondeterministic finite automaton* (*NFA* for short) $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ consists of a finite set of states Q, an alphabet Σ, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and an initial state $q_0 \in Q$ as well as a set $F \subseteq Q$ of accepting states. For a given input word $w \in \Sigma^*$, a *run* starting in q_0 is a sequence $q_0 q_1 \cdots q_{|w|}$ of states such that $(q_{i-1}, a_i, q_i) \in \Delta$ for each $i \in \{1, \ldots, |w|\}$ where $w = a_1 \cdots a_{|w|}$. We say that $\mathcal{A}$ *accepts* w if it has an *accepting* run from q_0, i.e., with last state accepting: $q_{|w|} \in F$. The language $L(\mathcal{A}) \subseteq \Sigma^*$ consists of those words that are accepted by $\mathcal{A}$.

For the rest of the paper, let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ be an NFA. We assume that each state $q \in Q$ is reachable from q_0 and that each state has at least one outgoing transition for each letter (missing transitions can be added to a sink state).

Lookahead Delegators. A k-delegator has to choose a transition from Δ deterministically by looking ahead on the next k input symbols. It is further required to still accept $L(\mathcal{A})$.

▶ **Definition 1.** For $k \in \mathbb{N}$, a *k-lookahead delegator for $\mathcal{A}$* (or *k-delegator* for short) is a function $f : Q \times \Sigma\Sigma^{\leq k} \to Q$ such that

a) $\big(q, a, f(q, aw)\big) \in \Delta$ for each $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^{\leq k}$, and
b) $f^*(q_0, w) \in F$ for each $w \in L(\mathcal{A})$, where we define $f^* : Q \times \Sigma^* \to Q$ as follows: for $w = a_0 \cdots a_{|w|}$, let $q_i = f(q_{i-1}, a_i \cdots a_{\min(|w|, i+k)})$ for $0 < i \leq |w|$, and let $f^*(q_0, w) = q_{|w|}$.

Note that our notion of k-lookahead follows [4, 3] by counting the additional lookahead whereas in [9], this is understood as $(k+1)$-lookahead as they count the current input symbol as a part of the lookahead. Hence, in the setting of Definition 1, a 0-lookahead delegator for an NFA can be identified with a deterministic subset of the transitions such that the same language is accepted. We say that $\mathcal{A}$ *has a bounded lookahead delegator* if it has a k-lookahead delegator for some $k \in \mathbb{N}$.

▶ **Example 2.** For the alphabet $\Sigma = \{a, b\}$, consider the NFA $\mathcal{A}$ depicted in Figure 1a which accepts the language $L(\mathcal{A}) = \{aa\} \cup \Sigma^*\{b, aaa\}$. The only nondeterministic choice of a transition is at state q_0 for symbol a. A function $f : Q \times \Sigma\Sigma^{\leq 2} \to Q$ is a 2-delegator for $\mathcal{A}$ if $f(q_0, aa) = q_1$ and $f(q_0, aaa) = q_0$. For all other cases, it suffices that f is consistent with Δ, i.e., $\big(q_0, a, f(q_0, aw)\big) \in \Delta$ for $w \in \Sigma^{\leq 2}$. Example 4 will justify why this yields a 2-delegator.

Left Quotients. For $u \in \Sigma^*$ and $L \subseteq \Sigma^*$, let $u^{-1}L = \{v \in \Sigma^* \mid uv \in L\}$ be the *left quotient* of u with L. This is the language containing each word that completes u to a word in L. Note that the composition of left quotients can be written as concatenation: $v^{-1}(u^{-1}L) = (uv)^{-1}L$. Further, we abbreviate $L = L(\mathcal{A})$ and $L_q = L(\mathcal{A}_q)$ for the language of $\mathcal{A}$ where q is taken as the initial state.

We now present a technical lemma that occurs as a key ingredient in the proofs and algorithms of the following three sections. A similar statement can be found in [9, Lemma 7] using a notion called blindness. It gives a language-theoretical characterization of the main property a k-delegator has to fulfill when it selects a transition. That is, whenever an a-successor p has to be chosen for some lookahead w and state q, then $(aw)^{-1}L_q = w^{-1}L_p$, i.e., $awv \in L_q$ iff $wv \in L_p$ for all $v \in \Sigma^*$. Note that the inclusion $(aw)^{-1}L_q \supseteq w^{-1}L_p$ holds generally, since $(q, a, p) \in \Delta$.

▶ **Lemma 3.** *$\mathcal{A}$ has a k-delegator iff there exists a set $Q' \subseteq Q$ such that $q_0 \in Q'$ and for each $q \in Q'$, $a \in \Sigma$, $w \in \Sigma^k$, there exists $p \in Q'$ such that $(q, a, p) \in \Delta$ and $(aw)^{-1}L_q = w^{-1}L_p$ hold.*

Proof. For the direction from left to right, let f be a k-delegator and Q' be the set of states reachable by f from q_0 with full lookahead, i.e., the smallest set such that $q_0 \in Q'$ and $f(q, aw) \in Q'$ for each $q \in Q'$, $a \in \Sigma$, and $w \in \Sigma^k$. For a contradiction, assume there are $q \in Q'$, $a \in \Sigma$, and $w \in \Sigma^k$ such that $(aw)^{-1}L_q \neq w^{-1}L_p$ for all $p \in Q'$ with $(q, a, p) \in \Delta$. Since q is reachable from q_0, fix a word $u \in \Sigma^*$ such that f leads to q and assume w.l.o.g. that q is the first state on that run with the above property. Let $p = f(q, aw)$. Then, there is a word $v \in (aw)^{-1}L_q \setminus w^{-1}L_p$, i.e., $awv \in L_q$ but $wv \notin L_p$. Hence, we have that $f^*(q_0, uawv) = f^*(q, awv) = f^*(p, wv) \notin F$ whereas $uawv \in L$ in contradiction to the definition of a k-delegator.

For the other direction, we construct a k-delegator f from a set Q' with the above properties. For each $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^{\leq k}$, we set $f(q, aw) = p$ as follows.

a) If $|w| = k$ and $q \in Q'$, then $p \in Q'$ becomes some state such that $(q, a, p) \in \Delta$ and $(aw)^{-1}L_q = w^{-1}L_p$ as directly guaranteed by the property.
b) If $|w| < k$ and $aw \in L_q$, then there is an a-successor $p \in Q$ of q such that $w \in L_p$.
c) If $|w| < k$ and $aw \notin L_q$, then fix some arbitrary a-successor $p \in Q$ of q.

Note that the case $|w| = k$ and $q \notin Q'$ cannot occur due to the above property. It easily follows by the definition of f^* that $f^*(q_0, v) \in F$ if $v \in L$. Hence, f is a k-delegator for $\mathcal{A}$. ◀

▶ **Example 4.** Let us reconsider the NFA $\mathcal{A}$ of Example 2 from the perspective of Lemma 3. Since q_0 is the initial state, a set Q' satisfying the property of Lemma 3 must contain q_0. The only nondeterministic choice happens at q_0 with letter a. The two a-successors of q_0 are q_0 itself and q_1. The left quotients that are relevant for Lemma 3 are listed in Figure 1b for some example words w, where L_i stands for L_{q_i}. For $w = \varepsilon$, one can see that neither the language $w^{-1}L_0$ nor $w^{-1}L_1$ is equivalent to $(aw)^{-1}L_0$ which indicates, according to Lemma 3, that there is no 0-delegator for $\mathcal{A}$. The row for $w = a$ shows analogously that there is also no 1-delegator. When the lookahead is increased to 2, then the condition of Lemma 3 is finally fulfilled. For $w = aa$, the left quotients are the same if and only if a delegator chooses q_0 as a-successor of q_0. The remaining two lines show that the choice does not matter for $w \in \{ab, b\}$.

3 Fixed Lookahead

In this section, we present for an arbitrary fixed number $k \in \mathbb{N}$ an algorithm for the problem k-Delegator, which is to decide the existence of a k-delegator for a given NFA $\mathcal{A}$ (and to compute such a delegator if it exists). The special case 0-Delegator corresponds to deciding whether the NFA $\mathcal{A}$ can be turned into an equivalent DFA just by removing transitions of the NFA. The polynomial time decidability of this very problem has already been mentioned in the survey article [2, Theorem 15] without a proof.

The rough idea behind our approach is to construct a game that simulates the delegation. The two players play a sequence of actions in alternation. One player has to choose the letters of an input word while the other has to choose appropriate transitions. The goal of the player in charge of the transitions is to play an accepting run if a word contained in the language $L(\mathcal{A})$ is formed by the player in charge of the input.

Before we proceed with the detailed definition of this game, we first introduce some terminology concerning 2-player games which follows [5].

Games. Formally, a *safety game* $\mathcal{G} = (V, V_0, E, S)$ is played between Player 0 and Player 1, and consists of

a) a finite directed graph (V, E),
b) a partition of the vertices into sets $V_0 \subseteq V$ for Player 0 and $V_1 = V \setminus V_0$ for Player 1, and
c) a set $S \subseteq V$ of safe vertices.

A *play* in $\mathcal{G}$ is a finite or infinite sequence $v_0 v_1 v_2 \cdots$ of vertices. It starts in some initial vertex $v_0 \in V$ and then, for all v_i, there has to be an edge $(v_i, v_{i+1}) \in E$ to its successor v_{i+1} which is chosen by Player 0 if $v_i \in V_0$ or by Player 1 if $v_i \in V_1$. Then, the safety condition states that the play is *won* by Player 0 if all vertices are safe: $v_i \in S$ for all i. Otherwise Player 1 wins the play. We note here that usually a player that cannot move (because there are no outgoing edges) loses. For our purpose, the above condition is more natural, that is, if a maximal play is finite because it ends in a vertex without outgoing edges, then Player 0 wins if all vertices of the play are safe, independent of whether the last vertex is in V_0 or in V_1. One can obtain the standard setting by adding self-loops to terminal vertices of Player 0.

For $\sigma \in \{0, 1\}$, let $V'_\sigma = \{v \in V_\sigma \mid (v, v') \in E \text{ for some } v' \in V\}$ be the set of non-terminal vertices of Player σ. A *strategy for Player* σ is a function $s : V^* V'_\sigma \to V$ that chooses a successor vertex $v_{i+1} = s(v_0 \cdots v_i)$ with $(v_i, v_{i+1}) \in E$ for each finite play $v_0 \cdots v_i \in V^* V'_\sigma$ ending in a non-terminal vertex v_i of Player σ. We say that s is *winning from a vertex* v_0 if Player σ wins every maximal play $v_0 v_1 v_2 \cdots$ resulting from this strategy, i.e., with $v_{i+1} = s(v_0 \cdots v_i)$ for every $v_i \in V'_\sigma$. Finally, we say that Player σ can *win* $\mathcal{G}$ *from a vertex* v_0 if he has a winning strategy from there.

Safety games are *determined*, i.e., either Player 0 or Player 1 has a winning strategy (see [5]). Further, the respective Player σ can always win with a *positional* strategy s, which means that the choice of the next move only depends on the current vertex, i.e., $s(v_0 \cdots v_i) = s(u_0 \cdots u_j)$ holds for any two finite plays $v_0 \cdots v_i, u_0 \cdots u_j \in V^* V'_\sigma$ with $v_i = u_j$. We then consider a strategy as a function $s : V'_\sigma \to V$.

Delegation Game. Now, we can define our game $\mathcal{G}_{\mathcal{A},k}$ according to the main idea explained at the beginning of this section, where Player 1 has to give the input word and Player 0 has to choose transitions. The positions of the game store the corresponding information: a lookahead of k letters (or less if the play goes towards the end of the input word) provided by Player 1, and the current state of $\mathcal{A}$ reached by Player 0. The goal of this construction is to show later in Lemma 7 that $\mathcal{A}$ has a k-delegator if and only if Player 0 has a winning strategy. Since this strategy will occur to be positional, we can directly use it as a function that deterministically chooses transitions, i.e., as a delegator.

The main challenge is now to create a safety game the number of states of which is polynomial in the number of automaton states. The winning condition should express that the state of Player 0 is accepting if the input word played by Player 1 is in the language, which depends not on the current position alone but on the whole play instead. Naïvely, one can implement this as a safety condition by additionally keeping track of the set of reachable

states by the input played by Player 1. However, this leads to a blowup that is exponential in the size of the automaton.

To solve this problem, we modify our game in such a way that Player 1 also has to choose a transition for each input, but after Player 0 has chosen one. We show that, since Player 0 has to make the choice of the transition first, the additional information on the transition chosen by Player 1 does not help Player 0 (because basically, Player 0 has to choose a transition according to Lemma 3, which only depends on the current state of Player 0). This means that in this modified game, a winning strategy for Player 0 still corresponds to a k-delegator.

To summarize, the game $\mathcal{G}_{\mathcal{A},k}$ goes as follows. First, Player 1 gives the initial content of the lookahead. Then, both players play in alternation. Player 0 chooses a transition for the next input letter. Afterwards, Player 1 also chooses a transition for it and simultaneously removes this input symbol (as both players have just processed it) and appends a new letter to the content of the lookahead, or he does not refill it if the input word should end. Consequently, a game position encodes the content of the lookahead as well as one state for each player. The safety condition for Player 0 now simply states that such vertices have to be avoided, where the state of Player 1 is accepting and the state of Player 0 is non-accepting although the lookahead is empty.

▶ **Definition 5.** Given an NFA $\mathcal{A}$ and $k \in \mathbb{N}$, we define the two player safety game $\mathcal{G}_{\mathcal{A},k} = (V, V_0, E, S)$ as follows:

a) $V = \{\top\} \cup \Big(\{0,1\} \times \Sigma^{\leq k+1} \times Q \times Q\Big)$, (initial vertex and simulation vertices)

b) $V_0 = \Big(\{0\} \times \Sigma^{\leq k+1} \times Q \times Q\Big)$,

c) $E \subseteq V \times V$ containing the following edges:

i) $\Big(\top, (0, w, q_0, q_0)\Big)$ for $w \in \Sigma^{\leq k+1}$, (initiate buffer)

ii) $\Big((0, aw, q, p), (1, aw, q', p)\Big)$ for $(q, a, q') \in \Delta$ and $w \in \Sigma^{\leq k}$, (Player 0 applying transition)

iii) $\Big((1, aw, q', p), (0, wb, q', p')\Big)$ for $(p, a, p') \in \Delta$, $w \in \Sigma^k$, and $a, b \in \Sigma$, (Player 1 applying transition, removing leftmost symbol, and refilling lookahead)

iv) $\Big((1, aw, q', p), (0, w, q', p')\Big)$ for $(p, a, p') \in \Delta$, $w \in \Sigma^{\leq k}$, and $a \in \Sigma$, (Player 1 applying transition and removing leftmost symbol without refilling)

d) $S = V \setminus \Big\{(0, \varepsilon, q, p) \;\Big|\; q \notin F \wedge p \in F\Big\}$. (Player 0 has to avoid $\overline{F} \times F$)

The number of vertices of $\mathcal{G}_{\mathcal{A},k}$ is in $\mathcal{O}\big((k+1) \cdot |\Sigma|^{k+1} \cdot |Q|^2\big)$ which is polynomial in $|Q|$ and $|\Sigma|$ for a fixed k. It is easy to see that the game can be constructed in time polynomial in the number of vertices.

▶ **Example 6.** Let us reconsider the NFA $\mathcal{A}$ of Example 2. A part of the 1-delegator game $\mathcal{G}_{\mathcal{A},1}$, that is reachable from the initial vertex $\top$, is depicted in Figure 2, in such a way that for Player 1, only one edge is enabled from each vertex whereas for Player 0, all successors are considered. This deterministic choice corresponds to a positional strategy for Player 1. The strategy always forces the play to an unsafe vertex no matter how Player 0 reacts. Hence, it is a positional winning strategy for Player 1 in $\mathcal{G}_{\mathcal{A},1}$ from $\top$. We show next that this is because there exists no 1-delegator for $\mathcal{A}$ (cf. Example 4).

▶ **Lemma 7.** *Player 0 has a positional winning strategy for $\mathcal{G}_{\mathcal{A},k}$ from $\top$ iff $\mathcal{A}$ has a k-lookahead delegator.*

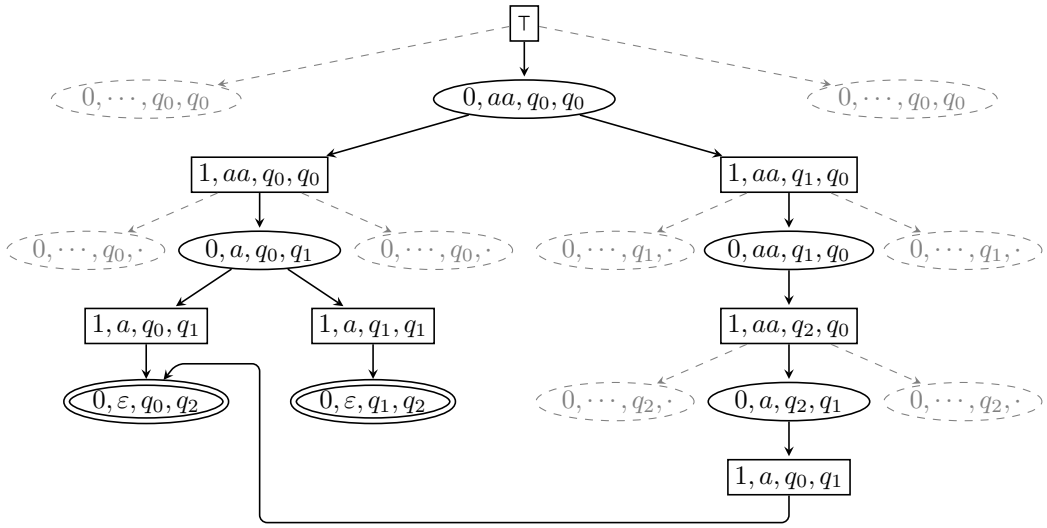


Figure 2 The (partial) 1-delegator game $\mathcal{G}_{\mathcal{A},1}$ showing a positional winning strategy for Player 1 (circled vertices belong to Player 0, boxed ones to Player 1, and doubly circled vertices are unsafe).

Proof. The key observation is that for (a strategy of) Player 0, it is not important to known which states Player 1 chooses since Lemma 3 states that left quotients are important rather than exact states. As long as the property is fulfilled locally, Player 1 can dually be assumed w.l.o.g. to just copy the choices of his opponent.

For the left to right direction, suppose Player 0 has a positional winning strategy s. We show the existence of a k-delegator for $\mathcal{A}$ by proving that the condition from Lemma 3 is satisfied. For this purpose, let Q' be the set consisting of all states $q \in Q$ such that a vertex of the form $(0, aw, q, q)$, with $a \in \Sigma$ and $w \in \Sigma^k$, can be reached with s for some sequence of moves of Player 1. Since trivially $q_0 \in Q'$, it remains to show that for each such vertex, there is a successor $(1, aw, p, q)$ with $(aw)^{-1}L_q = w^{-1}L_p$. Assume, to the contrary, that $(aw)^{-1}L_q \neq w^{-1}L_p$ for each $p \in Q$ with $(q, a, p) \in \Delta$. Then, no matter which p is chosen by Player 0, there is a word $v \in (aw)^{-1}L_q \setminus w^{-1}L_p$, i.e., $awv \in L_q$ but $wv \notin L_p$. Player 1 can win from $(1, aw, p, q)$ by continuing to play the word v since there is a sequence of states that accepts awv from q but none that accepts wv from p. This contradicts the property that s is a winning strategy for Player 0.

For the other direction, suppose $\mathcal{A}$ has a k-delegator f. We can naturally use it to define a positional winning strategy $s : V_0' \to V$ for Player 0 where $s(0, aw, q, p) = \big(1, aw, f(q, aw), p\big)$. One can easily see by the construction of $\mathcal{G}_{\mathcal{A},k}$ and s that $q = f^*(q_0, x)$ holds whenever a terminal vertex $(0, \varepsilon, q, p)$ is reached after Player 1 has played a complete word $x \in \Sigma^*$. Player 0 wins because $p \in F$ implies $x \in L$ and hence, $q \in F$. ◀

By combining Lemma 7 with the linear-time determinacy of safety games (see [5, Theorem 4.1]), we get the main result of this section. Linear-time determinacy means that for the winning player, there exists a positional winning strategy which can be computed in linear time (using a fixed-point algorithm that successively identifies the vertices from which Player 0 is losing, starting with the set of unsafe vertices).

Consequently, the existence of a k-lookahead delegator for $\mathcal{A}$ can be decided in time $\mathcal{O}\big((k+1) \cdot |\Sigma|^{k+1} \cdot |Q|^2\big)$ for a given NFA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ and a positive number k. This yields polynomial running time for a fixed k that we consider in this section.

▶ **Corollary 8.** *For each $k \in \mathbb{N}$, the problem k-*Delegator *can be solved in polynomial time.*

This generalizes [9, Theorem 2] where polynomial time decidability of k-Delegator for each fixed k is shown for unambiguous NFA.

As explained in Section 1, we consider the input to be a single NFA whereas in the original motivation, the input consists of several DFAs. It is shown in [7] that the problem 0-Delegator is EXPTIME-complete in the original setting. This is caused by the fact that the construction of a product of the DFAs yields an NFA that is exponentially larger.

4 Given Lookahead

We now consider the complexity of the problem Delegator, where an NFA and a bound k are given. Note that for deciding whether $\mathcal{A}$ has a k-lookahead delegator, the game-based approach from the previous section yields an algorithm that runs in time that is doubly exponential in the binary representation of k. However, using a different algorithm, we can show that the problem can be solved in polynomial space. The idea of the algorithm is to check whether the property of Lemma 3 holds. The main problem in checking this condition in polynomial space is that we cannot enumerate all words $w \in \Sigma^k$ because their length is exponential in the binary representation of k.

We therefore first introduce transition profiles, which can be used to circumvent this problem. Intuitively, a transition profile of a word w for a given NFA $\mathcal{A}$ describes the possible state transformations induced by w on $\mathcal{A}$, that is, the transition profile for w contains all pairs of states (p, q) such that there is a w-labeled path from p to q.

▶ **Definition 9.** For an NFA $\mathcal{A}$ and a word $w \in \Sigma^*$, we define the *transition profile* $\Delta_w \subseteq Q^2$:

$$
\begin{aligned}
&(q,q) \in \Delta_\varepsilon && \text{for each } q \in Q,\\
&(q,p) \in \Delta_a \;\Leftrightarrow\; (q,a,p) \in \Delta && \text{for each } q,p \in Q,\ a \in \Sigma,\\
&(q,p) \in \Delta_{wa} \;\Leftrightarrow\; \exists r \in Q : (q,r) \in \Delta_w \text{ and } (r,p) \in \Delta_a && \text{for each } q,p \in Q,\ a \in \Sigma,\ w \in \Sigma^*.
\end{aligned}
$$

The main idea for checking the condition of Lemma 3 in polynomial space is to use transition profiles that are induced by words of length k, instead of working directly with the words. This is justified by the simple observation that words with the same profile also have the same left quotient.

▶ **Lemma 10.** *Let $x, y \in \Sigma^*$ be such that $\Delta_x = \Delta_y$. Then, $x^{-1}L_q = y^{-1}L_q$ for all $q \in Q$.*

Proof. A trivial consequence of Definition 9 is that $\Delta_{xw} = \Delta_{yw}$ for all $w \in \Sigma^*$. Then,

$$
\begin{aligned}
w \in x^{-1}L_q \;\Leftrightarrow\; xw \in L_q \;&\Leftrightarrow\; \exists p \in F : (q,p) \in \Delta_{xw}\\
&\Leftrightarrow\; \exists p \in F : (q,p) \in \Delta_{yw} \;\Leftrightarrow\; yw \in L_q \;\Leftrightarrow\; w \in y^{-1}L_q.
\end{aligned}
$$

◀

▶ **Theorem 11.** *The problem* Delegator *is in PSPACE.*

Proof. Let an NFA $\mathcal{A}$ (with the usual components) and k be given. We show that for each $Q' \subseteq Q$, there is a nondeterministic PSPACE algorithm that checks whether the property of Lemma 3 is satisfied. Savitch's theorem (see [8, Theorem 7.5]) implies that there is also a deterministic PSPACE algorithm.

So let $Q' \subseteq Q$ with $q_0 \in Q$. The algorithm tests for each q and each a, whether for each word $w \in \Sigma^k$ there is an a-successor p of q such that $(aw)^{-1}L_q = w^{-1}L_p$. As mentioned above, we cannot enumerate all words $w \in \Sigma^k$ because their length is exponential in the binary

representation of k. Instead, we work with the transition profiles induced by the words w. Each such transition profile is of size polynomial in $\mathcal{A}$ and contains sufficient information to test $(aw)^{-1}L_q = w^{-1}L_p$. Lemma 10 allows us to restrict the test to transition profiles, as words with the same transition profile induce the same left quotient.

We now describe the algorithm. Given Q', $q \in Q'$, and $a \in \Sigma$, the algorithm proceeds as follows. For each transition profile $\tau \in 2^{Q \times Q}$:

a) Check if $\tau = \Delta_w$ for some word w of length k. If it is the case, do the following. Otherwise, move on to the next transition profile.
b) Let $p_1, \ldots, p_n$ be the a-successors of q in Q'. For $i \in \{1, \ldots, n\}$, let $R_i = \{p \in Q \mid (p_i, p) \in \tau\}$ be the set of states that are reached from p_i in the profile τ. Let $R = \bigcup_{1 \le i \le n} R_i$. Note that $L_{R_i} = w^{-1}L_{p_i}$ and $L_R = (aw)^{-1}L_q$, where for $S \subseteq Q$, we let $L_S = \bigcup_{s \in S} L_s$.
c) Check if there is an index $i \in \{1, \ldots, n\}$ such that $L_R = L_{R_i}$.

If the last test fails (meaning that there is no such index i), then Q' does not satisfy the property of Lemma 3. If the test passes for all q, all a, and all the relevant transition profiles (those passing the first test), then Q' has the desired property and thus, $\mathcal{A}$ has a k-lookahead delegator.

It remains to verify that the steps of the algorithm can be carried out in polynomial space. The first test uses the idea of checking reachability in a directed graph in logarithmic space. In our setting, we use a counter for counting up to k (note that the number of bits needed for the counter corresponds to the size of the binary representation of k), and successively guess k steps to reach the transition profile τ. That is, we start with the transition profile Δ_ε of the empty word. In each step, we guess a letter $b \in \Sigma$ and extend the current transition profile Δ_v to Δ_{vb}. After k steps, we check whether the resulting profile Δ_w is equal to τ. At each moment, we only need to store the counter and the intermediate transition profile, which requires polynomial space.

The second step just computes (in LOGSPACE) some sets from the transition profile τ.

The third step requires us to test n equivalences $L_R = L_{R_i}$, where the languages are given by NFAs with the sets R and R_i as initial states, respectively. Since equivalence of NFAs can be tested in polynomial space (see [1]), this step is also in PSPACE. ◀

▶ **Theorem 12.** *The problem* DELEGATOR *is PSPACE-complete.*

Proof. The upper bound follows from Theorem 11.

For the lower bound, let M be some polynomially space bounded Turing machine that solves a PSPACE-hard problem. We show that the word problem for M can be reduced to the problem of the existence of a bounded lookahead delegator. The word problem for M is to decide for a given word whether M accepts w, which is clearly PSPACE-hard because M solves a PSPACE-hard problem.

Let h be the polynomial for the space bound of M. Given a word w, we construct an NFA $\mathcal{A}$ that has a $\big(2h(n) + 2\big)$-lookahead delegator iff M rejects w, where $n = |w|$.

As usual, we encode configurations of M by words of the form $\kappa = usv$, where uv is the content of the tape of M, and s the current control state. The head of M in configuration usv is on the first position of v. We can assume that $|uv| = h(n)$. A computation of M is then encoded by a word of the form $\#\kappa_0\#\kappa_1\#\cdots\kappa_\ell\#$, where κ_0 is the initial configuration of M on w, each κ_{i+1} is the successor configuration of κ_i, and κ_ℓ is an accepting configuration.

The core of the reduction is an NFA $\mathcal{A}_w$ that accepts a word if it does *not* encode an accepting computation of M on w (see [1, Lemma 10.2] for such a construction for regular expressions instead of NFAs). For this purpose, $\mathcal{A}_w$ uses a product of automata testing the following properties:

a) The word is not of the required form $\#\kappa_0\#\kappa_1\#\cdots\kappa_\ell\#$ where each κ_i is of the form $u_i s_i v_i$ with $|u_i v_i| = h(n)$.
b) The first configuration is not the initial configuration of M on w.
c) The last configuration is not an accepting configuration.
d) There is an i such that κ_{i+1} is not the M-successor configuration of κ_i.

The first three properties can be easily checked by DFAs of size linear in $h(n)$. The last property can be checked by an NFA that guesses at some symbol # that this corresponds to the index i, and then guesses a position j in κ_i and tests whether κ_{i+1} has been updated in a wrong way at position j (to detect this, the three symbols at positions $j-1$, j, and $j+1$ are sufficient). The size of such an NFA is also linear in $h(n)$ (it needs to count up to $h(n)$ for finding the corresponding cell j in κ_{i+1}). All the automata can be constructed in logarithmic space from M and w. The automaton $\mathcal{A}_w$ is the product of these four automata that accepts if one of its components accepts. Note that $\mathcal{A}_w$ has a $\big(2h(n)+2\big)$-lookahead delegator because it is sufficient to know the next two configurations to decide which transition to take in the NFA for the last property.

Further, note that $\mathcal{A}_w$ accepts all words if there is no accepting computation of M on w. And if there is such an accepting computation, then $\mathcal{A}_w$ does not accept the word encoding this computation.

We now embed $\mathcal{A}_w$ into an NFA $\mathcal{A}$ to obtain the desired reduction. Let Σ be the alphabet of $\mathcal{A}_w$, and let X, Y, Z be new letters. Define the languages

$$L_1 := X^* \cdot L(\mathcal{A}_w) \cdot \{Y, Z\} \text{ and } L_2 := X^* \cdot \Sigma^* \cdot \{Y\}.$$

Note that $L_2 \subseteq L_1$ iff $L(\mathcal{A}_w) = \Sigma^*$ iff M rejects w.

We construct $\mathcal{A}$ to accept the language $X \cdot (L_1 \cup L_2)$. For this purpose, $\mathcal{A}$ nondeterministically chooses from its initial state on the first X to either go to an automaton $\mathcal{A}_1$ for L_1 or to an automaton $\mathcal{A}_2$ for L_2. The automaton $\mathcal{A}_1$ is a simple extension of $\mathcal{A}_w$ by an X-loop at the beginning and transitions for processing the last Y or Z. The automaton $\mathcal{A}_2$ just consists of an X-loop, followed by a Σ-loop, followed by a transition for Y into an accepting state.

Now assume that M rejects w. Then, $L_2 \subseteq L_1$, as noted above, and a lookahead delegator for $\mathcal{A}$ can always choose the transition going to $\mathcal{A}_1$ from the initial state. We already noted that $\mathcal{A}_w$ has a $\big(2h(n)+2\big)$-lookahead delegator. Overall, we obtain a $\big(2h(n)+2\big)$-lookahead delegator for $\mathcal{A}$ in this case.

Now assume that M accepts w and that $\mathcal{A}$ has a k-lookahead delegator f for some k. Consider the decision of f on the lookahead word X^k. If f moves to $\mathcal{A}_1$, then pick the word v encoding the accepting computation of M on w, followed by the letter Y. $\mathcal{A}_1$ does not accept this word and therefore, f cannot be a k-lookahead delegator because $X^k v Y \in L$.

If f moves to $\mathcal{A}_2$, then consider any word v accepted by $\mathcal{A}_w$ followed by Z. Then, $X^k v Z \in L_1$ but $\mathcal{A}_2$ only accepts words ending with Y. Hence, also in this case, f cannot be a k-lookahead delegator.

This shows that $\mathcal{A}$ has a k-lookahead delegator for some k iff M rejects w. Furthermore, k can be chosen as $2h(n)+2$. ◀

We note that in [9, Theorem 3], it is shown that the problem DELEGATOR for unambiguous NFA is contained in co-NP.

5 Bounded Lookahead

We now turn to the problem of deciding, given an NFA $\mathcal{A}$, whether there exists a k such that $\mathcal{A}$ has a k-delegator, that is, we consider the problem BOUNDED-DELEGATOR. We show

that if $\mathcal{A}$ has some k-delegator, then $\mathcal{A}$ also has some K-delegator for a number K that is singly exponential in the size of $\mathcal{A}$. In the combination with the results from Section 4, we then obtain that Bounded-Delegator is PSPACE-complete, too.

The proof showing that there is this bound K uses a technique inspired by [6], where two player games with lookahead for one of the players are considered. In our setting, the main idea is the following. If the lookahead K is big enough, then each word that can occur as lookahead contains an infix that can be pumped such that the considered lookahead word can be extended to a word of length k (with k and K as explained above where we assume w.l.o.g. $k > K$). On this longer lookahead word of length k, one can query the existing delegator and use the same decision for a delegator with the smaller lookahead K.

The required pumping argument that we just mentioned is formalized by an extension of Lemma 10 where we use transition profiles (cf. Definition 9) again.

▶ **Lemma 13.** *Let $x, y, z \in \Sigma^*$ be such that $\Delta_x = \Delta_{xy}$. Then, $(xyz)^{-1}L_q = (xy^iz)^{-1}L_q$ for all $q \in Q$ and $i \in \mathbb{N}$.*

Proof. An easy induction shows that $\Delta_{xy} = \Delta_{xy^i}$ for all $i \in \mathbb{N}$. Consequently, $\Delta_{xyz} = \Delta_{xy^iz}$ holds and the claim follows directly by Lemma 10. ◀

Using a simple counting argument, we can show that each word of a certain length has a decomposition xyz as in Lemma 13.

▶ **Lemma 14.** *For the bound $K = 2^{|Q|^2}$, each word $w \in \Sigma^K$ can be decomposed as $w = xyz$ with $y \neq \varepsilon$ and $\Delta_x = \Delta_{xy}$.*

Proof. A word of length $2^{|Q|^2}$ has $2^{|Q|^2} + 1$ prefixes. Two different prefixes must have the same transition profile since there are at most $2^{|Q|^2}$ transition profiles. This implies the existence of the claimed decomposition. ◀

We now combine Lemma 13 and Lemma 14 to prove that the bound $K = 2^{|Q|^2}$ is the maximal "useful" lookahead.

▶ **Theorem 15.** *$\mathcal{A}$ has a K-lookahead delegator if it has a bounded lookahead delegator.*

Proof. Let f be a k-delegator for $\mathcal{A}$ where $k > K$ w.l.o.g. We show that the property on the right hand side of Lemma 3, which holds for k by assumption, also holds for K for the same set $Q' \subseteq Q$. To this end, we have to show that for every $q \in Q'$, $a \in \Sigma$, and $w \in \Sigma^K$, there is some $p \in Q'$ with $(q, a, p) \in \Delta$ and $(aw)^{-1}L_q = w^{-1}L_p$. We know that there is a decomposition $w = xyz$ with $y \neq \varepsilon$ and $\Delta_x = \Delta_{xy}$. Choose $i \in \mathbb{N}$ and a proper prefix y' of y such that $|xy^iy'| = k$. Let y'' be such that $y = y'y''$. Finally, we pick some $p \in Q'$ with $(q, a, p) \in \Delta$ such that $(axy^iy')^{-1}L_q = (xy^iy')^{-1}L_p$ the existence of which is guaranteed by Lemma 3 for k-lookahead. With Lemma 13, we can now show that the desired property holds for K-lookahead:

$$
\begin{aligned}
(axyz)^{-1}L_q &= (axy^{i+1}z)^{-1}L_q && \text{(Lemma 13)}\\
&= (axy^iy'y''z)^{-1}L_q && (y = y'y'')\\
&= (y''z)^{-1}\big((axy^iy')^{-1}L_q\big) && \text{(composition of left quotients)}\\
&= (y''z)^{-1}\big((xy^iy')^{-1}L_p\big) && \text{(Lemma 3 for } k\text{-lookahead)}\\
&= (xy^iy'y''z)^{-1}L_p && \text{(composition of left quotients)}\\
&= (xy^{i+1}z)^{-1}L_p && (y = y'y'')\\
&= (xyz)^{-1}L_p && \text{(Lemma 13)}
\end{aligned}
$$

◀

Since the bound K is singly exponential in the number of states of $\mathcal{A}$ and therefore has a binary representation that is polynomial in the size of $\mathcal{A}$, Theorem 11 implies that Bounded-Delegator also is in PSPACE. Furthermore, our reduction showing that Delegator is PSPACE-hard also shows that Bounded-Delegator is PSPACE-hard (see proof of Theorem 12).

▶ **Corollary 16.** *The problem* Bounded-Delegator *is PSPACE-complete.*

In [9, Theorem 4], it is shown that Bounded-Delegator is in PSPACE for unambiguous NFA. This result is generalized by Corollary 16. However, the NFA constructed in the proof of Theorem 12 for the PSPACE lower bound is ambiguous, in general, and therefore, our completeness result does not extend to unambiguous automata.

6 Conclusion

We have shown that the existence of a k-lookahead delegator for an NFA $\mathcal{A}$ can be decided in $\mathcal{O}\big((k+1)\cdot|\Sigma|^{k+1}\cdot|Q|^2\big)$, where Σ is the input alphabet, and Q the state set of the given NFA. In particular, for a fixed k, the problem can be solved in polynomial time. We have furthermore shown that the problem becomes PSPACE-complete if the bound is either a part of the input or no bound is given. This gives a complete picture for the complexities of the decision problems for lookahead delegators for NFA.

As further research, we would like to analyze whether our techniques can be extended to decide the existence of lookahead delegators for some classes of infinite state automata (like counter or pushdown automata), as it has been considered in [3].

Acknowledgements. With thanks to the anonymous reviewers for their detailed feedback.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, New York, 1974.
2 Thomas Colcombet. Forms of Determinism for Automata (Invited Talk). In *STACS 2012*, volume 14 of *LIPIcs*, pages 1–23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
3 Zhe Dang, Oscar H. Ibarra, and Jianwen Su. Composability of Infinite-State Activity Automata. In *ISAAC*, pages 377–388, 2004.
4 Cagdas Evren Gerede, Richard Hull, Oscar H. Ibarra, and Jianwen Su. Automated composition of e-services: lookaheads. In *ICSOC*, pages 252–262, 2004.
5 Erich Grädel. Back and Forth Between Logics and Games. In *Lectures in Game Theory for Computer Scientists*, pages 99–145. Springer, 2011.
6 Michael Holtmann, Lukasz Kaiser, and Wolfgang Thomas. Degrees of Lookahead in Regular Infinite Games. *Logical Methods in Computer Science*, 8(3), 2012.
7 Anca Muscholl and Igor Walukiewicz. A Lower Bound on Web Services Composition. *Logical Methods in Computer Science*, 4(2), 2008.
8 Christos H. Papadimitriou. *Complexity Theory*. Addison Wesley, 1994.
9 Bala Ravikumar and Nicolae Santean. On the Existence of Lookahead Delegators for NFA. *Int. J. Found. Comput. Sci.*, 18(5):949–973, 2007.

Fair Matchings and Related Problems *

Chien-Chung Huang[1], Telikepalli Kavitha[2], Kurt Mehlhorn[3], and Dimitrios Michail[4]

1 **Chalmers University, Sweden**
huangch@chalmers.se
2 **Tata Institute of Fundamental Research, India**
kavitha@tcs.tifr.res.in
3 **Max-Planck Institut für Informatik, Germany**
mehlhorn@mpi-inf.mpg.de
4 **Harokopio University of Athens, Greece**
michail@hua.gr

Abstract

Let $G = (A \cup B, E)$ be a bipartite graph, where every vertex ranks its neighbors in an order of preference (with ties allowed) and let r be the worst rank used. A matching M is *fair* in G if it has maximum cardinality, subject to this, M matches the minimum number of vertices to rank r neighbors, subject to that, M matches the minimum number of vertices to rank $(r-1)$ neighbors, and so on. We show an efficient combinatorial algorithm based on LP duality to compute a fair matching in G. We also show a scaling based algorithm for the fair *b-matching* problem.

Our two algorithms can be extended to solve other profile-based matching problems. In designing our combinatorial algorithm, we show how to solve a generalized version of the minimum weighted vertex cover problem in bipartite graphs, using a single-source shortest paths computation—this can be of independent interest.

1998 ACM Subject Classification F.2.2 Computations on discrete structures

Keywords and phrases Matching with Preferences, Fairness and Rank-Maximality, Bipartite Vertex Cover, Linear Programming Duality, Complementary Slackness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.339

1 Introduction

Let $G = (A \cup B, E)$ be a bipartite graph on n vertices and m edges, where each $u \in A \cup B$ has a list ranking its neighbors in an order of preference (ties are allowed). Such an instance is usually referred to a *stable marriage* instance with incomplete lists and ties. A matching is a collection of edges, no two of which share an endpoint.

The focus in stable marriage problems is to find matchings that are *stable* [6]. However, there are many applications where stability is not a proper objective: for instance, in matching students with counselors or applicants with training posts, we cannot compromise on the size of the matching and a *fair* matching is a natural candidate for an optimal matching in such problems.

▶ **Definition 1.** A matching M is fair in $G = (A \cup B, E)$ if M has maximum cardinality, subject to this, M matches the minimum number of vertices to rank r neighbors, and subject

* This work is based on two preprints [11, 17].

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 339–350

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

to that, M matches the minimum number of vertices to rank $(r-1)$ neighbors, and so on, where r is the worst rank used in the preference lists of vertices.

The fair matching problem can be solved in polynomial time as follows: for an edge e with incident ranks i and j, let $w(e) = n^{i-1} + n^{j-1}$. It is easy to see that a maximum cardinality matching of minimum weight (under weight function w) is a fair matching in G. Such a matching can be computed via the maximum weight matching algorithm by resetting e's weight to $4n^r - n^{i-1} - n^{j-1}$, where r is the largest rank used in any preference list.

However this approach can be expensive even if we use the fastest maximum-weight bipartite matching algorithms [1, 3, 4, 5]. The running time will be $O(rmn)$ or $\tilde{O}(r^2 m\sqrt{n})$. Note that these complexities follow from the customary assumption that an arithmetic operation takes $O(r)$ time on weights of the order n^r. We present two different techniques to efficiently compute fair matchings and a generalization called fair *b-matchings*.

A combinatorial technique. Our first technique is an iterative combinatorial algorithm for the fair matching problem. The running time of this algorithm is $\tilde{O}(r^* m\sqrt{n})$ or $\tilde{O}(r^* n^\omega)$ with high probability, where r^* is the largest rank used in a fair matching and $\omega \approx 2.37$ is the exponent of matrix multiplication. This algorithm is based on linear programming duality and in iteration i, we solve the following "dual problem" – dual to a variant of the maximum weight matching problem.

> *Generalized minimum weighted vertex cover problem.* Let $G_i = (A \cup B, E_i)$ be a bipartite graph with edge weights given by $w_i : E_i \to \{0, 1, \ldots, c\}$. Let $K_{i-1} \subseteq A \cup B$ satisfy the property that there is a matching in G that matches all $v \in K_{i-1}$. Find a cover $\{y_u^i\}_{u \in A \cup B}$ so that $\sum_{u \in A \cup B} y_u^i$ is minimized subject to (1) for each $e = (a, b)$ in E_i, we have $y_a^i + y_b^i \geq w_i(e)$, and (2) $y_u^i \geq 0$ if $u \notin K_{i-1}$.

When $K_{i-1} = \emptyset$, the above problem reduces to the standard weighted vertex cover problem. We show that the generalized minimum weighted vertex cover problem, where y_v^i for $v \in K_{i-1}$ can be negative, can be solved via a single-source shortest paths subroutine in directed graphs, by a non-trivial extension of a technique of Iri [13].

A scaling technique. Our second technique uses scaling in order to solve the fair matching problem, by the aforementioned reduction to computing a maximum weight matching using exponentially large edge weights. It starts by solving the problem when each edge weight is 0 and then iteratively solves the problem for better and better approximations of the edge weights. This technique is applicable in the more generalized problem of computing fair *b-matchings*, where each vertex has a capacity associated with it. We solve the fair b-matching problem, in time $\tilde{O}(rmn)$ and space $O(m)$, by solving the capacitated transshipment problem, while carefully maintaining "reduced costs" whose values are within polynomial bounds. Brute-force application of the fastest known minimum-cost flow algorithms would suffer from the additional cost of arithmetic and an $O(rm)$ space requirement. For instance, using [9] would result in $\tilde{O}(r^2 mn)$ running time and $O(rm)$ space.

1.1 Background

Fair matchings are a special case of the *profiled-based* matching problems. Fair matchings have not received much attention in the literature so far. Except the two preprints [11, 17] on which this work is based, the only work dealing with fair matchings is the Ph.D. thesis of

Sng [23], which has an algorithm to find a fair b-matching[1] in $O(rQ \min\{m \log n, n^2\})$ time, where $Q = \sum_{v \in V} q(v)$ and $q(v)$ is the capacity of vertex v.

The first profiled-based matching problem was introduced by Irving [14] and is called the "rank-maximal matching" problem.[2] This problem has been well-studied [15, 16, 18, 20].

▶ **Definition 2.** A matching M in $G = (A \cup B, E)$ is rank-maximal if M matches the maximum number of vertices to rank 1 neighbors, subject to this constraint, M matches the maximum number of vertices to rank 2 neighbors, subject to the above two constraints, M matches the maximum number of vertices to rank 3 neighbors, and so on.

However the rank-maximal matching problem has been studied so far in a more restricted model called the the *one-sided* preference lists model. In this model, only vertices of A have preferences over neighbors while vertices in B have no preferences. Note that a problem in the one-sided preference lists model can also be modeled as a problem with two-sided preference lists by making every $b \in B$ assign rank r to every edge incident on it, where r is the worst rank in the preference lists of vertices in A.

The current fastest algorithm to compute a rank-maximal matching in the one-sided preference lists model takes time $O(\min\{r^* m\sqrt{n}, mn, r^* n^\omega\})$ [15], where r^* is the largest rank used in a rank-maximal matching. In the one-sided preference lists setting, each edge has a unique rank associated with it, thus the edge set E is partitioned into $E_1 \dot\cup E_2 \dot\cup \cdots \dot\cup E_r$ – this partition enables the problem of computing a rank-maximal matching to be reduced to computing r^* maximum cardinality matchings in certain subgraphs of G.

Our fair matching algorithm can be easily modified to compute a rank-maximal matching in the two-sided preference lists model. Thus the latter problem can be solved in time $\tilde{O}(r^* m\sqrt{n})$ or $\tilde{O}(r^* n^\omega)$ with high probability, which almost matches its running time for the one-sided case. Another problem that our algorithm can solve is the "maximum cardinality" rank-maximal matching problem. A matching M is a *maximum cardinality rank-maximal* matching if M has maximum cardinality, and within the set of maximum cardinality matchings, M is rank-maximal.

Organization of the paper. Section 2.1 contains our algorithm for the generalized bipartite vertex cover problem, Section 2.2 has our algorithm for fair matchings. Section 3 has our scaling algorithm. The omitted details can be found in the full version of the paper.

2 Our Combinatorial Technique for fair matchings

Recall that our input here is $G = (A \cup B, E)$ and r is the worst or largest rank used in any preference list. The notion of *signature* will be useful to us in designing our algorithm. We first define edge weight functions w_i, for $1 \le i \le r-1$. The value $w_i(e)$, where $e = (a, b)$, is defined as follows:

$$w_i(e) = \begin{cases} 2 & \text{if both } a \text{ and } b \text{ rank each other as rank} \le r - i \text{ neighbors} \\ 1 & \text{if exactly one of } \{a, b\} \text{ ranks the other as a rank} \le r - i \text{ neighbor} \\ 0 & \text{otherwise} \end{cases}$$

▶ **Definition 3.** For any matching M in G, let $\mathsf{signature}(M)$ be $(|M|, w_1(M), \ldots, w_{r-1}(M))$, where $w_i(M) = \sum_{e \in M} w_i(e)$, for $1 \le i \le r-1$.

[1] Sng used the term "generous maximum matching".
[2] Irving originally called it the "greedy matching" problem.

Thus $\mathsf{signature}(M)$ is an r-tuple, where the first coordinate is the size of M, the second coordinate is the number of vertices that get matched to neighbors ranked $r-1$ or better, and so on. Let OPT denote a fair matching. Then $\mathsf{signature}(\mathsf{OPT}) \succeq \mathsf{signature}(M)$ for any matching M in G, where $\succeq$ is the lexicographic order on signatures.

Let us introduce the constant function $w_0 : E \to \{1\}$ so that the first coordinate of $\mathsf{signature}(M)$ is also captured via an edge weight function. Thus $|M| = w_0(M) = \sum_{e \in M} w_0(e)$. For any matching M and $0 \leq j \leq r-1$, let $\mathsf{signature}_j(M) = (w_0(M), \ldots, w_j(M))$ denote the $(j+1)$-tuple obtained by truncating $\mathsf{signature}(M)$ to its first $j+1$ coordinates.

▶ **Definition 4.** A matching M is $(j+1)$-optimal if $\mathsf{signature}_j(M) = \mathsf{signature}_j(\mathsf{OPT})$.

Our algorithm runs for r^* iterations, where $r^* \leq r$ is the largest index i such that $w_{i-1}(\mathsf{OPT}) > 0$. For any $j \geq 0$, in the $(j+1)$-st iteration, our algorithm solves the minimum weighted vertex cover problem in a subgraph G_j. This involves computing a maximum w_j-weight matching M_j in the graph G_j under the constraint that all vertices of a *critical* subset $K_{j-1} \subseteq A \cup B$ have to be matched. In the first iteration which corresponds to $j = 0$, we have $G_0 = G$ and $K_{-1} = \emptyset$.

The problem of computing M_j will be referred to as the primal program of the $(j+1)$-st iteration and the minimum weighted vertex cover problem becomes its dual. We will show M_j to be $(j+1)$-optimal. The problem of computing M_j can be expressed as a linear program (rather than an integer program) as the constraint matrix is totally unimodular and hence the corresponding polytope is integral. This linear program and its dual are given below. (Let $\delta(v)$ be the set of edges incident on vertex v.)

$$\max \sum_{e \in E} w_j(e) x_e^j$$

$$\sum_{e \in \delta(v)} x_e^j \leq 1 \qquad \forall v \in A \cup B$$

$$\sum_{e \in \delta(v)} x_e^j = 1 \qquad \forall v \in K_{j-1}$$

$$x_e^j \geq 0 \qquad \forall e \text{ in } G_j.$$

$$\min \sum_{v \in V} y_v^j$$

$$y_a^j + y_b^j \geq w_j(e) \qquad \forall e = (a,b) \text{ in } G_j$$

$$y_v^j \geq 0 \qquad \forall v \in (A \cup B) \setminus K_{j-1}.$$

▶ Proposition 1. M_j and y^j are the optimal solutions to the primal and dual programs respectively, iff the following hold:

1. if u is unmatched in M_j (thus u has to be outside K_{j-1}), then $y_u^j = 0$;
2. if $e = (u,v) \in M_j$, then $y_u^j + y_v^j = w_j(e)$.

Proposition 1 follows from the complementary slackness conditions in the linear programming duality theorem. This suggests the following strategy once the primal and dual optimal solutions M_j and y^j are found in the $(j+1)$-st iteration.

- to prune "irrelevant" edges: if $e = (u,v)$ and $y_u^j + y_v^j > w_j(e)$, then no optimal solution to the primal program of the $(j+1)$-st iteration can contain e. So we prune such edges from G_j and let G_{j+1} denote the resulting graph. The graph G_{j+1} will be used in the next iteration.
- to grow the critical set K_{j-1}: if $y_u^j > 0$ and $u \notin K_{j-1}$, then u *has to* be matched in every optimal solution of the primal program of the $(j+1)$-st iteration. Hence u should be added to the critical set. Adding such vertices u to K_{j-1} yields the critical set K_j for the next iteration.

Below we first show how to solve the dual problem and then give the main algorithm.

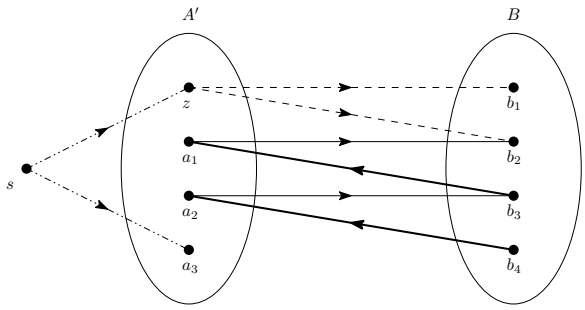


Figure 1 The bold edges are edges of M_j and are directed from B to A' while the edges of $E_j \setminus M_j$ are directed from A' to B.

2.1 Solving the dual problem

For any $0 \leq j \leq r-1$, let $G_j = (A \cup B, E_j)$ be the subgraph that we work with in the $(j+1)$-st iteration and let $K_{j-1} \subseteq A \cup B$ be the critical set of vertices in this iteration. Recall that for each $e \in E_j$, we have $w_j(e) \in \{0, 1, 2\}$. We now show how to solve the dual problem efficiently for a more general edge weight function, i.e., $w_j(e) \in \{0, 1, \ldots, c\}$ for each $e \in E_j$.

Let M_j be the optimal solution of the primal program (we discuss how to compute it at the end of this section). We know that M_j matches all vertices in K_{j-1}. We now describe our algorithm to solve the dual program using M_j. Our idea is built upon that of Iri [13], who solved the special case of $K_{j-1} = \emptyset$. Recall that if a vertex v is not matched in M_j, then $v \notin K_{j-1}$.

- Add a new vertex z to A and let $A' = A \cup \{z\}$. Add an edge of weight 0 from z to each vertex in $B \setminus K_{j-1}$. For convenience, we call the edges from z to these vertices "virtual" edges. The matching M_j still remains an optimal feasible solution after this transformation. [Note that there are only $O(n)$ virtual edges.]
- Next direct all edges $e \in E_j \setminus M_j$ from A' to B and set the edge weight $d(e) = -w_j(e)$; also direct all edges in M_j from B to A' and let the edge weight $d(e) = w_j(e)$.
- Create a *source vertex* s and add a directed edge of weight 0 from s to each *unmatched* vertex in A'. See Figure 1.

Let $\mathcal{R}$ denote the set of all vertices in $A' \cup B$ that are reachable from s. In Figure 1, $\mathcal{R} = \{z, a_3, b_1, b_2\}$.

▶ **Lemma 5.** *By the above transformation,*

1. $B \setminus K_{j-1} \subseteq \mathcal{R}$.
2. *There is no edge between* $A' \cap \mathcal{R}$ *and* $B \setminus \mathcal{R}$.
3. M_j *projects on to a perfect matching between* $A' \setminus \mathcal{R}$ *and* $B \setminus \mathcal{R}$.

Proof. Part (1) holds because there is a directed edge from s to z and directed edges from z to every vertex in $B \setminus K_{j-1}$. To show part (2), it is trivial to see that there can be no edge from $A' \cap \mathcal{R}$ to $B \setminus \mathcal{R}$ (by the definition of $B \setminus \mathcal{R}$). If there is an edge (b, a) from $B \setminus \mathcal{R}$ to $A' \cap R$, then this has to be an edge in M_j and hence it is a's only incoming edge. So for a to be reachable from s, it has to be the case that b is reachable from s, contradicting that $b \in B \setminus \mathcal{R}$.

For part (3), observe that if $b \in B \setminus \mathcal{R}$ is unmatched in M_j, then $b \notin K_{j-1}$ and such a vertex can be reached via z, contradicting the assumption that $b \in B \setminus \mathcal{R}$. If $a \in A' \setminus \mathcal{R}$ is unmatched in M_j, then such a vertex can be reached from s, contradicting the assumption

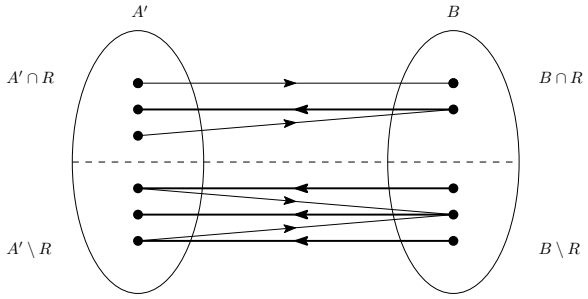


Figure 2 The set $A' \cup B$ in the graph H_j is split into two parts: $(A' \cup B) \cap \mathcal{R}$ and $(A' \cup B) \setminus \mathcal{R}$.

that $a \in A' \setminus \mathcal{R}$. So all vertices in $(A' \cup B) \setminus \mathcal{R}$ are matched in M_j. By (2), a vertex $b \in B \setminus \mathcal{R}$ cannot be matched to vertices in $A' \cap \mathcal{R}$. If a vertex $a \in A' \setminus \mathcal{R}$ is matched to a vertex B in $\mathcal{R}$, then a is also in $\mathcal{R}$, a contradiction. This proves part (3). ◀

Note that there may exist some edges in $E_j \setminus M_j$ that are directed from $A' \setminus R$ to $B \cap \mathcal{R}$. Furthermore, some vertices of $A \setminus K_{j-1}$ can be contained in $A \setminus \mathcal{R}$. Delete all edges from $A' \setminus \mathcal{R}$ to $B \cap \mathcal{R}$ from G_j; let H_j denote the resulting graph. By Lemma 5.3, no edge of M_j has been deleted, thus M_j belongs to H_j and M_j is still an optimal matching in the graph H_j. Moreover, H_j is split into two parts: one part is $(A' \cup B) \cap \mathcal{R}$, which is isolated from the second part $(A' \cup B) \setminus \mathcal{R}$. See Figure 2.

Next add a directed edge from the source vertex s to each vertex in $B \setminus \mathcal{R}$. Each of these edges e has weight $d(e) = 0$. By Lemma 5.3, all vertices can be reached from s now. Also note that there can be no negative-weight cycle, otherwise, we can augment M_j along this cycle to get a matching of larger weight while still keeping the same set of vertices matched, which leads to a contradiction to the optimality of M_j.

Apply the single-source shortest paths algorithm [7, 21, 22, 24] from the source vertex s in this graph H_j where edge weights are given by $d(\cdot)$. Such algorithms take $O(m\sqrt{n})$ time or $\tilde{O}(n^\omega)$ time when the largest edge weight is $O(1)$. Let d_v be the distance label of vertex $v \in A' \cup B$.

We define an initial vertex cover as follows. If $a \in A'$, let $\tilde{y}_a := d_a$; if $b \in B$, let $\tilde{y}_b := -d_b$. (We will adjust this cover further later.)

▶ **Lemma 6.** *The constructed initial vertex cover $\{\tilde{y}_v\}_{v \in A' \cup B}$ for the graph H_j satisfies the following properties:*

1. *For each vertex $v \in ((A \cup B) \cap \mathcal{R}) \setminus K_{j-1}$, $\tilde{y}_v \geq 0$.*
2. *If $v \in (A \cup B) \setminus K_{j-1}$ is unmatched in M_j, then $\tilde{y}_v = 0$.*
3. *For each edge $e = (a, b) \in H_j$, we have $\tilde{y}_a + \tilde{y}_b \geq w_e^j$.*
4. *For each edge $e = (a, b) \in M_j$, we have $\tilde{y}_a + \tilde{y}_b = w_e^j$.*

Proof. For part (1), suppose that $a \in (A \cap \mathcal{R}) \setminus K_{j-1}$ and $\tilde{y}_a < 0$. By Lemma 5.2 and the fact that all edges from $A' \setminus \mathcal{R}$ to $B \cap \mathcal{R}$ are absent, the shortest path from s to a cannot go through $(A \cup B) \setminus \mathcal{R}$. So there exists an alternating path P (of even length) starting from some *unmatched* vertex $a' \in (A' \cap \mathcal{R}) \setminus K_{j-1}$ and ending at a. The distance from a' to a along path P must be negative, since $d_a = \tilde{y}_a < 0$. Therefore,

$$\sum_{e \in M_j \cap P} w_e < \sum_{e \in P \setminus M_j} w_e.$$

Note that it is possible that the first edge $e = (a', b) \in P$ is a virtual edge, i.e., $a' = z$ and the first edge e connects z to some vertex $b \in (B \cap \mathcal{R}) \setminus K_{j-1}$. In this case, $d_e = 0$ and b is not an element of the critical set K_{j-1}. Therefore, irrespective of whether the first edge is virtual or not, we can replace the matching M_j by $M_j \oplus P$ (ignoring the first edge in P if it is virtual), thereby creating a feasible matching with larger weight than M_j, a contradiction.

So we are left to worry about the case when vertex $b \in (B \cap \mathcal{R}) \setminus K_{j-1}$. Recall that $\tilde{y}_b = -d_b$. We claim that $d_b \leq 0$. Suppose not. Then the shortest distance from s to b is strictly larger than 0. But this cannot be, since there is a path composed of edges (s, z) and (z, b), and such a path has total weight exactly 0. This completes the proof of part (1).

To show part (2), by Lemma 5.3, an unmatched vertex must be in $\mathcal{R}$. First, assume that this unmatched vertex is $a \in (A \cap \mathcal{R}) \setminus K_{j-1}$. By our construction, there is only one path from s to a, which is simply the directed edge from s to a and its weight is 0. So $y_a = d_a = 0$. Next assume that this unmatched vertex is $b \in (B \cap \mathcal{R}) \setminus K_{j-1}$. Suppose that $\tilde{y}_b > 0$. Then $d_b = -\tilde{y}_b < 0$. By Lemma 5.2 and the fact that all edges from $A' \setminus \mathcal{R}$ to $B \cap \mathcal{R}$ have been deleted, the shortest path from s to b cannot go through $(A \cup B) \setminus \mathcal{R}$. So the shortest path from s to b must consist of the edge from s to some unmatched vertex $a \in (A' \cap \mathcal{R}) \setminus K_{j-1}$, followed by an augmenting path P (of odd length) ending at b. As in the proof of (1), we can replace M_j by $M_j \oplus P$ (irrespective of whether the first edge in P is virtual or not) so as to get a matching of larger weight while preserving the feasibility of the matching, a contradiction. This proves part (2).

For parts (3) and (4), first consider an edge $e = (a, b)$ outside M_j in H_j. Such an edge is directed from a to b. So $\tilde{y}_a - w_e^j = d_a + d(e) \geq d_b = -\tilde{y}_b$. This proves part (3). Next consider an edge $e = (a, b) \in M_j$. Such an edge is directed from b to a. Furthermore, e is the only incoming edge of a, implying that e is part of the shortest path tree rooted at s. As a result, $-\tilde{y}_b + w_e^j = d_b + d(e) = d_a = \tilde{y}_a$. This shows part (4). This completes the proof of Lemma 6. ◀

At this point, we possibly still do not have a valid cover for the dual program due to the following two reasons.

- Some vertex $a \in A \setminus K_{j-1}$ has $\tilde{y}_a < 0$. (However it cannot happen that some vertex $b \in B \setminus K_{j-1}$ has $\tilde{y}_b < 0$, since Lemma 5.1 states that such a vertex is in $\mathcal{R}$ and Lemma 6.1 states that $\tilde{y}_b$ must be non-negative.)
- The edges deleted from G_j (to form H_j) are not properly covered by the initial vertex cover $\{\tilde{y}_v\}_{v \in A \cup B}$.

We can remedy these two defects as follows. Define $\delta = \max\{\delta_1, \delta_2, 0\}$,

$$\text{where} \quad \delta_1 = \max_{e=(a,b) \in E} \{w_e^j - \tilde{y}_a - \tilde{y}_b\} \quad \text{and} \quad \delta_2 = \max_{a \in A \setminus K_{j-1}} \{-\tilde{y}_a\}.$$

In $O(n+m)$ time, we can compute δ. If $\delta = 0$, the initial cover is already a valid solution to the dual program. In the following, we assume that $\delta > 0$ exists (if the initial cover is already a valid solution for the dual program, then the proof that it is also optimal is just the same as in Theorem 7.) We build the final vertex cover as follows.

1. For each vertex $u \in (A \cup B) \cap \mathcal{R}$, let $y_u = \tilde{y}_u$;
2. For each vertex $a \in A \setminus \mathcal{R}$, let $y_a = \tilde{y}_a + \delta$;
3. For each vertex $b \in B \setminus \mathcal{R}$, let $y_b = \tilde{y}_b - \delta$.

▶ **Theorem 7.** *The final vertex cover $\{y_v\}_{v \in A \cup B}$ is an optimal solution for the dual program.*

Given M_j, it follows that the dual problem can be solved in time $O(m\sqrt{n})$ or $\tilde{O}(n^\omega)$. The problem of computing M_j can be solved by the following folklore technique: form a new graph $\tilde{G}_j$ by taking two copies of G_j and making the two copies of a vertex $u \notin K_{j-1}$ adjacent using an edge of weight 0. A maximum weight *perfect* matching in $\tilde{G}_j$ yields a maximum weight matching in G_j that matches all vertices in K_{j-1}, i.e., an optimal solution to the primal program of the j-th iteration. Since $c = O(1)$, a maximum weight perfect matching in $\tilde{G}_j$ can be found in $O(m\sqrt{n}\log n)$ time by the fastest bipartite matching algorithms [1, 3, 5], or in $\tilde{O}(n^\omega)$ time with high probability by Sankowski's algorithm [22].

2.2 Our main algorithm

We now present our algorithm to compute a fair matching. Recall that r is the worst rank in the problem instance and r^* is the worst rank in a fair matching. We first present an algorithm that runs for r iterations and we show later in this section how to terminate our algorithm in r^* iterations.

1. *Initialization.* Let $G_0 = G$ and $K_{-1} = \emptyset$.
2. For $j = 0$ to $r - 1$ do
   a. Find the optimal solution $\{y_u^j\}_{u \in A \cup B}$ to the dual program of the $(j+1)$-st iteration.
   b. Delete from G_j every edge (a, b) such that $y_a^j + y_b^j > w_j(e)$. Call this subgraph G_{j+1}.
   c. Add all vertices with positive dual values to the critical set, i.e., $K_j = K_{j-1} \cup \{u\}_{y_u^j > 0}$.
3. Return the optimal solution to the primal program of the last iteration.

The solution returned by our algorithm is a maximum (w_{r-1})-weight matching in the graph G_{r-1} that matches all vertices in K_{r-2}. By Proposition 1, this is, in fact, a matching in the subgraph G_r that matches all vertices in K_{r-1}. Lemma 9 proves the correctness of our algorithm. Lemma 8 guarantees that our algorithm is never "stuck" in any iteration due to the infeasibility of the primal or dual problem.

▶ **Lemma 8.** *The primal and dual programs of the $(j+1)$-st iteration are feasible, for $0 \le j \le r - 1$.*

▶ **Lemma 9.** *For every $0 \le j \le r - 1$, the following hold:*
1. *any matching M in G_j that matches all $v \in K_{j-1}$ is j-optimal;*
2. *conversely, a j-optimal matching in G is a matching in G_j that matches all $v \in K_{j-1}$.*

Proof. We proceed by induction. The base case is $j = 0$. As $K_{-1} = \emptyset$, $G_0 = G$, and all matchings are, by definition, 0-optimal, the lemma holds vacuously.

For the induction step $j \ge 1$, suppose that the lemma holds up to $j-1$. As $K_{j-1} \supseteq K_{j-2}$ and G_j is a subgraph of G_{j-1}, M is a matching in G_{j-1} that matches all vertices of K_{j-2}. Thus by induction hypothesis, M is $(j-1)$-optimal. For each edge $e = (a, b) \in M$ to be present in G_j, e must be a tight edge in the j-th iteration, i.e., $y_a^{j-1} + y_b^{j-1} = w_{j-1}(e)$. Furthermore, as $K_{j-1} \supseteq \{u\}_{y_u^{j-1} > 0}$, we have

$$w_{j-1}(M) = \sum_{e=(a,b) \in M} w_{j-1}(e) = \sum_{e=(a,b) \in M} y_a^{j-1} + y_b^{j-1} \ge \sum_{u \in A \cup B} y_u^{j-1},$$

where the final inequality holds because all vertices v with positive y_v^{j-1} are matched in M. By linear programming duality, M must be optimal in the primal program of the j-th iteration. So the j-th primal program has optimal solution of value $w_{j-1}(M)$.

Recall that by definition, OPT is also $(j-1)$-optimal. By (2) of the induction hypothesis, OPT is a matching in G_{j-1} and OPT matches all vertices in K_{j-2}. So OPT is a feasible solution of the primal program in the j-th iteration. Thus $w_{j-1}(\mathsf{OPT}) \leq w_{j-1}(M)$. However, it cannot happen that $w_{j-1}(\mathsf{OPT}) < w_{j-1}(M)$, otherwise, $\mathsf{signature}(M) \succ \mathsf{signature}(\mathsf{OPT})$, since both OPT and M have the same first $j-1$ coordinates in their signatures. So we conclude that $w_{j-1}(\mathsf{OPT}) = w_{j-1}(M)$, and this implies that M is j-optimal as well. This proves (1).

In order to show (2), let M' be a j-optimal matching in G. Since M' is j-optimal, it is also $(j-1)$-optimal and by (2) of the induction hypothesis, it is a matching in G_{j-1} that matches all vertices in K_{j-2}. So M' is a feasible solution to the primal program of the j-th iteration. As $\mathsf{signature}(M')$ has $w_{j-1}(\mathsf{OPT})$ in its j-th coordinate, M' must be an optimal solution to this primal program; otherwise there is a j-optimal matching with a value larger than $w_{j-1}(\mathsf{OPT})$ in the j-th coordinate of its signature, contradicting the optimality of OPT. By Proposition 1.2, all edges of M' are present in G_j and by Proposition 1.1, all vertices $u \notin K_{j-2}$ with $y_u^{j-1} > 0$, in other words, all vertices in $K_{j-1} \setminus K_{j-2}$ have to be matched by the optimal solution M'. This completes the proof of (2). ◀

Since our algorithm returns a matching in G_r that matches all vertices in K_{r-1}, we know from Lemma 9.1 that this matching is r-optimal, thus the matching returned is fair. As mentioned earlier, our algorithm can be modified so that it terminates in r^* iterations. For that, we need to know the value of r^*.

We continue to use the weight function $w_0 : E \to \{1\}$, however instead of $w_1, \ldots, w_{r-1}$, we should use the weight functions $\tilde{w}_1, \ldots, \tilde{w}_{r^*-1}$ where for $1 \leq i \leq r^* - 1$, $\tilde{w}_i$ is defined as: for any edge $e = (a, b)$, $\tilde{w}_i(e)$ is 2 if both a and b rank each other as rank $\leq r^* - i + 1$ neighbors, it is 1 if exactly one of $\{a, b\}$ ranks the other as a rank $\leq r^* - i + 1$ neighbor, otherwise it is 0. The value r^* can be computed right at the start of our algorithm as follows.

- Let M^* be a maximum cardinality matching in G. The value r^* is the smallest index j such that the subgraph $\bar{G}_j$ admits a matching of size $|M^*|$, where $\bar{G}_j$ is obtained by deleting all edges $e = (a, b)$ from G where either a or b (or both) ranks the other as a rank $> j$ neighbor.
- We compute r^* by first computing M^* and then computing a maximum cardinality matching in $\bar{G}_1, \bar{G}_2, \ldots$ and so on till we see a subgraph $\bar{G}_j$ that admits a matching of size $|M^*|$. This index $j = r^*$ and it can be found in $O(r^* m\sqrt{n})$ time [12] or in $O(r^* n^\omega)$ time [10, 19].

We now bound the running time of our algorithm. We showed how to solve the dual program in $O(m\sqrt{n})$ time once we have the solution to the primal program and we have seen that the primal program can be solved in $O(m\sqrt{n}\log n)$ time. Alternatively, both the primal and dual problems can be solved in $\tilde{O}(n^\omega)$ time with high probability. Theorem 10 follows.

▶ **Theorem 10.** *A fair matching M in $G = (A \cup B, E)$ can be computed in $\tilde{O}(r^* m\sqrt{n})$ time, or in $\tilde{O}(r^* n^\omega)$ time with high probability, where r^* is the largest rank incident on an edge in M, $n = |A \cup B|$, $m = |E|$, and $\omega \approx 2.37$ is the exponent of matrix multiplication.*

In the full version of the paper, we show how our algorithm can be adapted to find a rank-maximal matching and similarly, a maximum cardinality rank-maximal matching.

▶ **Theorem 11.** *A rank-maximal (similarly, a maximum cardinality rank-maximal) matching in $G = (A \cup B, E)$ with two-sided preference lists, can be computed in $\tilde{O}(r^* m\sqrt{n})$ time, or in $\tilde{O}(r^* n^\omega)$ time with high probability, where r^* is the largest rank used in such a matching.*

3 The fair b-matching problem: our scaling technique

The fair matching problem can be generalized by introducing capacities on the vertices. We are given $G = (A \cup B, E)$ as before, along with the capacity function $q : V \to \mathbb{Z}_{>0}$. What we seek is a subset E' of E where each vertex $v \in A \cup B$ is incident to at most $q(v)$ edges in E'. Such a subset E' is a *b-matching*. Our goal here is to find a fair b-matching, i.e., a b-matching M which has the largest possible size, subject to this constraint, M matches the minimum number of vertices to their rank r neighbors, and so on.

The fair b-matching problem can be reduced to the minimum-cost flow problem as follows. Add two additional vertices s and t. For each vertex $a \in A$, add an edge (s, a) with capacity $q(a)$ and cost zero; for each vertex $b \in B$, add an edge (b, t) with capacity $q(b)$ and cost zero. Every edge (a, b) where $a \in A, b \in B$ has capacity one and is directed from A to B. If the incident ranks on edge e are i and j, then e will be assigned a cost of $-(4n^r - n^{i-1} - n^{j-1})$. The resulting instance has a trivial upper bound of $n^2/4$ on the maximum s-t flow. We also add an edge from t to s with zero cost and capacity larger than the $n^2/4$ upper bound. It is easy to verify that a minimum-cost circulation yields a fair b-matching.

We note however, that the above reduction involves costs that are exponential in the size of the original problem. We now present a general technique in order to handle these huge costs – we focus on solving the *capacitated transshipment* version of the minimum-cost flow problem [8]. Let $G = (V, E)$ be a directed network with a cost $c : E \to \mathbb{Z}$ and capacity $u : E \to \mathbb{Z}_{\geq 0}$ associated with each edge. With each $v \in V$, a real number $b(v)$ is associated, where $\sum_{v \in V} b(v) = 0$. If $b(v) > 0$, then v is a supply node, and if $b(v) < 0$, then v is a demand node. We assume G to be *symmetric*, i.e., $e \in E$ implies that the reverse arc $e^R \in E$. The reversed edges are added in the initialization step. The cost and capacity functions satisfy $c(e) = -c(e^R)$ for each $e \in E$, $u(e) \geq 0$ for the original edges and $u(e^R) = 0$ for the additional edges. From now on, E denotes the set of original and artificial edges.

A *pseudoflow* is a function $x : E \to \mathbb{Z}$ satisfying the *capacity* and *antisymmetry* constraints: for each $e \in E$, $x(e) \leq u(e)$ and $x(e) = -x(e^R)$. This implies $x(e) \geq 0$ for the original edges. For a pseudoflow x and a node v, the *imbalance* $imb_x(v)$ is defined as $imb_x(v) = \sum_{(w,v) \in E} x(w, v) + b(v)$. A flow is a pseudoflow x such that, $imb_x(v) = 0$ for all $v \in V$. The cost of a pseudoflow x is $cost(x) = \sum_{e \in E} c(e)x(e)$. The minimum-cost flow problem asks for a flow of minimum cost.

For a given flow x, the residual capacity of $e \in E$ is $u_x(e) = u(e) - x(e)$. The residual graph $G(x) = (V, E(x))$ is the graph induced by edges with positive residual capacity. A *potential* function is a function $\pi : V \to \mathbb{Z}$. For a potential function π, the *reduced cost* of an edge $e = (v, w)$ is $c^\pi(v, w) = c(v, w) + \pi(v) - \pi(w)$. A flow x is optimal if and only if there exists a potential function π such that $c^\pi(e) \geq 0$ for all residual graph edges $e \in E(x)$. For a constant $\varepsilon \geq 0$, a flow is ε-optimal if $c^\pi(e) \geq -\varepsilon$ for all $e \in E(x)$ for some potential function π. Consider an ε-optimal flow x and any original edge e. If $c^\pi(e) < -\varepsilon$, the residual capacity of e must be zero and hence e is saturated, i.e., $x(e) = u(e)$. If $c^\pi(e) > \varepsilon$, we have $c^\pi(e^R) = -c^\pi(e) < -\varepsilon$ and hence the residual capacity of e^R must be zero. Thus e^R is saturated, i.e., $x(e^R) = u(e^R) = 0$. So e is unused.

We are now ready to describe our scaling algorithm, which is presented in a concise form in Figure 3. The details can be found in the full version of the paper. We conclude this section with Theorem 13, which follows from the edge cost values used in our reduction.

▶ **Lemma 12** (from [9], see also [8]). *Given the edge cost function $\tilde{c}_i$ and a 3-optimal flow x_{i-1} with respect to the zero potential function, in time $O(mn \log(n^2/m))$ one can compute a flow x_i and a potential function $\tilde{\pi}$ such that x_i is 1-optimal with respect to the potential*

1. *Reduction.*
   a. Add two additional vertices s and t. For each vertex $a \in A$, add an edge (s, a) with capacity $q(a)$ and cost zero; for each vertex $b \in B$, add an edge (b, t) with capacity $q(b)$ and cost zero. Add an edge from t to s with zero cost and capacity larger than $n^2/4$.
   b. Direct any edge (a, b) where $a \in A$ and $b \in B$ from A to B, set its capacity to one and cost to $-(4n^r - n^{i-1} - n^{j-1})$.
   c. Set the demand/supply values of all vertices to zero. Add, if required, additional edges to ensure that G is symmetric.
2. *Initialization Phase.*
   a. Multiply all edge costs by $2^{1+\lceil \log n \rceil}$ to make them divisible by the same amount.
   b. Let $K = \lceil \log C \rceil$ where C is the magnitude of the largest edge cost and let $\mathcal{E}_i$, $1 \leq i \leq K$, denote the set of all edges having a 1 in the i-th bit of their cost.
   c. Initialize x_0 to any feasible flow and reduced cost $c_0(e) = 0$ for any $e \in E$.
3. *Scaling Phase.* For $i = 1$ to K do
   a. Let $\tilde{c}_i(e) = 2c_{i-1}(e) + (1 \text{ if } e \in \mathcal{E}_i \text{ else } 0) \times sign(e)$, where $sign(e) = \pm 1$ depending on the sign of the original cost $c(e)$. The flow x_{i-1} is 3-optimal with respect to the cost function $\tilde{c}_i$ and the zero potential function, i.e., the potential of all the vertices is 0.
   b. Use Lemma 12 (from [9]) stated below with input (i) the flow x_{i-1}, (ii) $\tilde{c}_i$ as the edge cost function and (iii) the zero potential function, to compute a 1-optimal flow and a potential function $\tilde{\pi}$ which proves the 1-optimality. Let x_i be this flow.
   /* Potentials are only decreased, starting from zero, during the computation and $\tilde{\pi}(v) \geq -d \cdot n$ for some constant d and all v. Constant d depends on the way the techniques of [9] are applied to refine a 3-optimal flow to a 1-optimal flow. */
   c. Compute new reduced costs as $c_i(u, v) = \tilde{c}_i(u, v) + \tilde{\pi}(u) - \tilde{\pi}(v)$.
   d. Let d be the constant from Lemma 12. If any edge $e \in E$ has $|c_i(e)| > d \cdot n + 1$, then fix it to empty or saturated by removing it (and its reversal) from the graph and modifying the imbalances of both its endpoints accordingly.
4. Return the b-matching induced by the flow x_K and any flow on edges which were fixed to either empty or saturated.

Figure 3 The scaling algorithm for the fair b-matching problem.

function $\tilde{\pi}$. Potentials are only decreased, starting from zero, during the computation and $\tilde{\pi}(v) \geq -d \cdot n$ for some constant d and all v.

▶ **Theorem 13.** *Given $G = (A \cup B, E)$ and a capacity function $q : A \cup B \to \mathbb{Z}_{>0}$, the fair b-matching problem can be solved in time $O(rmn \log(n^2/m) \log n)$ using space $O(m)$.*

Acknowledgements. We are grateful to the anonymous reviewers for their careful comments. Special thanks to the reviewer who pointed out Sng's thesis [23].

References

1 J. B. Orlin and R. K. Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. In *Mathematical Programming* 54(1): 41-56, 1992.
2 R. K. Ahuja, T. L. Magnanti and J. B. Orlin. Network Flows: Theory, Algorithms, and Applications. Prentice Hall (1993).
3 R. Duan and H.-H. Su. A Scaling Algorithm for Maximum Weight Matchings in Bipartite Graphs. In 23rd *SODA*: 1413-1424, 2012.

4 M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *J.ACM* 34(3), 596-615, 1987.

5 H. Gabow and R. Tarjan. Faster scaling algorithms for network problems. In *SIAM J. Comput.* 18: 1013-1036, 1989.

6 D. Gale and L.S. Shapley. College admissions and the stability of marriage. In *American Mathematical Monthly* 69: 9-15, 1962.

7 A. V. Goldberg. Scaling Algorithms for the Shortest Paths Problem. In *SIAM J. Comput.* 24(3): 494-504, 1995.

8 A. V. Goldberg, E. Tardos and R. E. Tarjan. Network Flow Algorithms. In *Paths, Flows and VLSI-Design*: 101-164, Springer Verlag, 1990.

9 A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. In *Math. Oper. Res.* 15: 430-466, 1990.

10 N. J. A. Harvey. Algebraic Structures and Algorithms for Matching and Matroid Problems. In *SIAM J. Comput.* 39(2): 679-702, 2009.

11 C.-C. Huang and T. Kavitha. Weight-maximal Matchings. In the 2nd International Workshop on Matching under Preferences, July 2012.

12 J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. In *SIAM J. Comput.* 2: 225-231, 1973.

13 M. Iri. A new method of solving transportation-network problems. In *Journal of the Operations Research Society of Japan* 3: 27-87, 1960.

14 R. W. Irving. Greedy Matchings. University of Glasgow, Computing Science Department Research Report, TR-2003-136, 2003.

15 R.W. Irving, T. Kavitha, K. Mehlhorn, D. Michail and K. E. Paluch. Rank-maximal matchings. In *ACM Transactions on Algorithms* 2(4): 602-610, 2006.

16 T. Kavitha and C. D. Shah. Efficient Algorithms for Weighted Rank-Maximal Matchings and Related Problems. In *17th ISAAC*: 153-162, 2006.

17 K. Mehlhorn and D. Michail. Network Problems with Non-Polynomial Weights and Applications. Available at `www.mpi-sb.mpg.de/~mehlhorn/ftp/HugeWeights.ps`.

18 D. Michail. Reducing rank-maximal to maximum weight matching. In *Theoretical Computer Science* 389(1-2): 125-132, 2007.

19 M. Mucha and P. Sankowski. Maximum Matchings via Gaussian Elimination. In *45th FOCS*: 248-255, 2004.

20 K. Paluch. Capacitated Rank-Maximal Matchings. In *8th CIAC*: 324-335, 2013.

21 P. Sankowski. Shortest Paths in Matrix Multiplication Time. In *13th ESA*: 770-778, 2005.

22 P. Sankowski. Maximum weight bipartite matching in matrix multiplication Time. In *Theoretical Computer Science* 410: 4480-4488, 2009.

23 C. Sng Efficient Algorithms for bipartite matching problems with preferences Ph.D. thesis, University of Glasgow, 2008.

24 R. Yuster and U. Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *46th FOCS*: 90-100, 2005.

Ranking with Diverse Intents and Correlated Contents *

Jian Li[1] **and Zeyu Zhang**[2]

1 **IIIS, Tsinghua University, Beijing, China**
lijian83@mail.tsinghua.edu.cn
2 **Department of Mathematics, Tsinghua University, Beijing, China**
zyzhang92@gmail.com

Abstract

We consider the following document ranking problem: We have a collection of documents, each containing some topics (e.g. sports, politics, economics). We also have a set of users with diverse interests. Assume that user u is interested in a subset I_u of topics. Each user u is also associated with a positive integer K_u, which indicates that u can be satisfied by any K_u topics in I_u. Each document s contains information for a subset C_s of topics. The objective is to pick one document at a time such that the average *satisfying time* is minimized, where a user's satisfying time is the first time that at least K_u topics in I_u are covered in the documents selected so far.

Our main result is an $O(\rho)$-approximation algorithm for the problem, where ρ is the algorithmic integrality gap of the linear programming relaxation of the set cover instance defined by the documents and topics. This result generalizes the constant approximations for generalized min-sum set cover and ranking with unrelated intents and the logarithmic approximation for the problem of ranking with submodular valuations (when the submodular function is the coverage function), and can be seen as an interpolation between these results. We further extend our model to the case when each user may be interested in more than one sets of topics and when the user's valuation function is XOS, and obtain similar results for these models.

1998 ACM Subject Classification I.1.2 Algorithms

Keywords and phrases Approximation Algorithm, Diversification, min-sum Set Cover

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.351

1 Introduction

1.1 Background

In a typical information retrieval application, we have a set of users and a set of documents. Each user issues a query and we would like to present the user with a rank list of the documents. Hopefully, the top-ranked documents are relevant to the user and our general objective is to maximize the overall user satisfaction. In many IR applications, the *probabilistic ranking principle* (PRP) is considered as a common rule of thumb to rank the documents [33]. PRP states that we should rank the documents in descending order by their probability of relevance and it is the "optimal" way to rank the documents in the sense that PRP minimizes the expected loss (also known as the Bayes risk) under 1/0 loss [28]. However, the 0/1 loss metric does not directly relate to the users' satisfaction and sometimes the ranking given by

* This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301 and the National Natural Science Foundation of China Grant 61202009, 61033001, 61061130540, 61073174.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 351–362

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

PRP is clearly suboptimal. Indeed, even the original paper [33] provided such an example (the example was discovered by W.S.Cooper).

▶ **Example 1.** [33] The class of users consists of two subclasses $\mathcal{U}_1$ and $\mathcal{U}_2$. $\mathcal{U}_1$ has 100 users and $\mathcal{U}_2$ has 50 users. Any user from $\mathcal{U}_1$ would be satisfied with any document s_1–s_9, but no others. Any user from $\mathcal{U}_2$ would be satisfied with only s_{10}. If we consider any document s_1–s_9 on its own, it has a probability of 2/3 of being relevant to the next user (the ranking algorithm does not know which subclass the user belongs to). Similarly, s_{10} has a probability of 1/3 of being relevant. Therefore, by PRP, the ranking should be $s_1, s_2, \ldots, s_9, s_{10}$. But this means that $\mathcal{U}_1$ users can be satisfied with s_1 while $\mathcal{U}_2$ users have to see nine irrelevant documents before they retrieve s_{10}. Consider the ranking $s_1, s_{10}, s_2, \ldots, s_9$. $\mathcal{U}_1$ users are still satisfied by the first document, but $\mathcal{U}_2$ users are satisfied with the second document, which is much better than the ranking defined by PRP.

The action of placing several documents aimed at different types of users at the top positions of the rank list (e.g. place s_1 and s_2 as the top-2 in the above example) is called *diversification*. It is a widely accepted fact that *diversification* of the ranking result is helpful in minimizing the risk of user dissatisfaction in a multiuser scenario (See, e.g., [32, 16, 20, 4]). Example 1 is a simple yet instructive illustration why the diverse intents and the correlations of the documents (s_1–s_9 are correlated in a way that any of them could satisfy a $\mathcal{U}_1$ user) are the major reasons for diversification.

1. **Diverse intentions.** Different users may have different intents towards the same query (e.g., a keyword). However, the ranking algorithm does not know the actual type of an individual user but has to use the same ranking function for the same query. In Example 1, there are two user types $\mathcal{U}_1$ and $\mathcal{U}_2$, and the next user could be either of them. Considering another real life example, the keyword “Michael Jordan" may refer to the famous NBA player in one query, and the U.C. Berkeley Professor in another search.
2. **Correlations among documents.** Typically, the utility a user can obtain from a set of documents is not the sum of the utilities from individual documents in the set. This is because of the similarity (or dissimilarity) of the documents. For instance, the utility of two very similar documents is not much more than the utility of one of them (e.g., documents s_1 and s_2 in Example 1). Such correlations can be seen as another cause of diversification of the ranking result (see e.g., [16]).

1.2 Problem Formulation

In this section, we propose our model for diversification, which captures both the diversity of users' intents and the correlations of the documents.

▶ **Definition 2** (Ranking with Diverse Intents and Correlated Contents (RDC))**.** Here, we have a set $\mathcal{U}$ of users, a set $\mathcal{S}$ of documents, and a set $\mathcal{E}$ of topics. Each user u is interested in a subset I_u of topics. Each user u is also associated with a positive integer K_u which is less or equal to $|I_u|$. Each document s contains a subset C_s of topics and $\mathcal{E} = \bigcup_{s\in\mathcal{S}} C_s$. The objective is to pick an ordering of all documents such that the average *satisfying time* is minimized, where a user's satisfying time $t_u = \min\{t \mid$ at least K_u topics in I_u are covered by the first t selected documents$\}$.

It is not hard to see that our RDC model captures both the diversity of the users' intents (i.e., each user is interested in a different subset of topics) and the correlations among documents (i.e., different documents may have some common topics). Now, we discuss some closely related prior work and their relations with our model.

1. **Ranking with multiple intents (R-Multi) [4]:** Azar et al. proposed the following combinatorial model to capture the diversity of user preferences. We have a set $\mathcal{U}$ of users and a set $\mathcal{S}$ of documents. User u can be satisfied with any K_u document from a subset I_u of documents. The objective is the same as ours, to minimize the cumulative users' satisfying time. We can see that it is a special case of RDC where each document contains a distinct topic.
2. **Ranking with unrelated intents (R-Unrel) [2]:** This model is a generalization of R-Multi. For each user u and a document s, there is a nonnegative number A_{us} that is the amount of utility u can get from s. u is satisfied if she accumulates at least K_u units of utility. The objective is same as R-Multi. It is also not hard to see that R-Unrel is a special case of RDC where each document contains A_{us} distinct topics [1].
3. **Ranking with submodular intents (R-Submod) [3]:** The model is a generalization of both R-Multi and R-Unrel. For each user u, there is a nonnegative submodular function $f_u : \{0,1\}^{\mathcal{S}} \to \mathbb{R}^+ \cup \{0\}$. u is satisfied if the set S of documents she gets is such that $f_u(S) \geq 1$. The objective is same as before. R-Submod generalizes RDC (as well as R-Multi and R-Unrel). If the submodular function f_u is the *coverage function* [2], R-Submod is equivalent to RDC.

1.3 Our results

We find that the approximability of RDC is closely related with the (algorithmic) integrality gap of the underlining set cover instance induced by the documents and topics. In particular, we can show the following result. Let $F \subseteq \mathcal{E}$ be a subset of topics. We denote $\mathsf{SC}(F)$ the set cover instance formed by the subsets $C_s : s \in \mathcal{S}$ and the set of topics in F. Let $\mathsf{LP}(\mathsf{SC}(F))$ be the natural linear programming relaxation for $\mathsf{SC}(F)$:

$$\begin{aligned} \text{minimize} \quad & \sum_{s\in\mathcal{S}} x_s & \\ \text{subject to} \quad & \sum_{s:e\in C_s, s\in\mathcal{S}} x_s \geq 1 & \forall e \in F \\ & x_s \geq 0 & \forall s \in \mathcal{S} \end{aligned}$$

▶ **Theorem 3.** *Suppose for any $F \subseteq \mathcal{E}$, there is a polynomial time algorithm that can produce a solution for $\mathsf{SC}(F)$ whose cost is at most ρ times the optimal value of $\mathsf{LP}(\mathsf{SC}(F))$. There is a polynomial time factor $O(\rho)$ approximation algorithm for RDC.*

First, we can see that Theorem 3 produces $O(1)$ factor approximation for both R-Multi and R-Unrel. As we mentioned before, if we view R-Multi and R-Unrel as special cases of RDC, the induced set cover instances have very simple structure: each set (document) consists of a disjoint set of elements (topics). In both R-Multi and RDC, the integrality gap of $\mathsf{LP}(\mathsf{SC}(F))$ is 1 for any $F \subseteq \mathcal{E}$ and we can find an integral optimal solution in polynomial time (the algorithm trivially includes all subsets that contains at least one element in F). Hence, $\rho = 1$ and we have a constant factor approximation algorithm. Therefore, our result generalizes the constant approximations for R-Multi in [5, 34, 24] and that for R-Unrel in [2].

[1] It is a polynomial time reduction if $\max A_{us}$ is a positive polynomially bounded integer. However, our analysis works even if A_{us} is exponentially large, since the corresponding set cover instance $\mathsf{LP}(\mathsf{SC}(F))$ can be solved (trivially) in polynomial time.

[2] The set of documents and the set of topics in I_u form a set cover instance, where I_u is the subset of topics which user u is interested in. For $S \subseteq \mathcal{S}$, the coverage function $f(S)$ is the number of topics in I_u covered by some document in S.

For R-Submod, Azar et al. showed that there is an $O(\log \frac{1}{\epsilon})$ approximation for the problem where ϵ is the minimum non-zero marginal value for f_is [3]. If the submodular functions are coverage function, the result translates to an $O(\log |\mathcal{E}|)$-approximation. It is well known that we can round any fractional solution of $\mathsf{LP}(\mathsf{SC}(F))$ to an integral solution such that the cost of the integer solution is at most $\log |\mathcal{E}|$ of the value of the fractional solution. Hence, Theorem 3 also gives an $O(\log |\mathcal{E}|)$-approximation, reproducing the result in [3] for R-Submod with coverage functions (with a somewhat larger constant hidden in the big-O notation).

Our result can be seen as an interpolation between the constant approximation for R-Multi and R-Unrel (which induce trivial set cover instances) and the logarithmic approximation for R-Submod(which may induce arbitrary set cover instances). Besides the above implications on previous problems, Theorem 3 is also interesting since typically the set cover instances induced by the documents and topics are much easier to approximate than general set cover problem. We provide some useful examples.

1. In R-Multi and R-Unrel, the induced set cover instances are trivial and can be solved optimally.
2. Consider another interesting example where each topic is covered by at most d documents. It is known that we can obtain a d-approximation by a simple deterministic rounding or primal-dual techniques (see e.g. [36]). Hence, in this case, we have an $O(d)$-approximation for RDC.
3. Suppose the VC dimension of the set system $(\mathcal{E}, \mathcal{S})$ is d. It is well known that we can achieve an approximation factor of $O(d \log \tau)$ via the LP approach [17], where τ is the optimum LP value ($O(d \log \mathsf{OPT})$ is known even earlier via a non-LP approach [11]). In many cases, $O(d \log \tau)$ can be much smaller than $O(\log |\mathcal{E}|)$.
4. For some geometric set cover problems, we can achieve sub-logarithmic factor approximation algorithms using LP approaches. For example, if each subset corresponds to a unit disk in the plane and each element corresponds to a point, there is a constant approximation [31]. For general disk graphs, a $2^{O(\log^* |\mathcal{E}|)}$-approximation is known [35]. Several sub-logarithmic factor approximation algorithms are known for certain geometric set cover problems via other techniques, such as local search or dynamic programming [1, 30, 19, 12]. However, it is not clear how to combine those techniques with our LP approach. We leave this as an interesting open question.

Even though the real world topics and documents may not necessarily have low VC-dimension or match any geometric set cover instance, it is still our general belief that the real world instance do not form arbitrary set system and the particularity of those instances may help us to develop sub-logarithm factor approximations, which further implies that RDC can be approximated within the same factor (up to a constant). Exploring the particularity of the real world instances is left as an open question of great importance.

1.4 Related work

Azar et al. [4] introduced R-Multi and first gave an $O(\log n)$ factor approximation algorithm. Bansal et al. [5] improved the approximation ratio to a constant (a few hundreds). Subsequently, the constant was further reduced to about 28 in [34], and then to 12.4 [24]. An important special case of R-Multi, where $K_u = 1$ for each u, is called the *min-sum set cover* problem. Feige et al. [18] developed a 4-approximation and proved that it is NP-hard to achieve an approximation factor of $4 - \epsilon$ for any constant $\epsilon > 0$. In fact, it is conjectured that R-Multi can also be approximated within a factor of 4 [24]. Another special case of R-Multi where $K_u = |I_u|$ has also been studied under the name of *minimum latency set cover* and it is known that there is a polynomial time approximation algorithm with factor 2 [22, 25],

which is also optimal assuming a variant of the Unique Games Conjecture [8]. Im et al. [23] considered a generalization of R-Submod where there is metric switching cost and gave a poly-logarithmic factor approximation algorithm for it.

There is a huge literature on search result diversification in IR and DB literature. We refer interested readers to [16, 20, 9] and the references therein. In practice, the overall satisfying time as defined above is not a direct measure of the overall user satisfaction. Alternative measures have been proposed in the literature, such as discounted cumulative gain (DCG) and mean average precision (MAP). Bansal et al. considered R-Multi with DCG being the objective function and obtained an $O(\log \log n)$-approximation [7].

2 A Constant Factor Approximation Algorithm

In this section, we will prove Theorem 3 by giving a randomized LP rounding algorithm.

2.1 The LP Relaxation

We use the following linear program relaxation. Here we use boolean variable x_{st} to represent whether document s is selected at time t. y_{ut} indicates if user u is satisfied after time t. z_{st} represents if document s has been selected at time t.

(**LP**) :

$$\text{minimize} \quad \sum_{u\in\mathcal{U}}\sum_{t=1}^{n}(1-y_{ut}) \tag{1}$$

$$\text{subject to} \quad \sum_{t=1}^{n} x_{st} = 1 \qquad \forall s \in \mathcal{S} \tag{2}$$

$$\sum_{s\in\mathcal{S}} x_{st} = 1 \qquad \forall t \in [n] \tag{3}$$

$$z_{st} = \sum_{t'=1}^{t} x_{st'} \qquad \forall t \in [n] \tag{4}$$

$$\sum_{e\in I_u}(y_{ut} - \min\{\sum_{s:e\in C_s} z_{st}, 1\}) \le (|I_u| - K_u)y_{ut} \qquad \forall u \in \mathcal{U}, t \in [n] \tag{5}$$

$$x_{st}, y_{ut}, z_{st} \in [0,1] \qquad \forall s \in \mathcal{S}, u \in \mathcal{U}, t \in [n] \tag{6}$$

Constraints (2) and (3) make sure that a document can be selected only once and each time we pick one document. The meaning of z_{st} is captured in constraints (4). Constraints (5) guarantee that a user u is satisfied if less than $|I_u| - K_u$ topics havn't been covered. However, it is known that the integrality gap of this LP is unbounded (even for R-Multi) [5]. To remedy this, [5] uses the *knapsack cover* constraints to replace the simple covering constraints (5) In our case, we define $S(e,u,F) = \{s \mid e \in C_s, s \in T_2(u,F)\}$ where $T_1(u,F)$ is the set of all documents that cover at least $K_u - |F|$ topics in $I_u \backslash F$, and $T_2(u,F) = \mathcal{S}\backslash T_1(u,F)$. And we use the following constraints instead of (5):

$$y_{ut}(K_u - |F|) \le (K_u - |F|) \sum_{s\in T_1(u,F)} z_{st} + \sum_{e\in I_u\backslash F}\sum_{s\in S(e,u,F)} z_{st} \qquad \forall u \in \mathcal{U}, t \in [n], F \subseteq I_u, |F| \le K_u \tag{7}$$

Constraints (7) differ from the knapsack cover constraints in [5] in that we handle sets T_1 and T_2 seperately, for technical reason that will be clear from the analysis.

Now we show that (**LP**) is indeed an LP relaxation of RDC. We just need to prove that any feasible solution to RDC satisfies constraints (7): If $y_{ut} = 0$, the inequality must be true because the left side is 0. If $y_{ut} = 1$, there are two cases. The first case is that at least one document in $T_1(u, F)$ has been selected, which means $\sum_{s \in T_1(u,F)} z_{st} \geq 1$. The other case is that at least $(K_u - |F|)$ topics have related document in $T_2(u, F)$, which means $\sum_{e \in I_u \setminus F} \sum_{s \in S(e,u,F)} z_{st} \geq (K_u - |F|)$. Therefore both cases satisfy the inequality. So we have proved the following lemma:

▶ **Lemma 4.** *The optimal value* $\mathsf{OPT}_{\mathsf{LP}}$ *of (**LP**) is at most the optimal total satisfying time of RDC.*

2.2 A Randomized Rounding Algorithm

Assume (x^*, y^*, z^*) be the optimal fractional solution to (**LP**). We also assume that for any $F \subseteq \mathcal{E}$, there is a poly-time algorithm **AlgoSC** which can produce an integral solution for $\mathsf{SC}(F)$ whose cost is at most ρ times the value of the *fractional* optimal solution to $\mathsf{LP}(\mathsf{SC}(F))$. Our randomized rounding scheme consists of $\lceil \log n \rceil + 1$ rounds, where in the k-th round, we perform the following procedure.

- Let $t = 2^k$, $G_k = \emptyset$ and $p_e = \min\{1, 50 \sum_{s:e \in C_s} z^*_{st}\}$, $\forall e \in \mathcal{E}$.
- Let $P_k = \{e \in \mathcal{E}, p_e = 1\}$. Let the set $H_k \subseteq \mathcal{S}$ be the solution of **AlgoSC**$(\mathsf{SC}(P_k))$.
- For each $s \in \mathcal{S} \setminus H_k$, add document s to G_k independently with probability $\min\{1, 50 z^*_{st}\}$.
- If there are more than $(70+\rho) \cdot 2^k$ documents in $H_k \cup G_k$, we say this round is "overflowed" and select nothing, else we select all the documents in $H_k \cup G_k$ in arbitrary order in this round.

Our algorithm builds on the ideas developed in [5] (as well as [2]). A key technical difference between our algorithm and [5] is that we need to deal with those topics that are almost covered (i.e., the set P_k) and the rest separately. It will be clear soon from the analysis, for a particular user u, independent rounding (step 3) can guarantee that, at a cost not much more than the fractional optimal, topics in $I_u \setminus P_k$ are covered with constant probability. For these topics, we can use a Chernoff-like concentration result for submodular functions to show this. Topics in $I_u \cap P_k$ are handled separately by **AlgoSC** to make sure they are covered in the k-th round. This is where the approximiblity of the set cover instance jumps in.

2.3 The Analysis

Constraints (4) and (6) show that the optimal solution z^*_{st} is monotonically non-decreasing with t for all $s \in \mathcal{S}$. Thus it's easy to see that y^*_{ut} is monotonically non-decreasing with t for all $u \in \mathcal{U}$.

For each $u \in \mathcal{U}$, let $t^*_u = \max\{t \in [n] \mid y^*_{ut} \leq \frac{1}{2}\}$, then $\sum_{t=1}^{n}(1 - y^*_{ut}) \leq \sum_{t=1}^{t^*_u}(1 - y^*_{ut}) \leq \frac{1}{2} t^*_u$. Thus we have the fact that $\mathsf{OPT}_{\mathsf{LP}} \geq \frac{1}{2} \sum_u t^*_u$.

Before we start to prove our Theorem 3, we need the following Chernoff-type bounds:

▶ **Lemma 5.** *If $X_1, X_2, \ldots, X_n$ are independent $\{0,1\}$-valued random variables with $X = \sum_i X_i$ such that $\mathbb{E}[X] = \mu$, then we have that*

1. [29] $\Pr[X < (1-\delta)\mu] \leq e^{-\frac{\delta^2}{2}\mu}$.
2. [10] $\Pr[X > \mu + \beta] \leq \exp(-\frac{\beta^2}{2\mu + \frac{2}{3}\beta})$.

▶ **Lemma 6.** [13] *Let $f : \{0,1\}^n \to \mathbb{R}^+$ be a monotone submodular function with marginal values in $[0,1]$. Let $\mu = \mathbb{E}[f(X_1,\ldots,X_n)]$. Then for any $\delta > 0$,*

$$\Pr\Big[f(X_1,\ldots,X_n) \le (1-\delta)\mu\Big] \le e^{-\frac{\delta^2}{2}\mu}.$$

We now give the following lemma:

▶ **Lemma 7.** *For any user $u \in \mathcal{U}$ and a non-overflowed round k such that $2^k \ge t^*_u$, the probability that $H_k \cup G_k$ does not satisfy u is at most 0.023.*

Proof. Fix a user u. Consider constraints (7) for $F = P_k \cap I_u$ and $t = 2^k$. If $|F| \ge K_u$, u is clearly satisfied in this phase because all the documents in H_k are selected in this round. Therefore, we consider the case where $|F| < K_u$. From constraints (7) we know that:

$$(K_u - |F|) \sum_{s\in T_1(u,F)} z^*_{st} + \sum_{e\in I_u\setminus F} \sum_{s\in S(e,u,F)} z^*_{st} \ge y^*_{ut}(K_u - |F|) \ge \frac{1}{2}(K_u - |F|).$$

Here, either $\sum_{s\in T_1(u,F)} z^*_{st}$ must be greater or equal to $\frac{1}{5}$, or $\sum_{e\in I_u\setminus F}\sum_{s\in S(e,u,F)} z^*_{st}$ must be greater or equal to $\frac{3}{10}(K_u - |F|)$.

In the case of $\sum_{s\in T_1(u,F)} z^*_{st} \ge \frac{1}{5}$. If there is an $s \in T_1(u,F)$ such that $50 z_{st} \ge 1$, then s is selected and user u is satisfied. Otherwise, since we select documents independently in our algorithm and the expected number of selected documents in $T_1(u,F)$ is

$$\mathbb{E}\Big[|(G_k \cup H_k) \cap T_1(u,F)|\Big] = \sum_{s\in T_1(u,F)\setminus H_k} 50 z^*_{st} + |T_1(u,F) \cap H_k| \ge \sum_{s\in T_1(u,F)} 50 z^*_{st} \ge 10.$$

From Lemma 5 (4a), we know that the probability that $H_k \cup G_k$ contains less than one document in $T_1(u,F)$ is

$$\Pr\Big[|(G_k \cup H_k) \cap T_1(u,F)| < (1 - \frac{9}{10})\cdot 10\Big] \le \exp(-\frac{(\frac{9}{10})^2}{2}\cdot 10) < 0.018.$$

Therefore the probability that user u is not satisfied by $H_k \cup G_k$ in this case is at most 0.018.

In the case of $\sum_{e\in I_u\setminus F}\sum_{s\in S(e,u,F)} z^*_{st} \ge \frac{3}{10}(K_u - |F|)$, assume boolean vector $\mathbf{w} = \{w_s\} \in \{0,1\}^{|T_2(u,F)|}$ indicates the selected documents in $T_2(u,F)$. Let submodular function $f(\mathbf{w}) = \sum_{e\in I_u\setminus F} \min\{1, \sum_{s\in S(e,u,F)} w_s\}$, i.e. the number of topics in $I_u\setminus F$ that the selection of documents $\mathbf{w}$ covers. Suppose $\overline{\mathbf{z}_t} = \{\overline{z^*_{st}}\}_{s\in T_2(u,F)}$ to be a random 0/1 vector that is obtained as follows: Independently set $\overline{z^*_{st}}$ to be 0 with probability $(1 - 50z^*_{st})$ if $s \in T_2(u,F)\setminus H_k$, and 1 otherwise. Since for any $e \in I_u\setminus F$, $\sum_{s:e\in C_s} z^*_{st} < \frac{1}{50}$ (see the definition of F and P_k), we can find:

$$\begin{aligned}
\mathbb{E}\Big[f(\overline{\mathbf{z}_t})\Big] &= \sum_{e\in I_u\setminus F} \Pr\Big[e \text{ is covered by } \overline{\mathbf{z}_t}\Big] \\
&= \sum_{e\in (I_u\setminus F)\setminus \bigcup_{s\in H_k} C_s} (1 - \prod_{s\in S(e,u,F)} (1 - 50 z^*_{st})) + |\{e \mid e \in (I_u\setminus F) \cap \bigcup_{s\in H_k} C_s\}| \\
&\ge \sum_{e\in I_u\setminus F} (1 - \prod_{s\in S(e,u,F)} (1 - 50 z^*_{st})) \\
&\ge \sum_{e\in I_u\setminus F} (1 - \prod_{s\in S(e,u,F)} e^{-50 z^*_{st}}) \\
&\ge \sum_{e\in I_u\setminus F} (1 - \exp(-\sum_{s\in S(e,u,F)} 50 z^*_{st})) \\
&\ge \sum_{e\in I_u\setminus F} (1 - \tfrac{1}{e}) \sum_{s\in S(e,u,F)} 50 z^*_{st} \\
&\ge (1 - \tfrac{1}{e}) 15 (K_u - |F|)
\end{aligned}$$

where the penultimate inequality is because $(1-e^{-x}) \geq (1-\frac{1}{e})x$, $\forall x \in [0,1]$.

From Lemma 6, we know that

$$\Pr[f(\overline{\mathbf{z}_t}) \leq |K_u| - |F|] = \Pr[f(\overline{\mathbf{z}_t}) \leq (1 - \frac{14e-15}{15e-15})] \leq e^{-(\frac{14e-15}{15e-15})^2 \frac{15e-15}{2e}} < 0.023.$$

This shows that the probability that user u is not satisfied by $H_k \cup G_k$ in this case is at most 0.023, which complete the proof of Lemma 7. ◀

▶ **Lemma 8.** *The probability that the algorithm "overflowed" in round k is at most 0.03.*

Proof. First we show $|H_k| \leq \rho \times 2^k$. It is easy to see that $\mathbf{z} = \{z_{st}\}_{s\in\mathcal{S}}$ is a feasible solution of $\mathsf{LP}(\mathsf{SC}(P_k)))$. By our assumption on **AlgoSC**, we have that

$$|H_k| \leq \rho \sum_{s\in\mathcal{S}} z^*_{st} = \rho \sum_{s\in\mathcal{S}} \sum_{t'=1}^{t} x^*_{st'} = \rho \sum_{t'=1}^{t} \sum_{s\in\mathcal{S}} x^*_{st'} = \rho \cdot 2^k.$$

where the last equation is because $t = 2^k$ in round k.

Therefore, it is suffice to show that $\Pr[|G_k| \geq 70 \cdot 2^k] < 0.03$. In our setting,

$$\mathbb{E}\Big[|G_k|\Big] = \sum_{s\in\mathcal{S}\setminus H_k} \min\{1, 50z^*_{st}\} \leq \sum_{s\in\mathcal{S}} 50z^*_{st} = \sum_{s\in\mathcal{S}} \sum_{t'=1}^{t} 50x^*_{st'} = \sum_{t'=1}^{t} \sum_{s\in\mathcal{S}} 50x^*_{st'} = 50 \cdot 2^k.$$

From Lemma 5 (4b), we know that

$$\Pr\Big[|G_k| > 50 \cdot 2^k + 20 \cdot 2^k\Big] \leq \exp(-\frac{400 \cdot 2^{2k}}{100 \cdot 2^k + \frac{40}{3} \cdot 2^k}) < 0.03.$$

◀

Now we will prove our Theorem 3.

Proof. Let $\mathbf{Satisfy}(u)$ denote the satisfying time of u in our algorithm. From constraints (2), (3) and (4), we know that $z^*_{sn} = 1$ for all $s \in \mathcal{S}$, thus all the users must be satisfied after $\lceil \log n \rceil + 1$ rounds. If some user u is satisfied before the $\lceil \log t^*_u \rceil$th round, the satisfying time $\mathbf{Satisfy}(\mathrm{u}) \leq 2^{\lceil \log t^*_u \rceil}$.

Otherwise, if some user u is satisfied after the $\lceil \log t^*_u \rceil$th round, since we select at most $(70+\rho)2^k$ documents in each round, the satisfying time of user u is at most $2 \cdot (70+\rho)2^k$ if he is satisfied in the k-th round. From Lemma 7 and Lemma 8, we know that the probability that user u isn't satisfied after the k-th round where $2^k \geq t^*_u$ is less than $1-(1-0.023)\times(1-0.03) < 0.053$. Notice that the probability is independent in each round, we get the expected total satisfying time:

$$\begin{aligned}
\mathbb{E}\Big[\sum_{u\in\mathcal{U}} \mathbf{Satisfy}(\mathrm{u})\Big] &\leq \sum_{u\in\mathcal{U}} (2 \cdot (70+\rho)2^{\lceil \log t^*_u \rceil} + \sum_{i=\lceil \log t^*_u \rceil + 1}^{\lceil \log n \rceil + 1} (70+\rho)2^i \cdot 0.053^{i - \lceil \log t^*_u \rceil}) \\
&\leq \sum_{u\in\mathcal{U}} ((140+2\rho)t^*_u + (70+\rho)t^*_u \sum_{i=1}^{\infty} 0.106^i) \\
&< (149 + 2.12\rho) \sum_{u\in\mathcal{U}} t^*_u \\
&< (298 + 4.3\rho)\mathsf{OPT}_{\mathsf{LP}} = O(\rho\mathsf{OPT}).
\end{aligned}$$

This complete the proof of Theorem 3 from Lemma 4. ◀

2.4 Solving the LP

In order to use the ellipsoid method to find the optimal solution, a polynomial-time separation oracle is needed [21]. (7) contains exponentially many inequalities and it is infeasible to verify all of them. However, we can use the trick mentioned in [2]: Instead of finding a solution which satisfies all constraints, we only need one that satisfies the constraints we need in the analysis. Note that in our analysis, we only consider one knapsack inequality in each round where $F = P_k \cap I_u$ and $t = 2^k$. Thus if there is a solution satisfies all these $\lceil \log n \rceil + 1$ inequalities in (7) as well as other constraints in (2), (3), (4) and (6), it is enough for our algorithm even if it is not a feasible solution for (**LP**). Therefore in each iteration of the ellipsoid method, we just need to examine polynomial many constraints.

3 Extensions

3.1 Ranking with Groups of Intents and Correlated Contents (RGC)

Now we extend RDC to the problem that each user u may interest in more than one sets of topics $I_{u1}, I_{u2}, \ldots, I_{up}$, and a user is satisfied if at least one of these groups is satisfied, where p is at most polynomial of n. Same as RDC, a set I_{ui} is satisfied if K_{ui} topics in I_{ui} are covered. This time we could change our relaxed LP as follows:

$$
\begin{array}{lll}
\text{minimize} & \sum_{u\in\mathcal{U}} \sum_{t=1}^{n} (1 - y_{ut}) & \\
\text{subject to} & \sum_{t=1}^{n} x_{st} = 1 & \forall s \in \mathcal{S} \\
& \sum_{s\in\mathcal{S}} x_{st} = 1 & \forall t \in [n] \\
& y_{ut} \le \max_i\{g_{uit}\} & \forall u \in \mathcal{U}, t \in [n] \\
& z_{st} = \sum_{t'=1}^{t} x_{st'} & \forall t \in [n] \\
& g_{uit}(K_{ui} - |F|) \le (K_{ui} - |F|) & \sum_{s\in T_1(u,i,F)} z_{st} + \sum_{e\in I_u\setminus F} \sum_{s: e\in C_s, s\in T_2(u,i,F)} z_{st} \\
& & \forall i \in [p], u \in \mathcal{U}, t \in [n], F \subseteq I_{ui}, |F| \le K_{ui} \\
& x_{st}, y_{ut}, z_{st}, g_{uit} \in [0,1] & \forall s \in \mathcal{S}, u \in \mathcal{U}, t \in [n], i \in [p]
\end{array}
$$

where $T_1(u, i, F)$ is the set of all documents that cover at least $K_{ui} - |F|$ objects in $I_{ui}\setminus F$, $T_2(u, i, F) = \mathcal{S}\setminus T_1(u, i, F)$, and g_{uit} indicates if for user u, group i is satisfied after time t.

The algorithm and analysis are almost the same as in RDC, so we won't talk about it more. Finally we get Theorem 9.

▶ **Theorem 9.** *Suppose for any $F \subseteq \mathcal{E}$, there is a polynomial time algorithm that can produce a solution for $\mathsf{SC}(F)$ whose cost is at most ρ times the optimal value of $\mathsf{LP}(\mathsf{SC}(F))$. There is a polynomial time factor $O(\rho)$ approximation algorithm for RGC.*

3.2 Ranking with XOS Valuations (RXOS)

Notice that all the problems we have mentioned in this paper are special cases of R-Submod, where the users' satisfying functions are submodular functions of the set of documents. There is another family of valuations called XOS. An XOS function is a set function which is the maximum of several additive set functions. An additive set function $f : \{0,1\}^{\mathcal{S}} \to \mathbb{R}^+ \cup \{0\}$ has the form $f(F) = \sum_{s\in F} A_s$, $\forall F \subseteq S$, where A_s is a constant associated with each element $s \in S$. Since the family of submodular functions is contained in XOS [26], R-Submod is a

special case of RXOS.[3] Suppose for each user, the XOS function contains only polynomial number of additive set functions which are non-negative. We can give an O(1)-approximation algorithm. (If the number of additive set functions is exponential, the best approximate rate we can hope for is $O(\log n)$ since RXOS generates R-Submod.)

For each user u, suppose the additive set functions be $f_{ui}, i \in [p]$, where $f_{ui}(F) = \sum_{s\in F} A_{uis}$ for $F \subseteq \mathcal{S}$. Without lose of generality, we can let the satisfying time be $t_u = \min\{t \mid \max_i f_{ui}(\text{the first } t \text{ selected documents}) \geq 1\}$. Now we could have the following LP relaxation:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{u\in\mathcal{U}} \sum_{t=1}^{n} (1 - y_{ut}) \\
\text{subject to} \quad & \sum_{t=1}^{n} x_{st} = 1 && \forall s \in \mathcal{S} \\
& \sum_{s\in\mathcal{S}} x_{st} = 1 && \forall t \in [n] \\
& y_{ut} \leq \max_i\{g_{uit}\} && \forall u \in \mathcal{U}, t \in [n] \\
& z_{st} = \sum_{t'=1}^{t} x_{st'} && \forall t \in [n] \\
& g_{uit}(1 - \sum_{s\in F} A_{uis}) \leq (1 - \sum_{s\in F} A_{uis}) \sum_{s\in T_1(u,i,F)} z_{st} + \sum_{s\in T_2(u,i,F)} A_{uis} z_{st} && \\
& && \forall i \in [p], u \in \mathcal{U}, t \in [n], F \subseteq \mathcal{S}, \sum_{s\in F} A_{uis} \leq 1 \\
& x_{st}, y_{ut}, z_{st}, g_{uit} \in [0, 1] && \forall s \in \mathcal{S}, u \in \mathcal{U}, t \in [n], i \in [p]
\end{aligned}
$$

where $T_1(u, i, F) = \{s \mid A_{uis} \geq (1 - \sum_{e\in F} A_{uie}), s \in \mathcal{S}\}$, $T_2(u, i, F) = \mathcal{S}\backslash T_1(u, i, F)$, and g_{uit} indicates if $f_{ui}(\{\text{the first } t \text{ selected documents}\}) \geq 1$.

This time we do not need to consider the set cover instances, and the k-th round of our algorithm can be:

- Let $t = 2^k$, $G_k = \emptyset$.
- For each $s \in \mathcal{S}$, add document s to G_k independently with probability $\min\{1, 50z^*_{st}\}$
- If there are more than $70 \cdot 2^k$ documents in G_k, we say this round is "overflowed" and select nothing, else we select all the documents in G_k in arbitrary order in this round.

All the discussions are the same as in section 2.3 except there is no H_k, and in the case that $\sum_{s\in T_2(u,i,F)} A_{uis} z_{st} \geq \frac{3}{10}(1 - \sum_{s\in F} A_{uis})$, we could have $\mathbb{E}\left[\sum_{s\in T_2(u,i,F)} A_{uis} \overline{z^*_{st}}\right] \geq 15(1 - \sum_{s\in F} A_{uis})$ and use Lemma 6 directly.

Finally we can have the following theorem:

▶ **Theorem 10.** *Suppose for each user, the XOS function contains only polynomial additive set functions which are non-negative. There is an $O(1)$-approximation for RXOS.*

4 Final Remarks

As we mentioned in the introduction, the real world document-topic instance do not form arbitrary set system and may be easier to approximate than the general combinatorial set cover problem. There is a huge literature on algorithms for classifying or clustering the documents and modeling document-topic relations. Many of those works leverage the underlining special structure of the document-topic instance to achieve good classification or

[3] The number of additive set functions which are needed to represent a submodular function may be exponential

clustering. It is an interesting further direction to explore the connections to those works and see whether the assumptions made or the special structures used in those works would translate to interesting set cover instances that are easier to approximate.

We could extend our model in several ways to capture other factors that may affect the search result. For example, we can capture that each user only has limited patience. We can also incorporate uncertainty into the user preferences. Namely, a user is interested in a particular document with a certain probability. The resulting stochastic version of the problem may have a similar flavor with the sequential trial optimization defined in [15], the probabilistic ranking problem [27] or the stochastic matching problem in [14, 6].

Finally, we note that our approximation algorithm is mainly of theoretical interests since we need to use the ellipsoid algorithm to solve a linear program with exponential constraints, which is computationally expensive in practice. Hence, developing more efficient algorithms for RDC (even with worse performance guarantee) is of great practical interests.

Acknowledgements. Jian Li would like thank Yossi Azar for a stimulating discussion about the RDC model and for providing the manuscript [2].

References

1 Christoph Ambühl, Thomas Erlebach, Matúš Mihalák, and Marc Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 3–14. Springer, 2006.

2 Y. Azar and I. Gamzu. Ranking with Unrelated Intents, 2010.

3 Yossi Azar and Iftah Gamzu. Ranking with submodular valuations. In *Proceedings of the 22nd Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 1070–1079. SIAM, 2011.

4 Yossi Azar, Iftah Gamzu, and Xiaoxin Yin. Multiple intents re-ranking. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 669–678. ACM, 2009.

5 Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *Proceedings of the 21st Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 1539–1545. Society for Industrial and Applied Mathematics, 2010.

6 Nikhil Bansal, Anupam Gupta, Jian Li, Julián Mestre, Viswanath Nagarajan, and Atri Rudra. When lp is the cure for your matching woes: Improved bounds for stochastic matchings. *Algorithmica*, 63(4):733–762, 2012.

7 Nikhil Bansal, Kamal Jain, Anna Kazeykina, and Joseph Seffi Naor. Approximation algorithms for diversified search ranking. In *Automata, Languages and Programming*, pages 273–284. Springer, 2010.

8 Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 453–462. IEEE, 2009.

9 Allan Borodin, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proceedings of the 31st symposium on Principles of Database Systems*, pages 155–166. ACM, 2012.

10 Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. A sharp concentration inequality with applications. *Random Structures & Algorithms*, 16(3):277–292, 2000.

11 Hervé Brönnimann and Michael T Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(1):463–479, 1995.

12 Timothy M Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.

13 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding for matroid polytopes and applications. *arXiv preprint arXiv:0909.4348*, 2009.

14 Ning Chen, Nicole Immorlica, Anna R Karlin, Mohammad Mahdian, and Atri Rudra. Approximating matches made in heaven. In *Automata, Languages and Programming*, pages 266–278. Springer, 2009.

15 Edith Cohen, Amos Fiat, and Haim Kaplan. Efficient sequences of trials. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 737–746. Society for Industrial and Applied Mathematics, 2003.

16 Marina Drosou and Evaggelia Pitoura. Search result diversification. *ACM SIGMOD Record*, 39(1):41–47, 2010.

17 Guy Even, Dror Rawitz, and Shimon Moni Shahar. Hitting sets when the vc-dimension is small. *Information Processing Letters*, 95(2):358–362, 2005.

18 Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

19 Matt Gibson and Imran A Pirwani. Algorithms for dominating set in disk graphs: breaking the logn barrier. In *Algorithms–ESA 2010*, pages 243–254. Springer, 2010.

20 S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.

21 Martin Grötschel, László Lovász, and Alexander Schrijver. Geometric algorithms and combinatorial optimization. 1988.

22 Refael Hassin and Asaf Levin. An approximation algorithm for the minimum latency set cover problem. In *Algorithms–ESA 2005*, pages 726–733. Springer, 2005.

23 Sungjin Im, Viswanath Nagarajan, and Ruben van der Zwaan. Minimum latency submodular cover. In *Automata, Languages, and Programming*, pages 485–497. Springer, 2012.

24 Sungjin Im, Maxim Sviridenko, and Ruben van der Zwaan. Preemptive and non-preemptive generalized min sum set cover. *Mathematical Programming*, pages 1–25, 2012.

25 David Karger, Cliff Stein, and Joel Wein. Scheduling algorithms. *CRC Handbook of Computer Science*, 1997.

26 Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006.

27 Jian Li, Barna Saha, Amol Deshpande. A unified approach to ranking in probabilistic databases *The VLDB Journal*, 20(2):249-275,2011.

28 C.D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press New York, NY, USA, 2008.

29 Rajeev Motwani. *Randomized algorithms*. Cambridge university press, 1995.

30 Nabil Hassan Mustafa and Saurabh Ray. Ptas for geometric hitting set problems via local search. In *Proceedings of the 25th annual symposium on Computational geometry*, pages 17–22. ACM, 2009.

31 Saurav Pandit, Sriram V Pemmaraju, and Kasturi Varadarajan. Approximation algorithms for domatic partitions of unit disk graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 312–325. Springer, 2009.

32 F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *ICML*, pages 784–791, 2008.

33 S.E. Robertson. The probability ranking principle in IR. *Journal of documentation*, 33(4):294–304, 1993.

34 Martin Skutella and David P Williamson. A note on the generalized min-sum set cover problem. *Operations Research Letters*, 39(6):433–436, 2011.

35 Kasturi Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the 42nd ACM symposium on Theory of computing*, pages 641–648. ACM, 2010.

36 Vijay V Vazirani. *Approximation algorithms*. springer, 2001.

Separating Regular Languages by Locally Testable and Locally Threshold Testable Languages

Thomas Place*, **Lorijn van Rooijen***, **and Marc Zeitoun***

LaBRI, Bordeaux University, France, `firstname.lastname@labri.fr`

Abstract

A separator for two languages is a third language containing the first one and disjoint from the second one. We investigate the following decision problem: given two regular input languages, decide whether there exists a locally testable (resp. a locally threshold testable) separator. In both cases, we design a decision procedure based on the occurrence of special patterns in automata accepting the input languages. We prove that the problem is computationally harder than deciding membership. The correctness proof of the algorithm yields a stronger result, namely a description of a possible separator. Finally, we discuss the same problem for context-free input languages.

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

Keywords and phrases Automata, Logics, Monoids, Locally testable, Separation, Context-free

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.363

1 Introduction

Context. The strong connection between finite state devices and descriptive formalisms, such as first-order or monadic second-order logic, has been a guideline in computer science since the seminal work of Büchi, Elgot and Trakhtenbrot. This bridge has continuously been fruitful, disseminating tools and bringing a number of applications outside of its original research area. For instance, compiling logical specifications into various forms of automata has become one of the most successful methods in automatic program verification [26].

One of the challenging issues when dealing with a logical formalism is to precisely understand its expressiveness and its limitations. While solutions to *decide* such logics often use a compilation procedure from formulas to automata, capturing the expressive power amounts to the opposite translation: given a language, one wants to know whether one can reconstruct a formula that describes it. In other words, we want to solve an instance of the *membership problem*, which asks whether an input language belongs to some given class.

For regular languages of finite words, the main tool developed to capture this expressive power is the syntactic monoid [16]: this is a finite, computable, algebraic abstraction of the language, whose properties make it possible to decide membership. An emblematic example is the membership problem for the class of first-order definable languages, solved by Schützenberger [19] and McNaughton and Papert [14], which has led to the development of algebraic methods for obtaining decidable characterizations of logical or combinatorial properties.

The separation problem and its motivations. We consider here the *separation problem* as a generalization of the membership problem. Assume we are given two classes of languages $\mathcal{C}$ and $\mathcal{S}$. The question is, given *two* input languages from $\mathcal{C}$, whether we can separate them by a language from $\mathcal{S}$. Here, we say that a language *separates* K from L if it contains K and is

* Work supported by Agence Nationale de la Recherche ANR 2010 BLAN 0202 01 FREC.

disjoint from L. An obvious necessary condition for separability is that the input languages K, L be disjoint. A separator language *witnesses* this condition.

One strong motivation for this problem is to understand the limits of logics over finite words. Notice that membership reduces to separation when $\mathcal{C}$ is closed under complement, because checking that a language belongs to $\mathcal{S}$ amounts to testing that it is $\mathcal{S}$-separable from its complement. Deciding $\mathcal{S}$-separation is also more difficult than deciding membership in $\mathcal{S}$, as one cannot rely on algebraic tools tailored to the membership problem. It may also be computationally harder, as we shall see in this paper. Thus, solving the separation problem requires a deeper understanding of $\mathcal{S}$ than what is sufficient to check membership: one not only wants to decide whether $\mathcal{S}$ is powerful enough to *describe* a language, but also to decide whether it can *discriminate* between two input languages. This discriminating power provides more accurate information than the expressive power.

While our main concern is theoretical, let us mention some motivating applications. In model checking, reachable configurations of a system can be represented by a language. Separating this from the language representing bad configurations proves to be effective for verifying safety of a system. Craig interpolation is a form of separation used in this context [12, 9]. In this line of work, Leroux [10] simplified the proof that reachability in vector addition systems is decidable [11]: he proved that a recursively enumerable set of separators witnesses non-reachability. Finally, questions in database theory also motivated separation questions [5].

Although the separation problem frequently occurs, it has not been systematically studied, even in the restricted, yet still challenging case of regular languages.

Contributions. In general, elements of $\mathcal{C}$ cannot always be separated by an element of $\mathcal{S}$ and there is no minimal separator wrt. inclusion. We are interested in the following questions:

- can we *decide* whether one can separate two given languages of $\mathcal{C}$ by a language of $\mathcal{S}$?
- what is the *complexity* of this decision problem?
- if separation is possible, can we *compute* a separator, and at which cost?

A motivating but difficult objective is to understand separation by first-order definable languages, whose decidability follows from involved algebraic methods [7, 8]. A first step is to look at easier subclasses. Indeed, the question was raised and solved for separation by piecewise-testable languages [5, 18], and unambiguous languages [18], which sit in the lower levels of the quantifier-alternation hierarchy of first-order logic. In this paper, we look at yet another widely studied class, whose properties are orthogonal to those of the above classes.

We investigate the separation problem by locally and locally threshold testable languages. A language is *locally testable (LT)* if membership of a word can be tested by inspecting its prefixes, suffixes and infixes up to some length (which depends on the language). The membership problem for this class was raised by McNaughton and Papert [14], and solved independently by McNaughton and Zalcstein [27, 13] and by Brzozowski and Simon [4]. This class has several generalizations. The most studied one is that of *locally threshold testable languages (LTT)*, where counting infixes is allowed up to some threshold. These are the languages definable in FO(+1), *i.e.*, first-order logic with the successor relation (but without the order). Again, membership is decidable [24], and can actually be tested in Ptime [17].

Our results are as follows: we show that separability of regular languages by LT and LTT languages is decidable, first for a fixed threshold, by reduction to fixed parameters: we provide a bound on the length of infixes that define a possible separator. This reduces the problem to a finite number of candidate separators, and hence entails decidability. For LTT-separability, we also provide a bound for the threshold. We further get an equivalent formulation on automata in terms of forbidden patterns for the languages to be separable, which yields an

NEXPTIME algorithm. We also obtain lower complexity bounds: even starting from DFAs, the problem is NP-hard for LT and LTT (while membership to LTT is in PTIME). Finally, we discuss the separation problem starting from context-free input languages rather than regular ones. Due to lack of space, several proofs only appear in the journal version of the paper.

The main arguments rely on pumping in monoids or automata. The core of our proof is generic: we show that if one can find two words, one in each input language, that are close enough wrt. the class of separators, then the languages are not separable. Here, "close enough" is defined in terms of parameters of the input languages, such as the size of input NFAs.

Related work. In the context of semigroup theory, it has been proven [1] that the separation problem can be rephrased in purely algebraic terms. Solving the separation problem for a class $\mathcal{S}$ amounts to computing the so-called *pointlike sets* for the algebraic variety corresponding to $\mathcal{S}$. While it has been shown that the varieties corresponding to both LT and LTT have computable pointlike sets [2, 20, 21], this approach suffers two drawbacks. First, being purely algebraic, the proofs provide no insight on the underlying class $\mathcal{S}$. In particular, they provide only yes/no answers without giving any description of what an actual separator might be.

Finally, the separation problem for the class of piecewise-testable languages has recently been shown PTIME-decidable, independently and with different techniques in [5] and [18].

2 Preliminaries

Words and Languages. We fix a finite alphabet A. We denote by A^* the free monoid over A. The empty word is denoted by ε. If w is a word, we set $|w|$ as the *length*, or *size* of w. When w is nonempty, we view w as a sequence of $|w|$ positions labeled over A. We number positions from 0 (for the leftmost one) to $|w| - 1$ (for the rightmost one).

Infixes, Prefixes, Suffixes. An *infix* of a word w is a word w' such that $w = u \cdot w' \cdot v$ for some $u, v \in A^*$. Moreover, if $u = \varepsilon$ (resp. $v = \varepsilon$) we say that w' is a *prefix* (resp. *suffix*) of w.

Let $0 \leqslant x < y \leqslant |w|$. We write $w[x, y]$ for the infix of w starting at position x and ending at position $y - 1$. By convention, we set $w[x, x] = \varepsilon$. Observe that by definition, when $x \leqslant y \leqslant z$, we have $w[x, z] = w[x, y] \cdot w[y, z]$.

Profiles. For $k \in \mathbb{N}$, let $k_\ell = \lfloor k/2 \rfloor$ and $k_r = k - k_\ell$. A *k-profile* is a pair of words (w_ℓ, w_r) of lengths at most k_ℓ and k_r, respectively. Given $w \in A^*$ and x a position of w, the *k-profile of x* is the pair (w_ℓ, w_r) defined as follows: $w_\ell = w[\max(0, x - k_\ell), x]$ and $w_r = w[x, \min(x + k_r, |w|)]$ (see Figure 1). A k-profile (w_ℓ, w_r) *occurs in a word w* if there exists some position x within w whose k-profile is (w_ℓ, w_r). Similarly, if $n \in \mathbb{N}$, we say that (w_ℓ, w_r) *occurs n times in w* if there are n distinct positions in w where (w_ℓ, w_r) occurs.

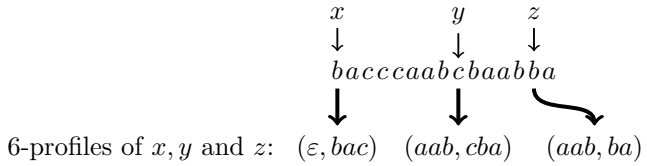


Figure 1 Illustration of the notion of k-profile for $k = 6$.

Intuitively, the k-profile is the description of the infix of w that is centered at position x. Observe in particular that the k-profiles that occur in a word determine the prefixes and suffixes of length $k-1$ of this word. This is convenient, since we only have to consider one object instead of three in the usual presentations of the classes LT and LTT.

We denote by A_k the set of k-profiles over the alphabet A. It is of size exponential in k.

Separability. Given languages L, L_1, L_2 over A^*, we say that L *separates* L_1 from L_2 if

$$L_1 \subseteq L \text{ and } L_2 \cap L = \varnothing.$$

Given a class $\mathcal{S}$ of languages, we say that the pair (L_1, L_2) is $\mathcal{S}$-*separable* if some language $L \in \mathcal{S}$ separates L_1 from L_2. When $\mathcal{S}$ is closed under complement, (L_1, L_2) is $\mathcal{S}$-separable if and only if (L_2, L_1) is, in which case we simply say that L_1 and L_2 are $\mathcal{S}$-separable.

Automata. A *nondeterministic finite automaton* (NFA) over A is denoted by a tuple $\mathcal{A} = (Q, A, I, F, \delta)$, where Q is the set of states, $I \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states and $\delta \subseteq Q \times A \times Q$ the transition relation. The *size* of an automaton is its number of states plus its number of transitions. If δ is a function, then $\mathcal{A}$ is a *deterministic* finite automaton (DFA). We denote by $L(\mathcal{A})$ the language of words accepted by $\mathcal{A}$.

Monoids. Let L be a language and M be a monoid. We say that L *is recognized by* M if there exists a monoid morphism $\alpha : A^* \to M$ together with a subset $F \subseteq M$ such that $L = \alpha^{-1}(F)$. It is well known that a language is accepted by an NFA if and only if it can be recognized by a *finite monoid*. Further, one can compute from any NFA a finite monoid recognizing its accepted language.

3 Locally Testable and Locally Threshold Testable Languages

In this paper, we investigate two classes of languages. Intuitively, a language is locally testable if membership of a word in the language only depends on the *set* of infixes, prefixes and suffixes up to some fixed length that occur in the word. For a locally threshold testable language, membership may also depend on the *number* of occurrences of such infixes, which may thus be counted up to some fixed threshold.

In this section we provide specific definitions for both classes. We start with the larger class of locally threshold testable languages. In the following, we say that two numbers are *equal up to threshold* d if either both numbers are equal, or both are greater than or equal to d.

Locally Threshold Testable Languages. Let L be a language, we say that L is *locally threshold testable* (LTT) if it is a boolean combination of languages of the form:

1. $uA^* = \{w \mid u \text{ is a prefix of } w\}$, for some $u \in A^*$.
2. $A^*u = \{w \mid u \text{ is a suffix of } w\}$, for some $u \in A^*$.
3. $\{w \mid w \text{ has } u \text{ as an infix at least } d \text{ times}\}$ for some $u \in A^*$ and $d \in \mathbb{N}$.

LTT languages can actually be defined in terms of first-order logic. A language is LTT if and only if it can be defined by an FO(+1) formula [2, 25], *i.e.*, a first-order logic formula using predicates for the equality and next position relations, but not for the linear order.

We also define an index on LTT languages. Usually, this index is defined as the smallest size of infixes, prefixes and suffixes needed to define the language. However, since we only work with k-profiles, we directly define an index based on the size of k-profiles. Given words

w, w' and natural numbers k, d, we write $w \equiv_k^d w'$ if for every k-profile (w_ℓ, w_r), the number of positions x such that (w_ℓ, w_r) is the k-profile of x is equal up to threshold d in w and w'. One can verify that for all $k, d \in \mathbb{N}$, $\equiv_k^d$ is an equivalence relation of finite index.

For $k, d \in \mathbb{N}$ we denote by $\mathrm{LTT}[k, d]$ the set of the finitely many languages that are unions of $\equiv_k^d$-classes. We have $\mathrm{LTT} = \bigcup_{k,d} \mathrm{LTT}[k, d]$. Given $L \subseteq A^*$, the smallest $\mathrm{LTT}[k, d]$-language containing L is

$$[L]_{\equiv_k^d} = \{w \in A^* \mid \exists u \in L \text{ such that } u \equiv_k^d w\}.$$

As it is often the case, there is no smallest LTT language containing a given regular language.

Locally Testable Languages. The class of locally testable languages is the restriction of LTT languages in which infixes cannot be counted. A language L is *locally testable* (LT) if it is a boolean combination of languages of the form 1, 2 and the following restriction of 3:

4. $A^*uA^* = \{w \mid w \text{ has } u \text{ as an infix}\}$ for some $u \in A^*$.

No simple description of LT in terms of first-order logic is known. However, there is a simple definition in terms of temporal logic. A language is LT if and only if it can be defined by a temporal logic formula using operators X (next), Y (yesterday), and G (globally).

Given two words w, w' and a number k, we write $w \equiv_k w'$ for $w \equiv_k^1 w'$. For all $k \in \mathbb{N}$, we denote by $\mathrm{LT}[k]$ the set of languages that are unions of $\equiv_k$-classes, and $\mathrm{LT} = \bigcup_k \mathrm{LT}[k]$. Given $L \subseteq A^*$ and $k \in \mathbb{N}$, the smallest $\mathrm{LT}[k]$-language containing L is

$$[L]_{\equiv_k} = \{w \in A^* \mid \exists u \in L \text{ such that } u \equiv_k w\}.$$

4 Separation for a Fixed Threshold

In this section, we prove that if the counting threshold d is fixed, it is decidable whether two languages can be separated by an LTT language of counting threshold d (*i.e.*, by an $\mathrm{LTT}[k, d]$ language for some k). In particular, this covers the case of LT, which corresponds to $d = 1$. All results in this section are for an arbitrary d. Our result is twofold.

- First, we establish a bound on the size of profiles that need to be considered in order to separate the languages. This bound only depends on the size of monoids recognizing the languages, and it can be computed. One can then use a brute-force algorithm that tests separability by all the finitely many $\mathrm{LTT}[k, d]$ languages, where k denotes this bound.
- The second contribution is a criterion on the input languages to check separability by an $\mathrm{LTT}[k, d]$ language for some k. This criterion can be defined equivalently on automata or monoids recognizing the input languages, in terms of the absence of common patterns.

The section is organized into three subsections: our criterion is stated in the first one, and the second and last ones are devoted to the statement and proof of the theorem.

4.1 Patterns

In this section we define our criterion that two languages must satisfy in order to be separable. The criterion can be defined equivalently on automata or monoids recognizing the languages.

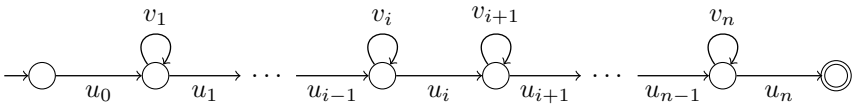


Figure 2 An $\mathcal{A}$-compatible $\mathcal{P}$-decomposition $u_0v_1u_1v_2\cdots v_nu_n$.

Block Patterns. A *block* is a triple of words $\mathfrak{b} = (v_\ell, u, v_r)$ where v_ℓ, v_r are nonempty. Similarly, a *prefix block* is a pair of words $\mathfrak{p} = (u, v_r)$ with v_r nonempty, and a *suffix block* is a pair of words $\mathfrak{s} = (v_\ell, u)$ with v_ℓ nonempty. Let $d \in \mathbb{N}$. A *d-pattern* $\mathcal{P}$ is either a word w, or a triple $(\mathfrak{p}, f, \mathfrak{s})$ where $\mathfrak{p}$ and $\mathfrak{s}$ are respectively a prefix and a suffix block, and f is a function mapping blocks to the set $\{0, \ldots, d\}$, such that all but finitely many blocks are mapped to 0.

Decompositions. Let w be a word and let $\mathcal{P}$ be a d-pattern. We say that w *admits a $\mathcal{P}$-decomposition* if w admits a decomposition $w = u_0v_1u_1v_2\cdots v_nu_n$ with $n \geqslant 0$ and such that either $n = 0$ and $\mathcal{P} = u_0 = w$, or $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$ and the following conditions are verified:

1. $\mathfrak{p} = (u_0, v_1)$ and $\mathfrak{s} = (v_n, u_n)$.
2. for all blocks $\mathfrak{b}$, if $f(\mathfrak{b}) < d$, then there are exactly $f(\mathfrak{b})$ indices i such that $(v_i, u_i, v_{i+1}) = \mathfrak{b}$.
3. for all blocks $\mathfrak{b}$, if $f(\mathfrak{b}) = d$, then there are at least d indices i such that $(v_i, u_i, v_{i+1}) = \mathfrak{b}$.

Sometimes, we just say $\mathcal{P}$-decomposition to mean $\mathcal{P}$-decomposition *of some word*. Let $\alpha : A^* \to M$ be a morphism into a monoid M, and let $s \in M$. A $\mathcal{P}$-decomposition is *(α, s)-compatible* if $\alpha(w) = s$ and $\alpha(u_0\cdots v_i) = \alpha(u_0\cdots v_i)\cdot\alpha(v_i)$, for $1 \leqslant i \leqslant n$. Similarly, if $\mathcal{A}$ is an automaton, we say that a $\mathcal{P}$-decomposition is *$\mathcal{A}$-compatible* if there is an accepting run for w and each infix v_i labels a loop in the run, for $1 \leqslant i \leqslant n$, as pictured in Figure 2 (where edges denote sequences of transitions).

Common Patterns. Let $d \in \mathbb{N}$ and M_1, M_2 be two monoids together with morphisms $\alpha_1 : A^* \to M_1$ and $\alpha_2 : A^* \to M_2$ and accepting sets $F_1 \subseteq M_1$, $F_2 \subseteq M_2$. We say that M_1, M_2 have a *common d-pattern* if there exist a d-pattern $\mathcal{P}$, two elements $s_1 \in F_1$, $s_2 \in F_2$, and two $\mathcal{P}$-decompositions of (possibly different) words that are respectively (α_1, s_1)-compatible and (α_2, s_2)-compatible. Similarly, if $\mathcal{A}_1, \mathcal{A}_2$ are automata, we say that $\mathcal{A}_1, \mathcal{A}_2$ have a *common d-pattern* if there exist a d-pattern $\mathcal{P}$ and two $\mathcal{P}$-decompositions of words that are respectively $\mathcal{A}_1$-compatible and $\mathcal{A}_2$-compatible. In particular, by the very definition, $\mathcal{A}_1$ and $\mathcal{A}_2$ have a common 1-pattern if there are successful paths in $\mathcal{A}_1, \mathcal{A}_2$ of the form shown in Figure 2 with the same *set* of triples (v_i, u_i, v_{i+1}).

A useful property about common patterns is that whether such a pattern exists only depends on the recognized languages, and not on the choice of $\mathcal{A}_1, \mathcal{A}_2, M_1, M_2$.

▶ **Proposition 1.** *Fix $d \in \mathbb{N}$. Let L_1, L_2 be regular languages and let $M_1, M_2, \mathcal{A}_1, \mathcal{A}_2$ be arbitrary monoids and automata recognizing L_1, L_2, respectively. Then M_1, M_2 have a common d-pattern if and only if $\mathcal{A}_1, \mathcal{A}_2$ have a common d-pattern.*

4.2 Separation Theorem for a Fixed Threshold

We can now state our main theorem for this section.

▶ **Theorem 2.** *Fix $d \in \mathbb{N}$. Let L_1, L_2 be regular languages and let $M_1, M_2, \mathcal{A}_1, \mathcal{A}_2$ be arbitrary monoids and automata recognizing L_1, L_2 respectively. Set $k = 4(|M_1||M_2| + 1)$. Then the following conditions are equivalent:*

1. *L_1 and L_2 are* LTT$[l,d]$*-separable for some l.*
2. *L_1 and L_2 are* LTT$[k,d]$*-separable.*
3. *The language $[L_1]_{\equiv_k^d}$ separates L_1 from L_2.*
4. *M_1, M_2 do not have a common d-pattern.*
5. *$\mathcal{A}_1, \mathcal{A}_2$ do not have a common d-pattern.*

Observe that Item 3b is essentially a *delay theorem* [22] for separation restricted to the case of LTT: we prove that the size of profiles (*i.e.*, infixes) that a potential separator needs to consider can be bounded by a function of the size of the monoids recognizing the languages. By restricting Theorem 2 to the case $d = 1$, we get the following separation theorem for LT.

▶ **Theorem 3.** *Let L_1, L_2 be regular languages and let $M_1, M_2, \mathcal{A}_1, \mathcal{A}_2$ be arbitrary monoids and automata recognizing L_1, L_2 respectively. Let $k = 4(|M_1||M_2| + 1)$. Then the following conditions are equivalent:*

1. *L_1 and L_2 are* LT*-separable.*
2. *L_1 and L_2 are* LTT$[k]$*-separable.*
3. *The language $[L_1]_{\equiv_k}$ separates L_1 from L_2.*
4. *M_1, M_2 do not have a common 1-pattern.*
5. *$\mathcal{A}_1, \mathcal{A}_2$ do not have a common 1-pattern.*

Theorem 2 and Theorem 3 yield algorithms for deciding LT- and LTT-separability for a fixed threshold. Indeed, the algorithm just tests all the finitely many LTT$[k,d]$ languages as potential separators. This brute-force approach yields a very costly procedure. It turns out that a better algorithm can be obtained from Items 3d and 3e (the proof is available in the full version of the paper). This yields the following corollary.

▶ **Corollary 4.** *Let $d \in \mathbb{N}$. It is decidable whether two given regular languages are* LTT$[l,d]$*-separable for some $l \in \mathbb{N}$. In particular, it is decidable whether they are* LT*-separable.*

More precisely, given NFAs $\mathcal{A}_1, \mathcal{A}_2$, deciding whether $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are LT*-separable is in* co-NEXPTIME*. It is* co-NP*-hard, even starting from DFAs.*

It remains to prove Theorem 2. The implications (3c) ⇒ (3b) ⇒ (3a) are immediate by definition. We now prove (3a) ⇒ (3e) ⇒ (3d) ⇒ (3c). The implication (3e) ⇒ (3d) is immediate from Proposition 1. The implication (3a) ⇒ (3e) is a consequence of the following proposition.

▶ **Proposition 5.** *Let $d \in \mathbb{N}$ and let $\mathcal{A}_1, \mathcal{A}_2$ be NFAs. If $\mathcal{A}_1, \mathcal{A}_2$ have a common d-pattern, then, for all $k \in \mathbb{N}$, there exist w_1, w_2 accepted respectively by $\mathcal{A}_1, \mathcal{A}_2$ such that $w_1 \equiv_k^d w_2$.*

An immediate consequence of Proposition 5 is that as soon as $\mathcal{A}_1, \mathcal{A}_2$ have a common d-pattern, the recognized languages cannot be separated by an LTT$[k,d]$ language for any k. This is exactly the contrapositive of (3a) ⇒ (3e). We now prove Proposition 5.

Proof of Prop. 5. Let $\mathcal{P}$ be a common d-pattern of $\mathcal{A}_1, \mathcal{A}_2$. If $\mathcal{P} = w \in A^*$, then by definition, $w \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, so it suffices to choose $w_1 = w_2 = w$. Otherwise, $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$ and there are w_1, w_2 having an $\mathcal{A}_1$-, respectively $\mathcal{A}_2$-compatible $\mathcal{P}$-decomposition. Let $w_1 = u_0 v_1 u_1 v_2 \cdots v_n u_n$ and $w_2 = u'_0 v'_1 u'_1 v'_2 \cdots v'_m u'_m$ be these decompositions. For $k \in \mathbb{N}$, set

$$\begin{aligned} w_1 &= u_0 (v_1)^{k(d+1)} u_1 (v_2)^{k(d+1)} \cdots (v_n)^{k(d+1)} u_n \\ w_2 &= u'_0 (v'_1)^{k(d+1)} u'_1 (v'_2)^{k(d+1)} \cdots (v'_m)^{k(d+1)} u'_m \end{aligned}$$

By definition of compatibility, $w_1 \in L(\mathcal{A}_1)$ and $w_2 \in L(\mathcal{A}_2)$. From the fact that $(\mathfrak{p}, f, \mathfrak{s})$ is a d-pattern, it then follows that $w_1 \equiv_k^d w_2$. ◀

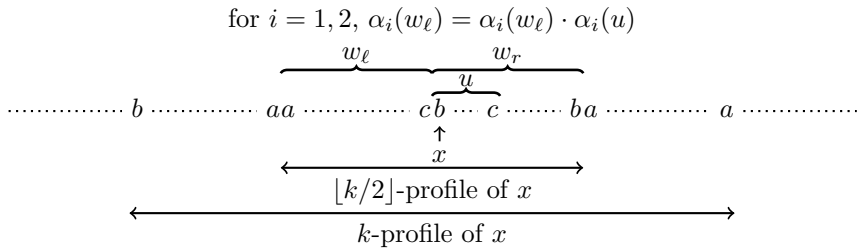


Figure 3 A position x admitting a k-loop u: $\alpha_i(w_\ell) = \alpha_i(w_\ell \cdot u)$, for $i = 1, 2$.

The remaining and most difficult direction, (3d) $\Rightarrow$ (3c) is a consequence of the next proposition whose proof is outlined in the next subsection.

▶ **Proposition 6.** *Let $\alpha_1 : A^* \to M_1$ and $\alpha_2 : A^* \to M_2$ be morphisms, and $k = 4(|M_1||M_2| + 1)$. Let $d \in \mathbb{N}$ and let w_1, w_2 be words such that $w_1 \equiv_k^d w_2$. Then there exists a d-pattern $\mathcal{P}$, an $(\alpha_1, \alpha_1(w_1))$-compatible $\mathcal{P}$-decomposition, and an $(\alpha_2, \alpha_2(w_2))$-compatible $\mathcal{P}$-decomposition.*

Before explaining how to show Proposition 6, let us explain how to conclude the proof of Theorem 2. We prove the contrapositive of (3d) $\Rightarrow$ (3c). If Item 3c does not hold, then by definition there must exist $w_1 \in L_1$ and $w_2 \in L_2$ such that $w_1 \equiv_k^d w_2$. If $w_1 = w_2$, $L_1 \cap L_2 \neq \emptyset$, therefore, M_1, M_2 have a common d-pattern. Otherwise, by Proposition 6 we get a d-pattern $(\mathfrak{p}, f, \mathfrak{s})$. Since $w_1 \in L_1$ and $w_2 \in L_2$, the d-pattern $(\mathfrak{p}, f, \mathfrak{s})$ is common to both M_1 and M_2, which ends the proof.

4.3 Proof of Proposition 6

We set w_1, w_2, k and d as in the statement of the proposition. Observe first that if $w_1 = w_2 = w$, then it suffices to take $\mathcal{P} = w$. Therefore, we suppose for the remainder of the proof that $w_1 \neq w_2$. We proceed as follows: we construct two new words w_1', w_2' from w_1, w_2 admitting respectively a $(\alpha_1, \alpha_1(w_1))$-compatible $\mathcal{P}$-decomposition and a $(\alpha_2, \alpha_2(w_2))$-compatible $\mathcal{P}$-decomposition, for some d-pattern $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$. In this outline, we only provide the construction, the proof of correctness is available in the full version of the paper.

The construction of w_1', w_2', amounts to duplicating infixes in w_1, w_2 verifying special properties. We first define these special infixes that we will call k-loops.

k-loops. Let $w \in A^*$, x be a position in w, and (w_ℓ, w_r) be the $\lfloor k/2 \rfloor$-profile of x. We say that x *admits a k-loop* if there exists a nonempty prefix u of w_r such that $\alpha_1(w_\ell) = \alpha_1(w_\ell \cdot u)$ and $\alpha_2(w_\ell) = \alpha_2(w_\ell \cdot u)$. In this case, we call the smallest such u *the k-loop of x*. See Figure 3.

For our construction to work, we need k-loops to have three properties that we state now. The first two are simple facts that are immediate from the definition: k-loops are determined by profiles and can be duplicated without modifying the image of the word under α.

▶ **Fact 7.** *Let x be a position. Whether x admits a k-loop, and if so, which k-loop x admits, only depends on the $\lfloor k/2 \rfloor$-profile of x.*

▶ **Fact 8.** *Let w be a word and let x be a position within w such that x admits a k-loop u. Then for $i = 1, 2$, $\alpha_i(w[0, x]) = \alpha_i(w[0, x]) \cdot \alpha_i(u)$.*

The last property we need is that k-loops occur frequently in words, *i.e.*, at least one of $\lfloor k/4 \rfloor$ consecutive positions must admit a k-loop. This follows from pumping arguments:

▶ **Lemma 9.** *Let w be a word and let $x_1, \ldots, x_{\lfloor k/4 \rfloor}$ be $\lfloor k/4 \rfloor$ consecutive positions in w. Then, there exists at least one position x_i with $i < \lfloor k/4 \rfloor$ that admits a k-loop.*

Construction of w'_1, w'_2. We can now construct w'_1 and w'_2. If w, u are words and x is a position of w, the *word constructed by inserting u at position x* is the word $w[0,x] \cdot u \cdot w[x,|w|]$. From w_1 (resp. w_2), we construct w'_1 (resp. w'_2) by inserting simultaneously all infixes $(u_x)^{k'}$ in w_1 (resp. w_2) at any position x that admits a k-loop, and where u_x is the k-loop of x. Using Fact 7, Fact 8 and Lemma 9 one can then verify that w'_1 admits a $(\alpha_1, \alpha_1(w_1))$-compatible $\mathcal{P}$-decomposition and w'_2 admits a $(\alpha_2, \alpha_2(w_2))$-compatible $\mathcal{P}$-decomposition, for some d-pattern $\mathcal{P} = (\mathfrak{p}, f, \mathfrak{s})$. The proof is available in the full version of the paper.

5 Separation by LTT Languages

This section is devoted to LTT. Again, our theorem actually contains several results. In the case of LTT, two parameters are involved: the size k of profiles and the counting threshold d. The first result in our theorem states that the bound on k of Theorem 2 still holds for full LTT. This means that two languages are LTT-separable if and only if there exists some counting threshold d such that they are LTT$[k,d]$-separable with the same bound k as in Theorem 2. It turns out that this already yields an algorithm for testing LTT-separability. The algorithm relies on the decidability of Presburger arithmetic and is actually adapted in a straightforward manner from an algorithm of [3] for deciding membership in LTT.

While this first result gives an algorithm for testing separability, it gives no insight about an actual separator. Indeed, the procedure does not produce the actual counting threshold d. This is obtained in the second part of our theorem: we prove that two languages are LTT-separable if and only if they are LTT$[k,d]$-separable, where k is as defined in Theorem 2, and d is bounded by a function of the size of the monoids (or automata) recognizing the input languages. Note that this result also gives another (brute-force) algorithm for testing LTT-separability. We now state our theorem. Recall that A_k denotes the set of k-profiles.

▶ **Theorem 10.** *Let L_1, L_2 be regular languages and let $M_1, M_2, \mathcal{A}_1, \mathcal{A}_2$ be arbitrary monoids and automata recognizing L_1, L_2. Set n to be either $\max(|M_1|, |M_2|) + 1$ or $\max(|\mathcal{A}_1|, |\mathcal{A}_2|) + 1$. Let $k = 4(|M_1||M_2| + 1)$ and $d = (|A_k|n)^{|A_k|}$. Then, the following conditions are equivalent:*

1. *L_1 and L_2 are* LTT*-separable.*
2. *There exists $d' \in \mathbb{N}$ such that L_1 and L_2 are* LTT$[k,d']$*-separable.*
3. *There exists $d' \in \mathbb{N}$ such that M_1, M_2 do not have a common d'-pattern.*
4. *There exists $d' \in \mathbb{N}$ such that $\mathcal{A}_1, \mathcal{A}_2$ do not have a common d'-pattern.*
5. *L_1 and L_2 are* LTT$[k,d]$*-separable.*
6. *The language $[L_1]_{\equiv_k^d}$ separates L_1 from L_2.*

Observe that decidability of LTT-separability is immediate from Item 3e by using the usual brute-force algorithm. As it was the case for a fixed counting threshold, this algorithm is slow. In the full version of the paper, we obtain a faster algorithm by using Items 3c and 3d.

▶ **Corollary 11.** *It is decidable whether two given regular languages are* LTT*-separable. More precisely, given NFAs $\mathcal{A}_1, \mathcal{A}_2$, deciding whether $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ are* LTT*-separable is in* 2-Expspace*. It is* co-Np*-hard, even starting from DFAs.*

By definition, a language is LTT if it is LTT$[k,d]$ for some natural numbers k, d. Hence, the equivalence between Items 3a, 3b, 3c and 3d is an immediate consequence of Theorem 2.

Therefore, we only need to prove Items 3e and 3f, *i.e.*, the bound on the threshold d. Unfortunately, these are exactly the items we need for Corollary 11. However, we will prove that by reusing an algorithm of [3], Corollary 11 can also be derived directly from Item 3b.

We now explain how to derive the first part of Corollary 11 from Item 3b without relying on the actual bound on the counting threshold. The bound itself is proved in the full version.

5.1 Decidability of LTT-separability as a consequence of Theorem 2

As we explained, the equivalence of Item 3b to LTT-separability is immediate from Theorem 2. We explain how to combine this fact with an algorithm of [3] to obtain decidability directly.

In [3], it is proved that once k is fixed, Parikh's Theorem [15] can be used to prove that whether a language is LTT$[k, d]$ for some d can be rephrased as a computable Presburger formula. Decidability of membership in LTT can then be reduced to decidability of Presburger Arithmetic. For achieving this, two ingredients were needed: *a*) a bound on k, and *b*) the translation to Presburger arithmetic. It turns out that in [3], only the proof of *a*) was specific to membership. On the other hand, separation was already taken care of in *b*), because the intuition behind the Presburger formula was testing *separability* between the input language and its complement. In our setting, we have already replaced *a*), *i.e.*, bounding k, by Item 3b. Therefore, the argument can be generalized. We explain in the remainder of this subsection how to construct the Presburger formula. The argument makes use of the notion of commutative image, which we now recall.

The *commutative image* of a word $w \in A^*$, denoted $\pi(w)$, is a vector of length $|A|$ of natural numbers counting, for all $a \in A$, how many occurrences of a there are in w. This notion can be easily generalized in order to count profiles rather than just letters. Let $k \in \mathbb{N}$. The *k-image* of w, $\pi_k(w)$, is a vector of length $|A_k|$ of numbers counting for every k-profile (w_ℓ, w_r) the number of positions in w with k-profile (w_ℓ, w_r). If L is a language, the *k-image* of L, $\pi_k(L)$ is the set $\{\pi_k(w) \mid w \in L\} \subseteq \mathbb{N}^{A_k}$. The definition of $\equiv_k^d$ yields the following fact.

▶ **Fact 12.** *Let $w, w' \in A^*$ and $k, d \in \mathbb{N}$. Then $w \equiv_k^d w'$ if and only if $\pi_k(w)$ and $\pi_k(w')$ are equal componentwise up to threshold d.*

A well-known result about commutative images is Parikh's Theorem [15], which states that if L is context-free (so in particular if L is regular), then $\pi(L)$ is semilinear, *i.e.*, Presburger definable [6]. As explained in [3], Parikh's Theorem extends without difficulty to k-images.

▶ **Theorem 13.** *Let L be a context-free language and let $k \in \mathbb{N}$. Then $\pi_k(L)$ is semilinear. Moreover, a Presburger formula for this semilinear set can be computed.*

Proof. For $k = 1$, this is Parikh's Theorem. When $k > 1$, let $L' \subseteq A_k^*$ such that $w' \in L'$ if there exists $w \in L$ of the same length, and such that a position in w' is labeled by the k-profile of the same position in w. One can verify that L' is (effectively) context-free, and that the k-image of L is the commutative image of L', which is semilinear by Parikh's Theorem. ◀

We can now explain how to decide LTT-separability. By Item 3b in Theorem 10, L_1, L_2 are LTT-separable if and only if they are LTT$[k, d]$-separable for $k = 4(|M_1||M_2| + 1)$ (where M_1, M_2 are monoids recognizing L_1, L_2) and some natural number d. Therefore, whether L_1, L_2 are LTT-separable can be rephrased as follows: does there exist some threshold d such that there exist no words $w_1 \in L_1, w_2 \in L_2$ such that $w \equiv_k^d w'$? By Fact 12, this can be expressed in terms of k-images: does there exist a threshold d such that there exist no vectors of natural numbers $\bar{x}_1 \in \pi_k(L_1), \bar{x}_2 \in \pi_k(L_2)$ that are equal up to threshold d? It follows from Theorem 13 that the above question can be expressed as a computable Presburger formula. Decidability of LTT-separability then follows from decidability of Presburger Arithmetic.

6 The Case of Context-Free Languages

In order to prove decidability of LTT-separability for regular languages, we needed three ingredients: Parikh's Theorem, decidability of Presburger Arithmetic and Item 3b in Theorem 10. Since Parikh's Theorem holds not only for regular languages but also for context-free languages, we retain at least two of the ingredients in the context-free setting.

In particular, we can reuse the argument of Section 5 to prove that once the size k of the profiles is fixed, separability by LTT is decidable for context-free languages. For any fixed $k \in \mathbb{N}$, we write $\mathrm{LTT}[k] = \bigcup_{d\in\mathbb{N}} \mathrm{LTT}[k,d]$.

▶ **Theorem 14.** *Let L_1, L_2 be context-free languages and $k \in \mathbb{N}$. It is decidable whether L_1, L_2 are $\mathrm{LTT}[k]$-separable.*

An interesting consequence of Theorem 14 is that LTT[1]-separability of context-free languages is decidable. A language is LTT[1] if and only if it can be defined by a first-order logic formula that can only test equality between positions, but not ordering. This result is surprising since membership of a context-free language in this class is undecidable. We give a proof of this fact below, which is a simple adaptation of the proof of Greibach's Theorem (which is in particular used to prove that regularity of a context-free language is undecidable).

▶ **Theorem 15.** *Let L be a context-free language. It is undecidable to test whether $L \in \mathrm{LTT}[1]$.*

Proof. We reduce universality of context-free languages to this membership problem. Fix L a context-free language over A and let $\# \notin A$. Let $K \notin \mathrm{LTT}[1]$ be some context-free language and set $L_1 = (K \cdot \# \cdot A^*) \cup (A^* \cdot \# \cdot L)$. Clearly, a context-free grammar for L_1 can be computed from a context-free grammar for L. We show that $L = A^*$ iff $L_1 \in \mathrm{LTT}[1]$.

If $L = A^*$, then $L_1 = A^* \cdot \# \cdot A^* \in \mathrm{LTT}[1]$. Conversely, assume that $L_1 \in \mathrm{LTT}[1]$, and suppose by contradiction that $L \neq A^*$. Pick $w \in A^*$ such that $w \notin L$. By definition, $K = \{u \mid u\#w \in L_1\}$. One can verify that LTT[1] is closed under right residual. Therefore, $K = L_1(\#w)^{-1} \in \mathrm{LTT}[1]$ which is a contradiction by definition of K. ◀

Theorems 14 and 15 may seem contradictory. Indeed in the setting of regular languages, membership can be reduced to separability (a language belongs to a class if the class can separate it from its complement). However, context-free languages are not closed under complement, which makes the reduction false in this larger setting.

An interesting question is whether decidability extends to full LT and LTT-separability of context-free languages. This would also be surprising since membership of a context-free language in LT or LTT is undecidable. Such a result would require to generalize our third ingredient, Item 3b in Theorem 10, to context-free languages. This means that we would need a method for computing a bound on the size of the infixes that a potential separator has to consider. It turns out that this is not possible.

▶ **Theorem 16.** *Let L_1, L_2 be context-free languages. It is undecidable to test whether L_1, L_2 are* LT*-separable. It is undecidable to test whether L_1, L_2 are* LTT*-separable.*

It was already known [23] that separability by a regular language is undecidable for context-free languages. The proof of Theorem 16 is essentially the same since the reduction provided in [23] actually works for any class of regular separators that contains languages of the form $K_1A^* \cup K_2$ where K_1, K_2 are finite languages. Since this is clearly the case for both LT and LTT, Theorem 16 follows.

7 Conclusion

We proved separation theorems for both LT and LTT. In both cases, algorithms to test separability, in co-NEXPTIME and 2-EXPSPACE respectively, are derived from these theorems. Another contribution is a description of possible separators, given by bounds defining them.

Several questions remain open in this line of research. A first one is to obtain tight complexity bounds for both classes. While we have co-NEXPTIME and 2-EXPSPACE upper bounds for LT and LTT respectively, we have only co-NP lower bounds. The upper bounds rely on a reduction to the case $k = 1$, *i.e.*, a translation to the special case when the size of infixes is fixed to 1. This translation is exponential wrt. the size of the input automata. Improving the upper bounds would likely require improving this reduction.

Another question is to consider other fragments for separability. A natural generalization of LTT is LTT+MOD, in which infixes can now also be counted modulo constants. The most interesting fragment is of course full first-order logic. While the problem was shown decidable [7, 8], the proofs rely on involved algebraic techniques and give an algorithm that provides only a yes/no answer. Furthermore, the techniques bring no insight on the expressive power of first-order logic. It remains a challenging open problem to obtain a combinatorial proof that FO-separability is decidable, as well as a description of a separator.

References

1 J. Almeida. Some algorithmic problems for pseudovarieties. *Publ. Math. Debrecen*, 54(suppl.):531–552, 1999. Automata and formal languages, VIII (Salgótarján, 1996).

2 D. Beauquier and J. E. Pin. Languages and scanners. *Theor. Comp. Sci.*, 84(1):3–21, 1991.

3 M. Bojańczyk. A new algorithm for testing if a regular language is locally threshold testable. *Inf. Process. Lett.*, 104(3):91–94, 2007.

4 J. Brzozowski and I. Simon. Characterizations of locally testable events. *Discrete Mathematics*, 4(3):243–271, 1973.

5 W. Czerwinski, W. Martens, and T. Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proc. of ICALP'13*, pages 150–161, 2013.

6 S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

7 K. Henckell. Pointlike sets: the finest aperiodic cover of a finite semigroup. *J. Pure Appl. Algebra*, 55(1-2):85–126, 1988.

8 K. Henckell, J. Rhodes, and B. Steinberg. Aperiodic pointlikes and beyond. *Internat. J. Algebra Comput.*, 20(2):287–305, 2010.

9 T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *Proc. of POPL'04*, pages 232–244. ACM, 2004.

10 J. Leroux. Vector addition systems reachability problem (a simpler solution). In *The Alan Turing Centenary Conference, Turing-100*, volume 10, pages 214–228, 2012.

11 E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.

12 K. L. McMillan. Applications of Craig interpolants in model checking. In N. Halbwachs and L. D. Zuck, editors, *Proc. of TACAS'05*, pages 1–12. Springer, 2005.

13 R. McNaughton. Algebraic decision procedures for local testability. *Math. Systems Theory*, 8(1):60–76, 1974.

14 R. McNaughton and S. Papert. *Counter-free automata*. The M.I.T. Press, 1971.

15 R. J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.

16 J. E. Pin. Syntactic semigroups. In G. Rozenberg and A. Salomaa, editors, *Handbook of language theory, Vol. I*, pages 679–746. Springer, 1997.

17 J. E. Pin. Expressive power of existential first-order sentences of Büchi's sequential calculus. *Discrete Math.*, 291(1–3):155–174, 2005.

18 T. Place, L. van Rooijen, and M. Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Proc. of MFCS'13*, 2013.

19 M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

20 B. Steinberg. On pointlike sets and joins of pseudovarieties. *IJAC*, 8(2), 1998.

21 B. Steinberg. A delay theorem for pointlikes. *Sem. Forum*, 63(3):281–304, 2001.

22 H. Straubing. Finite semigroup varieties of the form $V * D$. *J. Pure Appl. Algebra*, 36, 1985.

23 T. G. Szymanski and J. H. Williams. Noncanonical extensions of bottom-up parsing techniques. *SIAM J. Comput.*, 5(2), 1976.

24 D. Thérien and A. Weiss. Graph congruences and wreath products. *J. Pure Appl. Algebra*, 36:205–215, 1985.

25 W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25(3):360–376, 1982.

26 M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. of LICS'86*, pages 332–344. IEEE Computer Society, 1986.

27 Y. Zalcstein. Locally testable languages. *J. Comput. Syst. Sci.*, 6(2):151–167, 1972.

On the Structure and Complexity of Rational Sets of Regular Languages

Andreas Holzer[1], Christian Schallhart[2], Michael Tautschnig[3], and Helmut Veith[1]

1 **Vienna University of Technology, Austria**
2 **University of Oxford, UK**
3 **Queen Mary, University of London, UK**

Abstract

In the recently designed and implemented test specification language FQL, relevant test goals are specified as regular expressions over program locations. To transition from single test goals to test suites, FQL describes suites as regular expressions over finite alphabets where each symbol corresponds to a regular expression over program locations. Hence, each word in a test suite expression yields a test goal specification. Such test suite specifications are in fact rational sets of regular languages (RSRLs). We show closure properties of general and finite RSRLs under common set theoretic operations. We also prove complexity results for checking equivalence and inclusion of star-free RSRLs and for checking whether a regular language is a member of a general or star-free RSRL. As the star-free (and thus finite) case underlies FQL specifications, the closure and complexity results provide a systematic foundation for FQL test specifications.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Rational Sets, Regular Languages, Test Specification in FQL, Closure Properties, Decision Problems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.377

1 Introduction

Despite the success of model checking and theorem proving, software testing has a dominant role in industrial practice. In fact, state-of-the-art development guidelines such as the avionic standard DO-178B [27] are heavily dependent on test coverage criteria. It is therefore quite surprising that the formal specification of coverage criteria has been a blind spot in the formal methods and software engineering communities for a long time.

In a recent thread of papers [14, 12, 17, 16, 15, 6], we have addressed this situation and introduced the FShell Query Language (FQL) to specify and tailor coverage criteria, together with FShell, a tool to generate matching test suites for ANSI C programs. At the semantic core of FQL, test goals are described as regular expressions whose alphabet are the edges of the program control flow graph (CFG). For example, to cover a particular CFG edge c, one can use the regular expression $\Sigma^\star\ c\ \Sigma^\star$. Importantly, however, a coverage criterion usually induces not just a single test goal, but a (possibly large) number of test goals – e.g. *all* basic blocks of a program. FQL therefore employs regular languages which can express sets of regular expressions. To this end, the alphabet contains not only the CFG edges but also *postponed regular expressions* over these edges, written within quotes.

For example, "$\Sigma^\star$" $(a+b+c+d)$ "$\Sigma^\star$" describes the language {"$\Sigma^\star$" a "$\Sigma^\star$", "$\Sigma^\star$" b "$\Sigma^\star$", "$\Sigma^\star$" c "$\Sigma^\star$", "$\Sigma^\star$" d "$\Sigma^\star$"}. Each of these words is a regular expression that will then serve as a test goal. Following [1], we call such languages *rational sets of regular languages (RSRL)*.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 377–388

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

The goal of this paper is to initiate a systematic study of RSRLs from a theoretical point of view, considering closure properties and complexity of common set-theoretic operations. Thus, this paper is a first step towards a systematic foundation of FQL. In particular, a good understanding of set-theoretic operations is necessary for systematic algorithmic optimization and manipulation of test specifications. First results on query optimization for FQL have been obtained in [6].

A rational set of regular languages is given by a regular language K over alphabet Δ, and a *regular language substitution* $\varphi : \Delta \to 2^{\Sigma^\star}$, mapping each symbol $\delta \in \Delta$ to a regular language $\varphi(\delta)$ over alphabet Σ. We extend φ to words $w \in \Delta^+$ with $\varphi(\delta \cdot w) = \varphi(\delta) \cdot \varphi(w)$, and set $\varphi(L) = \bigcup_{w \in L} \varphi(w)$ for $L \subseteq \Delta^+$. The class of rational sets of a monoid $(M, \cdot, e)$ is the smallest subclass of M such that *(i)* $\emptyset$ is a rational set, *(ii)* each singleton set $\{m\}$ for $m \in M$ is a rational set, and if N_1 and N_2 are rational sets *(iii)* then $N_1 \cdot N_2$ is a rational set where $\cdot$ on rational sets is defined by the point-wise application of the monoid's $\cdot$ operation, *(iv)* $N_1 \cup N_2$ is a rational set, and *(v)* $N_1^\star$ is a rational set [9, 22].

▶ **Definition 1** (Rational Sets of Regular Languages, RSRLs [1])**.** Given a finite alphabet Σ, the *rational sets of regular languages* are the rational sets over the monoid $(2^{\Sigma^\star}, \cdot, \{\varepsilon\})$, where ε denotes the empty word. We represent a rational set of regular languages $\mathcal{R}$ as tuple (K, φ), where $K \subseteq \Delta^+$ is a regular language over a finite alphabet Δ, and φ is a regular language substitution $\varphi : \Delta \to 2^{\Sigma^\star}$, such that $\mathcal{R} = \{\varphi(w) \mid w \in K\}$. We say that RSRL $\mathcal{R}$ is *Kleene star free,* if there exists $(K, \varphi) = \mathcal{R}$ such that K is finite (and hence Kleene-star free).

Depending on context, we refer to $\mathcal{R}$ as a set of languages or as a pair (K, φ), but we always write $L \in \mathcal{R}$ iff $\exists w \in K : L = \varphi(w)$. Consider the above specification "$\Sigma^\star$" $(a+b+c+d)$ "$\Sigma^\star$" over base alphabet $\Sigma = \{a, b, c, d\}$. To represent this specification as RSRL $\mathcal{R} = (K, \varphi)$, we set $\Delta = \{\delta_{\Sigma^\star}\} \cup \Sigma$, containing a fresh symbol $\delta_{\Sigma^\star}$ for the quoted expression "$\Sigma^\star$". We set $K = L(\delta_{\Sigma^\star}\ (a + b + c + d)\ \delta_{\Sigma^\star})$ with $\varphi(\delta_{\Sigma^\star}) = \Sigma^\star$ and $\varphi(\sigma) = \{\sigma\}$ for $\sigma \in \Sigma$. Thus K contains the words $\delta_{\Sigma^\star}\ a\ \delta_{\Sigma^\star}, \ldots$ with $\varphi(\delta_{\Sigma^\star}\ a\ \delta_{\Sigma^\star}) = L(\Sigma^\star\ a\ \Sigma^\star) \in \mathcal{R}$, as desired.

Note that the RSRL above is finite with exactly four elements. This is of course not atypical: in concrete testing applications, FQL generates finite sets of test goals, since it relies on *Kleene star free* RSRLs only. For future applications, however, it is well possible to consider infinite sets of test goals e.g. for unbounded integer and real valued variables or for path coverage criteria which are either matched partially, or by abstract executions. In this paper, we are therefore considering the general, finite, and Kleene star free case.

▶ **Example 2.** Consider the alphabets $\Delta = \{\delta_1, \delta_2\}$ and $\Sigma = \{a, b\}$. Then, **(1)** with $\varphi(\delta_1) = L(a^\star)$, $\varphi(\delta_2) = \{ab\}$, and $K = L(\delta_1 \delta_2^\star \delta_1)$, we obtain the rational set of regular languages $\{L(a^\star(ab)^i a^\star) \mid i \in \mathbb{N}\}$; **(2)** with $\varphi(\delta_1) = L(a^\star)$, $\varphi(\delta_2) = \{a\}$, and $K = L(\delta_1 \delta_2^\star)$, we obtain $\varphi(w_1) \supset \varphi(w_2)$ for all $w_1, w_2 \in K$ with $|w_1| < |w_2|$; **(3)** with $\varphi(\delta_1) = \{\varepsilon, a\}$, $\varphi(\delta_2) = \{aa\}$, and $K = L(\delta_1 \delta_2^\star)$, we have $|\varphi(w)| = 2$ and $\varphi(w) \cap \varphi(w') = \emptyset$ for all $w \neq w' \in K$.

In the finite case we make an additional distinction for the subcase where the regular expressions in Δ, i.e., the set of postponed regular expressions, are fixed. This has practical relevance, because in the context of FQL, the results of the operations on RSRL will be better readable by engineers if Δ is unchanged.

Contributions and Organization

In Section 3, we show *closure properties* for general and finite RSRLs, considering the operators product, Kleene star, complement, union, intersection, set difference, and symmetric difference.

We also consider the case of finite RSRLs with a fixed language substitution φ, as this case is of particular interest for testing applications. In Section 4, we prove the *complexity results* of the decision problems equivalence, inclusion, and membership for Kleene star free RSRLs. To prove an upper bound on the complexity of the membership problem, we expand the decidability proof in [1] and give a first complete and explicit algorithm for the problem. We close in Section 5 in discussing how our results reflect back to design decisions for FQL.

2 Related Work

Afonin et al. [1] introduced RSRLs and studied the decidability of whether a regular language is contained in an RSRL and the decidability of whether an RSRL is finite. Although Afonin et al. shortly discuss possible upper bounds for the membership decision problem, their analysis is incomplete due to gaps in their algorithmic presentation (see also a more detailed discussion in Section 4.5). Closely connected to the membership problem is the question, whether a regular language L is expressible via a combination of a given set of regular languages L_i. Motivated by query rewriting for graph databases, Calvanese et al. [7] show the complexity of determining the maximal rewriting of a regular language L with given regular languages L_i. In earlier work, Hashiguchi [11] shows that it is decidable whether a regular language L is expressible via a finite application of a subset of the regular operators concatenation, union, and star to regular languages L_i. Afonin et al. [1] realized that distance automata [10] enable a decision algorithm for the membership problem for RSRL. Although this construction relies on distance automata, the properties analyzed by Krob [23] and Colcombet and Daviaud [8] are not applicable in our context. Kirsten [20, 21] generalizes distance automata to distance desert automata and uses these automata to show the first complexity result for determining whether a regular language is of a certain star height. Berstel [5] surveys closure properties of rational and recognizable subsets of monoids and thereby also the relationship between rational and recognizable subsets. Yet, most stated results do not apply to RSRLs, hence we investigate closure properties of RSRLs. Pin [26] introduced the term *extended automata* for RSRLs as an example of recognizable languages that can be characterized by constraint systems over symbols and substrings occurring in words of the language, but he did not further investigate any of their properties. In our own related work on FQL [14, 13, 12, 17, 16, 15, 6], we deal with practical issues arising in test case generation. Beyond RSRLs, FQL provides an additional language layer to extract suitable alphabets from the programs e.g. referring with a single symbol to all basic blocks of the program under scrutiny.

Let us finally discuss other work whose terminology is similar to RSRLs without direct technical relation. Barceló et al. define *rational relations*, which are relations between words over a common alphabet, whereas we consider sets of regular languages [3]. Barceló et al. also investigate *parameterized regular languages* [4], where words are obtained by replacing variables in expressions with alphabet symbols. *Metaregular languages* deal with languages recognized by automata with a time-variant structure [2, 28]. Lattice Automata [24] only consider lattices that have a unique complement element, whereas RSRLs are not closed under complement (no RSRL has an RSRL as complement).

3 Closure Properties

We investigate the closure properties of RSRLs, considering standard set theoretic operators, such as union, intersection, and complement, and variants thereof, fitting RSRLs. In

particular, we apply those operators also to pairs in the *Cartesian product* of RSRLs, and *point-wise* to each element in an RSRL and another given regular language.

▶ **Definition 3** (Operations on RSRL). Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be RSRLs and let R be a regular language. Then, we define the following operations on RSRLs:

Operation	Definition		
Product	$\mathcal{R}_1 \cdot \mathcal{R}_2$	$=$	$\{L_1 \cdot L_2 \mid L_1 \in \mathcal{R}_1, L_2 \in \mathcal{R}_2\}$
Kleene Star	$\mathcal{R}_1^\star$	$=$	$\bigcup_{i\in\mathbb{N}} \mathcal{R}_1^i$
Point-wise	$\dot{\mathcal{R}_1^\star}$	$=$	$\{L^\star \mid L \in \mathcal{R}_1\}$
Complement	$\overline{\mathcal{R}_1}$	$=$	$\{L \subseteq 2^{\Sigma^\star} \mid L \notin \mathcal{R}_1\}$
Point-wise	$\dot{\overline{\mathcal{R}_1}}$	$=$	$\{\overline{L} \mid L \in \mathcal{R}_1\}$
Binary Operators	$\mathcal{R}_1 \cap \mathcal{R}_2,\ \mathcal{R}_1 \cup \mathcal{R}_2,\ \mathcal{R}_1 - \mathcal{R}_2$		(standard def.)
Point-wise	$\mathcal{R}_1 \dot{\cup} / \dot{\cap} / \dot{-} R$	$=$	$\{L \cup / \cap / - R \mid L \in \mathcal{R}_1\}$
Cartesian	$\mathcal{R}_1 \bowtie / \boxtimes / \overset{\times}{-} \mathcal{R}_2$	$=$	$\{L_1 \cup / \cap / - L_2 \mid L_1 \in \mathcal{R}_1, L_2 \in \mathcal{R}_2\}$
Symmetric Difference	$\mathcal{R}_1 \Delta \mathcal{R}_2$	$=$	$\{L \mid L \in ((\mathcal{R}_1 \cup \mathcal{R}_2) - (\mathcal{R}_1 \cap \mathcal{R}_2))\}$

We analyze *three different classes* of RSRLs for being closed under these operators: **(1)** General RSRLs, **(2)** finite RSRLs, and **(3)** finite RSRLs with a fixed language substitution φ. For closure properties, we do *not distinguish* between Kleene star free and finite RSRLs, since every finite RSRL is expressible as Kleene star free RSRL (however, given an RSRL with Kleene star, it is non-trivial to decide whether the given RSRL is finite or not [1]). Therefore, all closure properties for finite RSRLs apply to Kleene star free RSRLs as well. Hence, cases **(2-3)** correspond to FQL. Case **(3)** is relevant for usability in practice, allowing to apply the corresponding operators without constructing a new language substitution. This does not only significantly reduce the search space but also provides more intuitive results to users.

▶ **Theorem 4** (Closure Properties of RSRL). *The following table summarizes the closure properties for RSRLs.*

Operation	Closure Property		
	General	Finite RSRLs	
(+ closed − not closed ? unknown)		General	Fixed Subst.
Product	+	+	+
Kleene Star	+	−	−
Point-wise	−	+	−
Complement	−	−	−
Point-wise	−	+	−
Union	+	+	+
Point-wise	−	+	−
Cartesian	−	+	−
Intersection	?	+	+
Point-wise	−	+	−
Cartesian	−	+	−
Difference	?	+	+
Point-wise	−	+	−
Cartesian	−	+	−
Symmetric	?	+	+

As most proofs for Theorem 4 are straightforward, we only exemplify the proofs for point-wise operators using the point-wise union operator (cf. Proposition 6) and show the rest of the proofs in an extended online version of this paper [18]. The following set of regular languages is not an RSRL and we use it to prove the non-closure of RSRLs under the point-wise union operator.

▶ **Example 5.** Consider the set $\mathcal{M} = \{\{b\} \cup \{a^i \mid 1 \leq i \leq n+1\} \mid n \in \mathbb{N}\} \subseteq 2^{\{a,b\}^\star}$. $\mathcal{M}$ contains infinitely many languages, therefore, any RSRL $\mathcal{R} = (K, \varphi)$, with $\mathcal{M} = \mathcal{R}$, requires a regular language K containing infinitely many words. By L_n we denote the set $\{b\} \cup \{a^i \mid 1 \leq i \leq n+1\}$. Then, $L_0 \subsetneq L_1 \subsetneq \ldots L_{i-1} \subsetneq L_i \subsetneq L_{i+1} \subsetneq \ldots$. There must be a word $w = uvz \in K$ such that $uv^iz \in K$, for all $i \geq 1$ (cf. pumping lemma for regular languages [19]). Furthermore, there must be such a word $w = uvz$ such that $\varphi(u) \neq \emptyset$, $\varphi(v) \neq \emptyset$, $\varphi(v) \neq \{\varepsilon\}$, and $\varphi(z) \neq \emptyset$. This is due to the fact that we have to generate arbitrary long words a^i. We can assume that $b \notin \varphi(v)$ because otherwise $b^i \in \varphi(v^i)$, for all $i \geq 1$. Therefore, $a^k \in \varphi(v)$ for some $k \geq 1$. Since $b \in \varphi(uvz)$ has to be true, we can assume w.l.o.g. that $b \in \varphi(u)$. But, then $ba^k \ldots \in \varphi(uvz)$. This is a contradiction to the fact that, for all $n \geq 1$, $ba^k \ldots \notin L_n$.

▶ **Proposition 6** (Closure of Point-wise Union)**.** *The set $\mathcal{R}_1 \cup R$ is, in general, not an RSRL.*

Proof. Let $\mathcal{R}_1 = (L(\delta_1\delta_2^\star), \varphi)$ with $\varphi(\delta_1) = \{a\}$ and $\varphi(\delta_2) = L(a + \varepsilon)$ and let $R = \{b\}$. Then, $\mathcal{R}_1 \cup R = \{\{b\} \cup \{a^i \mid 1 \leq i \leq n+1\} \mid n \in \mathbb{N}\}$ which is not an RSRL, as shown in Example 5. ◀

4 Decision Problems

Given a regular language $R \subseteq \Sigma^\star$ and an RSRL $\mathcal{R} = (K, \varphi)$ over the alphabets Δ and Σ, the *membership problem* is to decide whether $R \in \mathcal{R}$ holds. Given another $\mathcal{R}' = (K', \varphi')$, also over the alphabets Δ' and Σ, the *inclusion problem* asks whether $\mathcal{R} \subseteq \mathcal{R}'$ holds, and the *equivalence problem,* whether $\mathcal{R} = \mathcal{R}'$ holds.

▶ **Theorem 7** (Equivalence, Inclusion, and Membership for Kleene star free RSRLs)**.** *Membership, inclusion, and equivalence are* PSPACE*-complete for Kleene star free represented RSRLs.*

This holds true, since in case of Kleene star free represented RSRLs (given explicitly as (K, φ) with K finite), we can enumerate the regular expressions defining all member languages in PSPACE. Given the PSPACE-completeness of regular language equivalence, we compare a given regular expression with all member languages, solving the membership problem in PSPACE. Doing so for all languages of another RSRL solves the inclusion problem, and checking mutual inclusion yields an algorithm for equivalence. This approach does *not* immediately generalize to finite RSRLs, since finite RSRLs $\mathcal{R} = \{\varphi(w) \mid w \in K\}$ may be generated from an infinite K with Kleene stars.

In the general case, the situation is quite different: Previous work shows that the membership problem is decidable [1], but without turning the construction into a concrete algorithm or determining an upper bound for complexity of the problem. Taking this work as starting point, in the remainder of this section, we give an 2EXPSPACE upper bound on the complexity of the problem, discussing the relationship with [1] at the end of the section. The decidability of inclusion and equivalence remains open.

4.1 Membership for general RSRLs

By definition, the membership problem is equivalent to asking whether there exists a $w \in K$ with $\varphi(w) = R$. For checking the existence of such a w, we have to check possibly infinitely many words in K efficiently. To render this search feasible, we **(A)** rule out irrelevant parts of K, and **(B)** treat subsets of K at once. This leads to the procedure $\mathsf{membership}(K, R, \varphi)$ shown in Algorithm 1, which first enumerates with $M' \in \mathsf{enumerate}(K, R, \varphi)$ a sufficient set of sublanguages (Line 1), and then checks each of those sublanguages individually (Line 2).

Algorithm 1: membership(R, K, φ)

input : regular languages $R \subseteq \Sigma^\star$, $K \subseteq \Delta^\star$,
regular language substitution φ with $\varphi(\delta) \subseteq \Sigma^\star$ for all $\delta \in \Delta$
returns : **true** iff $\exists w \in K : \varphi(w) = R$ (i.e., iff $R \in (K, \varphi)$)

```
foreach M' ∈ enumerate(R, K, φ) do
    if basiccheck(R, M', φ) then return true;
return false;
```

More specifically, we employ the following optimizations: We rule out **(A.1)** all words w with $\varphi(w) \not\subseteq R$, and **(A.2)** all words w whose language $\varphi(w)$ differs from R in the *length of its shortest word.* We subdivide the remaining search space **(B)** into finitely many suitable languages M' and check the existence of a $w \in M'$ with $\varphi(w) = R$ in a single step.

We discuss a mutually fitting design of these steps below and consider the resulting complexity. However, due to space limitations, we put the necessary proofs into an extended online version of this paper [18].

(A.1) Maximal Rewriting

To rule out all w with $\varphi(w) \not\subseteq R$, we rely on the notion of a *maximal φ-rewriting $M_\varphi(R)$ of R*, taken from [7]. $M_\varphi(R)$ consists of the words w with $\varphi(w) \subseteq R$, i.e., we set $M_\varphi(R) = \{w \in \Delta^+ \mid \varphi(w) \subseteq R\}$. Furthermore, all subsets $M \subseteq M_\varphi(R)$ are called *rewritings* of R, and if $\varphi(M) = R$ holds, M is called *exact* rewriting.

▶ **Proposition 8** (Regularity of maximal rewritings [7][1]). *Let $\varphi : \Delta \to 2^{\Sigma^\star}$ be a regular language substitution. Then the* maximal φ-rewriting $M_\varphi(R)$ *of a regular language $R \subseteq \Sigma^\star$ is a regular language over Δ.*

As all words w with $\varphi(w) = R$ must be element of $M_\varphi(R)$, we restrict our search to $M = M_\varphi(R) \cap K$.

(A.2) Minimal Word Length

We restrict the search space further by checking the *minimal word length,* i.e., we compare the length of the respectively shortest word in R and $\varphi(w)$. If R and $\varphi(w)$ have different minimal word lengths, $R \neq \varphi(w)$ holds, and hence, we rule out w. We define the minimal word length $\mathsf{minlen}(L)$ of a language L with $\mathsf{minlen}(L) = \min\{|w| \mid w \in L\}$, leading to the definition of language strata.

▶ **Definition 9** (Language Stratum). Let L be a language over Δ, and $\varphi : \Delta \to 2^{\Sigma^\star}$ be a regular language substitution, then the *B-stratum of L,* denoted as $L[B, \varphi]$, is the set of words in L which generate via φ languages of minimal word length B, i.e., $L[B, \varphi] = \{w \in L \mid \mathsf{minlen}(\varphi(w)) = B\}$.

Starting with $M = M_\varphi(R) \cap K$, we restrict our search further to $M[\mathsf{minlen}(R), \varphi]$.

[1] This proposition is not trivial, as φ is not a homomorphism mapping each word to a single word, but a substitution mapping each word w to a language $\varphi(w)$; if $\varphi(w)$ would yield only words, we would immediately obtain $M_\varphi(R) = \overline{\varphi^{-1}(\overline{R})}$ for $\varphi^{-1}(L) = \{w \mid \varphi(w) \cap L \neq \emptyset\}$.

(B) 1-Word Summaries

It remains to subdivide $M[\mathsf{minlen}(R), \varphi]$ into finitely many subsets M', which are then checked efficiently without enumerating their words $w \in M'$. Here, we only discuss the property of these subsets M' which enables such an efficient check, and later we will describe an enumeration of those subsets M'. When we check a subset M', we do not search for a single word $w \in M'$ with $\varphi(w) = R$ but for a finite set $F \subseteq M'$ with $\varphi(F) = R$. The soundness of this approach will be guaranteed by the existence of *1-word summaries:* A language $M' \subseteq \Delta^\star$ has 1-word summaries, if for all finite subsets $F \subseteq M'$ there exists a summary word $w \in M'$ with $\varphi(F) \subseteq \varphi(w)$. The property we exploit is given by the following proposition.

▶ **Proposition 10** (Membership Condition for Summarizable Languages, adapting [1]). *Let $M' \subseteq \Delta^\star$ be a regular language with 1-word summaries and $\varphi(M') \subseteq R$. Then there exists a $w \in M'$ with $\varphi(w) = R$ iff there exists a finite subset $F \subseteq M'$ with $\varphi(F) = \varphi(M') = R$.*

Putting it together

First, combining **A.2** and **B**, we obtain Lemma 11, to subdivide the search space $M[B, \varphi]$ into a set $\mathsf{rep}(M, B, \varphi)$ of languages M' with 1-word summaries. Second, in Theorem 12, building upon Lemma 11 and **A.1**, we fix $B = \mathsf{minlen}(R)$ and iterate through these languages M'. We check each of them at once with our membership condition from Proposition 10. In terms of Algorithm 1, Lemma 11 provides the foundation for $\mathsf{enumerate}(K, R, \varphi)$ and Proposition 10 underlies $\mathsf{basiccheck}(R, M', \varphi)$.

▶ **Lemma 11** (Summarizable Language Representation, adapting [1]). *Let $M \subseteq \Delta^\star$ be a regular language and $\varphi : \Delta \to 2^{\Sigma^\star}$ be a regular language substitution. Then, for each bound $B \geq 0$, there exists a family $\mathsf{rep}(M, B, \varphi)$ of union-free regular languages $M' \in \mathsf{rep}(M, B, \varphi)$ with 1-word summaries, such that $M[B, \varphi] \subseteq \bigcup_{M' \in \mathsf{rep}(M,B,\varphi)} M' \subseteq M$ holds.*

▶ **Theorem 12** (Membership Condition, following [1]). *Let $\mathcal{R} = (K, \varphi)$ be a RSRL and $\varphi : \Delta \to 2^{\Sigma^\star}$ be a regular language substitution. Then, for a regular language $R \subseteq \Sigma^\star$, we have $R \in \mathcal{R}$, iff there exists an $M' \in \mathsf{rep}(M_\varphi(R) \cap K, \mathsf{minlen}(R), \varphi)$ with a finite subset $F \subseteq M'$ with $\varphi(F) = \varphi(M') = R$.*

We obtain the space complexity of membership, depending on the *size of the expressions* which represent the involved languages. More specifically, the complexity depends on the expression sizes $||R||$ and $||K||$ and the summed size $||\varphi|| = \Sigma_{\delta \in \Delta} ||\varphi(\delta)||$ of the expressions in the co-domain of φ.

▶ **Theorem 13** ($\mathsf{membership}(R, K, \varphi)$ runs in 2ExpSpace). *More precisely, it runs in* DSpace $\left(||K||^r 2^{2^{(||R||+||\varphi||)^s}} \right)$ *for some constants r and s.*

We prove Theorem 13 in Section 4.4, relying on the algorithms presented in Sections 4.2 and 4.3.

4.2 Implementing $\mathsf{basiccheck}(R, M', \varphi)$

Since Lemma 11 produces only languages $M' = N_1 S_1^\star N_2 \dots N_m S_m^\star N_{m+1}$ with 1-word summaries, we restrict our implementation to such languages and exploit these restrictions subsequently. So, given such a language M' over Δ, and a regular language substitution $\varphi : \Delta \to 2^{\Sigma^\star}$, we need to check whether there exists a finite $F \subseteq M'$ with $\varphi(F) = \varphi(M') = R$.

Algorithm 2: basiccheck(R, M', φ)

input : regular languages $R \subseteq \Sigma^\star$, $M' \subseteq \Delta^\star$, and
regular language substitution φ with $\varphi(\delta) \subseteq \Sigma^\star$ for all $\delta \in \Delta$
requires : M' is union-free and $\varphi(M') \subseteq R$
returns : **true** iff $\exists$ finite $F \subseteq M' : \varphi(F) = \varphi(M') = R$

1 **build** $A_{M'}$;
2 **if** $A_{M'}$ *limited* **then**
3 **if** $\varphi(M') = R$ **then return true**;
4 **return false**;

We implement this check with the procedure basiccheck(R, M', φ), splitting the condition of Proposition 10 into two parts, namely **(1)** whether there exists a finite $F \subseteq M'$ with $\varphi(F) = \varphi(M')$, and **(2)** whether $\varphi(M') = R$ holds. While the latter condition amounts to regular language equivalence, the former requires distance automata as additional machinery.

▶ **Definition 14** (Distance Automaton [10])**.** A *distance automaton* over an alphabet Δ is a tuple $\mathcal{A} = \langle \Delta, Q, \rho, q_0, F, d \rangle$ where $\langle \Delta, Q, \rho, q_0, F \rangle$ is an NFA and $d : \rho \to \{0, 1\}$ is a distance function, which can be extended to a function on words as follows. The distance function $d(\pi)$ of a path π is the sum of the distances of all edges in π. The distance $\mu(w)$ of a word $w \in L(\mathcal{A})$ is the minimum of $d(\pi)$ for all paths π accepting w.

A distance automaton $\mathcal{A}$ is called *limited* if there exists a constant U such that $\mu(w) < U$ for all words $w \in L(\mathcal{A})$.

In our check for **(1)**, we build a distance automaton which is limited iff a finite F with $\varphi(F) = \varphi(M')$ exists. Then, we rely on the PSpace-decidability [25] of the limitedness of distance automata to check whether F exists or not.

Distance-automaton Construction

Here, we exploit the assumption that M' is a union-free language over Δ: Given the regular expression defining M', we construct the distance automaton $A_{M'}$ following the form of this regular expression:

- $\delta \in \Delta$: We construct the finite automaton A_δ with $L(A_\delta) = \varphi(\delta)$. We extend A_δ to a distance automaton by labeling each transition in A_{δ_i} with 0.
- $e \cdot f$: Given distance automata A_e and A_f with $A_e = (Q_e, \Sigma, \rho_e, q_{0,e}, F_e, d_e)$ and $A_f = (Q_f, \Sigma, \rho_f, q_{0,f}, F_f, d_f)$, we set $A_{e\cdot f} = (Q_e \uplus Q_f, \Sigma, \rho_e \cup \rho_f \cup \rho, q_{0,e}, F_f, d_{e\cdot f})$ where $\rho = \{(q, \varepsilon, q_{0,f}) \mid q \in F_e\}$ and $d_{e\cdot f} = d_e \cup d_f \cup \{(t, 0) \mid t \in \rho\}$, i.e., we connect each final state of A_e to the initial state of A_f and assign the distance 0 to these connecting transitions.
- $e^\star$: We construct the distance automaton $A_e = (Q_e, \Sigma, \rho_e, q_{0,e}, F_e, d_e)$. Then, $A_{e^\star} = (Q_e, \Sigma, \rho_e \cup \rho, q_{0,e}, F_e \cup \{q_{0,e}\}, d_{e^\star})$, where $\rho = \{(q, \varepsilon, q_{0,e}) \mid q \in F_e\}$ and $d_{e^\star} = d_e \cup \{((q, \varepsilon, p), 1) \mid (q, \varepsilon, p) \in \rho\}$, i.e., we connect each final state of A_e to the initial states of A_e and assign the corresponding transitions the distance 1.

If the resulting distance automaton $A_{M'}$ is limited, then there exists a finite subset $F \subseteq M'$ such that $\varphi(F) = \varphi(M')$. This implies that **(1)** holds.

So, given M' and R together with all languages in the domain of φ as regular expressions, basiccheck(R, M', φ) in Algorithm 2 first builds $A_{M'}$ (Line 1) and checks its limitedness (Line 2), amounting to condition **(1)**. For condition **(2)**, basiccheck verifies that $\varphi(M')$ and R are equivalent (Line 3) and returns **true** if both checks succeed.

Algorithm 3: enumerate(R, K, φ)

input : regular languages $R \subseteq \Sigma^\star$, $K \subseteq \Delta^\star$, and
regular language substitution φ with $\varphi(\delta) \subseteq \Sigma^\star$ for all $\delta \in \Delta$
yields : $L \in \mathsf{rep}(M, \mathsf{minlen}(R), \varphi)$ for $M = M_\varphi(R) \cap K$

1 $M := M_\varphi(R) \cap K$;
2 **for** $L \in \mathsf{unionfreedecomp}(M)$ **do** $\mathsf{unfold}(L, \varphi, \mathsf{minlen}(R))$;

Algorithm 4: unfold(L, φ, B)

input : union-free regular language $L \subseteq \Delta^\star$, written as
$L = N_1 S_1^\star N_2 \dots N_m S_m^\star N_{m+1} \subseteq \Delta^\star$ with $N_i \in \Delta^\star$ and union-free $S_h \subseteq \Delta^\star$,
regular language substitution φ with $\varphi(\delta) \subseteq \Sigma^\star$ for all $\delta \in \Delta$, and bound B
yields : $L' \in \mathsf{rep}(L, B, \varphi)$

1 **if** $\forall S_h \forall w \in S_h \ : \ \varepsilon \in \varphi(w)$ **then yield** L;
2 **else**
3 fix S_h arbitrarily with $\exists w \in S_h \ : \ \varepsilon \notin \varphi(w)$;
4 $E := S_h \cap \Delta_\varepsilon^\star$; // $\Delta_\varepsilon = \{\delta \in \Delta \mid \varepsilon \in \varphi(\delta)\}$
5 $L_0 := N_1 S_1^\star N_2 \dots \ N_h E^\star N_{h+1} \dots N_m S_m^\star N_{m+1}$;
6 $\mathsf{unfold}(L_0, \varphi, B)$;
// $L_p := N_1 S_1^\star N_2 \dots \ N_h E^\star \bar{E}_p S_h^\star N_{h+1} \dots N_m S_m^\star N_{m+1}$ (see text)
7 **for** $p \in \mathsf{critical}(S_h)$ *with* $\mathsf{minlen}(\varphi(L_p)) \leq B$ **do** $\mathsf{unfold}(L_p, \varphi, B)$;

▶ **Lemma 15** (basiccheck(R, M', φ) runs in PSPACE)**.** basiccheck(R, M', φ) *runs in* PSPACE*, which is optimal up to the assumption that* PSPACE *does not collapse with a lower class, as it solves a* PSPACE*-complete problem.*

4.3 Implementing enumerate(K, R, φ)

Our enumeration algorithm must produce the languages $\mathsf{rep}(M, B, \varphi)$, guaranteeing that all $M' \in \mathsf{rep}(M, B, \varphi)$ have 1-word summaries, and that $M[B, \varphi] \subseteq \bigcup_{M' \in \mathsf{rep}(M,B,\varphi)} M' \subseteq M$ holds (as specified by Lemma 11). To this end, we rely on a sufficient condition for the existence of 1-word summaries. First we show this condition with Proposition 16, before turning to the enumeration algorithm itself.

▶ **Proposition 16** (Sufficient Condition for 1-Word Summaries)**.** *Let L be a union-free language over Δ, given as $L = N_1 S_1^\star N_2 \dots N_m S_m^\star N_{m+1}$, with words $N_h \in \Delta^\star$ and union-free languages $S_h \subseteq \Delta^\star$. If $\varepsilon \in \varphi(w)$ for all $w \in S_h$ and all S_h, then L has 1-word summaries.*

We are ready to design our enumeration algorithm, shown in Algorithm 3, and its recursive subprocedure in Algorithm 4. Both algorithms do not return a result but yield their result as an enumeration: Upon invocation, both algorithms run through a sequence of **yield** statements, each time appending the argument of **yield** to the enumerated sequence. Thus, the algorithm never stores the entire sequence but only the stack of the invoked procedures.

Initializing the recursive enumeration, Algorithm 3 obtains the maximum rewriting $M := M_\varphi(R) \cap K$ of R (Line 1) and iterates over the languages L in the union-free decomposition of M (Line 2) to call for each L the recursive procedure unfold, shown in Algorithm 4. In turn, Algorithm 4 takes a union free language $L = N_1 S_1^\star N_2 \dots N_m S_m^\star N_{m+1}$ and a bound B to unfold the Kleene-star expressions of L until the precondition of Proposition 16 is satisfied or $\mathsf{minlen}(\varphi(L)) > B$.

More specifically, unfold exploits a rewriting, based on the following terms: Given a union free language S_h, let $E = S_h \cap \Delta_\varepsilon^\star$ with $\Delta_\varepsilon = \{\delta \in \Delta \mid \varepsilon \in \varphi(\delta)\}$ denote all words w in S_h with $\varepsilon \in \varphi(w)$ and let $\bar{E} = S_h \setminus E$. Since $\bar{E}$ is in general not union free, we need to split $\bar{E}$ further. To this end, we define $\mathsf{ufs}(S_h, p)$ recursively for an integer sequence $p = \langle p_H \mid p_T \rangle$ with head element p_H and tail sequence p_T. Intuitively, a sequence p identifies a subexpression in S_h by recursively selecting a nested Kleene star expression; $\mathsf{ufs}(S_h, p)$ unfolds S_h such that this selected expression is instantiated at least once. Formally, for $S_h = \alpha_1\beta_1^\star\alpha_2 \dots \alpha_n\beta_n^\star\alpha_{n+1}$ we set $\mathsf{ufs}(S_h, \varepsilon) = S_h$ and $\mathsf{ufs}(S_h, p) = \alpha_1 \dots \alpha_{p_H}\beta_{p_H}^\star \mathsf{ufs}(\beta_{p_H}, p_T)\beta_{p_H}^\star\alpha_{p_H+1} \dots \alpha_{n+1}$. Consider $S_h = A^\star(B^\star C^\star)^\star D^\star$ (with all $\alpha_i = \varepsilon$ for brevity), then we obtain

$$\begin{aligned}
\mathsf{ufs}(S_h, \langle 2,1\rangle) &= A^\star \ (B^\star C^\star)^\star \quad \mathsf{ufs}(B^\star C^\star, \langle 1\rangle) \quad (B^\star C^\star)^\star \ D^\star \\
&= A^\star \ (B^\star C^\star)^\star \ (B^\star \quad \mathsf{ufs}(B, \varepsilon) \quad B^\star \quad C^\star) \ (B^\star C^\star)^\star \ D^\star \\
&= A^\star \ (B^\star C^\star)^\star \ (B^\star \quad (B) \quad B^\star \quad C^\star) \ (B^\star C^\star)^\star \ D^\star
\end{aligned}$$

instantiating B at position $\langle 2,1\rangle$ at least once. Let $\mathsf{critical}(S_h)$ be integer sequences which identify a subexpression of S_h which directly contain a symbol δ with $\varepsilon \notin \varphi(\delta)$ (and not only via another Kleene-star expression). Then, we write $\bar{E} = \bigcup_{p \in \mathsf{critical}(S_h)} \bar{E}_p$, with $\bar{E}_p = \mathsf{ufs}(S_h, p)$. This discussion leads to the following rewriting:

▶ **Proposition 17** (Rewriting for 1-Word Summaries)**.** *For every union free language $S_h^\star$, we have $S_h^\star = E^\star \cup \bigcup_{p \in \mathsf{critical}(S_h)} E^\star \bar{E}_p S_h^\star$. All languages in the rewriting, i.e., $E^\star$ and $E^\star \bar{E}_p S_h^\star$, are union free, $E^\star$ has 1-word summaries, and $\mathsf{minlen}(S_h^\star) < \mathsf{minlen}(E^\star \bar{E}_p S_h^\star)$ holds for all $p \in \mathsf{critical}(S_h)$.*

If L already satisfies the precondition imposed by Proposition 16, Algorithm 4 **yield**-s L and terminates (Line 1). Otherwise, it fixes an arbitrary S_h violating this precondition and rewrites L recursively with Proposition 17 (Lines 3-7). **(1)** *Termination:* In each recursive call, unfold either eliminates in L_0 an occurrence of a subexpression S_h violating the precondition of Proposition 16 (Line 6), or increases the minimum length in L_p, eventually running into the upper bound B (Line 7). **(2)** *Correctness:* Setting $B = \infty$, unfold **yield**-s a possibly infinite sequence of union free languages which have 1-word summaries such that their union equals the original language L: As the generation of these languages is based on the equality of Proposition 17 each rewriting step is sound and complete, leading to an infinite recursion tree whose leaves **yield** the languages in the sequence. The upper bound on minimum length only cuts off languages L_p producing words of minimum length beyond B, i.e., $L_p \cap L[B, \varphi] = \emptyset$, and in consequence, it is safe to drop L_p, since we only need to construct $\mathsf{rep}(L, B, \varphi)$ with $\mathsf{rep}(L, B, \varphi) \supseteq L[B, \varphi]$.

4.4 Upper Bound of the Complexity

The proof of Theorem 13 is based on the size of the maximum rewriting $M = M_\varphi(R) \cap K$ of $||K||2^{(||R||+||\varphi||)^l}$ for some constant l, shown in [7], and unfold's complexity: In Proposition 18, we show an upper bound on the space complexity of unfold, leading to the complexity of enumerate in Lemma 19 and the desired proof of Theorem 13.

▶ **Proposition 18** ($\mathsf{unfold}(L, \varphi, B)$ runs in $\mathrm{DSpace}(B^2||L||^4 + ||\varphi||)$)**.**

▶ **Lemma 19** ($\mathsf{enumerate}(R, K, \varphi)$ runs in $\mathrm{DSpace}\left(||K||^4 2^{2^{(||R||+||\varphi||)^k}}\right)$)**.**

Proof of Theorem 13. The enumeration runs in $\mathrm{DSpace}\left(||K||^4 2^{2^{(||R||+||\varphi||)^k}}\right)$, producing expressions for basiccheck at most of the same size (Lemma 19). Since basiccheck is in PSpace (Lemma 15), we obtain the overall complexity $\mathrm{DSpace}\left(||K||^r 2^{2^{(||R||+||\varphi||)^s}}\right) \subseteq 2\mathrm{ExpSpace}$ for some constants r and s. ◀

4.5 Differences to Afonin and Khazova [1]

Afonin and Khazova show that the membership problem is decidable. In determining an upper bound for the complexity of membership problem, we had to expand their approach significantly: In general, we follow a top-down approach to describe the overall algorithm, whereas Afonin and Khazova go bottom-up, focusing on the building blocks enabling the decision procedure. More specifically, basiccheck is described in [1], while enumerate is omitted, as [1] deals with decidability only, deeming the bound on the enumeration size irrelevant. Hence Algorithms 3 and 4 are new, as well as the construction in Section 4.3, leading to Proposition 17. Based on the new algorithms, we contribute Theorem 13, together with Proposition 18, and Lemma 19. Moreover, in [1], the overall algorithm and the proof for Theorem 12 are only described in a brief paragraph. Finally, Section 4.1, albeit technically not new, provides a much more conceptual and hopefully accessible description of the algorithm.

5 Conclusion

Motivated by applications in test case specifications with FQL, we have studied general and finite RSRLs. While we showed that general RSRLs are not closed under most common operators, *finite* RSRLs are closed under all operators except Kleene stars and complementation (Theorem 4). This shows that our restriction to Kleene star free and hence finite RSRLs in FQL results in a natural framework with good closure properties. Likewise, the proven PSPACE-completeness results for Kleene star free RSRLs provide a starting point to develop practical reasoning procedures for Kleene star free RSRLs and FQL. Experience with LTL model checking shows that PSPACE-completeness often leads to algorithms which are feasible in practice. In contrast, for general and possibly infinite RSRLs, we have described a 2EXPSPACE membership checking algorithm – leaving the question for matching lower bounds open. Nevertheless, reasoning on general RSRLs seems to be rather infeasible.

Last but not least, RSRLs give rise to new and interesting research questions, for instance the decidability of inclusion and equivalence for general RSRLs, and the closure properties left open in this paper. In our future work, we want to generalize RSRLs to other base formalisms. For example, we want φ to substitute symbols by context-free expressions, thus enabling FQL test patterns to recognize e.g. matching of parentheses or emptiness of a stack.

Acknowledgements. This work received funding in part by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF), by the Vienna Science and Technology Fund (WWTF) grant PROSEED, and by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement DIADEM no. 246858.

References

1 S. Afonin and E. Hazova. Membership and Finiteness Problems for Rational Sets of Regular Languages. In *DLT*, pages 88–99, 2005.
2 G. A. Agasandyan. Variable-Structure Automata. *Soviet Physics Doklady*, 1967.
3 P. Barceló, D. Figueira, and L. Libkin. Graph Logics with Rational Relations and the Generalized Intersection Problem. In *LICS*, pages 115–124, 2012.
4 P. Barceló, J. L. Reutter, and L. Libkin. Parameterized Regular Expressions and their Languages. *Theor. Comput. Sci.*, 474:21–45, 2013.
5 J. Berstel. *Transductions and Context-Free Languages*. Teubner Studienbücher, Stuttgart, 1979.

6 D. Beyer, A. Holzer, M. Tautschnig, and H. Veith. Information Reuse for Multi-goal Reachability Analyses. In *ESOP*, pages 472–491, 2013.

7 D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of Regular Expressions and Regular Path Queries. *JCSS*, 64:443–465, 2002.

8 T. Colcombet and L. Daviaud. Approximate Comparison of Distance Automata. In *STACS*, pages 574–585, 2013.

9 S. Eilenberg and M. P. Schützenberger. Rational Sets in Commutative Monoids. *J. Algebra*, 13:173–191, 1969.

10 K. Hashiguchi. Limitedness Theorem on Finite Automata with Distance Functions. *J. Comput. Syst. Sci.*, 24(2):233–244, 1982.

11 K. Hashiguchi. Representation Theorems on Regular Languages. *J. Comput. Syst. Sci.*, 27(1):101–115, 1983.

12 A. Holzer, V. Januzaj, S. Kugele, B. Langer, C. Schallhart, M. Tautschnig, and H. Veith. Seamless Testing for Models and Code. In *FASE'11*, pages 278–293, 2011.

13 A. Holzer, D. Kroening, C. Schallhart, M. Tautschnig, and H. Veith. Proving Reachability using FShell (Competition Contribution). In *TACAS*, pages 538–541, 2012.

14 A. Holzer, C. Schallhart, M. Tautschnig, and H. Veith. How did You Specify Your Test Suite? In *ASE*, pages 407–416, 2010.

15 A. Holzer, M. Tautschnig, C. Schallhart, and H. Veith. FSHELL: Systematic Test Case Generation for Dynamic Analysis and Measurement. In *CAV*, pages 209–213, 2008.

16 A. Holzer, M. Tautschnig, C. Schallhart, and H. Veith. Query-Driven Program Testing. In *VMCAI*, pages 151–166, 2009.

17 A. Holzer, M. Tautschnig, C. Schallhart, and H. Veith. An Introduction to Test Specification in FQL. In *HVC*, pages 9–22, 2010.

18 Andreas Holzer, Christian Schallhart, Michael Tautschnig, and Helmut Veith. On the Structure and Complexity of Rational Sets of Regular Languages. *CoRR*, abs/1305.6074, 2013.

19 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

20 D. Kirsten. Distance Desert Automata and the Star Height One Problem. In *FoSSaCS*, pages 257–272, 2004.

21 D. Kirsten. Distance Desert Automata and the Star Height Problem. *ITA*, 39(3):455–509, 2005.

22 S. C. Kleene. Representation of Events in Nerve Nets and Finite Automata. *RAND Corporation Memorandum*, 1951.

23 D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Intl. Journal of Algebra and Computation*, 4(3):405–425, 1994.

24 O. Kupferman and Y. Lustig. Lattice Automata. In *VMCAI*, pages 199–213. 2007.

25 H. Leung and V. Podolskiy. The limitedness problem on distance automata: Hashiguchi's method revisited. *TCS*, 310(1–3):147–158, 2004.

26 J.-E. Pin. Mathematical Foundations of Automata Theory. Lecture Notes, 2011.

27 RTCA DO-178B. Software Considerations in Airborne Systems and Equipment Certification, 1992.

28 A. Salomaa. On Finite Automata with a Time-Variant Structure. *Information and Control*, 13(2):85 – 98, 1968.

Geometric Avatar Problems

Mario E. Consuegra and Giri Narasimhan

School of Computing and Information Sciences, Florida International University, Miami, USA, {mcons004,giri}@fiu.edu

Abstract

We introduce the concept of *Avatar problems* that deal with situations where each entity has multiple copies or "avatars" and the solutions are constrained to use exactly one of the avatars. The resulting set of problems show a surprising range of hardness characteristics and elicit a variety of algorithmic solutions. Many avatar problems are considered. In particular, we show how to extend the concept of ϵ-kernels to find approximation algorithms for geometric avatar problems. Results for metric space graph avatar problems are also presented.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Avatar problems, choice

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.389

Every man has the power to choose,
but no power to escape the necessity of choice.
– **Ayn Rand**

1 Introduction

We introduce a family of optimization problems which we call *Avatar problems*. The main feature of this family of problems is that their input entities have multiple replicas (or copies, or *avatars*), but their output is constrained to use exactly one of the copies. Avatar problems manifest themselves in many practical applications. For example, if disk storage systems have multiple copies of data items, then disk scheduling algorithms may process requests by visiting any one of the copies of each requested data to optimize the total access cost. Given any optimization (or decision) problem, its avatar version is required to achieve the same optimization (or decision) over all possible instances where each instance is created by assigning each element a_i to **exactly** one of k possible values. In this paper we investigate the complexity of *avatar* versions of classical algorithmic problems.

Avatar versions of NP-hard problems are easily shown to be NP-hard. However, designing good approximation algorithms often requires the solution of other avatar problems (e.g., avatar TSP can be well approximated by approximating avatar MST or avatar matching). This suggests the need for solving a family of avatar problems, over and beyond those with direct relevance to practical applications.

Related problems include *generalized MST* (MST spanning at least one vertex from each given set) [20], *group TSP* (minimum length tour visiting at least one vertex from each given group) [9], and *TSP with neighborhoods* (minimum length tour that visits each given neighborhood) [8, 22], all of which are NP-hard, even in Euclidean space. A closely related model is the indecisive (uncertain) points model [16], where input points have spatial uncertainty, but their true locations are known to be from a set of (possibly infinite) possibilities. Jørgensen et al. [16] suggest many applications for their model; these are applicable to the

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 389–400

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

avatar model as well. For example, privacy considerations may prevent a database from storing the precise location of a person with a certain illness, but may provide a zip code; sensors may have limited accuracy and may provide approximate location data. A major difference with the avatar model is that in the uncertainty model, the power of choice lies with an adversary. Also see [5, 10–12, 14, 19, 21].

The concept of switching graphs [15, 18] introduces another related model. In the area of scheduling, a related problem is the job interval selection problem (JISP) [7]. In this problem the input is a set of n jobs assigned to a worker. Each job is a set of one or more intervals on the real line, and we must select one interval for each job such that we schedule as many jobs as possible by picking non-overlapping intervals. In the k-avatar version of JISP each job is a set of at most k intervals. Avatar problems share some overlap with the area of parameterized complexity. The study of the complexity of a *k-avatar* problem as k goes from 1 to ∞ provides better understanding of the complexity landscape of the problem.

Results

The main results are summarized here, and is indicative of how "choice" affects the complexity of these problems in different ways.

1. In Section 2, we tackle two seemingly simple problems, minGap and maxGap, where the avatar versions result in natural optimization problems. We design a $\mathcal{O}(n^2 \log n)$-time algorithm for the 2-avatar maximum minGap problem for inputs in $\mathcal{R}^d$, and a 2-approximation $\mathcal{O}(n^3 \cdot k^3 \log(nk))$-time algorithm for the k-avatar minimum maxGap problem for points on a line.
2. In Section 3, we extend the concept of ϵ-kernels to the avatar world and show how to compute it efficiently for a k-avatar point set in $\mathbb{R}^d$. This enabled us to design a polynomial-time algorithm for finding an ϵ-approximate smallest convex hull for the k-avatar convex hull problem in $\mathbb{R}^d$. The ϵ-kernel result was also used to design polynomial-time $(1+\epsilon)$-approximation algorithms for the avatar versions of the following geometric problems: smallest volume axis-aligned enclosing hyperbox.
3. Reachability is a fundamental problem with linear-time algorithms for non-avatar inputs. Surprisingly, we show in Section 4 that for unweighted graphs, the k-avatar reachability problem is NP-complete. For weighted graphs, we show that the k-avatar shortest path problem is inapproximable to any constant factor unless $P = NP$.

We establish some basic notation for this paper. Let $L = \{a_1, a_2, \ldots, a_n\}$ be a set of n k-avatar entities. In other words, for each entity $a_i \in L$, one can assign a_i to one of the k avatar values from the set $Av(a_i) = \{v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(k)}\}$. An *avatar assignment* for entities in L, denoted by $A(\cdot)$, is an assignment of a single avatar value to each entity in L. Thus, $A(a_i) \in Av(a_i)$. Let $A(L)$ denote the set of values assigned to each element in L.

2 Avatar Minimum and Maximum Gaps

Given a set of points $\{x_1, \ldots, x_n\}$ on a line, we define the *minGap* (resp. *maxGap*) as the smallest (resp. largest) gap between consecutive items in the sorted order. The avatar version of the *maximum minGap* and *minimum maxGap* problems is: *Given a set of n k-avatar entities, find an avatar assignment that results in the maximum minGap (resp. minimum maxGap).* More formally, we are given a set of k-avatar entities $L = \{a_1, a_2, \ldots, a_n\}$, where each entity a_i can be assigned one of k values from the set $\{v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(k)}\}$.

2.1 Avatar Maximum minGap

We present a polynomial-time algorithm for the 2-avatar version of *maximum minGap* problem. It is clear that the minGap must be between a pair of points from the set of all avatar values, $\bigcup_{a_i \in L} Av(a_i) = \{v_1^{(1)}, \ldots, v_1^{(k)}, v_2^{(1)}, \ldots, v_2^{(k)}, \ldots, v_n^{(1)}, \ldots, v_n^{(k)}\}$. We first solve the decision problem of determining if there exists an avatar assignment so that the minGap is at least B; this is achieved by giving a polynomial-time reduction to 2SAT. The construction creates two complementary boolean variables, x_i and $\neg x_i$, to represent the two avatars of entity a_i. For every pair of values that are not avatars of each other and that have a distance of at most B, a clause is created to ensure that the corresponding boolean variables are not simultaneously set to true; a conjunction of these clauses generates an instance of 2SAT. It is easily shown that the resulting 2SAT formula is satisfiable if and only if the original 2-Avatar Maximum minGap problem has a minGap that is no smaller than B. Given the linear time algorithm for 2SAT [3], it is not difficult to see that the above algorithm takes $\mathcal{O}(n^2)$ time, and that the maximum minGap can be found in $\mathcal{O}(n^2 \log n)$ time by doing a binary search on the sorted list of all interpoint distances.

The above reduction to 2SAT for the 2-avatar minGap problem readily generalizes to the case where the entity values are points in d-dimensional space. However, the k-avatar minGap problem is NP-complete, and can be proved by a trivial adaptation of the proof of NP-Completeness of the problem of finding a *System of q-Distant Representatives* proved by Fiala et al. [13].

▶ **Theorem 1.** *The 2-avatar maximum minGap problem for n points in $\mathbb{R}^d$ can be solved in $\mathcal{O}(n^2 \log n)$ time. The corresponding k-avatar problem for $k > 2$ is NP-hard.*

2.2 Avatar Minimum maxGap

The avatar minimum maxGap problem appears to be harder than the avatar maximum minGap problem. While an exact polynomial time algorithm for the minimum maxGap problem remains open, below we present an approximation algorithm for the k-avatar minimum maxGap problem for points on a line.

Let B^* be the length of the minimum MaxGap, where the minimum is over all possible avatar assignments. We will perform binary search on the sorted list of all interpoint distances in order to find good lower and upper bounds B_{low} and B_{upp} for B^* such that $B_{\text{low}} \leq B^* \leq B_{\text{upp}}$. Establishing bounds for the ratio between the lower and upper bounds gives an approximation for B^*. A sorted list of interpoint distances can be computed in $O(n^2k^2 \log nk)$. For a given value of B during this binary search we need to solve the decision problem of determining if there exists an avatar assignment so that the maxGap is at most B. The algorithm described below will give an approximate solution to this decision problem in the following sense. If the algorithm says "NO", then maxGap is greater than B. If the algorithm says "YES", then the maxGap is at most $2B$.

Let V denote the set of kn avatar values mapped on to the real line. Any avatar assignment is a subset of n points from V. A partition of the line into infinite number of disjoint abutting cells each of size B (see Fig. 1) is called a *valid* partition if there exists an avatar assignment such that all the points in the assignment are contained in a sequence of consecutive non-empty cells. Therefore, it follows that if there exists an avatar assignment for L such that the resulting point set has maxGap at most B then every partition of the line into infinite cells of size B is *valid*. The consequence is that if there is any partition of the line into infinite cells of size B that is not valid, then we know for sure that the maxGap for every assignment is greater than B. The difficulty is that the converse need not be true.

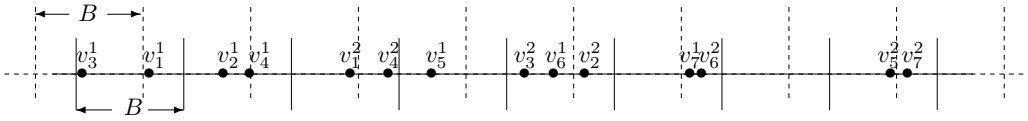


Figure 1 Two different infinite partitions of the line are shown. The partition with dotted vertical lines is *valid*, while the partition with solid vertical lines is not *valid*. The valid partition is achieved by an avatar assignment that picks all choices with superscript 2.

Even though the assigned values appear in a sequence of consecutive cells, the maxGap could be between two items in adjacent cells that are nearly $2B$ apart, a key observation that leads to a 2-approximate algorithm. For example, in Fig. 1, v_6^2 and v_5^2 are in adjacent cells (of the partition with vertical dotted lines) but are almost $2B$ apart.

Given B, a fixed infinite partition of the line into cells of size B, and a fixed sequence of consecutive cells, we check if that partition is valid for some avatar assignment of L by a reduction to Network Flow. Briefly, we construct a bipartite network where one partition P has vertices corresponding to entities $a_i \in L$ and the other partition Q has vertices corresponding to cells of the partition. There is an edge from a vertex $p \in P$ to a vertex $q \in Q$ if the entity corresponding to p has an avatar in the cell corresponding to q. Finally, the reduction involves showing that the network has a flow of n if and only if the partition is valid. For lack of space, details of the algorithm are omitted from this draft. As mentioned above, we perform binary search on the sorted list of interpoint distances until we find two adjacent gaps B_{i-1} and B_i in the list of gaps such that

1. $B_{i-1} \leq B_i$,
2. the algorithm returns `NO` for all partitions into cells of length B_{i-1}, and
3. Returns `YES` for at least one partition into cells of length B_i.

Thus, $B_{i-1} < B^*$. Since the smallest possible gap attainable that is larger than B_{i-1} is B_i, we have $B_i \leq B^*$. Also, since we have a partition into cells of length B_i for which we can find an avatar assignment where all the chosen points are in a set of adjacent cells such that each cell in that set contains a chosen point, we can use that avatar assignment to produce an assignment with a maximum gap that is no larger than $2 \cdot B_i$. Hence we have that $B_i \leq B^* \leq 2 \cdot B_i$. This gives us a polynomial-time 2-approximation algorithm for the 1D k-avatar minimum maxGap problem. The hardness of the avatar minimum maxGap for points in $\mathbb{R}^d$ remains open, even for $d = 1$.

▶ **Theorem 2.** *The k-avatar minimum maxGap problem for points on a line has a 2-approximate algorithm that runs in $O(n^3 \cdot k^3 \log(nk))$ time.*

3 Avatar Convex Hulls

Let L be a set of k-avatar entities where each entity can be assigned one of k different points in d-dimensional space. A k-avatar convex hull of L is a minimal convex set that contains at least one avatar for each entity $a \in L$. The aim is to minimize a specific measure such as the perimeter, surface area, or volume. The computational complexity of the problem of computing the avatar minimum convex hull remains an open problem. Related work includes results on the minimum and maximum convex hull for a set of points with imprecise locations [17, 23], and a recent paper by Abdullah et al. [1] for the model of uncertain points.

A *smallest avatar convex hull* is a convex hull that has minimum perimeter over all possible avatar assignments. Using the concept of ϵ-kernels, we present an algorithm that

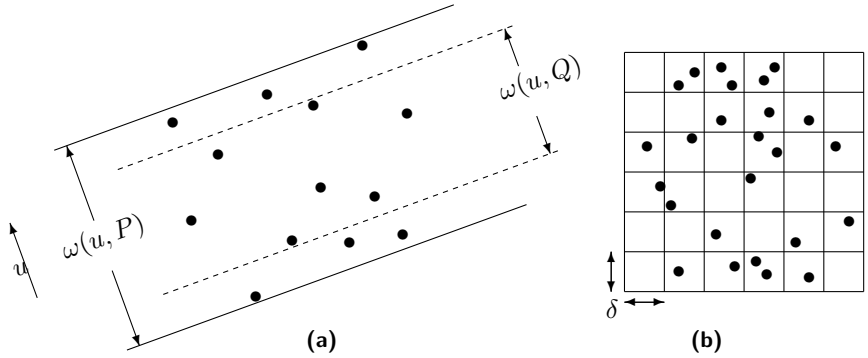


Figure 2 (a) Directional Width; (b) ϵ-grid Z.

finds an ϵ-approximate smallest avatar convex hull for the k-avatar convex hull problem in $\mathbb{R}^d$. The results can be extended to minimum area/volume convex hulls.

3.1 Approximate Avatar Convex Hulls

For any point set $X \subset \mathbb{R}^d$, let $\omega(u, X)$ denote the *directional width* of X in direction u (see Fig. 2 (a)). A subset $Q \subseteq P$ is called an ϵ-approximation of P if for any direction $u \in S^{d-1}$ we have $(1-\epsilon)\omega(u, P) \leq \omega(u, Q) \leq (1+\epsilon)\omega(u, P)$. Our proposed algorithm finds an ϵ-approximate smallest convex hull $\mathcal{CH}(Q)$ by returning a set of avatar points $Q \subseteq A'(L)$ for some avatar assignment $A'(L)$ such that $(1-\epsilon)\omega(u, \mathcal{CH}^*(L)) \leq \omega(u, \mathcal{CH}(Q)) \leq (1+\epsilon)\omega(u, \mathcal{CH}^*(L))$, where $\mathcal{CH}^*(L)$ is the minimum avatar convex hull of L. Using the terminology of Agarwal et al. [2], one can think of the set Q as the avatar equivalent of an ϵ-kernel. This is formalized in the following definition of an *avatar ϵ-kernel* whose width along any direction is within a $1-\epsilon$ factor of the width of the optimal hull along that direction.

▶ **Definition 3.** Given a set L of n k-avatar entities in R^d, we say that a point set Q is an *avatar ϵ-kernel* of L if and only if $(1-\epsilon)\omega(u, \mathcal{CH}^*(L)) \leq \omega(u, Q) \leq (1+\epsilon)\omega(u, \mathcal{CH}^*(L)), \forall u \in \mathbb{S}^{d-1}$, where $\mathbb{S}^{d-1}$ is the unit hypersphere centered at the origin.

The following procedure for finding a *diameter-oriented bounding box* $\mathcal{B}$ of a set S of points in R^d was described by Barequet and Har-Peled [4]. Let $\mathcal{D}(\mathcal{S})$ be the diameter of S and let $s_1, t_1 \in S$ s.t. $|s_1t_1| = \mathcal{D}(S)$. Let H be a hyperplane perpendicular to s_1t_1 and let Q be the orthogonal projection of S onto H. We again compute two points $s_2, t_2 \in Q$ s.t. $|s_2t_2| = \mathcal{D}(Q)$. Once again we project Q onto a hyperplane H' perpendicular to s_1t_1 and s_2t_2 and determine the diameter $\mathcal{D}(Q')$ of the projection Q onto H' and select two more points $s_3, t_3 \in Q'$ s.t. $|s_3t_3| = \mathcal{D}(Q')$. After d iterations of this process we have a diameter-oriented bounding box $\mathcal{B}(S)$ of S with the diameter in each iteration determined by the direction from s_i to t_i, for $i = 1, 2, \ldots, d-1$.

Note that $\mathcal{CH}^*$ must cover a set S of $2 \cdot d$ avatar points of an avatar assignment such that the diameter-oriented bounding box $\mathcal{B}(\mathcal{S})$ is exactly the same as the diameter-oriented bounding box $\mathcal{B}(\mathcal{CH}^*)$. See Algorithm 1 for the pseudocode for the following procedure. We pick all possible subsets of $2 \cdot d$ avatar points of L, of which there are $\binom{n \cdot k}{2 \cdot d}$. For each such subset S_i, we first check that no two points in S_i are in the same avatar set, then we find the diameter-oriented bounding box $B_i = \mathcal{B}(S_i)$. If every entity in L has an avatar point inside B_i then it is possible that $B_i = \mathcal{B}(\mathcal{CH}^*)$, otherwise we can discard B_i. We find

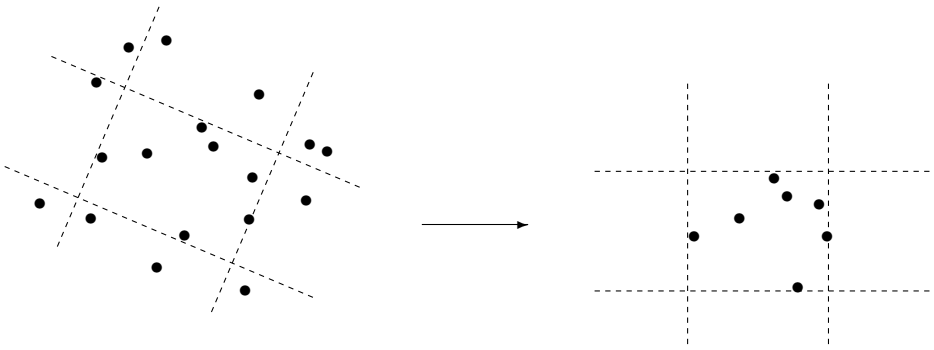


Figure 3 Affine transform of space inside diameter-oriented bounding box of $2 \cdot d$ points.

an ϵ-approximate minimum avatar convex hull $\mathcal{CH}_i$ of all the points inside B_i and output the smallest one, which we refer to as $\mathcal{CH}_{\min}$. Since $\mathcal{B}(\mathcal{CH}^*) = B_i$ for some i, $\mathcal{CH}_{\min}$ will ϵ-approximate $\mathcal{CH}^*$. The following lemma from [2] is useful for this proof. We say that a point set is α-*fat* if its convex hull (a) is contained in a hypercube $\mathcal{H}$ and (b) contains a copy of $\mathcal{H}$ sharing the same center as $\mathcal{H}$, but shrunk by a factor $\alpha < 1$.

▶ **Lemma 4.** **[2]** *For any point set P with non-zero volume in $\mathbb{R}^d$ there exists an affine transform M s.t. $M(P)$ is an α-fat point set (for some $\alpha <$) where the hypercube $\mathbb{C} = [-1, +1]^d$ is the smallest enclosing box of $M(P)$ and s.t. a subset $Q \subseteq P$ is an ϵ-kernel of P iff $M(Q)$ is an ϵ-kernel of $M(P)$.*

Algorithm 1 COMPUTING ϵ-APPROXIMATE SMALLEST AVATAR CONVEX HULL

Require: L: set of n k-avatar entities; μ: a measure function of the size of the perimeter of a convex hull, $T(.)$ affine transform procedure

`let` $\mathcal{CH}_{min} =$ `null`

`let` S be the set of all possible sets of $2d$ avatar points of L.

for $S_i \in S$ **do**
  if no two points in S_i are avatars of each other **then**
    `let` $\mathcal{B}(S_i)$ be the *diameter-oriented bounding box*
    `let` B_i be the set of all avatar points inside $\mathcal{B}(S_i)$ such that every entity is represented in B_i
    `let` $\mathcal{CH}_i$ be the ϵ-approximate smallest avatar convex hull of $T(B_i)$ computed by Algorithm 2
    $\mathcal{CH}_{min} =$ `Min`$(\mathcal{CH}_{min}, \mathcal{CH}_i)$
  end if
end for
return $\mathcal{CH}_{min}$

It is known that for a diameter-oriented bounding box B with largest side $\mathcal{D}$, if we appropriately expand or contract the box along each direction until it becomes a hypercube of side $\mathcal{D}$ and scale it to the hypercube $\mathbb{C}$, then the transformed point set is an α-fat point set (for some $\alpha <$) in $\mathbb{C}$ [4]. This is illustrated by an example in Fig. 3. This transformation $T(B)$ of B as well as the transformed points can be computed in linear time, i.e., $O(n \cdot k)$ time. By Lemma 4, to compute an ϵ-approximate avatar convex hull of all the points in $\mathcal{B}$, we only need to compute an ϵ-approximate avatar convex hull of all the points in $\mathbb{C}$, which is computed as follows.

As in [4], let δ be the largest value such that $\delta \leq (\epsilon/\sqrt{d})\alpha$ and $1/\delta$ is an integer. We then partition the bounding hypercube into a uniform grid with cells of side length δ (see

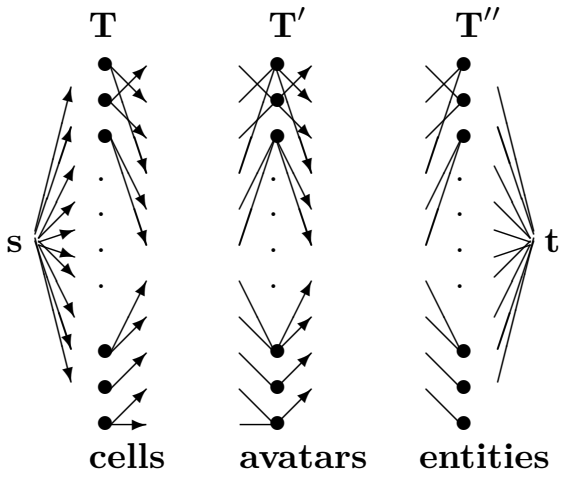


Figure 4 Reduction to network flow used to determine if a set of cells is legal.

Fig. 2(b)). However, applying the algorithm of Barequet and Har-Peled [4] does not help us to compute ϵ-kernels in $\mathbb{C}$ because now we must make sure not to pick two or more avatars of the same entity.

We need one other idea to compute ϵ-kernels in $\mathbb{C}$. The following procedure computes the ϵ-kernel in $T(B)$ (see Algorithm 2). Consider all possible assignments of binary values (0/1) to the cells in the grid (see Fig. 2 (b)). For the i^{th} binary assignment let Q_i be the set of cells that are assigned a value of 1. We call the set Q_i *legal* if each avatar entity has at least one element in at least one of the cells of Q_i, and it is possible to pick a representative point from each cell such that no two cells have representative points that are avatars of the same entity. Since there are $1/\delta^d$ cells, there are at most $2^{1/\delta^d}$ legal sets. In particular, if $A_{OPT}(\cdot)$ is the avatar assignment that leads to the optimal avatar convex hull, then it is easy to see that one of these legal sets must contain exactly the collection of cells with points from $A_{OPT}(\cdot)$.

It is clear that for any box B_i, with largest side of length $\mathcal{D}_i$, if we expand the box along each direction until it becomes a hypercube of side $\mathcal{D}_i$ and scale it to the unit hypercube $\mathbb{C}$, we are left with an α-fat point set in $\mathbb{C}$ (for some $\alpha <$) since $\mathcal{CH}(B_i)$ must cover all the points in S_i and hence it must touch each face of $\mathbb{C}$. This transformation $T(B_i)$ of B_i can be found in time linear in the number of points in B_i, which is equal to $O(nk)$. By Lemma 4, we know that finding an ϵ-approximate avatar convex hull of all the points in $\mathbb{C}$ gives us directly an ϵ-approximate avatar convex hull of all the points in $\mathcal{B}_i$.

We can determine if a given set of grid cells Q_i is legal by solving a network flow problem as follows. (See Fig. 4.) Create a set of vertices T such that each vertex in T represents a different cell in Q_i. Create a source vertex s with directed edges to each vertex in T. Create a set of vertices T' such that each vertex in T' represents a distinct point in some cell in Q_i. Add an edge from $u \in T$ to $u' \in T'$ if the cell in Q_i corresponding to u contains the point corresponding to u'. Create another set of vertices T'' such that each vertex in T'' corresponds to an avatar entity. Add an edge from $u' \in T'$ to $u'' \in T''$ if u' is a possible assignment for the avatar entity u''. Finally add a sink vertex t and connect all vertices in T'' to t by an edge. All edges have capacity 1. A maximum flow of size $|T|$ from s to t will identify a representative point in each cell such that no two points are avatars of the same entity. It is easy to see that such a flow exists if and only if the corresponding set of cells Q_i is legal. The following theorem formalizes the result.

Algorithm 2 ϵ-APPROXIMATION OF SMALLEST AVATAR CONVEX HULL FOR α-FAT AVATAR POINT SET

Require: P: an α-fat set (for some $\alpha <$) of k-avatar points inside the unit hypercube $\mathbb{C}$
μ: measure function of the size of the perimeter
let δ be the largest integer s.t. $\delta \leq (\epsilon/\sqrt{d})\alpha$
let Z be a d-dimensional grid of cell size δ
for each assignment of binary values (0/1) to the cells in the grid Z **do**
`let` Q_i be the set of cells assigned with a 1 in the i^{th} binary assignment
if Q_i is *legal* **then**
`let` $Q'_i \subseteq Q_i$ be the collection of highest and lowest cells in every hypercolumn containing at least one cell of Q_i
`let` Q' be the set of representative points of cells in Q'_i
`let` $\mathcal{CH}_i = \mathcal{CH}(Q')$
if $\mu(\mathcal{CH}_i) < \mu(\mathcal{CH}_{min})$ **then**
$\mathcal{CH}_{min} = \mathcal{CH}_i$
end if
end if
end for
return $\mathcal{CH}_{min}$

▶ **Theorem 5.** *There is an algorithm that finds an ϵ-approximate min-perimeter k-avatar convex hull in time $O((nk)^{(2d+3)} \cdot \frac{n}{\delta^d} \cdot (2d)^2 \cdot 2^{\frac{1}{\delta^d}(\frac{2}{\delta^{d-1}})^{\lfloor \frac{d}{2} \rfloor}})$ by finding an avatar ϵ-kernel Q of L, which by Definition 3 satisfies $(1-\epsilon)\omega(u, \mathcal{CH}^*(L)) \leq \omega(u, \mathcal{CH}(Q)), \quad \forall u \in S^{d-1}$. Note that the choice of constant δ depends on k, ϵ, and α.*

The proof is sketched as follows. Given a legal set, Q_i, let $Q'_i \subseteq Q_i$ be the collection of highest and lowest cells in every hypercolumn containing at least one cell of Q_i. Let Q (resp., Q') be the set of representative points of cells in Q_i (resp., Q'_i). It is easy to see that Q is an ϵ-kernel of Q'. We argue that $A_{OPT}(\cdot)$, the avatar assignment that leads to the optimal avatar convex hull, occupies a collection of cells (call this set of cells Q_{OPT}), which would have been considered by our algorithm. While the algorithm may not have picked the points in the optimal avatar assignment, it is sure to pick one representative point from each of the cells in Q_{OPT}. Since for each point in the legal set, there is at least one representative point that is within distance $\epsilon \cdot \alpha$ for every point in the optimal avatar assignment, we immediately have an avatar ϵ-kernel of the original input.

3.2 Approximate Smallest Volume Enclosing Hyperbox

Using ϵ-kernels we prove the following theorem.

▶ **Theorem 6.** *Given an exact algorithm for finding the min-volume axis-aligned enclosing hyperbox that runs in time $O(n^a)$, there exists an algorithm that finds a $(1+\epsilon)$-approximate smallest volume axis-aligned avatar enclosing hyperbox in time $O((nk)^{(2d+3)} \cdot \frac{n}{\delta^d} \cdot (2d)^2 \cdot 2^{\frac{1}{\delta^d}(\frac{2}{\delta^{d-1}})^{\lfloor \frac{d}{2} \rfloor}} + (\frac{2}{\delta^{d-1}})^a)$. Note that the choice of constant δ depends on k, ϵ, and α.*

Proof. We can compute a $(1+\epsilon)$-approximate smallest volume axis-aligned enclosing hyperbox $B(L)$ containing an avatar of each entity in L after finding an ϵ'-kernel of L, for some constant ϵ'. Let $\mathcal{CH}(L)$ be the smallest avatar convex hull of a set L of k-avatar points. If

Q is a k-avatar ϵ'-kernel of L such that $Q \subset \mathcal{CH}(L)$, then we have:

$$\begin{aligned}(1-\epsilon')\cdot\omega(u,L) &\leq \omega(u,Q), \quad \forall u \in S^{d-1}\\ (1-\epsilon')\cdot\omega(u,L) &\leq \omega(u,Q), \quad \forall u \in [d] = \{e_1, e_2, \ldots, e_d\}\\ (1-\epsilon')^d \prod_{u\in[d]} \omega(u,L) &\leq \prod_{u\in[d]} \omega(u,Q)\end{aligned}$$

There exists a constant c (function of ϵ' and d), such that $(1-c\epsilon') \leq (1-\epsilon')^d$, thus implying the following:

$$\begin{aligned}(1-c\epsilon') \prod_{u\in[d]} \omega(u,L) &\leq \prod_{u\in[d]} \omega(u,Q)\\ (1-c\epsilon')\cdot \mathit{Volume}(B(L)) &\leq \mathit{Volume}(B(Q))\end{aligned}$$

By choosing $\epsilon = \frac{1}{1-c\epsilon'}$, we obtain a $(1+\epsilon)$-approximation of the smallest volume axis-aligned enclosing rectangle, since

$$1 \leq \frac{(1+\epsilon)\cdot \mathit{Volume}(B(Q))}{\mathit{Volume}(B(L))} \leq (1+\epsilon)$$

◀

Similar results can be achieved for an approximate min-diameter (see Section 3.3) and min-perimeter axis-aligned avatar enclosing box.

3.3 $(1+\epsilon)$-Approximate Avatar Diameter

This section gives yet another result using ϵ-kernels.

▶ **Definition 7.** Define the minimum *avatar* diameter $diam(L)$ of a set L of avatar points as the diameter of the *avatar assignment* $A(L)$ with the smallest diameter.

▶ **Theorem 8.** *Given an exact algorithm for finding the diameter of a convex hull that runs in time $O(n^a)$, there exists an algorithm that computes a $(1+\epsilon)$-approximate smallest k-avatar diameter in time $O((nk)^{(2d+3)} \cdot \frac{n}{\delta^d} \cdot (2d)^2 \cdot 2^{\frac{1}{\delta^d}} (\frac{2}{\delta^{d-1}})^{\lfloor \frac{d}{2} \rfloor} + (\frac{2}{\delta^{d-1}})^a)$. Note that the choice of constant δ depends on k, ϵ, and α.*

Proof. We can use the procedure described in Section 3 to find an *avatar* ϵ'-kernel Q. (We find Q by finding an ϵ'-approximate smallest convex hull $\mathcal{CH}(Q)$ of L.) Our measure function is $\mu(.) = diam(.)$. This measure function $diam(.)$ has the property that $diam(L) = diam(\mathcal{CH}^*(L))$. Let $\bar{u} \in S^{d-1}$ be the direction of $diam(L)$. Then we have that:

$$\begin{aligned}(1-\epsilon')\omega(u,L) &\leq \omega(u,Q), \quad \forall u \in S^{d-1}\\ (1-\epsilon')\cdot diam(L) &\leq \omega(\bar{u},Q)\\ &\leq diam(Q)\\ diam(L) &\leq (1+\epsilon)\cdot diam(Q), \quad \text{where } \epsilon = \frac{1}{1-\epsilon'}\end{aligned}$$

Thus we can just return $(1+\epsilon)\cdot diam(Q)$, which gives us an $(1+\epsilon)$-approximate minimum avatar diameter knowing that:

$$\begin{aligned}diam(Q) \leq diam(L) &\leq (1+\epsilon)\cdot diam(Q)\\ 1 \leq \frac{(1+\epsilon)\cdot diam(Q)}{diam(L)} &\leq (1+\epsilon)\end{aligned}$$

◀

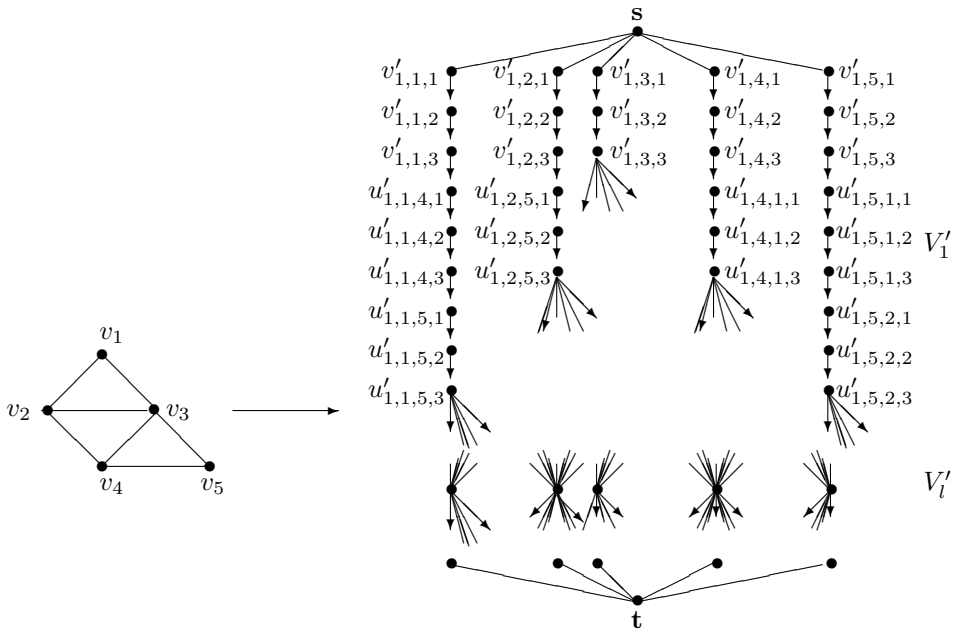


Figure 5 Sketch of reduction from CLIQUE to avatar vertex reachability.

4 Avatar Problems in Graphs and Metric Spaces

In this section we consider the hardness of the avatar versions of vertex reachability and shortest paths in unweighted graphs. The results easily generalize to weighted graphs and metric spaces. Vertex reachability has ties to *rainbow connectivity* problems from the graph theory literature [6]. As before, in order to set the stage, we provide some formal definitions.

4.1 Avatar graph reachability

A k-avatar graph $G(V, E, L, \mathcal{A})$ (or simply an "avatar" graph) consists of the following: a set of vertices V; a set of edges E connecting pairs of vertices in V; a set of entities $L = \{a_1, \ldots, a_m\}$; and a collection of disjoint avatar sets $\mathcal{A} = \{A_1, \ldots, A_m\}$ such that $\forall i$, $A_i \subseteq V$ is the avatar set for entity i, $|A_i| \leq k$, and $A_i \cap A_j = \phi$ if $i \neq j$. An *avatar path* in G is a path p such that no two vertices on the path p are avatars of the same entity.

The *k-avatar reachability problem* is stated as follows: *Given an avatar graph G and two vertices s and t in G determine wether there is an avatar path p from s to t.* Reachability is a fundamental graph problem and can be solved in linear time using simple techniques such as DFS or BFS. Surprisingly enough, in the avatar setting it turns out to be NP-complete, even for $k = 2$.

▶ **Theorem 9.** *The k-avatar reachability problem is NP-complete, for $k \geq 2$.*

Proof. The reduction is from the CLIQUE problem. Let graph $G_C(V, E)$ and integer k denote an instance of the CLIQUE problem. (G_C, k) is a YES instance if and only if G_C contains a clique of size k. We construct graph $G_A(V', E')$ as follows (see Fig. 5): create $k + 2$ layers of vertex sets, $V'_0, V'_1, \ldots, V'_{k+1}$. Let $V'_0 = \{s\}$ and $V'_{k+1} = \{t\}$. For $l = 1, \ldots, k$, let $V'_l = \{v'_{l,i,j} : 1 \leq i \leq |V|, 1 \leq j \leq k\}$. Vertices $v'_{x,i,y}$ and $v'_{y,i,x}$ correspond to avatars of the same entity A_i (prevents picking same vertex from 2 different layers). Add edges

$(v'_{l,i,j}, v'_{l,i,j+1})$ for all l, i, and j; for $0 \leq l < k$, add edges $(v'_{l,i,k}, v'_{l+1,j,1})$ for all i, j. Note that the vertices $v'_{l,i,1}, \ldots, v'_{l,i,k}$ in layer l form k connected subpaths. Denote the subpath from $v'_{l,i,1}$ to $v'_{l,i,k}$ by $S_{l,i}$. It is important to note that for each vertex $v_i \in V$, there is exactly one corresponding subpath in each layer V'_l. Now for each pair of non-adjacent vertices $v_i, v_j \in V$ from G_C, add vertices $u'_{l,i,j,x}$ and $u'_{l,j,i,x}$, for all $1 \leq x \leq k$ and $1 \leq l \leq k$ to G_A. Add edges $(u'_{l,i,j,x}, u'_{l,i,j,x+1})$ and $(u'_{l,j,i,x}, u'_{l,j,i,x+1})$, $1 \leq x < k$. Update subpaths $S_{l,i}$ by taking each outgoing edge from it and make it outgoing from $u'_{l,i,j,k}$ instead, and then adding an edge from it to $u'_{l,i,j,1}$, effectively making $u'_{l,i,j,1}$ the new last vertex of subpath $S_{l,i}$. Finally, let each pair of vertices $u'_{x,i,j,y}, u'_{y,j,i,x}$ be avatars of each other, for all $1 \leq x, y \leq k$.

It is not hard to see that the size of the graph G_A is polynomial in n. More importantly, we claim that if the set $\{v_{i_1}, \ldots, v_{i_k}\}$ is a clique in G_C, then a 2-avatar path can be found from s to t in G_A by starting at s (layer V'_0), moving from each layer to the next, selecting in each layer a subpath corresponding to a distinct vertex from the clique. Intuitively, if the path in level l goes through vertices of the form $v'_{l,i,j}$, then vertex i is chosen as the l-th vertex in the clique. Furthermore, the vertices of the form $u'_{l,i,j,x}$ that are required to be visited by the path (and its avatars) ensure that the other vertices picked for the clique are indeed adjacent to i. The converse is proved by starting from a 2-avatar path and selecting the clique vertices based on the subpaths traversed in each layer. Hence there is a clique of size k in G_C if and only if there is a 2-avatar $s \rightsquigarrow t$ path in G_A. It is readily shown that 2-avatar reachability is in NP, thus completing the proof that it is NP-complete. ◀

4.2 Avatar Shortest Paths

Given an unweighted (or unit-weighted) k-avatar graph and two vertices s and t in the graph, find the shortest length avatar path from s to t. Given that reachability is hard in the avatar setting, the shortest path would be expected to be at least as hard. The following theorem highlights its inapproximability.

▶ **Theorem 10.** *The k-avatar shortest path problem is APX-Hard for $k \geq 2$.*

The key to the proof is a gap-preserving reduction from the maximum clique problem, which is known to be APX-hard, implying immediately that it cannot be approximated to within any constant factor in polynomial time. The proof is omitted here because of limited space.

▶ **Lemma 11.** *There is a gap-preserving reduction from the max-clique problem to the 2-avatar shortest path problem that transforms a graph $G_{\text{clique}}(V, E)$ to a graph $G_{\text{avatar}}(V', E')$ such that:*
1. *if $OPT_{\text{clique}}(G) \geq k \cdot |V|$, $OPT_{\text{avatar}}(s-t) \leq m$, and*
2. *if $OPT_{clique}(G) < \alpha \cdot (k \cdot |V|)$, $OPT_{\text{avatar}}(s-t) > (2-\alpha) \cdot m$,*
where $m = (k^2 \cdot |V|^3) + 1$, and α and k are any constants such that $0 \leq \alpha, k \leq 1$. Here $OPT_{\text{clique}}(G)$ is the size of the maximum clique in $G_{\text{clique}}(V, E)$, and $OPT_{\text{avatar}}(s-t)$ is the length of the shortest 2-avatar path from a vertex s to a vertex t in $G_{\text{avatar}}(V', E')$.

Open Problems: Open problems from this work include determining the time complexity of the k-avatar versions of minimum MaxGap problem and convex hull.

Acknowledgments. This work was partly supported by NSF Grant (CNS-1018262) and the NSF Graduate Research Fellowship (DGE-1038321). The authors thank Shin-ichi Tanigawa, Daniel Rodriguez, Joshua Kirstein, and Ning Xie for useful discussions on earlier drafts. The detailed reviews by anonymous FSTTCS reviewers is gratefully acknowledged.

References

1 A. Abdullah, S. Daruki, and J. M. Phillips. Range counting coresets for uncertain data. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*, SoCG '13, pages 223–232, New York, NY, USA, 2013. ACM.

2 P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, July 2004.

3 B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

4 G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Journal of Algorithms*, 38(1):91 – 109, 2001.

5 R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman. Efficient update strategies for geometric computing with uncertainty. *Theory of Computing Systems*, 38(4):411–423, 2005.

6 G. Chartrand, G. L. Johns, K. A. McKeon, and P. Zhang. The rainbow connectivity of a graph. *Networks*, 54(2):75–81, 2009.

7 J. Chuzhoy, R. Ostrovsky, and Y. Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Mathematics of Operations Research*, 31(4):730–738, 2006.

8 A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *Journal of Algorithms*, 48(1):135 – 159, 2003.

9 K. Elbassioni, A.V. Fishkin, N.H. Mustafa, and R. Sitters. Approximation algorithms for euclidean group TSP. In *Proc. ICALP*, pages 1115–1126. Springer, 2005.

10 T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihal'ák, and R. Raman. Computing minimum spanning trees with uncertainty. In *Proc. of the 25th Annual STACS*, pages 277–288, 2008.

11 T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62(1):1–18, 2007.

12 T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. In *Proc. of the 32nd annual ACM STOC*, pages 602–607. ACM, 2000.

13 J. Fiala, J. Kratochvíl, and A. Proskurowski. Systems of distant representatives. *Discrete Appl. Math.*, 145(2):306–316, January 2005.

14 A. Goel, S. Guha, and K. Munagala. Asking the right questions: Model-driven optimization using probes. In *Proc. of the 25th ACM Symposium on PODS*, pages 203–212. ACM, 2006.

15 J. Groote and B. Ploeger. Switching graphs. *Intl J Found Comp Sci*, 20(5):869–886, 2009.

16 A. Jørgensen, M. Löffler, and J. M. Phillips. Geometric computations on indecisive points. In F. Dehne, J. Iacono, and J.-R. Sack, editors, *Algorithms and Data Structures*, volume 6844 of *LNCS*, pages 536–547. Springer, 2011.

17 W. Ju and J. Luo. New algorithms for computing maximum perimeter and maximum area of the convex hull of imprecise inputs based on the parallel line segment model. In *Proceedings of the CCCG*, 2009.

18 B. Katz, I. Rutter, and G. J. Woeginger. An algorithmic study of switch graphs. *Acta Inf.*, 49(5):295–312, 2012.

19 S. Khanna and W.-C. Tan. On computing functions with uncertainty. In *Proc. of the 20th ACM Symposium on PODS*, pages 171–182. ACM, 2001.

20 Y.-S. Myung, C.-H. Lee, and D.-W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.

21 C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries. In *Proc. of VLDB*, pages 144–155. Morgan Kaufmann, 2000.

22 S. Safra and O. Schwartz. On the complexity of approximating TSP with neighborhoods and related problems. *Comput. Complex.*, 14:281–307, March 2006.

23 M. van Kreveld and M. Löffler. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, February 2010.

Clustering With Center Constraints

Parinya Chalermsook[1] and Suresh Venkatasubramanian[2]

1 **Max-Planck-Institut für Informatik, Germany**
2 **School of Computing, University of Utah, USA**

Abstract

In the classical MAXIMUM INDEPENDENT SET PROBLEM, we are given a graph G of "conflicts" and are asked to find a maximum conflict-free subset. If we think of the remaining nodes as being "assigned" (at unit cost each) to one of these independent vertices and ask for an assignment of minimum cost, this yields the VERTEX COVER problem. In this paper, we consider a more general scenario where the assignment costs might be given by a distance metric d (which can be unrelated to G) on the underlying set of vertices. This problem, in addition to being a natural generalization of vertex cover and an interesting variant of the k-MEDIAN problem, also has connection to constrained clustering and database repair.

Understanding the relation between the conflict structure (the graph) and the distance structure (the metric) for this problem turns out to be the key to isolating its complexity. We show that when the two structures are unrelated, the problem inherits a trivial upper bound from vertex cover and provide an almost matching lower bound on hardness of approximation. We then prove a number of lower and upper bounds that depend on the relationship between the two structures, including polynomial time algorithms for special graphs.

1998 ACM Subject Classification H.3.3 Clustering, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Clustering, vertex cover, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.401

1 Introduction

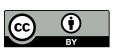
Let $G = (V, E)$ represent a set of *conflicts* between pairs of elements in V and $d : V \times V \to \mathbb{R}$ be a metric capturing the cost of *assigning* one vertex to another. In this paper, we consider the following problem, called *minimum edge-weighted independent set* (MWIS) [18]:

▶ Problem 1.1 (MWIS). Find an independent set of vertices $S \subset V$ such that the *assignment cost* $\sum_{v \in V \setminus S} \min_{s \in S} d(v, s)$ is minimized.

This problem is a natural generalization of VERTEX COVER (when $d(x, y) \equiv 1$ for all $x, y \in V$), and arises naturally in two distinct applications.

Constrained clustering. In a typical application of clustering, the goal is to group close-by objects into a small number of groups. Often the user has domain information about the data in the form of pairs of items that either should be linked together (MUST-LINK) or should not (CANNOT-LINK). Such a clustering problem is called *constrained clustering*[3] and has been studied extensively.

In general, these constraints are provided by users and it can be difficult to make clear judgments about whether two points should be in the same cluster or not, since this depends on the larger context of the clustering. An easier decision is to decide whether two points can serve as cluster *representatives* at the same time or not. Intuitively, if two points are close to each other, then we might expect one or the other to be a cluster center, but not



33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 401–412

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

both. This is less constraining than a MUST-LINK or CANNOT-LINK constraint: two points connected by a constraint can lie in the same cluster or in different clusters, as long as they are not both cluster centers at the same time.

This center-constrained clustering problem is captured by the above formulation. G captures the constraints, and d is the underlying distance function used for clustering. The goal is now to find a set of cluster centers that are not in conflict such that the total cost of assigning points to cluster centers is minimized (here the cost of a cluster is the sum of distances from points to the cluster representative - this is the standard k-median-type cost function). One attractive feature of this formulation is that it does not require us to specify the number of clusters, or in fact any parameter.

Database Repair. In a database, *integrity constraints* like functional dependencies are used to enforce semantic consistency of the data. A simple example of a functional dependency is a *key*: if two tuples have the same value for a key attribute, they must be identical in all attributes. For any database and set of such integrity constraints, a *conflict* is a pair of tuples that does not satisfy the integrity constraint[1].

When a database with conflicts is encountered, one approach to rectify the problem is to *repair* the database by changing tuples to eliminate conflicts. For example if a database had tuples $a = (1, 2)$ and $b = (1, 3)$, but the first attribute was required to be a key, then the database could be repaired either by modifying the first component of a or the second component of b. But each of these possibilities incurs a cost that we would want to keep as low as possible.

Now if we construct a graph where each tuple is a vertex, there is an edge between two tuples if they are in conflict with respect to integrity constraints, and a distance function d captures the cost of changing one tuple to another, the problem of database repair is precisely MWIS. This problem was first formulated by Kolahi et al.[18] and NP-hardness was established via its generalization of VERTEX COVER.

1.1 Our Contributions

In this paper, we study MWIS in its many forms. Our results reveal that the interaction between the conflict graph and the distance measure plays an important role in determining the complexity of the problem. Because of the problem's similarity to clustering (and k-median in particular) one might expect that techniques that have been used to deal with k-median and related problems might be useful in attacking MWIS and potentially give a good approximation algorithm. Our first result shows that this is, in fact, impossible. More formally, we show that there is no polynomial time approximation algorithm that guarantees the approximation ratio of $O(\frac{d_{\max}}{d_{\min}})$, unless $\mathsf{P} = \mathsf{NP}$. This matches the approximation ratio given by the vertex cover heuristic up to a constant factor.

Our proof uses the fact that the graph G and metric d can be completely unrelated to each other, and this suggests that establishing a relationship between G and d could make the problem easier. We have two ways to establish consistency between d and G.

- **Metric induced by graph (w-MWIS):** Given graph $G = (V, E)$ and weights $w : E \to \mathbb{R}$, the metric d is defined as the shortest path metric of (G, w). We denote this problem by w-MWIS.
- **Graph induced by metric (d-MWIS):** Given a metric d, the conflict graph G is defined to be the induced unit-disk graph: $E(G) = \{uv : d(u, v) \leq 1\}$. This problem is abbreviated by d-MWIS.

[1] In general, many different kinds of integrity constraints (but not all!) admit pairs of conflicting tuples.

Table 1 Summary of Our Results. All hardness results are proved under the assumption of P ≠ NP. VC refers to VERTEX COVER.

	w-MWIS				d-MWIS	
Bound	General	Unweighted	Tree-Width	Planar	General	$\mathbb{R}^d$
Upper	$\frac{2d_{\max}}{d_{\min}}$ [18]	VC[18]	Poly	??	$\frac{2d_{\max}}{d_{\min}}$ [18]	$(O(1/\epsilon), 1-\epsilon)$
Lower	$\Omega(\frac{d_{\max}}{d_{\min}})$	VC	Poly		NP-hard	$\Omega(\frac{d_{\max}}{d_{\min}})$

It turns out that d-MWIS captures the class of instances that usually arise in practice, especially when d has low dimensional structure (e.g. Euclidean space or metric of bounded doubling dimension), so algorithmic results for d-MWIS are interesting from a practical point of view.

d-MWIS is hard to approximate in a general metric. Therefore we consider the d-MWIS problem in $\mathbb{R}^d$ and prove that it is NP-hard even when $d = 2$ (and even in the bi-criteria setting). We also show a constant factor bi-criteria approximation algorithm, in the following sense: If C is the minimum possible clustering cost which guarantees that centers are of distance at least 1 apart, our algorithm always outputs a solution of cost $O(C/\epsilon)$ while ensuring that the centers are "almost feasible", i.e. the distance between each pair is at least $(1-\epsilon)$. Our algorithm combines tools from clustering and computational geometry.

Similarly, w-MWIS remains hard in general weighted graphs and so we further investigate two directions. In the first direction, we consider the unweighted graph, i.e. $w(e) = 1$ for all $e \in E(G)$, and observe that the problem is computationally equivalent to VERTEX COVER, so it is hard to approximate to within a factor of $(2-\epsilon)$ assuming UGC [17], 1.36 hard assuming P≠NP [10], and admits slightly better than 2 approximation [14, 16]. In the second direction, we restrict the underlying graph structure, showing that in graphs having bounded treewidth, d-MWIS is polynomial time solvable, while it is still NP-hard in planar graphs.

Finally, a special case of MWIS captures the k-median problem which has received a lot of attention in approximation algorithms [20, 8, 1]. That is, when graph G is simply a union of disjoint cliques, the problem generalizes k-median problem and is a special case of matroid median problem, a more general variant of the classical k-median problem [9, 13, 19]. We summarize these results in Table 1.

Organization. We show our hardness results in Section 3. Results related to d-MWIS and w-MWIS are presented in Section 4 and 5 respectively. The connection between MWIS and the classical k-median problem is deferred to the full version.

2 Preliminaries

Vertex Cover: Given a graph $G = (V, E)$, a set $C \subseteq V$ is a *vertex cover* of G if any edge $uv \in E$ satisfies $\{u, v\} \cap C \neq \emptyset$. The MINIMUM VERTEX COVER problem asks for the minimum cardinality vertex cover of G. A set $S \subseteq V$ is an *independent set* of G if any pair $u, v \in S$ satisfies $uv \notin E(G)$. S is an independent set iff $V \setminus S$ is a vertex cover.

Tree Decomposition: A tree decomposition of graph G is a tree T and collection of subsets of nodes $\mathcal{X} = \{X_t\}_{t \in V(T)}$ with the following properties:

- $V(G) = \bigcup_{t \in V(T)} X_t$.
- For any edge $uv \in E(G)$, there exists $t \in V(T)$ such that both u and v lie in X_t.
- For any $v \in V(G)$, if $v \in X_t \cap X_{t'}$ for $t, t' \in V(T)$, then $v \in X_{t''}$ for any t'' that lies on the unique path from t to t' in T.

The *width* of a tree decomposition is denoted by $\max_{t \in V(T)} (|X_t| - 1)$, and the treewidth of a graph is the minimum possible k such that there is a tree decomposition of width k for G. Bodlaender [4] presented a linear time algorithm that constructs a tree decomposition of width k if the treewidth of a graph is at most k.

Bi-criteria Approximation: Let $\beta < 1$. We say that an algorithm for d-MWIS is an (α, β) approximation if it returns the solution S of open facilities whose cost is at most OPT, i.e. $\sum_{v \in V} d(S, v) \leq \alpha \mathsf{OPT}$ and for each pair $u, v \in S$, we have $d(u, v) \geq \beta$.

3 Hardness of Approximation

In this section, we prove the following theorem.

▶ **Theorem 1.** *Unless $P = NP$, d-MWIS and w-MWIS have no polynomial time $O(\Delta/\delta)$ approximation algorithms, where $\Delta = \max d(x, y)$ and $\delta = \min d(x, y)$.*

The reduction starts from the hardness of approximation result for 3SAT. We will use the following tight result due to Hastad.

▶ **Theorem 2** ([15])**.** *Fix any $\epsilon > 0$ and assume that $P \neq NP$. Given a 3SAT formula, it is hard to distinguish between the following two cases in polynomial time:*

- *(Yes-Instance:) There is an assignment that satisfies every clause.*
- *(No-Instance:) Any assignment satisfies at most $7/8 + \epsilon$ fraction of the clauses.*

Construction: We will construct an instance of w-MWIS and argue later that it can be thought of as an instance of d-MWIS as well. We use a reduction idea introduced by Feige et al. [12]. This reduction has been a powerful tool in transforming any constraint satisfaction problem (CSP) into graph G such that the size of maximum independent set of the graph is "proportional" to the value of the input CSP, i.e. the maximum fraction of clauses that can be simultaneously satisfied.

Consider an instance Φ of 3SAT with n variables $\{x_1, \ldots, x_n\}$ and m clauses $C_1, \ldots, C_m$, as given by the above theorem. We show how to construct graph $G = (V, E)$. For each clause C_i, we create a set V_i containing 7 vertices, where each such vertex represents a satisfying assignment for clause C_i. Now vertex set V can be defined as $V = \bigcup_{i=1}^{m} V_i$, so $|V| = 7m$.

We proceed to define the set of edges E. We connect every pair of vertices in V_i by an edge, so each set V_i is a clique of 7 vertices. The distance between two vertices $u, v \in V_i$ is defined as $d(u, v) = 1$. Then we say that any two vertices $u \in V_i, v \in V_j$ where $i \neq j$ are *conflicting* if there is a variable x_ℓ in the formula Φ that belongs to both clauses C_i and C_j such that the assignment of u and v set $x_\ell = 0$ and $x_\ell = 1$ respectively. Then for any two conflicting vertices $u \in V_i$ and $v \in V_j$, we connect an edge uv with distance $d(u, v) = \rho$ where ρ is a parameter. It is easy to verify that d is indeed a metric.

Analysis: It can be argued that in the Yes-Instance, there is a solution of cost at most $6m$, while in the No-Instance the cost must be at least $7m\rho/10$. So we get a gap of $\Omega(\rho)$. We omit details in this extended abstract.

4 Graph Induced by Metric (d-MWIS)

The main result in this section is a $(1/\epsilon, 1 - \epsilon)$ bi-criteria approximation algorithm for Euclidean space. The idea of this algorithm follows the LP rounding algorithm for k-median

of Charikar et al [8]. The "natural" LP requires independent set constraints, and these yield an unbounded integrality gap even for the line metric. Hence, we need a stronger LP.

First, we argue that the problem already becomes non-trivial in $\mathbb{R}^2$ even if we only desire a bi-criteria approximation algorithms.

▶ **Theorem 3.** *d-MWIS is NP-hard even in $\mathbb{R}^2$. Moreover, finding $(1, 1-\epsilon)$ approximation is also NP-hard for any $\epsilon < 1/2$.*

The proof follows by showing a reduction from VERTEX COVER on coin graphs (which is NP-hard [5])and is deferred to the full version. We also note that the problem in 1-dimensional Euclidean space is polynomial time solvable by a simple dynamic program.

4.1 Geometric LP Relaxation

Since the independent set constraints cause large LP integrality gap, we use the LP with clique point constraints, as used successfully in many geometric packing problems (see, e.g., [11, 7, 6]). We have variables $x(u)$ to indicate the fact that the facility is open at u and variables $y(u, v)$ which say that the client at v is served by a facility at u. We first formulate the linear program, denoted by (LP).

(LP)

$$\min \sum_{u,v \in V(G)} d(u,v)y(u,v)$$

$$\text{s.t} \sum_{u:d(u,p)<1/2} x(u) \leq 1 \ (\forall p \in \mathbb{R}^2)$$

$$y(u,v) \leq x(u) \ \forall u, v \in V(G)$$

$$\sum_{v \in V(G)} y(u,v) \geq 1 \ \forall u \in V(G)$$

(LP')

$$\min \sum_{u,v \in V(G)} d(u,v)y(u,v)$$

$$\text{s.t.} \sum_{u:p \in r_u} x(u) \leq 1 \ (\forall p \in \mathcal{P})$$

$$y(u,v) \leq x(u) \ \forall u, v \in V(G)$$

$$\sum_{v \in V(G)} y(u,v) \geq 1 \ \forall u \in V(G)$$

The first set of constraints ensure that we do not open conflicting centers, while the second set of constraints force the LP to only assign clients to facilities that are open. Notice an important property of this LP that, once vector $\mathbf{x}$ is fixed, the values of $\mathbf{y}$ that minimize the LP cost can be computed efficiently, so we will sometimes refer to any LP solution as $\mathbf{x}$, instead of $(\mathbf{x}, \mathbf{y})$. The following lemma says that this LP is a valid formulation for MWIS.

▶ **Lemma 4.** *Any integral vector $\mathbf{x}$ is feasible for (LP) if and only if the set $S = \{u : x(u) = 1\}$ is a feasible solution to d-MWIS.*

Proof. First, assume that $\mathbf{x}$ is integral but not feasible for (LP), so it must violate some constraint $\sum_{p:d(u,p)<1/2} x(u) > 1$ for some point p. We will argue that the solution S is not feasible. Since $\mathbf{x}$ is integral, there must be at least two vertices u, u' such that $x(u) = x(u') = 1$ and that $d(u, p), d(u', p) < 1/2$, implying that $d(u, u') \leq d(u, p) + d(p, u') < 1$. Since $u, u' \in S$, the set S is not feasible for MWIS.

Now assume that the set S defined this way is not feasible for MWIS. We will argue that $\mathbf{x}$ violates some constraint of (LP). Since S is not feasible, there must be two facilities $u, u' \in S$ such that $d(u, u') < 1$. We pick the point p to be at the middle of the line segment connecting u and u', so we must have $d(u, p), d(u', p) < 1/2$, so the LP constraint is violated at point p. ◀

From now on, we think of each vertex $u \in V(G)$ as a geometric object, i.e. a ball r_u of radius $1/2$, and therefore the first set of LP constraints can also be seen as $\sum_{u:p \in r_u} x(u) \leq 1$.

If we consider the arrangement formed by the balls, then in each cell of this arrangement, for each point p in the cell, the sum $\sum_{d(u,p)<1/2} x(u)$ is a fixed constant. Hence, we need only choose one representative point in each cell to verify the constraints. Since the arrangement of balls in d dimensions has complexity $n^{O(d)}$ (by using a simple lifting map argument) and can be computed in similar time, we can write a new polynomial sized LP, denoted by (LP').

▶ **Lemma 5.** *Vector* $\mathbf{x}$ *is feasible for (LP) if and only if it is feasible for (LP').*

Proof. We only need to show that a feasible solution $\mathbf{x}$ for (LP') is always feasible for (LP). Suppose not. Then there must be a point $p \in \mathcal{P}$ such that $\sum_{u:p\in r_u} x(u) > 1$, and since the set of r_u containing p is maximal, there is a point $p' \in \mathcal{P}$ such that the same collection of disks contains p'. Hence, the constraint of (LP') is violated at p', a contradiction. ◀

4.2 LP Rounding Algorithm

Let ϵ be a parameter. We present $(O(1/\epsilon), 1-\epsilon)$ approximation algorithm. In the first step, we perform the clustering process as used in Charikar et al.[9] where clients are grouped into many clusters. Each cluster contains, roughly speaking, clients that will be served by the same facility. In the second step, we open facilities in some of these clusters. The property of the geometric LP guarantees that the opened clusters are "far".

Step 1: Clustering and Preprocessing. We define for each vertex $v \in V(G)$, the quantity $\bar{C}_v = \sum_u d(u,v)y(u,v)$. The term $\bar{C}_v$ represents the "fractional connecting cost" of v, and we can write $\mathsf{OPT} = \sum_{v\in V(G)} \bar{C}_v$. So our goal is to use this term to bound the actual cost created by our algorithm.

First, we say that a vertex $v \in V(G)$ is *heavy* if $\bar{C}_v \geq \epsilon/10$; otherwise, the vertex v is called *light*. Denote the sets of heavy and light vertices by V^h and V^l respectively. So we can partition vertices in $V(G)$ into $V(G) = V^h \cup V^l$, which will be handled separately by our algorithm: To deal with light vertices, we order the vertices in V^l by their values $\bar{C}_v$ in increasing order, i.e. $V^l = \{v_1, v_2, \ldots, v_{n'}\}$ such that $\bar{C}_{v_1} \leq \bar{C}_{v_2} \leq \ldots \leq \bar{C}_{v_{n'}}$. Initially, we define $\mathcal{C} = \emptyset$. We process the vertices in this order, starting from v_1. When v_i is processed, we consider $V_i = \left\{v_{i'} \in V^l : d(v_i, v_{i'}) \leq 3\bar{C}_{v_i}\right\}$ and check if there is another $j : j < i$ such that $v_j \in \mathcal{C}$ and $V_i \cap V_j \neq \emptyset$. If there is no such cluster, we add v_i to the collection of cluster centers $\mathcal{C}$; otherwise, we say that vertex v_i is *assigned* to center v_j.

When the above process finishes, we obtain a collection of cluster centers $\mathcal{C}$. The following observations follow immediately.

▶ Observation 4.1. For any vertex $v_i \in V^l$, either $v_i \in \mathcal{C}$ or vertex v_i is assigned to some other vertex $v_j \in \mathcal{C}$. In the second case, $d(v_i, v_j) \leq 3\bar{C}_{v_i} + 3\bar{C}_{v_j}$.

▶ Observation 4.2. For any $V_i, V_j \in \mathcal{V}$, we have $V_i \cap V_j = \emptyset$.

For each cluster center $v_j \in \mathcal{C}$, we define the combined demand D_j to be the number of vertices assigned to v_j, including v_j itself. Observe that $\sum_{j\in\mathcal{C}} D_j = |V^l|$. From now on, we will think of v_j as the representative of all clients assigned to it, and we will only try to open facilities to serve v_j. Now our goal is to solve the following new problem, instead of the original one: Given a collection of clients $\mathcal{C}$, the goal is to open the non-conflicting centers among the vertices in $\mathcal{C}$ so as to minimize the demand-weighted cost $\sum_{v_j\in\mathcal{C}} D_j d(S, v_j)$.

We ensure that the solution of this new problem can be turned into that of the old one without paying much cost, as stated in the following lemma.

▶ **Lemma 6.** *Let $S \subseteq \mathcal{C} \cup V^h$ be any subset of vertices that is maximal, w.r.t. V^h in the sense that any vertex $u \in V^h$ either belongs to S or is in conflict with some other vertex $u' \in S$. Then we have $\sum_{v \in V(G)} d(S,v) \leq O(1/\epsilon)\mathsf{OPT} + \sum_{v_j \in \mathcal{C}} D_j d(S, v_j)$*

Proof. Recall that $V(G) = V^l \cup V^h$. Since S is maximal, we know that the connecting cost of any vertex is never more than one. We first analyze the connecting cost of vertices in V^h. Consider vertex $v \in V^h$. Since $\bar{C}_v \geq \epsilon/10$, we have $d(S,v) \leq 1 \leq 10\bar{C}_v/\epsilon$. Summing over all $v \in V^h$, we get $\sum_{v \in V^h} d(S,v) \leq 10 \sum_{v \in V^h} \bar{C}_v/\epsilon \leq (10/\epsilon)\mathsf{OPT}$.

Now we bound the connecting cost of vertices v_i in V^l. If vertex v_i is the center of some cluster, i.e. $v_i \in \mathcal{C}$, we have the connecting cost $d(v_i, S)$. Otherwise from Observation 4.1, v_i is assigned to some cluster center $v_j \in \mathcal{C}, j < i$, so the connecting cost from v_i to the nearest opened facility is at most $d(S,v_i) \leq d(v_i,v_j) + d(S,v_j)$. Since $j < i$, it must be the case that $\bar{C}_{v_i} \geq \bar{C}_{v_j}$, so we have $d(v_i,v_j) \leq 3\bar{C}_{v_i} + 3\bar{C}_{v_j} \leq 6\bar{C}_{v_i}$. Finally, we have $d(S,v_i) \leq 6\bar{C}_{v_i} + d(S,v_j)$ in this case, so if we sum over all vertices v_i that have been assigned to v_j, we would obtain the bound $6\sum_{i:v_i \text{ assigned to } v_j} \bar{C}_{v_i} + D_j d(S,v_j)$.

Summing over all vertices gives $\sum_{v \in V^l} d(v,S) \leq 6\sum_{v \in V^l} \bar{C}_v + \sum_{v_j \in \mathcal{C}} D_j d(S,v_j)$. ◀

Step 2: Opening the Facilities. For each $v_j \in \mathcal{C}$, we simply open facility v_j. Then, we process heavy vertices in V^h in arbitrary order, and we open a facility if it does not conflict with any other already opened facility (remark that the conflicts might have been created already by opening all vertices $v_j \in \mathcal{C}$, but we avoid creating more conflict). This algorithm ensures that the resulting set S of opened facilities is a maximal set with respect to V^h, and observe that $\sum_{v_j \in \mathcal{C}} d(S,v_j) = 0$. So invoking Lemma 6 implies that $\sum_{v \in V(G)} d(S,v) \leq O(\mathsf{OPT}/\epsilon)$.

It only remains to analyze the distance between centers in S, which is done in the following lemma.

▶ **Lemma 7.** *Let S be the set of facilities opened by the algorithm. Then, for any pair $u, v \in S$, $d(u,v) \geq 1 - \epsilon$.*

Proof. Notice that we only need to analyze the pair of vertices in S that were once the clusters in $\mathcal{C}$ (because other vertices are added arbitrarily in a way that ensure no conflict). We will need the following claim.

▶ Claim 4.1. For all $v_i \in \mathcal{C}$, we have $\sum_{u \in V_i} x(u) \geq 2/3$

Proof. In fact, this claim can be seen as just a simple application of Markov's inequality: The term $\bar{C}_{v_i}$ is simply an expectation of connecting cost of v_i, where v_i is connected to u with probability $x(u)$. To be more formal, we write $\bar{C}_{v_i} = \sum_v y(v,v_i) d(v,v_i)$. Since $\sum_v y(v,v_i) = 1$, the terms $\{y(v,v_i)_v\}$ can be seen as distribution μ such that $\mathbb{E}_{v \sim \mu}[d(v,v_i)] = \bar{C}_{v_i}$. Applying Markov's inequality, the probability that $d(v,v_i) > 3\bar{C}_{v_i}$ is at most $1/3$, or in other words, $\sum_{v:d(v,v_i) \leq 3\bar{C}_{v_i}} y(v,v_i) \geq 2/3$. This implies that $\sum_{v:d(v,v_i) \leq 3\bar{C}_{v_i}} x(v) \geq 2/3$, as desired. ◀

Now consider $v_i, v_j \in \mathcal{C}$ and assume (for contradiction) that $d(v_i,v_j) < 1 - \epsilon$. Consider a point p whose distance to v_i and v_j are equal, so we have $d(p,v_i) = d(p,v_j) < 1/2 - \epsilon/2$. By triangle inequality, for any $u \in V_i$, the distance $d(u,p) \leq d(u,v_i) + d(v_i,p) \leq 1/2 - \epsilon/2 + 3\epsilon/10 < 1/2$ (also because $d(u,v_i) \leq 3\bar{C}_{v_i} \leq 3\epsilon/10$.)

So the balls r_u contain point p for all $u \in V_i$. By similar arguments, balls r_v contain point p for all $v \in V_j$, and this implies that $\sum_{u:p \in r_u} x(u) \geq \sum_{u \in V_i} x(u) + \sum_{u \in V_j} x(u) \geq 4/3$, from the claim and Observation 4.2; this contradicts the fact that the LP is feasible. Notice that point p may not belong to $\mathcal{P}$, but from the fact that $\mathcal{P}$ contains important points, we must have another point $p' \in \mathcal{P}$ such that $\sum_{u:p' \in r_u} x(u) > 4/3$. ◀

5 Metric Induced by Graphs (w-MWIS)

Recall that in the w-MWIS problem, we are given a graph G together with weight function $w : E(G) \to \mathbb{R}_{\geq 0}$. Metric d is then defined as an induced shortest path metric on (G, w). In this section, we consider special cases of w-MWIS when $w = 1$ and when the treewidth of graph G is bounded.

5.1 Unweighted Graphs

In this section, we consider the metric induced by unweighted graph (i.e. $w(e) = 1$ for all $e \in E(G)$) and prove that the problem is again equivalent to VERTEX COVER; the proof is simple and deferred to the full version.

▶ **Theorem 8.** *Unweighted w-MWIS is computationally equivalent to Vertex Cover. More formally, for any ρ, there is an algorithm that gives a ρ-approximation to w-MWIS on G if and only if there is a ρ-approximation to Vertex Cover G.*

One corollary of the above theorem is that, w-MWIS becomes NP-hard even in planar graphs, and there is a PTAS when the graph is unweighted (this follows from the fact that the minimum vertex cover problem admits a PTAS in planar graphs [2]).

▶ **Corollary 9.** MWIS *is NP-hard on unweighted planar graphs.*

5.2 Algorithm for weighted trees

In this section, we consider d-MWIS on trees and give a polynomial time algorithm for computing the optimal solution. Let (T, w) be the input where T is a tree and $w : E(T) \to \mathbb{R}$ is a weight function. Recall that VERTEX COVER is solvable in polynomial time on trees, and since VERTEX COVER is equivalent to w-MWIS when $w = 1$, our algorithm can be thought of as a strengthened version of vertex cover algorithm to handle a more general problem.

First we show some structural properties that suggest our dynamic programming. The solution for MWIS can be described by a set S of centers that are open, and given such set S, we can naturally define an assignment function $\alpha : V(G) \to S$ that assigns each vertex $v \in V(G)$ to its closest opened center. Let T denote the tree instance with the corresponding induced metric $d_T : V(T) \times V(T) \to \mathbb{R}$. We root the tree at arbitrary vertex $r \in V(T)$, and define, for each vertex $v \in V(T)$, the subtree T_v as the subtree of T rooted at v (including v itself). The following lemma characterizes the properties of the optimal solution on the tree.

▶ **Lemma 10.** *Let S^* be the set of opened centers in the optimal solution and $\alpha^* : V(T) \to S^*$ be the corresponding assignment. The following properties hold:*

- *For two vertices $u, v \in V(T)$ such that v is a parent of u in the tree, if $\alpha^*(v) \in T_u$ then $\alpha^*(u) = \alpha^*(v)$.*
- *For any two vertices u, v, we have $d_T(\alpha^*(u), u) \leq d_T(\alpha^*(v), u)$*

Proof. To prove the first property, assume this is not the case. Then there must be another vertex $w \in T_v \cap S^*$ such that $d_T(w, u) < d_T(\alpha^*(v), u)$, but this implies that $d_T(w, v) = d_T(w, u) + d_T(u, v) < d_T(\alpha^*(v), u) + d_T(u, v) = d_T(\alpha^*(v), v)$, a contradiction. The second property is obvious from the definition of α^*. ◀

Algorithm

Define the subproblem $C(v, x)$ as the minimum possible assignment cost of vertices in the subtree T_v with the constraint that vertex v is assigned to an opened center x (which may not necessarily belong to T_v). The optimal solution we are looking for is stored in the entry $C(r, \alpha^*(r))$ (which can be enumerated once the entries are computed correctly). The table entries are computed from the leaf to the root of the tree. The base case is defined on the leaves of the tree simply by: For each leaf $v \in V(T)$, we define $C(v, x) = d_T(v, x)$.

Now fix an entry $C(v, x)$. We show how to compute $C(v, x)$ once the subproblems have been computed. Let $v_1, \ldots, v_\ell$ be the children of v in the tree. For each child v_i such that $x \notin T_{v_i}$, we define the set of possible center candidates for v_i as

$$\Gamma_{v_i,x} = \{x\} \cup \{y \in T_{v_i} : yx \notin E(T), d_T(v_i, y) \leq d_T(v_i, x) \text{ and } d_T(v, y) \geq d_T(v, x)\}$$

Otherwise, if $x \in T_{v_i}$, define $\Gamma_{v_i,x} = \{x\}$. Then we can write our recurrence as:

$$C(v, x) = d_T(v, x) + \sum_{i=1}^{\ell} \min_{y \in \Gamma_{v_i,x}} C(v_i, y)$$

It follows from Lemma 10 that restricting our choices to $\Gamma_{v_i,x}$ still includes the optimal solution, so we only need to show that a feasible solution can be constructed from the table entries.

Correctness

We show that given the table entries $C(v, x)$ for all $v, x \in V(T)$, we can reconstruct a feasible solution that assigns vertices in T_v with total cost $C(v, x)$. This can be argued by induction on the tree structure as summarized in the following lemma.

▶ **Lemma 11.** *For each table entry $C(v, x)$, we can construct a corresponding partial solution $S^*_{(v,x)}$ (where $S^*_{(v,x)}$ is the set of opened centers) such that the assignment cost inside the subtree T_v is bounded by $\sum_{u \in T_v} d_T(S^*_{(v,x)}, u) \leq C(v, x)$. Moreover $S^*_{(v,x)} \subseteq T_v \cup \{x\}$, $x \in S^*_{(v,x)}$, and $S^*_{(v,x)}$ is an independent set in T.*

Proof. We will prove this lemma by induction on the distance of vertices from the leafs (i.e. in order from leafs to root). The base case when v is a leaf is trivial: We can simply define $S^*_{(v,x)} = \{x\}$ for all $x \in V(T)$, so this is clearly an independent set where $x \in S^*_{(v,x)}$ and $S^*_{(v,x)} \subseteq T_v \cup \{x\}$. Now consider a vertex v with children $v_1, \ldots, v_\ell$ and assume that the induction hypothesis holds for $C(v_i, y)$ for any v_i and y. Our goal is to show that it holds for $C(v, x)$.

For each $i \in [\ell]$, let $y_i \in \Gamma_{v_i,x}$ be the vertex such that $C(v_i, y_i)$ is minimized, so we can write $C(v, x)$ as $C(v, x) = d_T(v, x) + \sum_{i=1}^{\ell} C(v_i, y_i)$. We construct solution $S^*_{(v,x)}$ by $S^*_{(v,x)} = \{x\} \cup \bigcup_{i=1}^{\ell} S^*_{(v_i,y_i)}$, so it is immediate that $x \in S^*_{(v,x)}$. It only remains to show that (i) $S^*_{(v,x)}$ is an independent set, (ii) $S^*_{(v,x)} \subseteq T_v \cup \{x\}$, and (iii) that $\sum_{u \in T_v} d_T(S^*_{(v,s)}, u) \leq C(v, x)$.

To prove (i), assume (for contradiction) that there is an edge $w_i w_j$ for $w_i \in S^*_{(v_i,y_i)}$ and $w_j \in S^*_{(v_j,y_j)}$ for some $i \neq j$. If both $w_i \in T_{v_i}$ and $w_j \in T_{v_j}$, then it is impossible to have an edge between them, so there must be one of them that lies outside its tree. Assume that $w_i \notin T_{v_i}$ (the other case is symmetric), we must have $w_i = y_i$ because $S^*_{(v_i,y_i)} \subseteq T_{v_i} \cup \{y_i\}$. And since $y_i \in \Gamma_{v_i,x} \subseteq T_{v_i} \cup \{x\}$, the only possibility for y_i to be outside of T_{v_i} is that $y_i = x$. We now do case analysis.

- If $w_i \notin T_{v_i}$ but $w_j \in T_{v_j}$, either ($w_i = v$ and $w_j = v_j$) or $w_i \in T_{v_j}$. The latter is impossible because it would force $\Gamma_{v_j,x} = \{x\}$ (because $x \in T_{v_j}$), and hence both $w_i = x$ and w_j belong to $S^*_{(v_j,y_j)}$ by induction hypothesis, contradicting the fact that $S^*_{(v_j,y_j)}$ is independent. For the former case, $w_j = v_j$ implies that $w_j \notin \Gamma_{v_j,x}$, and in particular $w_j \neq y_j$. Now since $S^*_{(v_j,y_j)}$ is computed from $\{y_j\} \cup \bigcup_{v'_r} S^*_{(v'_r,y'_r)}$ where vertices v'_r are children of v_j, we must have w_j in some set $S^*_{(v'_r,y'_r)}$. This is only possible if $w_j = y'_r$ for some r, a contradiction to the fact that $w_j \notin \Gamma_{v'_r,y_j}$.
- If $w_i \notin T_{v_i}$ and $w_j \notin T_{v_j}$, then $w_i = y_i$ and $w_j = y_j$, but since $y_i \in T_{v_i} \cup \{x\}$, we must have $w_i = x$; similarly, since $y_j \in T_{v_j} \cup \{x\}$, it must be the case that $w_j = x$, a contradiction.

Next, we turn to prove (ii). Assume for contradiction that $S^*_{(v,x)} \not\subseteq T_v \cup \{x\}$. Denote by $z \in S^*_{(v,x)} \setminus (T_v \cup \{x\})$. Since $S^*_{(v,x)}$ is obtained from the union of $S^*_{(v_i,y_i)}$ and $\{x\}$, it must be the case that $z \in S^*_{(v_i,y_i)}$ for some i, and since y_i is the only node outside of T_{v_i}, we have $z = y_i \neq x$. This implies that $y_i \notin \Gamma_{i,x}$, a contradiction.

Finally, to prove (iii), we write $C(v,x)$ as $C(v,x) = d_T(v,x) + \sum_{i=1}^{\ell} C(v_i,y_i)$, where $C(v_i,y_i) \geq \sum_{u \in T_{v_i}} d_T(S^*_{(v_i,y_i)},u)$ by induction hypothesis. Since $x \in S^*_{(v,x)}$, we have that $d_T(S^*_{(v,x)},v) \leq d_T(v,x)$. Moreover, for each $u \in T_{v_i}$, $d_T(S^*_{(v,x)},u) \leq d_T(S^*_{(v_i,y_i)},u)$ because $S^*_{v_i} \subseteq S^*_v$, and $\sum_{u \in T_{v_i}} d_T(S^*_{(v_i,y_i)},u) \leq C(v_i,y_i)$ by induction hypothesis. This implies

$$\sum_{u \in T_v} d_T(S^*_{(v,x)},u) = d_T(S^*_{(v,x)},v) + \sum_i \sum_{u \in T_{v_i}} d_T(S^*_{(v,x)},u) \leq d_T(v,x) + \sum_{i=1}^{\ell} C(v_i,y_i) = C(v,x)$$

◀

5.3 Extension to Graphs of Bounded Treewidth

In this section, we extend the algorithm on trees to give polynomial time algorithm for graphs of bounded treewidth. The key ideas remain the same as the tree case, but the algorithm and analysis are more involved. For a subset $S \subseteq V(G)$, denote by $G[S]$ an induced subgraph of G on vertices in S. Let $(T,\mathcal{X})$ be a tree decomposition of G and assume that the tree is rooted at an arbitrary vertex $r \in T$. Assume that the treewidth of G is at most w, so we have $|X_t| \leq w$ for each $t \in T$. For each vertex $t \in T$, we define $G^{(t)}$ to be an induced subgraph of G on nodes $\bigcup_{t' \in T_t} X_{t'}$, i.e. $G^{(t)} = G[\bigcup_{t' \in T_t} X_{t'}]$. These subgraphs define our subproblems.

For each $t \in V(T)$, a *feasible partial assignment* for t is a function $\gamma : X_t \to V(G)$ such that there is no edge in the induced subgraph $G[\{\gamma(v) : v \in X_t\}]$. This partial assignment is used to memorize the closest opened centers to vertices in X_t, i.e. $\gamma(v)$ is supposed to represent the node in the optimal solution S^* that is closest to v. We now state a lemma similar in spirit to Lemma 10 (with proof deferred to a full version).

▶ **Lemma 12.** *Let S^* be the set of opened centers in the optimal solution and α^* be the corresponding assignment. Let $t, t' \in V(T)$ be two vertices such that t is a parent of t'. Then the following properties hold:*

- *If $\alpha^*(v) \in G^{(t')}$ for some $v \in X_t$, then $\alpha^*(u) = \alpha^*(v)$ for some $u \in X_{t'}$.*
- *For any vertex $u \in V(G^{(t')})$, if $\alpha^*(u) \notin G^{(t')}$, then $\alpha^*(u) = \alpha^*(v)$ for some $v \in X_t$.*

Algorithm: Now we show the algorithm that solves w-MWIS optimally in time $n^{O(w)}$. The algorithm is suggested by the above lemma. For each $t \in V(T)$ and feasible partial assignment $\gamma : X_t \to V(G)$, the table entry $C(t,\gamma)$ stores the minimum possible assignment cost inside

$G^{(t)}$ such that vertex $v \in X_t$ is assigned to $\gamma(v)$ for all $v \in X_t$. The optimal solution we want is in the entry $C(r, \gamma^*)$ where γ^* is a restriction of α^* on X_r.

To compute the entry $C(t, \gamma)$, consider the children $t_1, \ldots, t_\ell$ of t in the tree. For each $i \in [\ell]$, we define the set $\Gamma_{i,\gamma}$ as the set of all possible partial assignments $\gamma' : X_{t'} \to V(G)$ that satisfy the following properties:

1. For all $v \in X_t \cap X_{t_i}$, we have $\gamma'(v) = \gamma(v)$.
2. If $\gamma(v) \in G^{(t_i)}$ for some $v \in X_t$, then $\gamma'(u) = \gamma(v)$ for some $u \in X_{t_i}$.
3. For any vertex $u \in G^{(t_i)}$, if $\gamma'(u) \notin G^{(t_i)}$, then $\gamma'(u) = \gamma(v)$ for some $v \in X_t$.
4. There is no edge $\gamma(v)\gamma'(v') \in E(G)$ for any $v, v' \in X_t \cup X_{t_i}$.

The number of possible functions γ' is at most n^w (each vertex in X_{t_i} has at most n possibilities of γ'), so this is polynomial time computable if the treewidth of G is at most a constant. Now the table entry $C(t, \gamma)$ can be computed as follows. The base case of $C(t, \gamma)$ when t is a leaf can be computed easily by $C(t, \gamma) = \sum_{v \in X_t} d(v, \gamma(v))$. Otherwise, for any t, γ such that t has $t_1, \ldots, t_\ell$ as its children, we can write the recurrence:

$$C(t, \gamma) = \sum_{i=1}^{\ell} \min_{\gamma' \in \Gamma_{i,\gamma}} C(t_i, \gamma') - \sum_{v \in X_t} (n_v - 1) d(v, \gamma(v))$$

where n_v is the number of j such that $v \in X_{t_j}$. Notice that n_v does not depend on the choices of γ.

Correctness: The proof of the following lemma uses similar ideas as in the tree case but more complicated.

▶ **Lemma 13.** *For any vertices $t \in V(T)$ and feasible partial assignment γ for t, we can construct a partial solution $S^*_{(t,\gamma)}$ such that the total assignment cost inside $G^{(t)}$ is bounded by*

$$\sum_{u \in V(G^{(t)})} d(S^*_{(t,\gamma)}, u) \leq C(t, \gamma)$$

Moreover, $S^*_{(t,\gamma)} \subseteq V(G^{(t)}) \cup \{\gamma(u) : u \in X_t\}$, $\gamma(X_t) \subseteq S^*_{(t,\gamma)}$ and $S^*_{(t,\gamma)}$ is an independent set in G.

Acknowledgement. We thank the FSTTCS reviewers for many useful comments, espcially for suggesting a simplified connection between our problem and the standard k-median problem.

References

1 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004.

2 Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, January 1994.

3 Sugato Basu, Ian Davidson, and Kiri Lou Wagstaff. *Constrained clustering: Advances in algorithms, theory, and applications.* Chapman & Hall/CRC, 2009.

4 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

5 M.R. Cerioli, L. Faria, T.O. Ferreira, and F. Protti. On minimum clique partition and maximum independent set on unit disk graphs and penny graphs: complexity and approximation. *Electronic Notes in Discrete Mathematics*, 18(0):73 – 79, 2004.

6 Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In Claire Mathieu, editor, *SODA*, pages 892–901. SIAM, 2009.

7 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.

8 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *J. Comput. Syst. Sci.*, 65(1):129–149, 2002.

9 Moses Charikar and Shi Li. A dependent lp-rounding approach for the k-median problem. In *Automata, Languages, and Programming*, pages 194–205. Springer, 2012.

10 Irit Dinur and Shmuel Safra. The importance of being biased. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 33–42, New York, NY, USA, 2002. ACM.

11 Alina Ene, Sariel Har-Peled, and Benjamin Raichel. Geometric packing under non-uniform constraints. In Tamal K. Dey and Sue Whitesides, editors, *Symposium on Computational Geometry*, pages 11–20. ACM, 2012.

12 Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996.

13 MohammadTaghi Hajiaghayi, Rohit Khandekar, and Guy Kortsarz. Local search algorithms for the red-blue median problem. *Algorithmica*, 63(4):795–814, 2012.

14 Eran Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM J. Comput.*, 31(5):1608–1623, 2002.

15 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, July 2001.

16 George Karakostas. A better approximation ratio for the vertex cover problem. *ACM Transactions on Algorithms*, 5(4), 2009.

17 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

18 Solmaz Kolahi, Laks V. S. Lakshmanan, Jonathan Leung, Divesh Srivastava, and Suresh Venkatasubramanian. Data cleaning 2.0: Scalable generation of association-preserving repairs. Manuscript., 2013.

19 Ravishankar Krishnaswamy, Amit Kumar, Viswanath Nagarajan, Yogish Sabharwal, and Barna Saha. The matroid median problem. In Dana Randall, editor, *SODA*, pages 1117–1130. SIAM, 2011.

20 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *STOC*, pages 901–910. ACM, 2013.

On Infinite Words Determined by Stack Automata

Tim Smith

Northeastern University
Boston, MA, USA
smithtim@ccs.neu.edu

Abstract

We characterize the infinite words determined by one-way stack automata. An infinite language L determines an infinite word α if every string in L is a prefix of α. If L is regular or context-free, it is known that α must be ultimately periodic. We extend this result to the class of languages recognized by one-way nondeterministic checking stack automata (1-NCSA). We then consider stronger classes of stack automata and show that they determine a class of infinite words which we call multilinear. We show that every multilinear word can be written in a form which is amenable to parsing. Finally, we consider the class of one-way multihead deterministic finite automata (1:multi-DFA). We show that every multilinear word can be determined by some 1:multi-DFA, but that there exist infinite words determined by 1:multi-DFA which are not multilinear.

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

Keywords and phrases stack automaton, infinite word, pumping lemma, prefix language, multi-head finite automaton

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.413

1 Introduction

In this paper we study the complexity of infinite words in terms of what automata can determine them, focusing on the infinite words determined by one-way stack automata. Stack automata are a generalization of pushdown automata whose stack head, in addition to pushing and popping when at the top of the stack, can move up and down the stack in read-only mode. Stack automata were first considered by Ginsburg, Greibach, and Harrison [8, 7]; see [12] and [17] for more references and results. These automata can be restricted and generalized in a number of ways, yielding various language classes.

To associate these and other automata with infinite words, we follow Book [3] in using the concept of prefix languages. A prefix language is a language L such that for all $x, y \in L$, x is a prefix of y or y is a prefix of x. Every infinite prefix language determines an infinite word. Where C is a class of languages, we denote by $\omega(C)$ the class of infinite words determined by the prefix languages in C. Then for any class of automata, we can investigate the infinite words determined by the languages recognized by those automata. We give several results aimed at building up a classification of infinite words with respect to which classes of languges and automata can determine them.

We begin with the ultimately periodic words, those of the form $xyyy\cdots$, where x and y are strings and y is not empty. As observed in [3], every infinite regular prefix language determines an ultimately periodic word. Since the converse is also true, an infinite word is in $\omega(\mathrm{REG})$ iff it is ultimately periodic. It is further known that $\omega(\mathrm{CFL})$, the class of infinite words determined by context-free languages, equals $\omega(\mathrm{REG})$. This follows from a result of Book [3], who used the pumping lemma for context-free languages to show that every context-free prefix language is regular. Book showed the same for one-way deterministic

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 413–424

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

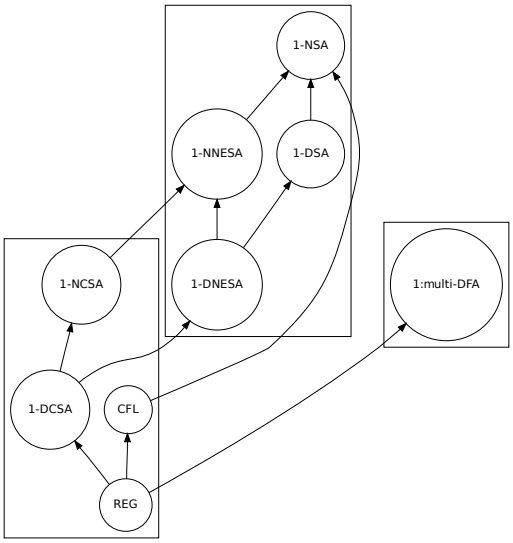


Figure 1 Relationships among the language classes considered in this paper. Arrows indicate inclusion of the lower class by the upper class. The three boxes correspond to the three types of infinite words determined by these languages.

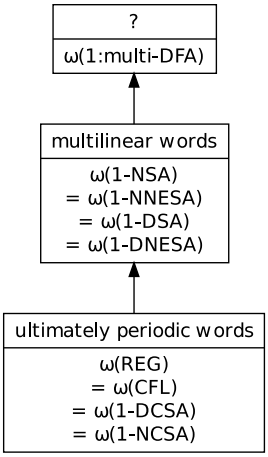


Figure 2 Three types of infinite words and the language classes which determine them. Arrows indicate proper inclusion.

checking stack automata (1-DCSA). We extend this result to the nondeterministic case (1-NCSA) using a weak pumping lemma for this class. That is, we show that every infinite word determined by a 1-NCSA is ultimately periodic.

Next, we consider a type of infinite word we call multilinear. A multilinear word consists of an initial segment q, followed by segments $r_1, \ldots, r_m$ which repeat in a way governed by linear polynomials. We show that these infinite words are determined by several classes of one-way stack automata. The most general of these is the class of one-way nondeterministic stack automata (1-NSA); various restrictions yield 1-DSA (deterministic stack automata), 1-NNESA (nondeterministic nonerasing stack automata), and 1-DNESA (deterministic nonerasing stack automata). We find with the help of a pumping lemma due to Ogden [15] that each of these classes determines exactly the multilinear infinite words. That is, $\omega(\text{1-NSA}) = \omega(\text{1-DSA}) = \omega(\text{1-NNESA}) = \omega(\text{1-DNESA})$.

Finally, we consider the class of one-way multihead deterministic finite automata (1:multi-DFA). We show that every multilinear word can be expressed in a form which is amenable to recognition by these automata. Then we show, using this form, that every such word can be determined by a 1:multi-DFA. We then give an example of an infinite word in $\omega(\text{1:multi-DFA})$ which is not multilinear. The problem of further characterizing the class of infinite words determined by 1:multi-DFA remains open.

1.1 Related work

The model used in this paper, in which infinite words are determined by languages of their prefixes, builds on Book's 1977 paper [3]. Book formulated the "prefix property" in order to allow languages to "approximate" infinite sequences, and showed that for certain classes of languages, if a language in the class has the prefix property, then it is regular. A follow-up

by Latteux [14] gives a necessary and sufficient condition for a prefix language to be regular. Languages whose complement is a prefix language, called "coprefix languages", have also been studied; see Berstel [2] for a survey of results on infinite words whose coprefix language is context-free. In Smith [16], prefix languages are used to categorize the infinite words determined by L systems, a type of parallel rewriting system.

Another approach is to consider sequence generators, devices which run indefinitely and output an infinite word piece by piece. This was the model of the seminal paper of Hartmanis and Stearns [10], as well as a 1970 follow-up by Fischer, Meyer, and Rosenberg [6], and several later papers beginning with Hromkovič, Karhumäki, and Lepistö [13], who investigated the computational complexity of infinite words generated by several kinds of iterated device. The question of what mechanisms and iterative devices suffice to generate particular infinite words has also been studied [5].

In another model, an automaton is associated with an infinite word α if when given a number n as input, it outputs the nth symbol of α. In 1972 Alan Cobham used this approach to associate finite automata with uniform tag sequences [4], leading to a literature on these "automatic sequences" [1].

1.2 Outline of paper

The paper is organized as follows. Section 2 gives preliminary definitions concerning prefix languages and automata. Section 3 gives results on ultimately periodic words and the languages and automata which determine them. Section 4 introduces multilinear infinite words and relates them to stack automata. Section 5 relates multilinear words to multihead finite automata. Section 6 gives our conclusions.

2 Preliminaries

An **alphabet** A is a finite set of symbols. A **word** is a concatenation of symbols from A. We denote the set of finite words by A^* and the set of infinite words by A^ω. A **string** x is an element of A^*. The length of x is denoted by $|x|$. We denote the empty string by λ. A **language** is a subset of A^*. A (symbolic) **sequence** S is an element of $A^* \cup A^\omega$. A **prefix** of S is a string x such that $S = xS'$ for some sequence S'. A **subword** (or factor) of S is a string x such that $S = wxS'$ for some string w and sequence S'. For $i \geq 1$, $S[i]$ denotes the ith symbol of S. For a string $x \neq \lambda$, x^ω denotes the infinite word $xxx\cdots$. Such a word is called **purely periodic**. An infinite word of the form xy^ω, where x and y are strings and $y \neq \lambda$, is called **ultimately periodic**.

2.1 Prefix languages

A **prefix language** is a language L such that for all $x, y \in L$, x is a prefix of y or y is a prefix of x. A language L **determines** an infinite word α iff L is infinite and every $x \in L$ is a prefix of α. For example, the infinite prefix language $\{\lambda, \texttt{ab}, \texttt{abab}, \texttt{ababab}, \ldots\}$ determines the infinite word $(\texttt{ab})^\omega$. The following propositions are basic consequences of the definitions.

▶ Remark. A language determines at most one infinite word.

▶ Remark. A language L determines an infinite word iff L is an infinite prefix language.

Notice that while a language determines at most one infinite word, an infinite word α may be determined by more than one language. Let $\text{Prefix}(\alpha) = \{x \mid x \text{ is a prefix of } \alpha\}$. We call $\text{Prefix}(\alpha)$ the **full** prefix language of α.

For a language class C, let $\omega(C) = \{\alpha \mid \alpha \text{ is an infinite word determined by some } L \in C\}$.

2.2 Automata

A **stack automaton** is a pushdown automaton with the extra ability to traverse its stack in read-only mode. In addition to moving its input head on the input tape, a stack automaton can move its stack head up and down to read symbols on the stack. Only when its stack head is at the top of the stack can it push or pop. A stack automaton is **nonerasing** if it never pops a symbol. A stack automaton is **checking** if it is nonerasing and if once it moves its stack head down from the top of the stack, it never again pushes a symbol. A stack automaton may be **deterministic** or **nondeterministic** and its input head may be **one-way** or **two-way**. See [12] and [17] for formal definitions and results.

Language Class	*Stack Automata*
1-NSA	one-way nondeterministic stack automata
1-DSA	one-way deterministic stack automata
1-NNESA	one-way nondeterministic nonerasing stack automata
1-DNESA	one-way deterministic nonerasing stack automata
1-NCSA	one-way nondeterministic checking stack automata
1-DCSA	one-way deterministic checking stack automata

From the definitions, we have

- 1-DCSA $\subseteq$ 1-NCSA, 1-DNESA $\subseteq$ 1-NNESA, 1-DSA $\subseteq$ 1-NSA,
- 1-DCSA $\subseteq$ 1-DNESA $\subseteq$ 1-DSA, and
- 1-NCSA $\subseteq$ 1-NNESA $\subseteq$ 1-NSA.

A **multihead finite automaton** is a finite automaton with one or more input heads. Here we are concerned only with 1:multi-DFA, the class of one-way multihead deterministic finite automata. This class is the union over all $i \geq 1$ of 1:i-DFA, the class of one-way i-head deterministic finite automata. Each such automaton begins with its input heads on the first symbol of the input. At each step, the automaton reads the input symbols under all of its heads and then changes state and moves any subset of its heads to the right. See [17] and [11] for formal definitions and results.

3 Ultimately periodic words

Recall that an infinite word is ultimately periodic if it has the form xy^ω, where x and y are strings and $y \neq \lambda$. Clearly every ultimately periodic word is determined by some regular language.

▶ **Theorem 1.** *Every ultimately periodic word α is in $\omega(REG)$.*

Proof. The infinite word α has the form xy^ω for some strings x and y where $y \neq \lambda$. Then the regular language xy^* determines α. So α is in ω(REG). ◀

As observed by Book [3], the converse, that every infinite regular prefix language determines an ultimately periodic word, is also true. In fact, as Book showed, the same holds for context-free languages.

▶ **Theorem 2** (Book). *Suppose α is in $\omega(CFL)$. Then α is ultimately periodic.*

Proof. Since α is in ω(CFL), some $L \in$ CFL determines α. Take any such L. Then L is infinite and every $s \in L$ is a prefix of α. By the pumping lemma for context-free languages, there is a string $uvxyz$ such that $|vy| \geq 1$ and for all $n \geq 0$, $uv^n xy^n z$ is in L. Suppose $|v| \geq 1$. Then because every string in L is a prefix of α, and every prefix of such a string is also a prefix of α, uv is a prefix of α, as are uvv, $uvvv$, and so on. Consequently $\alpha = uv^\omega$, so α is ultimately periodic. So say $|v| = 0$. Then $|y| \geq 1$ and $\alpha = uxy^\omega$, again making α ultimately periodic. ◀

Checking stack automata

The class of languages recognized by one-way checking stack automata is incomparable with the context-free languages. Nonetheless, this class too determines just the infinite words determined by regular languages. Book [3] proved the deterministic case (1-DCSA); our result holds in the nondeterministic case (1-NCSA) also. (Book showed that 1-NCSA contains non-regular prefix languages, but this does not imply that ω(1-NCSA) $\neq \omega$(REG).) We employ a weak pumping lemma for 1-NCSA which we obtain using results from Greibach [9].

For $k \geq 1$, a language L is **k-iterative** if there is an $n \geq 0$ such that for all $s \in L$ where $|s| \geq n$, there are strings x_1, y_1, x_2, y_2, …, x_k, y_k, x_{k+1} such that

- $s = x_1 y_1 x_2 y_2 \cdots x_k y_k x_{k+1}$,
- $|y_1 \cdots y_k| \geq 1$, and
- for all $i \geq 0$, $x_1 y_1^i x_2 y_2^i \cdots x_k y_k^i x_{k+1}$ is in L.

L is **weakly k-iterative** if it is either finite or contains an infinite k-iterative subset. Notice that every regular language is 1-iterative and every context-free language is 2-iterative, due to the pumping lemmas for these classes.

In proving the following lemma we use results from Greibach [9] formulated for a type of device called a one-way preset Turing machine. Greibach observes that a certain subclass of these devices, called nonwriting regular-based, can be regarded as checking stack automata. In particular, a one-way checking stack automaton can be simulated by a one-way nonwriting regular-based preset Turing machine, and vice versa, without changing the number of stack visits, crosses, or reversals by more than 1. Hence results for this subclass translate into facts about checking stack automata.

▶ **Lemma 3.** *Suppose L is in 1-NCSA. Then L is weakly k-iterative for some $k \geq 1$.*

Proof. A checking stack automaton M is **finite visit** if there is a $k \geq 1$ such that for every string s accepted by M, there is an accepting computation of M for s in which no stack position is visited more than k times. Suppose L is accepted by a finite visit 1-NCSA M. Then there is a $k \geq 1$ such that L is in the class k-VISIT(REGL) of [9]. Then by Lemma 4.22 of [9], L is weakly k-iterative. So say there is no such M. Then L is not in the class FINITEVISIT(REGL) of [9]. Then by Lemma 4.25 of [9], L is weakly 1-iterative. ◀

▶ **Theorem 4.** *Suppose α is in ω(1-NCSA). Then α is ultimately periodic.*

Proof. Since α is in ω(1-NCSA), some $L \in$ 1-NCSA determines α. Take any such L. Then L is infinite and every $s \in L$ is a prefix of α. By Lemma 3, L is weakly k-iterative for some $k \geq 1$. Then there is a string $x_1 y_1 x_2 y_2 \cdots x_k y_k x_{k+1}$ such that $|y_1 \cdots y_k| \geq 1$ and for all $i \geq 0$, $x_1 y_1^i x_2 y_2^i \cdots x_k y_k^i x_{k+1}$ is in L. Let j be the lowest number such that y_j is non-empty. Then $x_1 x_2 \cdots x_j y_j$ is a prefix of α, as are $x_1 x_2 \cdots x_j y_j y_j$, $x_1 x_2 \cdots x_j y_j y_j y_j$, and so on. Therefore $\alpha = x_1 x_2 \cdots x_j y_j^\omega$, so α is ultimately periodic. ◀

Summarizing, we have the following.

▶ **Theorem 5.** *$\omega(REG) = \omega(CFL) = \omega(1\text{-}DCSA) = \omega(1\text{-}NCSA)$, and α is in this class of infinite words iff α is ultimately periodic.*

Proof. Immediate from Theorems 1, 2, and 4 and the inclusions REG $\subseteq$ CFL and REG $\subseteq$ 1-DCSA $\subseteq$ 1-NCSA. ◀

4 Multilinear words

We now introduce the multilinear infinite words, a class which properly includes the ultimately periodic words. To our knowledge this type of infinite word has not previously been discussed. An infinite word is **multilinear** if it has the form

$$q \prod_{n\geq 0} r_1^{a_1 n+b_1} r_2^{a_2 n+b_2} \cdots r_m^{a_m n+b_m},$$

where $\prod$ denotes concatenation, q is a string, each r_i is a non-empty string, and m and each a_i and b_i are nonnegative integers such that $a_i + b_i > 0$. Examples:

- $\mathtt{ab} \prod_{n\geq 0} \mathtt{cd} = \mathtt{abcdcdcd}\cdots$
- $\prod_{n\geq 0} \mathtt{a}^{n+1}\mathtt{b} = \mathtt{abaabaaab}\cdots$
- $\prod_{n\geq 0} \mathtt{10}^{2n} = \mathtt{11001000010000001}\cdots$ (characteristic sequence of the perfect squares)

With the next few theorems we relate multilinear words to one-way stack automata.

▶ **Theorem 6.** *Suppose α is in $\omega(1\text{-}NSA)$. Then α is multilinear.*

Proof. Since α is in ω(1-NSA), some $L \in$ 1-NSA determines α. Take any such L. Then L is infinite and every $s \in L$ is a prefix of α. By Ogden's pumping lemma for one-way stack automata [15], there are

- strings μ and ν,
- strings ρ_i, σ_i, and τ_i for each $i \geq 0$,
- strings α_j, β_j, ϕ_j, χ_j and ψ_j for bounds on j implicit below, and
- positive integers m and p

such that, among other conditions,

(i) for each $j \geq 0$, $\mu\rho_0\rho_1\cdots\rho_j\sigma_j\tau_j\tau_{j-1}\cdots\tau_0\nu$ is in L,
(ii) for each $i \geq 1$, $\rho_i = \alpha_0\beta_1^{i-1}\phi_1\beta_2^{i-1}\alpha_1\beta_3^{i-1}\phi_2\beta_4^{i-1}\alpha_2\cdots\phi_{m-1}\beta_{2m-2}^{i-1}\alpha_{m-1}$,
(iii) $|\rho_0| = 0$ iff for all $i > 0$, $|\rho_i| = 0$,
(iv) for each $i \geq 0$, $\sigma_i = \chi_0\psi_1^i\chi_1\psi_2^i\chi_2\cdots\psi_{p-1}^i\chi_{p-1}$, and
(v) there is a j such that $|\psi_j| > 0$.

Suppose $|\rho_0| = 0$. Then by (iii), every ρ_i is empty, so by (i), we have that for each $j \geq 0$, $\mu\sigma_j\tau_j\tau_{j-1}\cdots\tau_0\nu$ is in L. Then for each $j \geq 0$, $\mu\sigma_j$ is a prefix of α. Then by (iv), for each $i \geq 0$, $\mu\chi_0\psi_1^i\chi_1\psi_2^i\chi_2\cdots\psi_{p-1}^i\chi_{p-1}$ is a prefix of α. Let j be the lowest number such that $|\psi_j| > 0$; by (v), there is such a j. Then for each $i \geq 0$, $\mu\chi_0\chi_1\cdots\chi_{j-1}\psi_j^i\chi_j\cdots\psi_{p-1}^i\chi_{p-1}$ is a prefix of α. Then $\mu\chi_0\chi_1\cdots\chi_{j-1}\psi_j$ is a prefix of α, as are $\mu\chi_0\chi_1\cdots\chi_{j-1}\psi_j\psi_j$, $\mu\chi_0\chi_1\cdots\chi_{j-1}\psi_j\psi_j\psi_j$, and so on. Therefore $\alpha = \mu\chi_0\chi_1\cdots\chi_{j-1}\psi_j^\omega$, which is ultimately periodic and hence multilinear.

So say $|\rho_0| > 0$. By (iii), some ρ_i other than ρ_0 is non-empty. By (ii), if ρ_1 is non-empty, then so are all subsequent ρ_is, and if ρ_1 is empty, then some β_j must be non-empty, and all subsequent ρ_is are again non-empty. So for all $i \geq 2$, $|\rho_i| > 0$. By (i), for each $j \geq 0$, $\mu\rho_0\rho_1 \cdots \rho_j$ is a prefix of α. Therefore $\alpha = \mu \prod_{n\geq 0} \rho_n = \mu\rho_0 \prod_{n\geq 0} \rho_{n+1} = \mu\rho_0 \prod_{n\geq 0} \alpha_0\beta_1^n\phi_1\beta_2^n\alpha_1\beta_3^n\phi_2\beta_4^n\alpha_2 \cdots \phi_{m-1}\beta_{2m-2}^n\alpha_{m-1}$, which is multilinear. ◀

▶ **Theorem 7.** *Suppose α is multilinear. Then α is in ω(1-DNESA).*

Proof. The infinite word α has the form $q \prod_{n\geq 0} r_1^{a_1 n+b_1} r_2^{a_2 n+b_2} \cdots r_m^{a_m n+b_m}$. Let A be a one-way deterministic nonerasing stack automaton, operating as follows. First, A checks that the input begins with q. In what follows, A will push counter symbols onto its stack; so far, the stack is empty. Next, for each i between 1 and m, A first checks the input for b_i occurrences of r_i. It then reads its stack, for each counter symbol checking the input for a_i occurrences of r_i. After checking r_m, A pushes a counter symbol onto its stack and proceeds as before. If any input symbol causes a check to fail, A rejects; otherwise, when A reaches end of input, it accepts. Now A recognizes Prefix(α), the full prefix language of α. Since Prefix(α) determines α, α is in ω(1-DNESA). ◀

▶ **Theorem 8.** *ω(1-NSA) = ω(1-DSA) = ω(1-NNESA) = ω(1-DNESA), and α is in this class of infinite words iff α is multilinear.*

Proof. Immediate from Theorems 6 and 7 and the inclusions 1-NSA $\supseteq$ 1-DSA $\supseteq$ 1-DNESA and 1-NSA $\supseteq$ 1-NNESA $\supseteq$ 1-DNESA. ◀

5 Multihead finite automata

In this section we relate multilinear infinite words to multihead finite automata. First, we show that every multilinear infinite word can be expressed in a certain form which is amenable to recognition by these automata. Then we show, using this form, that every multilinear infinite word can be determined by a one-way multihead deterministic finite automaton (1:multi-DFA).

Following the definition in the previous section, a multilinear infinite word can be viewed as a pair $[q, t]$, where t is a **term list** of m triples $[r_i, a_i, b_i]$. We say that two such pairs (or two term lists) are **equivalent** if they express the same multilinear word. Notice that if two term lists t_1, t_2 are equivalent, then the first term of t_1 begins with the same symbol as the first term of t_2. We say that t_1 and t_2 are **strongly equivalent** if they are equivalent and if the last term of t_1 begins with the same symbol as the last term of t_2. Any term i with $a_i > 0$ we call a **growth term**. Any pair $[q, t]$ can be **rotated**, yielding the equivalent pair $[q\ r_1^{b_1}, [t[2], \ldots, t[m], [r_1, a_1, b_1 + a_1]]]$. In the proofs below, we allow a growth term to temporarily have a negative b_i if it can later be "rotated away" (made nonnegative by repeated rotations). To this end, when $b_1 < 0$, we define rotation of the pair $[q\ r_1^{-b_1}, t]$ to yield the equivalent pair $[q, [t[2], \ldots, t[m], [r_1, a_1, b_1 + a_1]]]$. For use below, we give several conditions which a pair or term list may or may not satisfy.

- Condition 1. For every i from 1 to m, $b_i \geq 1$.
- Condition 2a. For every i from 1 to $m-1$, $r_i[1] \neq r_{i+1}[1]$.
- Condition 2b. If $m \geq 2$, $r_1[1] \neq r_m[1]$.

For example, take the multilinear infinite word

$$\alpha = \prod_{n\geq 0} (\mathtt{abc})^n \mathtt{aba}^n = \mathtt{ababcabaabcabcabaa}\cdots$$

The multilinear pair [λ, [[abc, 1, 0], [ab, 0, 1], [a, 1, 0]]] expresses α but does not meet any of the three conditions. However, the equivalent pair [ab, [[a, 1, 1], [b, 0, 1], [cab, 1, 1]]] meets all three conditions, giving

$$\alpha = \mathtt{ab} \prod_{n\geq 0} \mathtt{a}^{n+1}\mathtt{b}(\mathtt{cab})^{n+1}$$

We will show that every multilinear infinite word can be expressed as a pair satisfying conditions 1, 2a, and 2b. The proof outline is first to show that every multilinear term list has an equivalent term list satisfying condition 2a (Lemma 12), and next to show that every multilinear pair satisfying condition 2a is equivalent to a pair satisfying conditions 1, 2a, and 2b (Theorem 15). The notion of strong equivalence is used in the proof of Theorem 15, where we take a term list satisfying condition 2a, rotate it so that it satisfies 2b but now has a portion which does not satisfy 2a, and then replace that portion with a strongly equivalent one satisfying 2a, so that the whole then satisfies 2a and 2b.

▶ **Lemma 9.** *Given a multilinear term list* $[[r_1, a_1, b_1], [r_2, a_2, b_2]]$ *such that* $a_1 = 0$ *and* $a_2 > 0$*, there is a strongly equivalent term list meeting condition 2a.*

Proof. Let $s = r_1^{b_1}$. Suppose there is an i such that $s[i] \neq r_2[1]$. Take the first such i. If $i = 1$, we can just return $[[s, 0, 1], [r_2, a_2, b_2]]$. Otherwise, we split s at i and return $[[s[1]\cdots s[i-1]], 0, 1], [s[i]\cdots s[|s|]], 0, 1], [r_2, a_2, b_2]]$. So say there is no such i. Suppose there is an i such that $r_2[i] \neq r_2[1]$. Take the first such i. Since $[[s\ r_2, 0, 1], [r_2, a_2, b_2 - 1]]$ is equivalent to the original term list, we can return $[[s\ r_2[1]\cdots r_2[i-1], 0, 1], [r_2[i]\cdots r_2[|r_2|], 0, 1], [r_2, a_2, b_2 - 1]]$. So say there is no such i. Then every symbol in s and r_2 equals $r_2[1]$. So return $[[r_2[1], |r_2| \cdot a_2, |s| + |r_2| \cdot b_2]]$. ◀

▶ **Lemma 10.** *Given a multilinear term list* $[[r_1, a_1, b_1], [r_2, a_2, b_2]]$ *such that* $a_1 > 0$ *and* $a_2 > 0$*, there is a strongly equivalent term list meeting condition 2a.*

Proof. Suppose $r_1^\omega = r_2^\omega$. Then $r_1^{|r_2|} = r_2^{|r_1|}$, so by Theorem 1.5.3 of [1], there are $k, l > 0$ such that $r_1 = z^k$ and $r_2 = z^l$ for some string z. Then $[[z, \frac{|r_1|}{|z|} \cdot a_1 + \frac{|r_2|}{|z|} \cdot a_2, \frac{|r_1|}{|z|} \cdot b_1 + \frac{|r_2|}{|z|} \cdot b_2]]$ meets the condition. So say $r_1^\omega \neq r_2^\omega$. Let p be the longest common prefix of r_1^ω and r_2^ω. If $p = \lambda$, then $[[r_1, a_1, b_1], [r_2, a_2, b_2]]$ already meets the condition. Otherwise, let $c = (|p| \bmod |r_1|) + 1$, let $d = \frac{|p|}{|r_1|}$ (rounded down), let e $= (|p| \bmod |r_2|) + 1$, and let $f = \frac{|p|}{|r_2|}$ (rounded down). Then $r_1[c] \neq r_2[e]$. Suppose $c = 1$. Let u be the term list $[[r1, a1, b1 + d], [r_2[e]\cdots r_2[|r_2|], 0, 1], [r_2, a_2, b_2 - f - 1]]$. Then u is equivalent to the original term list. Its first term already starts with a different symbol than its second term, while by Lemma 9, its last two terms can be replaced with an equivalent term list meeting condition 2a. So say $c \neq 0$. Let u be the term list $[[r_1[1]\cdots r_1[c-1]], 0, 1], [r_1[c]\cdots r_1[|r_1|]r_1[1]\cdots r_1[c-1]], a1, b1 + d], [r_2[e]\cdots r_2[|r_2|], 0, 1], [r_2, a_2, b_2 - f - 1]]$. Then u is equivalent to the original term list. Its second term already starts with a different symbol than its third term, while by Lemma 9, its first two terms and last two terms can be replaced with equivalent term lists meeting condition 2a. ◀

▶ **Lemma 11.** *Given a multilinear term list* $[[r_1, a_1, b_1], [r_2, a_2, b_2]]$ *such that* $a_2 = 0$*, there is an equivalent term list meeting condition 2a.*

Proof. If $a_1 = 0$, then we can just join the terms, returning $[[r_1^{b_1} r_2^{b_2}, 0, 1]]$. So say $a_1 > 0$. Let $s = r_2^{b_2}$. Then $s = r_1^i\ x$ for some $i \geq 0$ and string x such that r_1 is not a prefix of x. If $x = \lambda$, return $[[r_1, a_1, b_1 + i]]$. Let p be the longest common prefix of r_1 and x. If $p = \lambda$, return $[[r_1, a_1, b_1 + i], [x, 0, 1]]$. Otherwise, $r_1 = py$ and $x = pz$ for some strings y, z. Let u be the term list $[[p, 0, 1], [yp, a_1, b_1]]$. By Lemma 9, there is an equivalent term list u' meeting condition 2a whose last term begins with the same symbol as y. If $z = \lambda$, then u' is equivalent to the original term list, and we are finished. Otherwise, append to u' the term $[z, 0, 1]$. Now u' is equivalent to the original term list. Further, since z begins with a different symbol than does y, u' meets condition 2a, and we are finished. ◀

▶ **Lemma 12.** *Given a multilinear term list t, there is an equivalent term list meeting condition 2a.*

Proof. We proceed by induction on $|t|$. If $|t| \leq 1$, then condition 2a is already met, so just return t. If $|t| = 2$, then by Lemmas 9, 10, and 11, the result holds. So say $|t| > 2$. Suppose for induction that the result holds for any term list of size less than $|t|$. Then by the induction hypothesis, there is a term list u equivalent to $[t[1], \ldots, t[m-1]]$ and meeting condition 2a. Let $x = [u[1], \ldots, u[|u|-1]]$ and let $y = u[|u|]$. Then again by the induction hypothesis, there is a term list v equivalent to $[y, t[m]]$ and meeting condition 2a. So $x + v$ is equivalent to t. Now, x and v each meet condition 2a. Further, since v is equivalent to $[y, t[m]]$, the first term of v begins with the same symbol as y. Since u met condition 2a, the last term of x begins with a different symbol than does y. Hence $x + v$ meets condition 2a. ◀

▶ **Lemma 13.** *Given a multilinear term list t whose last term is a growth term, there is a strongly equivalent term list meeting condition 2a.*

Proof. If $|t| = 1$, then the condition is already met, so just return t. Otherwise, by Lemma 12, there is a term list u equivalent to $[t[1], \ldots, t[m-1]]$ and meeting condition 2a. Let $x = [u[1], \ldots, u[|u|-1]]$ and let $y = u[|u|]$. By Lemmas 9 and 10, there is a term list v equivalent to $[y, t[m]]$, meeting condition 2a, and whose last term starts with the same symbol as does $t[m]$. So $x + v$ is strongly equivalent to t. Now, x and v each meet condition 2a. Further, since v is equivalent to $[y, t[m]]$, the first term of v begins with the same symbol as y. Since u met condition 2a, the last term of x begins with a different symbol than does y. Hence $x + v$ meets condition 2a. ◀

▶ **Lemma 14.** *Given a multilinear pair $[q, t]$ such that t contains exactly one growth term, there is an equivalent pair meeting conditions 2a and 2b.*

Proof. If $|t| = 1$, then $[q, t]$ already meets the conditions. So say $|t| > 1$. Rotate $[q, t]$ until $t[1]$ is the growth term. Let $s = r_2^{b_2} \cdots r_m^{b_m}$. Suppose $r_1^\omega = s\ r_1^\omega$. Then $r_1^\omega = s^\omega$, so by Theorem 1.5.3 of [1], there are $k, l > 0$ such that $r_1 = z^k$ and $s = z^l$ for some string z. Then $[q, t]$ is ultimately periodic, so $[q, [z, 0, 1]]$ is an equivalent pair which meets the conditions. So say $r_1^\omega \neq s\ r_1^\omega$. Let p be the longest common prefix of r_1^ω and $s\ r_1^\omega$. Then p is a prefix of $s\ r_1$. If $p = \lambda$, then $[q, t]$ already meets the conditions. Otherwise, let u be the term list $[[r_1, 0, 1], [r_1, a_1, b_1 - 1], [s, 0, 1]]$. The pair $[q, u]$ is equivalent to $[q, t]$. Now rotate $[q, u]$ and combine terms to give $[[r_1, a_1, b_1 - 1], [s\ r_1, 0, 1]]$. The string $s\ r_1$ has the form px for some $x \neq \lambda$ and p has the form $r_1^i\ y$ for some $i \geq 0$ and string y such that $r_1 = yz$ for some $z \neq \lambda$. Notice that $x[1] \neq z[1]$. If $y = \lambda$ then we have $[[r_1, a_1, b_1 - 1 + i], [x, 0, 1]]$ and we are finished. Otherwise, we have $[[r_1, a_1, b_1 - 1 + i], [y, 0, 1], [x, 0, 1]]$, which is equivalent to $[[y, 0, 1], [zy, a_1, b_1 - 1 + i], [x, 0, 1]]$. Rotating twice, we get $[[x, 0, 1], [y, 0, 1], [zy, a_1, b_1 - 1 + i + a_1]]$. By Lemma 13, there is an equivalent term list whose last term starts with the same symbol as does z. Then q paired with this term list meets the conditions. ◀

▶ **Theorem 15.** *Let α be a multilinear infinite word. Then α has the form*

$$q \prod_{n \geq 0} r_1^{a_1 n + b_1} r_2^{a_2 n + b_2} \cdots r_m^{a_m n + b_m}$$

for some $m \geq 0$, string q, non-empty strings r_i, and nonnegative integers a_i, b_i where $a_i + b_i > 0$, such that

- *for every i from 1 to m, $b_i \geq 1$,*
- *for every i from 1 to $m-1$, $r_i[1] \neq r_{i+1}[1]$, and*
- *if $m \geq 2$, $r_1[1] \neq r_m[1]$.*

Proof. α can be viewed as a pair $[q, t]$, where t is a term list of m triples $[r_i, a_i, b_i]$. We will give an equivalent pair meeting conditions 1, 2a, and 2b. First, by Lemma 12, there is a term list u which is equivalent to t and which meets condition 2a. We will give a pair $[q', u']$ which is equivalent to $[q, u]$ and meets conditions 2a and 2b. If u is empty, just set $q' = q$ and $u' = u$. Otherwise, suppose u contains no growth terms. Then u can be contracted into a single term, so set $q' = q$ and $u' = [r_1^{b_1} \cdots r_m^{b_m}, 0, 1]$. So suppose u contains exactly one growth term. Then by Lemma 14, there is a multilinear pair $[q', u']$ which is equivalent to $[q, u]$ and meets conditions 2a and 2b. Finally, suppose u contains more than one growth term. Let $u[i]$ be the first growth term in u. Let $[q', u']$ be the result of rotating $[q, u]$ by i terms. Now u' ends with the growth term $u[i]$. The first symbol of $u'[1]$ ($u[i+1]$) differs from the first symbol of $u'[m]$ ($u[i]$), so u' meets condition 2b. Further, u' meets condition 2a, except that the first symbol of $u'[m-i]$ may equal the first symbol of $u'[m-i+1]$. Using Lemma 13, replace the terms from $m-i$ to m with a strongly equivalent term list meeting condition 2a. Now u' meets conditions 2a and 2b. Finally, now that we have an equivalent pair which meets conditions 2a and 2b, we can rotate it until condition 1 is met. For each term i, if $a_i = 0$, then already $b_i > 0$, whereas if $a_i > 0$, then each rotation increases b_i to $b_i + a_i$. So repeatedly rotating $[q', u']$ will eventually cause it to meet condition 1. ◀

▶ **Theorem 16.** *Every multilinear infinite word is in ω(1:multi-DFA).*

Proof. Take any multilinear infinite word α. By Theorem 15, α can be expressed in a form $q \prod_{n \geq 0} r_1^{a_1 n + b_1} r_2^{a_2 n + b_2} \cdots r_m^{a_m n + b_m}$ meeting the conditions of that theorem. If $m = 1$, then α is ultimately periodic, so α is in $\omega(\text{REG}) \subseteq \omega$(1:multi-DFA). So say $m \geq 2$. Let A be a one-way 2-head deterministic finite automaton, operating as follows. First, A checks that the input begins with q, moving both heads to the right after each symbol. Next, A keeps one head stationary while using the other head to check the $n = 0$ subword, verifying that each r_i occurs b_i times. Now one head is at the beginning of the $n = 0$ subword and the other head is at the beginning of the $n = 1$ subword. For each $j \geq 1$, A checks the $n = j$ subword as follows. For each i from 1 to m, A first uses its right head to checks for a_i occurrences of r_i, keeping its left head stationary. Then A moves both heads to the right, checking for occurrences of r_i under the left head until no more are found, and rejecting if the symbol under the right head ever differs from the symbol under the left head. After $a_i(j-1) + b_i$ occurrences of r_i, the left head will encounter the first symbol of r_{i+1} (or r_1 if $i = m$). Since, by the conditions of Theorem 15, this symbol is different from the first symbol of r_i, A is now at the start of term $i+1$, and can proceed to check this term, and so on until it has checked all m terms, at which point it moves on to subword $n = j+1$. If any input symbol causes a check to fail, A rejects; otherwise, when A reaches end of input, it accepts. Now A recognizes Prefix(α), the full prefix language of α. Since Prefix(α) determines α, α is in ω(1:multi-DFA). ◀

Finally, we give a simple example of an infinite word in ω(1:multi-DFA) which is not multilinear.

▶ **Theorem 17.** *Not every infinite word in ω(1:multi-DFA) is multilinear.*

Proof. Let α be the infinite word $\prod_{n\geq 0} \mathtt{a}^{2^n}\mathtt{b} = \mathtt{a}^1\mathtt{b}\mathtt{a}^2\mathtt{b}\mathtt{a}^4\mathtt{b}\mathtt{a}^8\mathtt{b}\cdots$. Clearly α is not multilinear. Let A be a one-way 2-head deterministic finite automaton, operating as follows. Initially, A keeps one head stationary while using the other head to check that the input begins with `ab`. Subsequently, A moves both heads to the right. For each `a` under the left head, A checks for two occurrences of `a` under the right head, while for each `b` under the left head, A checks for one occurrence of `b` under the right head. If any input symbol causes a check to fail, A rejects; otherwise, when A reaches end of input, it accepts. Now A recognizes Prefix(α), the full prefix language of α. Since Prefix(α) determines α, α is in ω(1:multi-DFA). ◀

6 Conclusion

In this paper we have given several results aimed at building up a classification of infinite words with respect to which classes of automata can determine them. To associate automata with infinite words, we used the concept of prefix languages. This concept can be applied not just to automata, but to arbitrary language classes, offering many opportunities for further research. For a given language class, we can ask what class of infinite words it determines. From the other direction, for a given infinite word, we can ask in what language classes it can be determined. It is hoped that work in this area will help to establish a theory of the complexity of infinite words as determined by their prefix languages. One specific task would be to further characterize the infinite words determined by one-way multihead deterministic finite automata (1:multi-DFA), beyond the result established in this paper, that the multilinear infinite words are properly contained by this class.

Acknowledgements. I want to thank my advisor, Rajmohan Rajaraman, for supporting this work, encouraging me, and offering many helpful comments and suggestions.

References

1 Jean-Paul Allouche and Jeffrey Shallit. *Automatic Sequences: Theory, Applications, Generalizations*. Cambridge University Press, New York, NY, USA, 2003.

2 J. Berstel. Properties of infinite words : Recent results. In B. Monien and R. Cori, editors, *STACS 89*, volume 349 of *Lecture Notes in Computer Science*, pages 36–46. Springer Berlin Heidelberg, 1989.

3 Ronald V. Book. On languages with a certain prefix property. *Mathematical Systems Theory*, 10:229–237, 1977.

4 Alan Cobham. Uniform tag sequences. *Theory of Computing Systems*, 6:164–192, 1972. 10.1007/BF01706087.

5 Karel Culik and Juhani Karhumäki. Iterative devices generating infinite words. *Int. J. Found. Comput. Sci.*, 5(1):69–97, 1994.

6 Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Time-restricted sequence generation. *J. Comput. Syst. Sci.*, 4:50–73, February 1970.

7 Seymour Ginsburg, Sheila A. Greibach, and Michael A. Harrison. One-way stack automata. *J. ACM*, 14(2):389–418, April 1967.

8 Seymour Ginsburg, Sheila A. Greibach, and Michael A. Harrison. Stack automata and compiling. *J. ACM*, 14(1):172–201, January 1967.

9 S.A. Greibach. One way finite visit automata. *Theoretical Computer Science*, 6(2):175 – 221, 1978.

10 J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of The American Mathematical Society*, 117:285–306, 1965.

11 Markus Holzer, Martin Kutrib, and Andreas Malcher. Complexity of multi-head finite automata: Origins and directions. *Theor. Comput. Sci.*, 412(1-2):83–96, January 2011.

12 J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley series in computer science. Addison-Wesley, 1979.

13 Juraj Hromkovič, Juhani Karhumäki, and Arto Lepistö. Comparing descriptional and computational complexity of infinite words. In *Proceedings of the Colloquium in Honor of Arto Salomaa on Results and Trends in Theoretical Computer Science*, pages 169–182, London, UK, 1994. Springer-Verlag.

14 Michel Latteux. Une note sur la propriété de prefixe. *Mathematical Systems Theory*, 11:235–238, 1978.

15 William F. Ogden. Intercalation theorems for stack languages. In *Proceedings of the first annual ACM symposium on Theory of computing*, STOC '69, pages 31–42, New York, NY, USA, 1969. ACM.

16 Tim Smith. On infinite words determined by L systems. In Juhani Karhumäki, Arto Lepistö, and Luca Zamboni, editors, *Combinatorics on Words*, volume 8079 of *Lecture Notes in Computer Science*, pages 238–249. Springer Berlin Heidelberg, 2013.

17 K. Wagner and G. Wechsung. *Computational Complexity*. Mathematics and its Applications. Springer, 1986.

The Combinatorics of Non-determinism*

Olivier Bodini[1], Antoine Genitrini[2], and Frédéric Peschanski[2]

1 **Laboratoire d'Informatique de Paris-Nord, CNRS UMR 7030 – Institut Galilée – Université Paris-Nord, Villetaneuse, France**
Olivier.Bodini@lipn.univ-paris13.fr

2 **Laboratoire d'Informatique de Paris 6, CNRS UMR 7606 and Université Pierre et Marie Curie, Paris, France**
{Antoine.Genitrini,Frederic.Peschanski}@lip6.fr

Abstract

A deep connection exists between the interleaving semantics of concurrent processes and increasingly labelled combinatorial structures. In this paper we further explore this connection by studying the rich combinatorics of partially increasing structures underlying the operator of non-deterministic choice. Following the symbolic method of analytic combinatorics, we study the size of the computation trees induced by typical non-deterministic processes, providing a precise quantitative measure of the so-called "combinatorial explosion" phenomenon. Alternatively, we can see non-deterministic choice as encoding a family of tree-like partial orders. Measuring the (rather large) size of this family on average offers a key witness to the expressiveness of the choice operator. As a practical outcome of our quantitative study, we describe an efficient algorithm for generating computation paths uniformly at random.

1998 ACM Subject Classification F.1.2 Modes of Computation

Keywords and phrases Concurrency theory, Analytic combinatorics, Non-deterministic choice, Partially increasing trees, Uniform random generation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.425

1 Introduction

The mathematical structures underlying concurrency theory are most often considered as *abstract* entities from an algebraic point of view, and are much less often scrutinized under the meticulous lens of *concrete mathematics*. From the combinatorial point of view, even very basic process operators prove quite intricate. Indeed, we show in [3] that the study of the classic merge (or interleaving) operator requires non-trivial analytic combinatorics techniques.

In this paper we study the rich combinatorics of *partially increasing structures* underlying the operator of *non-deterministic choice* [11]. Note that this is a major departure from related studies involving regular shuffle in automata theory, e.g. [12]. Although we are still far from a full-fledged process algebra, the increase of expressiveness if compared to pure merge is noticeable. When unfolded as computation trees, the resulting process behaviours prove much more difficult to deal with. We show in this paper, however, that the analytic techniques we rely on – based on the *symbolic method* of *analytic combinatorics* [9] – scale relatively well to cover this enriched model. In Section 3 we give precise – average-case – results concerning the number of computation paths induced by typical non-deterministic

* This research was partially supported by the CNRS project *ALPACA* (PEPS INS2I 2012–2013) and by the A.N.R. project *MAGNUM*, ANR 2010-BLAN-0204.

processes, hence providing a quantitative measure for the so-called "combinatorial explosion" phenomenon. As we make clear in the paper, this phenomenon is not only a worst-case situation but it is also quite perceptible in the average case.

Our quantitative study also establishes deep links between the combinatorics of increasing structures and partial orders. Indeed, we can interpret non-deterministic choice as an encoding of a family of tree-like partial orders or *tree-posets* [2]. Measuring the (indeed exponential) average size of this family on average offers a key witness to the expressiveness of the choice operator. This is discussed in Section 4.

As a practical outcome of our quantitative study, we aim at developing techniques to analyze properties of the computation trees without explicitly constructing them. As an illustration, we describe in Section 5 an efficient algorithm for the generation of non-deterministic computation paths uniformly at random directly from the syntax of process specification. This algorithms is based on a compact polynomial representation of the uniform distribution of the computation paths.

2 Processes as combinatorial structures

In this section we define the mathematical objects that represent the syntax of process specifications on the one side, and the semantics of process behaviours on the other side.

2.1 Syntax: process trees

The syntax of formal languages is most often defined as a context-free grammar yielding tree structures: (abstract) syntax trees that will be called *process trees* in what follows. For the minimalist process calculus we discuss in this paper, the (semi-formal) grammar of process trees is the following :

$$\left[\begin{array}{lcll} P & ::= & P_{\|} \mid P_{+} & \text{(process)} \\ P_{\|} & ::= & \alpha \mid \alpha.\,(P \parallel \ldots) & \text{(prefixed parallel)} \\ P_{+} & ::= & P_{\|} + P_{\|} + \ldots & \text{(non-deterministic choice)} \end{array}\right.$$

If compared to most algebraic-oriented presentations, we do not use binary constructors. The main reason is that the parallel and choice operators are associative. We could of course encode the process terms using binary syntactic trees. However, if this is quite transparent for the tree structure, the encoding would have a sizable (and somewhat gratuitous) impact on the analytic developments. For similar reasons the prefixing construct is intermixed with the merge constructor for parallel processes, which means that each branch of a choice is prefixed. Hence we consider in essence what is often called *guarded choice* in the literature. To avoid an ambiguity in the model, we also require a choice to have at least two sub-processes.

The process specification that we will use as a running example in the paper is the following one: $a.\,\big(\ [b.(c \parallel d) + e] \parallel f.\,([g + h.(i \parallel j) + k] \parallel l)\ \big)$

Note that we use distinct labels for actions because our purely structural study does not reflect on the identity nor the nature of the atomic actions performed by the processes. So we do not distinguish among e.g. internal vs. external choice. In fact, we study the *effect* of non-determinism (i.e. the branching in computation trees) rather than its *cause*.

To formalize the process specifications as combinatorial objects, we rely on the *symbolic method* of analytic combinatorics [9]. Our starting point is the following combinatorial

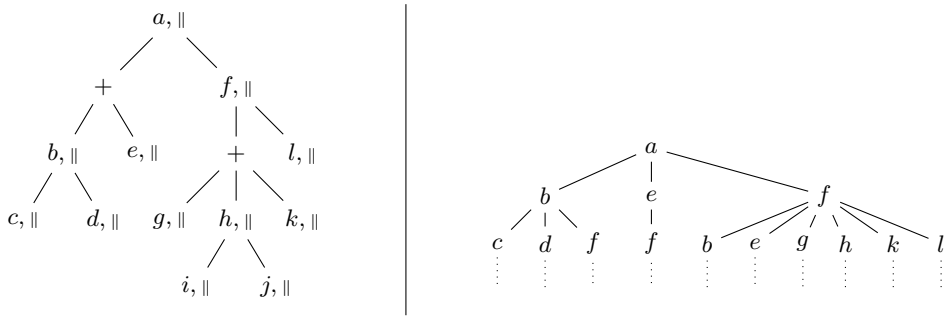


Figure 1 A process tree (left) and the first three layers of its computation tree (right).

specification of *process trees*:

$$\left[\begin{array}{lcll} \mathcal{A} & = & \mathcal{A}_{\|} + \mathcal{A}_{+} & \text{(process trees)} \\ \mathcal{A}_{\|} & = & \mathcal{Z} \times \textsc{Seq}\,(\mathcal{A}) & \text{(trees with action/parallel root, counted by } \mathcal{Z}) \\ \mathcal{A}_{+} & = & \mathcal{A}_{\|} \times \mathcal{A}_{\|} \times \textsc{Seq}(\mathcal{A}_{\|}) & \text{(trees with choice root)} \end{array}\right.$$

There is a strong correlation between this specification and the grammar discussed previously. The definition here is formal: this is the combinatorial class $\mathcal{A}$ of process trees. We remind the reader that a set is a combinatorial class if each element of the class has a known finite size. A further requirement is that there is only a finite number of elements in the class for a given size n. The $+$ operator on combinatorial classes denotes disjoint union. Thus a process tree is defined here as being either a merge tree in class $\mathcal{A}_{\|}$ or a non-deterministic choice in class A_{+}. A merge tree has a labelled root and its sub-trees form a (possibly empty) finite forest, as specified by the SEQ constructor. The $\mathcal{Z}$ mark explains that the label of the root node is counted and thus the size of a merge tree is the sum of the sizes of its sub-trees plus one. Finally, the non-deterministic choices are trees with a root that is not counted for the size and a sequence of at least two merge sub-trees. Since they are not counted, the choice nodes cannot nest otherwise arbitrarily large sub-trees of the same size could be constructed. In the same spirit, the prefixing of parallel nodes with actions is justified to avoid arbitrarily large intermixes of operators. Hence, as noted previously, there is a precise combinatorial motivation for guarding the choices.

The process tree representing the example given above is depicted on the left-hand side of Figure 1. For the sake of clarity we explicitly mark the nodes with $\|$ and $+$ labels but only the action could be shown without any ambiguity. According to the specification for class $\mathcal{A}$ given above, this tree has size 12 (only counting the atomic actions).

Following the principles of analytic combinatorics, the combinatorial class $\mathcal{A}$ admits a counting sequence A_n consisting of the number of objects of $\mathcal{A}$ of size n. This sequence is linked to a formal power series $A(z)$ such that $A(z) = \sum_{n\geq 0} A_n z^n$. The n-th coefficient of $A(z)$ is commonly denoted by $[z^n]A(z) = A_n$. Various analysis techniques can then deployed to "dissect" such power series, e.g. study convergence, find closed formulas and derive asymptotic results. Indeed, the class $\mathcal{A}$ is a quite classical tree model, as e.g. studied in [9, chapter I] .

▶ Fact 1. The combinatorial class $\mathcal{A}$ of process trees satisfies:

$$A(z) = \frac{1}{2}\left(1 - z - \sqrt{1 - 6z + z^2}\right); \qquad A_n \sim_{n\to\infty} \sqrt{\frac{3\sqrt{2}-4}{4\pi n^3}}\left(3 - 2\sqrt{2}\right)^{-n}.$$

The integer sequence A_n (precisely, shifted by 1) is known as *Large Schröder Numbers* and

appears naturally in lattice paths' context [13]. See for example [9, p. 475], or the *Online Encyclopedia of Integer Sequence*: OEIS A006318.

2.2 Semantics: computation trees

A classical semantic domain for process behaviours is the class of *computation trees* that we intend to study as combinatorial structures, a study that will cover most of the remaining material of this paper.

From an algebraic point of view, computation trees can be characterized quite concisely, for example using the following inference rules:

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ (act) \qquad \frac{P_i \xrightarrow{\alpha_i} P_i' \quad 1 \leq i \leq n}{P_1 + \ldots + P_i + \ldots + P_n \xrightarrow{\alpha_i} P_i'}\ (sum)$$

$$\frac{P_i \xrightarrow{\alpha_i} P_i' \quad 1 \leq i \leq n}{P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_n \xrightarrow{\alpha_i} P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_n}\ (par)$$

In combinatorics, these rules are interpreted as the transformation of a syntactic process tree T into a computation tree, denoted $[\![T]\!]$, with quite a constrained structure. In the special case of a process without any occurrence of the choice operator – as thoroughly studied in [3] – then each branch of the corresponding computation tree is a full traversal of the initial process. With the choice operator, one must first select exactly one branch for each choice – an operation we call a *global choice* – and only then traverse the resulting "choice-free" process. On the right-hand side of Figure 1 we depict the first three layers of the computation tree corresponding to our example process. Each node of the tree corresponds to a labelled transition that can be proved by the inference rules above. We further abstract from the process states since the transitions capture the required information. Note that the computation tree of Figure 1 is not represented entirely since it has a total of 1120 leaves! The calculation is detailed in Section 5.2.

There is a fruitful reinterpretation of this semantic construction in terms of partial orders. With pure parallel processes, the process trees can be seen as tree-like partial orders or *tree-posets*, as studied in e.g. [2]. The associated computation trees then encode the sets of their *linear extensions*, i.e. the strict orderings induced by the posets. In the presence of the non-deterministic choice, the process trees encode a family of tree-posets (one for each possible global choice) and the associated computation tree is the combination of all their linear extensions. For example, if we take the process tree of Figure 1, then $\langle a, e, f, l, g\rangle$ is a valid linear extension, whereas $\langle a, f, l, e, b, d, c\rangle$ is not because b and e occur in distinct branches of a choice node, and should thus be mutually exclusive. Also, $\langle a, e, l, f, g\rangle$ is invalid since f precedes l in the partial order.

3 Quantitative study I – number of computation paths

The understanding of the computation trees generated by non-deterministic processes as combinatorial objects requires a meaningful notion of size. The measure that conveys the most important information about the computation trees is their number of leaves, or equivalently the number of computation paths. Indeed, this measure directly relates (asymptotically, by a constant factor) to most other natural measures such as the total number of internal nodes (this is discussed at length in [3]).

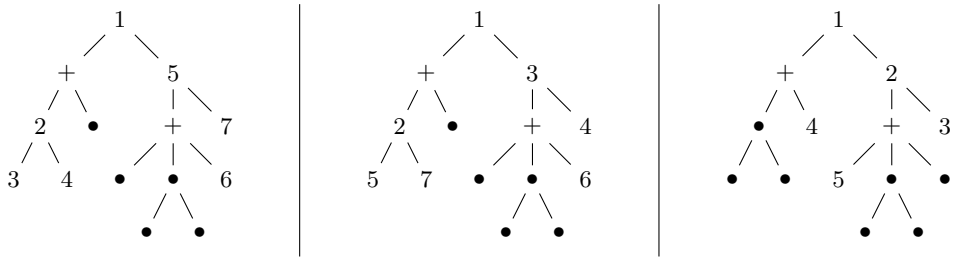


Figure 2 Three partially increasing trees based on the process tree of Figure 1.

3.1 Partially increasing trees

Consider T a process tree of size n. In [3] we show that the corresponding computation tree $[\![T]\!]$ has as many computation paths as the number of distinct ways to label the process tree T with increasing labels in range $[1..n]$.

If we add the choice operator, then the isomorphism with increasing trees is less direct since only the nodes that have been selected in a global choice must be labelled: exactly one branch for each choice node. All the other branches must be unselected in the resulting total number of possible computation paths. For this, we relax the total increasing labelling by allowing unlabelled nodes in the counting. To illustrate this combinatorial model, three distinct partial labellings for our running example are depicted on Figure 2. For example the leftmost case corresponds to a possible increasing labelling when the b and k branches are taken (a global choice labelled $\{b, k\}$). The unselected branches are labelled by the silent mark $\bullet$. Note that there are 1117 other possibilities of such partially increasing trees to count the total number of computation paths induced by the process tree of Figure 1.

The symbolic method can once again be used to formally define the combinatorial class of the partially increasing trees, denoted by $\mathcal{B}$, as follows:

$$\left[\begin{array}{lcl} \mathcal{B} & = & \mathcal{B}_{\|} + \mathcal{B}_{+} \\ \mathcal{B}_{\|} & = & \mathcal{W}^{\square\mathcal{W}} \star \mathcal{Z} \times \textsc{Seq}(\mathcal{B}) \\ \mathcal{B}_{+} & = & (\mathcal{B}_{\|} \times \mathcal{A}_{\|} \times \textsc{Seq}(\mathcal{A}_{\|})) + (\mathcal{A}_{\|} \times \textsc{Seq}(\mathcal{A}_{\|}) \times \mathcal{B}_{\|} \times \textsc{Seq}(\mathcal{A}_{\|})) \end{array}\right.$$

The specification above uses two distinct counting variables: $\mathcal{Z}$ for all the nodes and $\mathcal{W}$ only for the increasingly labelled nodes. A partially increasing tree in $\mathcal{B}$ may either be with a parallel or a choice root node, respectively in class $\mathcal{B}_{\|}$ and $\mathcal{B}_{+}$. The root node of a parallel node has two counts: one (totally) increasing in $\mathcal{W}$ and one for the total size in $\mathcal{Z}$ (including the unselected sub-trees marked with $\bullet$ in Figure 2). Its sub-trees consists of partially increasing trees in $\mathcal{B}$. The *box* notation $\mathcal{W}^{\square\mathcal{W}}$ specifies that the labelling counted by $\mathcal{W}$ must be increasing. This box operator introduces non-trivial differential functional equations, cf. [9, p. 139] for further details. The choice nodes can be formed out of two possibilities depending on where we select the branch in the semantics. In the first case, the leftmost branch of the choice has been selected, and thus below we need a partially increasing parallel tree in $\mathcal{B}_{\|}$ (we remind the reader that choice nodes cannot nest directly). The rest of the sub-trees are in $\mathcal{A}_{\|}$, which means that they are not labelled. The other possibility is that the branch we select is not the first one. This decomposition is required because a choice node must have at least two sub-trees to avoid any ambiguity with parallel nodes with only one sub-tree.

The most notable characteristic of partially increasing trees is that their specification mixes the box operator together with the unlabelled variant of the classical combinatorial

constructs. To our knowledge, this is the first study of such a mixed labelled/unlabelled combinatorial class, which represents, we believe, a contribution in the field of analytic combinatorics.

3.2 Average case analysis

The combinatorial class of partially increasing trees can be directly translated to the following system of generating functions:

$$\left[\begin{array}{ll} B_{\|}(z,w) & = \int_0^w \frac{z}{1-B_{\|}(z,t)-B_{+}(z,t)}\, dt \\ B_{+}(z,w) & = \frac{B_{\|}(z,w)\cdot A_{\|}(z)}{1-A_{\|}(z)} \cdot \left(1+\frac{1}{1-A_{\|}(z)}\right). \end{array}\right.$$

Here, the generating functions are bivariate with variable z for marking all the nodes and w marking only the increasingly labelled ones. We see here also the interpretation of the box operator as an integral. As usual with the symbolic method, the passage from the specification to the generating functions is completely automatic.

This system of generating functions is not trivial to study in analytic combinatorics. Resolving it requires the use of *holonomy theory* and related advanced techniques. In fact a computer algebra system must be used because some calculations are very intricate. We do however get a workable result.

▶ **Theorem 2.** *The generating function that enumerates the accumulated number of computation paths induced by all the process trees of a given size n is:*

$$\hat{B}(z) = \int_{w=0}^{\infty} (B_{\|}(z,w) + B_{+}(z,w)) \exp(-w) dw.$$

This function satisfies a linear homogeneous differential equation[1] *of order* 9 *whose coefficients are polynomials with highest degree* 15*.*

Now, using the holonomic equation of $\hat{B}(z)$, we can compute very efficiently its first terms (several thousands of terms can be obtained in a few seconds). For a more general complexity result, we are now interested in the asymptotic behaviour of the coefficients of $\hat{B}(z)$.

▶ **Theorem 3.** *Assuming the Hayman-admissibility of the function $\hat{B}$, the asymptotic of the average number of computation paths induced by all the process trees of a given size n is:*

$$\left(\frac{\left(6-4\sqrt{2}\right)n}{e}\right)^n \left(\frac{e\sqrt{2\pi}}{\sqrt{3\sqrt{2}-4}} n^{1/2} + \frac{1}{48}\frac{e\sqrt{2\pi}\left(12\sqrt{2}-7\right)}{\left(3\sqrt{2}-4\right)^{3/2}} n^{-1/2} + O(n^{-3/2})\right).$$

Roughly speaking, a function $C(z)$ is *Hayman-admissible* if the distribution of law $(\mathbb{P}(N = n) = a_n z^n / C(z))_z$ tends to a Gaussian distribution when z tends to infinity. The Hayman-admissibility of $\hat{B}$ is unequivocal, although the complete proof requires the difficult Wasow's theory [14]. Beyond H-admissibility, the proof of the theorem is based on saddle point analysis, cf. [9, Section V] for similar technical developments.

In the case of pure merge trees, we show in [3] that the average number of computation paths is $(n-1)!/2^n = \Theta(\sqrt{n}(0.5 \cdot n/e)^n)$. In comparison, the asymptotics for the process trees with choice nodes is approximately $\Theta(\sqrt{n}(0.34315 \cdot n/e)^n)$. It is interesting to note that

[1] The differential equation cited in Theorem 2 is not informative in itself, it is thus left omitted.

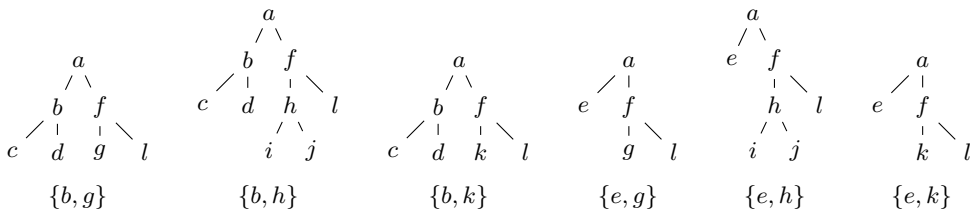


Figure 3 The global choices induced by the process tree of Figure 1.

the latter is exponentially smaller, which gives us a quantitative measure of the "cutting" effect induced by the choice operator. However, the reduction in size is not large enough for any algorithm requiring the explicit construction of the computation trees (even in some compacted form) to be of practical interest in general. Put in other terms, the combinatorial explosion phenomenon is thus not just a worst-case situation, but it is also quite perceptible in the average case.

4 Quantitative study II – average number of choices

In the previous section we obtained a precise although not really surprising negative result about the average number of computation paths. We now aim at measuring the choice itself. As explained previously, the choice operator – when interpreted globally – can be seen as encoding a family of computation trees resulting from choice-free process trees. In this section we study the average size of this family. This provides a rather precise characterization of the expressive power of the choice operator: the amount of information it encodes.

4.1 Generalized hook length formula

Let T be a process tree. A *global choice* of T is obtained by selecting exactly one sub-tree for each choice node of T. In Figure 3 we describe the set of all possible global choices for the process tree of Figure 1. There are indeed 6 possible global choices depending on the pairs of branches selected for the two choice nodes. Each choice can be identified by the root labels of the two selected sub-trees.

The important question of interest is the number of such possible global choices that can be expanded from typical process trees. Indeed, if we find that there are only a few possible choices on average, like for the example in Figure 1, then one might expand these choices and apply the efficient algorithms developed in [3] to analyze the computation trees without constructing them explicitly.

As a matter of fact, there is a tight connection between the possible number of such global choices, for a given process tree T, and the number of leaves of its computation tree $[\![T]\!]$ that we quantified in the previous section.

▶ **Lemma 4.** (Generalized hook length formula) *Let T be a process tree. The number ℓ_T of computation paths in $[\![T]\!]$ is given by the following formula:*

$$\ell_T = \sum_{C \text{ global choice of } T} \frac{|C|!}{\prod_{S \text{ sub-tree of } C} |S|}.$$

This theorem is justified as follows. If we make a global choice in a process tree (or if there is no choice to make), then we obtain a pure merge tree and in this case the number

of computation paths is given by the "standard" hook length formula (cf. [3]). Then by summing over all the possible global choices we obtain the desired result.

4.2 Choice expansion

Our objective is to count, for average process trees, the number of summands in the formula of Lemma 4. To this end we start with the following specification:

$$\left[\begin{array}{lcl} \bar{\mathcal{A}} & = & \bar{\mathcal{A}}_{\|} + \bar{\mathcal{A}}_{+} \\ \bar{\mathcal{A}}_{\|} & = & \mathcal{Z} \times \mathcal{Y} \times \textsc{Seq}\left(\bar{\mathcal{A}}\right) \\ \bar{\mathcal{A}}_{+} & = & \bar{\mathcal{A}}_{\|} \times \mathcal{A}_{\|} \times \textsc{Seq}(\mathcal{A}_{\|}) + \mathcal{A}_{\|} \times \textsc{Seq}(\mathcal{A}_{\|}) \times \bar{\mathcal{A}}_{\|} \times \textsc{Seq}(\mathcal{A}_{\|}) \end{array}\right.$$

Each tree in class $\bar{\mathcal{A}}$ is either a parallel or a choice tree. In the case of a parallel node, there are two distinct counts: $\mathcal{Z}$ counts the number of parallel nodes as previously, and $\mathcal{Y}$ only counts the nodes that are part of a given global choice. The sub-trees of parallel nodes are zero or more trees in $\bar{\mathcal{A}}$. In order to obtain a non ambiguous specification for $\bar{\mathcal{A}}_{+}$, we split it in two parts: either the first branch is in $\bar{A}_{\|}$, in which case the second and possibly the others are in $\mathcal{A}$, or it is another branch that belongs to $\bar{A}_{\|}$. In both cases, the choice nodes are not counted. Their sub-trees are formed by a single sub-tree in $\bar{\mathcal{A}}$, i.e. counted by both $\mathcal{Z}$ and $\mathcal{Y}$, and all the other sub-trees only counted by $\mathcal{Z}$ by combinatorial class $\mathcal{A}$ of "normal" computation trees, as defined in Section 2.

Working with the generating functions, we obtain the following result:

▶ **Theorem 5.** *The average number k of choices in a process tree of size n is, asymptotically, such that there are two constants A and B with $k = A \cdot B^n$. We have the following estimates*[2] *for the constants: $A \approx 1.4408$ and $B \approx 1.11062$. Moreover, the average size of a choice in a process tree of size n is, asymptotically, equal to $C \cdot n$ where $C \approx 0.49636$.*

This shows that on average the number of choices induced by non-deterministic processes is exponential. Thus, even if it is clearly more efficient to resolve the choices first and then work on the choice-free systems, this does not yield tractable algorithms in general, in the average (and not just the worst) case. From another perspective, this result appears to us as a fairly good quantitative witness of the expressiveness of the non-deterministic choice operator. The syntactic choice construct indeed provides a particularly succinct encoding at the semantic level of an arbitrarily large family of global choices.

5 Uniform random generation of computation paths

We describe in this section an algorithm to generate non-deterministic computation paths uniformly at random for a fixed non-deterministic process. This provides a basic building block for e.g. (uniform) random testing or statistical model checking (cf. [7]). As our quantitative study makes clear, the approaches of (1) constructing the computation trees or (2) expanding the non-deterministic choices first, both yield impractical algorithms even in the average case. Thankfully the symbolic method of analytic combinatorics leads to a more tractable approach.

[2] In Theorem 5 we have presented approximations instead of exact expressions for the constants, for the sake of brevity. In fact we have calculated the exact values but their expression is very intricate and does not bring any essential information.

5.1 Polynomial representation

Let T be a process tree, $T_1, T_2, \dots$ its children and $\mathrm{act}(T)$ the root action of T. We specify, in the sense of the symbolic method, the set $\mathcal{S}(T)$ of all runs of T:

$$\left[\begin{array}{ll} \mathcal{S}(T) & = \mathcal{S}^{\|}(T) + \mathcal{S}^{+}(T) \\ \mathcal{S}^{\|}(T) & = \mathcal{Z}^{\square}_{\mathrm{act}(T)} \star \textsc{Seq}_i(\mathcal{S}(T_i)) \\ \mathcal{S}^{+}(T) & = \mathcal{S}^{\|}(T_1) \times \mathcal{S}(T_2)^{\|} \times \dots \end{array}\right.$$

This is very similar to the specification of the semantic trees except that each parallel node is counted by a dedicated variable $\mathcal{Z}_{\mathrm{act}(T)}$ that records the action performed at the root of T. If a parallel node has a root action labelled a then the variable is denoted $\mathcal{Z}_a$. In fact, we consider the process T as a combinatorial class itself.

For our running example (cf. Figure 1), we get the following specification:

$$\mathcal{S}(T) = \mathcal{Z}_a^{\square} \star \left(\left(\mathcal{Z}_b^{\square} \star (\mathcal{Z}_c \times \mathcal{Z}_d) + \mathcal{Z}_e\right) \times \mathcal{Z}_f^{\square} \star \left(\left(\mathcal{Z}_g + \mathcal{Z}_h^{\square} \star (\mathcal{Z}_i \times \mathcal{Z}_j) + \mathcal{Z}_k\right) \times \mathcal{Z}_l\right)\right).$$

This is almost a *paraphrase* of the process tree under study. The idea, then, is to apply the *recursive method* of uniform random generation developed by Nijenhuis and Wilf [15] and latter generalized to combinatorial classes (cf. the papers [10, 16]). However this is not possible in a direct way, because, to our knowledge the box operator has not been fully integrated in these methods. For instance, even if we obtain a correct selection of the $\mathcal{Z}$'s, this would not tell us in which order the action labels must be taken. Thus we propose to adapt the recursive method by decomposing the approach in two distinct phases: (1) the selection of a global choice and (2) the generation of a computation path. For the first phase, we do not require to mark all the action labels but only those involving a non-deterministic choice, i.e. actions just below choice nodes. We thus apply a simple substitution on the specification, denoted as follows:

$$\mathcal{S}'(T) = S(T)\{\mathcal{Z}/\mathcal{Z}_v, y_w\mathcal{Z}/\mathcal{Z}_w\} \text{ label } w \text{ is at the root of a plus branch and } v \text{ otherwise.}$$

We thus keep a unique mark y_w for each branch the choices, and we anonymize all the other actions. Applied to our example, we obtain:

$$\mathcal{S}'(T) = \mathcal{Z}^{\square} \star \left(\left(y_b\mathcal{Z}^{\square} \star \mathcal{Z}^2 + y_e\mathcal{Z}\right) \times \mathcal{Z}^{\square} \star \left(\left(y_g\mathcal{Z} + y_h\mathcal{Z}^{\square} \star \mathcal{Z}^2 + y_k\mathcal{Z}\right) \times \mathcal{Z}\right)\right).$$

We now recall the principles of the recursive method, albeit adapted to fit the special requirements for the box operator. The generating function corresponding to a weighted specification $\mathcal{S}'$ is a polynomial defined inductively.

▶ Definition 6. Let $S'(T)$ be a weighted choice specification for a process tree T. The *polynomial* of T is $\mathscr{P}_{S'(T)}(x)$ in the single variable x depending on the parameters y_v's of $S'(T)$ and constructed according to the following rules:

$$\left[\begin{array}{ll} \mathscr{P}_{\mathcal{Z}^{\square} \star \textsc{Seq}(S'_i)}(x) = & \displaystyle\int_0^x \prod_i \mathscr{P}_{S'_i}(t)dt \\ \mathscr{P}_{\textsc{Seq}_{\geq 2}(y_{v_i} \times S'_i)}(x) = & \sum_i y_{v_i} \cdot \mathscr{P}_{S'_i} \end{array}\right.$$

Note that in particular we have $\mathscr{P}_{Z^{\square} \star \emptyset}(x) = x$. For our example we obtain:

$$\begin{aligned} \mathscr{P}(x) &= \int_0^x \left[y_b \cdot \textstyle\int_0^t u^2 du + y_e \cdot t\right] \times \left[\int_0^t \left(y_g \cdot u + y_h \cdot \textstyle\int_0^u v^2 dv + y_k \cdot u\right) \times u\, du\right] dt \\ &= \tfrac{x^9}{405} \cdot y_b y_h + \tfrac{x^7}{315} \cdot (5y_b(y_g + y_k) + 3y_e y_h) + \tfrac{x^5}{15} \cdot y_e(y_g + y_k) \end{aligned}$$

If compared to the specification $\mathcal{S}'(T)$ the polynomial $\mathscr{P}(x)$ provides a decomposition of the available global choices in terms of size given by the degree of each monomial in x. In our example, there are global choices of size 9, 7 and 5. The weight of each choice in term of the distribution on the computation paths corresponds to the coefficient for each monomial.

The algorithm to generate the polynomial $\mathscr{P}(x)$ only requires a single traversal of the initial process tree. Using a compact representation (based on directed acyclic graphs for sharing common sub-terms), the size of the polynomial is at most $\Theta(n^2)$. This is because in the worst case the maximum degree of this polynomial is n (the size of the process tree), and each of the coefficients contains at most n occurrences of the parameters y_v's.

5.2 Sampling algorithm

Algorithm 1: weighted random generation of a global choice

Data: a process tree T and its polynomial $\mathscr{P}_T(x) = \sum_d \frac{x^d}{C_d}\lambda_d$
Result: a set $C = \{v \mid v \text{ a label of } T\}$ identifying a global choice in T
$\pi_T := \{n_d = \Gamma(\frac{x^d}{C_d}\lambda_d\{y_v \leftarrow 1 \mid v \text{ a label of } T\})\}$
Pick $m \in [1; \sum_d \pi_T(n_d)]$ at random
Select d s.t. $S < m \leq S + n_d$ for $n_d \in \pi_T$ and $S = \sum_{d'<d} n_{d'}$ for $n_d, n_{d'} \in \pi_T$
return $V(\lambda_d) \stackrel{\text{def}}{=} \begin{cases} \{v\} \text{ if } \lambda_d = y_v \\ \bigcup_i \{V(\lambda_i)\} \text{ if } \lambda_d = \prod_i \lambda_i \\ V(\lambda_j) \text{ if } \lambda_d = \sum_i \lambda_i \text{ for } j \text{ identifying } w_j \end{cases}$
with w_j taken randomly in $W \stackrel{\text{def}}{=} \{w_i \mid w_i = \lambda_i\{y_v \leftarrow 1 \mid v \text{ a label}\},\ \lambda = \sum_i \lambda_i\}$

Based on the polynomial representation discussed previously, Algorithm 1 is used to sample a global choice with the correct relative weight according to the uniform distribution of computation paths. We illustrate this process with the same example as previously. In the first step of the algorithm we use the polynomial $\mathscr{P}(x)$ to construct π_T, an integer partition of the total number of computation paths of the process tree T. Each element of the partition is calculated by taking the Γ-transform (in x) of a monomial (of a given degree d; $\Gamma(\alpha_d \cdot x^d) = \alpha_d \cdot d!$) where all the constants y_v's are simply replaced by 1's. For our example we obtain the partition $\{n_9 = 896, n_7 = 208, n_5 = 16\}$ (thus the total number of computation paths is $\sum_d \pi_T(n_d) = 1120$). In the next step, we sample an integer m in $[1; 1120]$, for example $m = 900$. Consequently, we select the monomial in x^7 (choices of size 7) with weight 208 in the partition π_T. We are thus considering the coefficient $\lambda_7 = 5y_b(y_g + y_k) + 3y_e y_h$ in the root polynomial.

To compute $V(\lambda_7)$ in the next step, we must eliminate the outermost sum, which corresponds to the third (and most complex) case in the computation. For this we use the distribution W as defined in the algorithm. Each summand is associated in distribution W to a given weight w_i obtained by substituting all the y_v's of the choices of a given degree d and summand λ_i. One of the w_i's is taken arbitrarily (because the choice is non-deterministic). In our example, suppose we have $w_1 = 10$ for the left side and $w_2 = 3$ on the right side for the two summands of λ_7. To choose uniformly one of the two summands, we can pick a random integer in $[1; 13]$, for example 9. We thus select the left summand $5y_b(y_g + y_k)$. The top-most product is eliminated by simply computing the V of its operands. For the left operand, the label b is selected and in the right operand the sum is eliminated as already explained. If at the end we select the label k then we obtain $V(\lambda_7) = \{b, k\}$ which identifies a single choice in the starting process tree T. This global choice corresponds to the tree

named $\{b, k\}$ in Figure 3. Since we obtain a pure merge tree, the dynamic multiset random sampler of [3] can be used to obtain a computation path uniformly at random.

▶ **Theorem 7.** *Let T a process tree of size n. A computation path can be generated uniformly at random in $O(n^2)$ arithmetic operations and with $O(n^2)$ space complexity.*

The global choice is obtained from Algorithm 1 in time linear in the size of the polynomial $\mathscr{P}(x)$ (represented as a DAG) because each coefficient contains at most one occurrence of a parameter y_v. Once the choice is sampled, the generation of the linear extension from a choice-free process is achieved in time $O(n \log n)$ where n is the size of the sampled choice. Thus, the overall complexity of the proposed random generator is dominated by the construction of the polynomial $\mathscr{P}(x)$, which already involves $O(n^2)$ operations.

Random generation is only one of various questions we can answer thanks to the symbolic representation of the problem. For example, we can count the number of computation paths in quadratic time. Using a similar process, we can also compute the probability of a given computation prefix, which would be useful to guide the search of counter-example in statistical model-checking. The paper [3] gives the detail about these algorithmic variations, in the choice-free context.

6 Related work

The algebraic properties of "pure" non-deterministic processes have been extensively studied, cf. e.g. [1, 5]. However, the underlying *concrete* combinatorial objects remain mostly undiscovered.

Related forms of choice and shuffle (or parallel) operators naturally appear in the framework of regular languages. In terms of analytic combinatorics the regular shuffle on disjoint alphabets has already been studied with basic quantitative results in e.g. [8], and more thorough properties studied in the work of Mishna and Zabrocki [12]. They prove, interestingly, that the algebraic properties of the shuffle directly translates to the nature of the generating functions that encodes the enumeration problems. However the connection with the non-deterministic choice studied in the present paper is rather loose. First in regular language the non-deterministic automata can be determinized, which gives a too abstract view of concurrent processes in the general case. This indeed amounts to consider trace-equivalence *versus* bisimilarity when comparing languages vs. processes [11]. In both case a form of abstraction is proposed, although it is far more radical in the case of regular languages because one can simply forget about the tree structure of the process behaviours.

The application of uniform random generation of execution paths to software testing is discussed in [6]. The major difference is that the random generation is based on the semantic structure whereas in our approach only the (exponentially smaller) syntactic structure needs to be constructed.

The complexity of counting of linear extensions for arbitrary partial orders is ♯P-complete as shown by [4]. In [3] we show that for tree-like partial orders it is linear (whereas the previously known algorithm was quadratic [2]). With non-deterministic choices the present paper shows that a quadratic worst-case algorithm exists and we conjecture this is also a lower bound. We emphasize the fact that the algorithm computes the number of linear extensions for a potentially exponential number of tree-like partial orders.

7 Conclusion

The quantitative study of the non-deterministic choice operator is an important milestone towards our goal of reinterpreting concurrency theory from the analytic combinatorics point of view. In the next step in our study, we shall investigate various forms of synchronization, in general corresponding to reflecting the action labels within the semantics. This calls for a *non-strict* variant of increasing structures. This in turn would allow a deeper study of the *causes of non-determinism*, e.g. the combinatorial interpretations of internal choice vs. external choice.

Another interesting continuation of the work is to study the compaction of the computation trees by identifying common subtrees. This would amount to study the semantic trees up-to *bisimilarity*. Note that our algorithmic framework would not be affected by such study, since the explicit construction of the semantic trees (whether compacted or not) is not required.

Finally, as pointed out by a kind reviewer of the paper, the process trees we consider are finite trees whereas in practice more expressive classes must be considered. An important goal we seek next is to apply our algorithmic framework in the context of e.g. regular processes. We do think an approach in the spirit of partial-order unfoldings of processes is feasible.

References

1 L. Aceto. On relating concurrency and non-determinism. In *MFPS*, volume 598 of *LNCS*, pages 376–402. Springer, 1991.
2 M. D. Atkinson. On computing the number of linear extensions of a tree. *Order*, 7:23–25, 1990.
3 O. Bodini, A. Genitrini, and F. Peschanski. Enumeration and random generation of concurrent computations. In *DMTCS AofA'12 proceedings*, pages 83–96, 2012.
4 G. Brightwell and P. Winkler. Counting linear extensions is #P-complete. In *STOC*, pages 175–181, 1991.
5 S. Christensen. *Decidability and decomposition in process algebras*. PhD thesis, University of Edinburgh, 1993.
6 A. Denise, M.-C. Gaudel, S.-D. Gouraud, R. Lassaigne, J. Oudinet, and S. Peyronnet. Coverage-biased random exploration of large models and application to testing. *STTT*, 14(1):73–93, 2012.
7 M.-C. Gaudel et al. Coverage-biased random exploration of models. In *ETAPS Workshop on Model Based Testing*, volume 220, pages 3–14. ENTCS, 2008.
8 P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):207–229, 1992.
9 P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge Univ. Press, 2009.
10 P. Flajolet, P. Zimmermann, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theor. Comput. Sci.*, 132(2):1–35, 1994.
11 R. Milner. *A Calculus of Communicating Systems*. Springer Verlag, 1980.
12 M. Mishna and M. Zabrocki. Analytic aspects of the shuffle product. In *STACS*, pages 561–572, 2008.
13 R. P. Stanley. *Enumerative Combinatorics, Vol. 2*. Cambridge Univ. Press, 2001.
14 W. Wasow. *Asymptotic Expansions for Ordinary Differential Equations*. Dover phoenix editions. Dover, 2002.
15 H. S. Wilf and A. Nijenhuis. *Combinatorial algorithms : An update*. CBMS-NSF. Philadelphia, Pa. Society for Industrial and Applied Mathematics, 1989.
16 P. Zimmermann. Random generation of unlabelled combinatorial structures. Technical report, Algorithms Seminar, 1993–1994, volume 2381. INRIA, 1994.

Renting a Cloud

Barna Saha

AT&T Shannon Research Laboratory
180 Park Avenue, Florham Park, NJ, USA
barna@research.att.com

Abstract

We consider the problem of efficiently scheduling jobs on data centers to minimize the cost of renting machines from "the cloud." In the most basic cloud service model, cloud providers offer computers on demand from large pools installed in data centers. Clients pay for use at an hourly rate. In order to minimize cost, each client needs to decide on the number of machines to be rented and the duration of renting each machine. This suggests the following optimization problem, which we call Rent Minimization. There is a set $J = \{j_1, j_2, \ldots, j_n\}$ of n jobs. Job j_i is released at time $r_i \geq 0$, has a deadline of d_i, and requires $p_i > 0$ contiguous processing time, $r_i, d_i, p_i \in \mathbb{R}$. The jobs need to be scheduled on identical parallel machines. Machines may be rented for any length of time; however, the cost of renting a machine for $\ell \geq 0$ time units is $\lceil \ell/D \rceil$ dollars, for some given large real D; in particular, one pays \$2 whether the machine is rented for $D+1$ or $2D$ time units. The goal is to schedule all the jobs in a way that minimizes the incurred rental cost.

In this paper, we develop offline and online algorithms for Rent Minimization problem. The algorithms achieve a constant factor approximation for the offline version and $O(\log \frac{p_{max}}{p_{min}})$ for the online version, where p_{max} and p_{min} are the maximum and minimum processing time of the jobs respectively. We also show that no deterministic online algorithm can achieve an approximation factor better than $\log_3 \frac{p_{max}}{p_{min}}$ within a constant factor. Both of these algorithms use the well-studied problem of Machine Minimization as a subroutine. Machine Minimization is a special case of Rent Minimization where $D = \max_i d_i$. In the process of solving the Rent Minimization problem, in this paper, we also develop the first online algorithm for Machine Minimization.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, F.2.2 Nonnumerical Algorithms and Problems, F.1.2 Modes of Computation

Keywords and phrases Scheduling Algorithm, Online Algorithm, Approximation Algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.437

1 Introduction

Computing in "the cloud" has emerged as a popular way for companies to compute and to store data. The first commercial cloud service was Amazon Web Services's Elastic Compute Cloud (EC2). Since the appearance of EC2, many other companies have offered their own cloud services. Low maintenance overhead and the ability to pay only for what one uses have induced numerous clients to move their businesses onto the cloud. The clients rent machines from cloud vendors either in advance according to their projected job load or on demand. Payment is usually made at an hourly rate.

While obtaining adequate service, clients want to minimize rental cost. Motivated by this issue of reducing rental cost, we define Rent Minimization as follows. There is a set $J = \{j_1, j_2, \ldots, j_n\}$ of n jobs. Job j_i is released at time $r_i \geq 0$, has a deadline at time d_i, and requires p_i units of contiguous processing time. The jobs need to be scheduled on

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 437–448

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

identical parallel machines. Machines may be rented for any duration; however, the cost of renting a machine for $\ell \geq 0$ time units is $\$\lceil \ell/D \rceil$, for a given large real D; in particular, one pays \$2 whether the machine is open for $D + \epsilon$ or $2D$ time units. The goal is to schedule all the jobs in a way that minimizes the incurred rental cost. We call this problem Rent Minimization. To quote the Amazon pricing model, "*Pricing is per instance-hour consumed for each instance, from the time an instance is launched until it is terminated. Each partial instance-hour consumed will be billed as a full hour.*"–this is precisely the model used for defining Rent Minimization.

Rent Minimization is a generalization of the well-studied classical problem called Machine Minimization. The input to this problem is again a set of jobs each with a release time, deadline and processing time, and the goal is to use a minimum number of machines such that each job is allocated, between its release time and deadline, an interval (closed on the left and open on the right) of length equal to its processing time, on some machine. Garey and Johnson [9] show that for this case, deciding whether all the jobs can be scheduled on a single machine is already NP-hard. The problem admits a constant-factor approximation [5]; this paper improves upon the previous work of Chuzhoy, Guha, Khanna and Naor [6], which achieved an approximation factor of $O(\sqrt{\frac{\log n}{\log \log n}})$. (Taking an instance of Machine Minimization and setting $D = \max_{i=1}^{n} d_i$ converts the instance to one of Rent Minimization, for the cost of renting any machine, for any duration, is \$1.) There is also a discrete version of Machine Minimization, in which the discrete set of multiple intervals in which each job can be feasibly scheduled is given explicitly. The discrete version of the problem is markedly harder than the continuous version (single release time and deadline) and is known to be inapproximable beyond $\Omega(\log \log n)$ [7].

Our algorithm for the offline version of Rent Minimization is based on carefully classifying jobs based on processing times and difference between the release times and deadlines, so that by losing a constant factor either an algorithm for Machine Minimization can be used for them or there exists a combinatorial algorithm for scheduling the jobs. Therefore, we achieve an approximation ratio of $\Theta(\alpha)$ where α is the approximation factor for Machine Minimization. For the online version, the most nontrivial part is to design an online algorithm for Machine Minimization. To the best of our knowledge, no online algorithm for Machine Minimization was known before this work. Next, using the same classification of jobs, the online Rent Minimization problem can be solved.

Contributions

1. Following the pricing model of Amazon EC2, we define a new class of problems called Rent Minimization with the objective of minimizing the amount of money paid by cloud users while renting machineries from the cloud. We develop a constant factor offline algorithm for Rent Minimization. In the online version we provide an algorithm with approximation factor of $O(\log \frac{p_{max}}{p_{min}})$ which also matches the lower bound of any deterministic online algorithm within a constant factor.
2. We develop the first online algorithm for the well-studied problem of Machine Minimization. The algorithm achieves an approximation factor of $O(\log \frac{p_{max}}{p_{min}})$ which also matches the lower bound of any deterministic online algorithm for Machine Minimization within a constant factor.

Related Work

The *throughput maximization*, also known as *real time scheduling* problem, focuses on maximizing the number of jobs scheduled given a pre-fixed number of machines and can be

thought of as the dual of Machine Minimization. In a more general setting, jobs may have weights and goal is to maximize the total weight of the scheduled jobs in given number of machines. The best known approximation factor for throughput maximization for the unweighted case is $\frac{e}{e-1}+\epsilon$, for any constant ϵ [8], whereas, for the weighted case nothing better than 2-approximation is known [2]. In the discrete setting, even for the unweighted case with only 2 feasible intervals for each job, the throughput maximization is known to be MAX-SNP hard. However, no such hardness result is known for the continuous case. The problem has also been studied in the online setting for unit length jobs and equal length jobs both in the single processor and multi-processor setting, where decision to schedule a job must be made at the time job arrives [3, 4]. The jobs may also have heights indicating the resource requirements and multiple jobs can be run in parallel in a single machine as long as the total width of the jobs at any time is not more than 1 [1, 8].

2 Offline Algorithm

Let the intervals of the form $[(r-1)D, rD)$, for positive integers r, be *machine intervals*. We assume that the earliest release time of the jobs in J is 0. We partition the job sets J into three subsets, $J = J_1 \sqcup J_2 \sqcup J_3$.

1. $J_1 = \{j \mid p(j) \geq \frac{D}{2}\}$. These are the big jobs.
2. $J_2 = \{j \mid j \notin J_1, \exists a \in \mathbb{N}, r(j) < aD, r(j)+p(j) > aD, d(j)-p(j) < aD\}$. No matter in which valid intervals we schedule these jobs, their scheduling intervals always contain the time aD.
3. $J_3 = J \setminus \{J_1 \cup J_2\}$.

Let *rent-OPT* denote the renting cost of an optimal solution for Rent Minimization (RM). We first show how to schedule jobs in J_1 and J_2 at rental cost $O(\textit{rent-OPT})$. We next consider the following special case of RM and denote it by SRM.

Each machine is open for exactly one machine interval. The entire time period, from the earliest release time to the latest deadline, can be partitioned into s machine intervals, where s is polynomial in n.

Let *Srent-OPT* denote the renting cost of an optimal solution for SRM.

For jobs in J_3, we show that any solution for RM can be reduced to a solution in SRM by losing only a constant factor. Therefore, a constant-factor approximation for SRM implies a constant-factor approximation algorithm for RM as well. We next further partition the jobs in J_3 into two categories, $J_3 = J_3^1 \sqcup J_3^2$.

▶ **Definition 1.** Let $H_r = [(r-1)D, rD)$ be the rth machine interval.

1. $J_3^1 = \{j \in J_3 \mid \exists r \in \mathbb{N}, r(j) \in H_r, d(j) \in H_r \cup H_{r+1}\}$.
2. $J_3^2 = \{j \in J_3 \mid \exists s \in \mathbb{N}, r(j) \in H_r, d(j) \in H_t, t \geq r+2\}$.

Scheduling jobs in J_1 and J_2. We simply assign a new machine to each job $j \in J_1$ and keep the machine open exactly for $\lceil \frac{p(j)}{D} \rceil$ machine intervals. We show by doing so the rental cost can increase at most by a factor of 3.

▶ **Lemma 2.** *The rental cost where each job $j \in J_1$ is assigned to a new machine that is open exactly for $\lceil \frac{p(j)}{D} \rceil$ machine intervals within a cost of 2rent-cost.*

Proof. We know $\textit{rent-OPT} \geq \sum_{j \in J} \frac{p_j}{D} \geq \sum_{j \in J_1} \frac{p_j}{D}$. The rental cost for scheduling jobs in J_1 each to a new machine is $\sum_{j \in J_1} \lceil \frac{p_j}{D} \rceil \leq 2 \sum_{j \in J_1} \frac{p_j}{D} \leq 2(\textit{rent-OPT})$. ◀

For jobs in J_2, we again assign a new machine to each of them and rent the machine exactly for D time units. We show that by doing so the incurred rental cost for assigning jobs in J_2 is at most *rent-OPT*.

▶ **Lemma 3.** *The rental cost for assigning each job in J_2 to a new machine is at most rent-OPT.*

Proof. Consider any two jobs $j_1, j_2 \in J_2$ such that for some $a \in \mathbb{N}$, they have $r(j_1) < aD, r(j_1)+p(j_1) > aD, d(j_1)-p(j_1) < aD$ and $r(j_2) < aD, r(j_2)+p(j_2) > aD, d(j_2)-p(j_2) < aD$. Consider the time point aD. No matter where the jobs j_1 and j_2 are scheduled, the intervals in which they are scheduled contain the time point aD. Hence, j_1 and j_2 must be assigned to two different machines in OPT.

Fix a machine. If there are exactly l jobs in J_2 scheduled on that machine, then it is possible to find positive integers $a_1 < a_2 < \cdots < a_l$ such that for all i, one job $j_i \in J_2$ is scheduled during a half-open interval containing $a_i D$. The number of half-open intervals $[x, x + D)$ of length D needed to cover $\{a_1 D, a_2 D, ..., a_l D\}$ is l, since no such interval can cover two such points.

It follows that the rental cost associated with that one machine is at least l. Since we can sum over machines, it follows that *rent-OPT* $\geq |J_2|$. ◀

Hence, we get the following lemma.

▶ **Lemma 4.** *There exists a schedule in which each job $j \in J_1 \cup J_2$ is assigned to a new machine that is open exactly for $\lceil \frac{p(j)}{D} \rceil$ machine intervals and incurs a cost at most 3rent-cost.*

Proof. Follows from Lemma 2 and Lemma 3. ◀

Reduction to SRM – Making machine opening and closing time uniform. We now consider only the jobs in J_3. We next show that if there are jobs J_3' in an optimal schedule that are allocated intervals overlapping two consecutive machine intervals, then by blowing up the rental cost at most by a factor of 5, all these jobs can be scheduled completely inside a single machine interval. Moreover, machines can be assumed to be rented exactly for one machine interval, and opened and shut down at integral multiples of D.

We consider the first machine interval to start at the earliest release time (counting it as 0), and the last machine interval to end at the smallest multiple of D greater than or equal to the latest deadline. If a machine is open from time a to b in OPT, we instead open it at time $a' \leq a$, $a' = D\lfloor \frac{a}{D} \rfloor$, which is the largest multiple of D which is less than or equal to a. Similarly, we can shut down the machine at time $b' \geq b$, $b' = D\lceil \frac{b}{D} \rceil$. A minute's thought will confirm that overall we can increase *rent-OPT* at most by a factor of 2 in this process. We do not change the scheduling intervals of the jobs in J_3 in this process.

Now consider the subset $J_3' \subseteq J_3$ of jobs that are scheduled in OPT in intervals that overlap two machine intervals. Since these jobs have processing time less than $\frac{D}{2}$, they overlap at most two machine intervals. For any such job j, if $(a-1)D \leq r(j) \leq a(D)$, for some $a \in \mathbb{N}$, we either have $r(j) + p(j) \leq aD$, or $d(j) - p(j) \geq aD$; otherwise, j would belong to J_2. It is possible to allocate each of these jobs a new machine such that the job is scheduled entirely within one machine interval. The machine needs to be rented for just one machine interval. If there are l jobs of J_3' scheduled in a machine, then that machine is rented at least for l machine intervals, since each of these jobs contain a different boundary point of machine intervals. Therefore, *rent-OPT* $\geq |J_3'|$. Since each new machine that can be rented to allocate J_3' needs to be open exactly for D time units, the rental cost can increase at most by $|J_3'|$. Hence, we get the following lemma.

▶ **Lemma 5.** *There exists a schedule of J_3 such that each job is scheduled entirely within a single machine interval and rental cost is at most $3rent\text{-}OPT$.*

Since all jobs are scheduled within a single machine interval, without loss of generality, we can assume that each machine is open exactly for D time units.

Hence at the end, we have a bunch of machines each opened and closed at multiples of D, each remaining open for exactly D time units, and which together schedule all the jobs in J_3.

Making the number s of machine intervals polynomial in n. Can be found in the full-version of the paper[1]. Hence, we have an instance of SRM.

We now proceed with jobs in J_3^1 and J_3^2 respectively.

Scheduling jobs in J_3^1 and J_3^2. Let $J_{3,r}^1 = J_3^1 \cap \{j \mid r(j) \in H_r\}$. For each $J_{3,r}^1, r = 1, 2, ..., s$, we define an instance of Machine Minimization and invoke an algorithm for it such as [5] to schedule the jobs.

▶ **Lemma 6.** *The rental cost for assigning the jobs in J_3^1 is at most $2\alpha Srent\text{-}OPT$ where α is the approximation ratio of the best* Machine Minimization *algorithm.*

Proof. In an optimal solution for SRM the machine intervals used for scheduling jobs in $J_{3,r}^1$ are completely disjoint from jobs in $J_{3,r'}^1$ such that $|r - r'| \geq 2$. Therefore, the total rental cost paid by $J_{3,r}^1, r = 0, 2, 4, 6, ...$ is the sum of the rental cost paid by each $J_{3,r}^1, r = 0, 2, 4, 6, ...$ which is the total optimal cost for Machine Minimization for each of $J_{3,r}^1, r = 0, 2, 4, 6, ...$ Hence using the polynomial time algorithm for Machine Minimization with the best approximation factor, the total cost paid for $J_{3,r}^1, r = 0, 2, 4, 6, ...$ is at most α $Srent\text{-}OPT$. Similarly, the total rental cost paid for $J_{3,r}^1, r = 1, 3, 5, 7, ...$ is at most α $Srent\text{-}OPT$. Therefore, the overall cost paid is 2α $Srent\text{-}OPT$. ◀

We now consider scheduling jobs in J_3^2. The following lemma shows that every job in J_3^2 can be scheduled in machine intervals that do not contain its release time or deadline by losing a factor 3 in the approximation for SRM.

▶ **Lemma 7.** *There exists a schedule of jobs in J_3^2 with total rental cost at most $3Srent\text{-}OPT$ such that no job is scheduled in machine intervals containing its release time or deadline.*

Proof. Consider an optimal schedule. We perturb the optimal schedule for SRM in a way so that by paying at most $3Srent\text{-}OPT$ every job is scheduled in a machine interval that neither contains it release time nor deadline. Let H_r be one such interval where some jobs $J' \subseteq J_3^2$ are both released and scheduled and some jobs $J'' \subseteq J_3^2$ have their deadlines and are scheduled. There must be at least one such interval H_r for which either J' or J'' is non-empty, otherwise, the lemma trivially holds. If J' are scheduled in h_1 machines in H_r, then rent h_1 extra machines each for D time units in the following machine interval H_{r+1} and schedule the jobs in J' in these new h_1 machines in H_{r+1} exactly the way they were scheduled in H_r. That is if a job $j' \in J'$ was scheduled from time $[a, b)$ in H_r then schedule it at time $[a + D, b + D)$ in H_{r+1}. This is a feasible schedule for J' as the earliest of their deadlines can occur in machine interval H_{r+2} or higher. We charge the h_1 machines to machine interval H_{r+1}. If J'' are scheduled in h_2 machines in H_r, then rent h_2 extra

[1] `http://www.cs.umd.edu/~barna/renting-cloud.pdf`

machines each for D time units in the previous machine interval H_{r-1} and schedule the jobs in J'' in these new h_2 machines in H_{r-1} exactly the way they were scheduled in H_r. That is if a job $j'' \in J''$ is scheduled from time $[a, b)$ in H_r then schedule it at time $[a - D, b - D)$ in H_{r-1}. This is a feasible schedule for J'' as the latest release time can occur in machine interval H_{r-2} or lower. We charge h_2 machines to machine interval H_{r-1}.

Each machine interval is charged at most twice with at most the total number of machines in the machine intervals immediately before and after it. Therefore, the total rental cost can go up only by a factor of 3. ◀

We will schedule each job in J_3^2 only in machine intervals that do not contain its release time or deadline as follows. We treat each machine interval as a point on a line. Each job $j \in J_3^2$ can be scheduled in some consecutive machine intervals, denoted by an interval I_j. We pick minimum number of points to cover all the intervals and if I_j is covered by a point p_k then we assign job j to machine interval k. This problem of interval covering can be solved optimally, in fact by a single scan.

Algorithm A

1. Scan the right end point of the intervals in non-decreasing order. If the first right end point is p_k, insert p_k in the solution and let it cover all the intervals that contain it. Remove all the intervals that are covered. Continue.
2. If machine intervals indexed $i_1, i_2, ..., i_l$ are selected in this procedure to cover jobs $J_{i_1}, J_{i_2}, ..., J_{i_l}$, then pack jobs J_{i_j} greedily using minimum number of machines each opened for D time units in machine interval H_{i_j}.

It can easily be verified that step 1 of the above algorithm returns minimum number of points. Both step 1 and 2 of Algorithm A can also be implemented easily in an online model where jobs arrive online.

▶ **Lemma 8.** *The rental cost for assigning the jobs in J_3^2 is at most $9Srent\text{-}OPT$.*

Proof. Let $Srent\text{-}OPT'$ denote the optimal rental cost for jobs in J_3^2 where no job is scheduled in machine interval containing its release time or deadline. Then from Lemma 7 $Srent\text{-}OPT' \leq 3Srent\text{-}OPT$.

The total rental cost paid is $\sum_{j=1}^{l} \lceil \frac{2p(J_{i_j})}{D} \rceil$ where $p(J_{i_j}) = \sum_{j \in J_{i_j}} p(j)$. The factor 2 comes from the fact that each opened machine may have at most $D/2$ empty slots. Now,

$$\begin{aligned} &\textstyle\sum_{j=1}^{l} \lceil \frac{2p(J_{i_j})}{D} \rceil \leq l + \sum_{j:p(J_{i_j}) \geq D} \lceil \frac{2p(J_{i_j})}{D} \rceil \\ &\leq Srent\text{-}OPT' + 2Srent\text{-}OPT' \leq 9Srent\text{-}OPT, \end{aligned}$$

where the second inequality comes from the fact that both l and $\sum_{j:p(J_{i_j}) \geq D} \lceil \frac{p(J_{i_j})}{D} \rceil$ serve as a lower bound on $Srent\text{-}OPT'$. ◀

▶ **Theorem 9.** *There exists a polynomial-time approximation algorithm for* RM *that incurs a rental cost of $(30 + 6\alpha)rent\text{-}OPT$, where $rent\text{-}OPT$ is the optimal rental cost and α is the approximation factor of the algorithm from [5] for* Machine Minimization.

Proof. From Lemma 4, Lemma 6 and Lemma 8, we have the total cost to be at most $3rent\text{-}OPT + (2\alpha + 9)Srent\text{-}OPT$ which is at most $30 + 6\alpha rent\text{-}OPT$ from Lemma 5. ◀

Note that we have not tried to optimize the constants which we believe can be reduced further.

3 Online Algorithm for Machine Minimization

In this section, we give an online algorithm for Machine Minimization problem. We first consider the case where jobs have nearly uniform processing time. We assume, each job has processing time in $[p, 2p]$, $p > 0$. We decompose the jobs into two sets based on the length of their time window, that is the duration between their release time and deadline.

▶ **Definition 10.** We call a job j "*slack*" if its release time $r(j)$ and deadline $d(j)$ satisfy $d(j) - r(j) \geq 8p$. Else, we call a job "*non-slack*".

▶ **Definition 11.** For a job j, any interval of length $p(j)$ in $[r(j), d(j))$ is referred to as a valid interval. The valid interval in which the job j is scheduled is referred to as its scheduling interval.

Suppose there exists a schedule of all the jobs in t machines. Clearly, using $2t$ machines, we can schedule the slack jobs in t machines and non-slack jobs separately in another t machines. Hence, with a loss of a factor of 2 in the approximation, we can consider scheduling of slack jobs and non-slack jobs separately.

3.1 Scheduling Slack Jobs

Let J_s denote the set of slack jobs. First, we show that any optimal schedule of J_s can be transformed into another schedule by blowing up the number of machines by at most a factor of 3 such that the scheduling interval of no job $j \in J_s$ intersects $[r(j), r(j) + 2p)$, or $[d(j) - 2p, d(j))$, that is, we wait at least $2p$ time units from the release time of each job before scheduling it and the job finishes $2p$ time units before its deadline.

▶ **Lemma 12.** *If t denotes the minimum number of machines in which all slack jobs can be scheduled, then there exists a schedule of all slack jobs using $3t$ machines, such that the scheduling interval of no job intersects the first $2p$ time units or the last $2p$ time units of its time window.*

Proof. Consider $J'_s \subseteq J_s$ be the set of jobs with their scheduling intervals intersecting the first $2p$ time units from their release time in an optimal solution. Let $J''_s \subseteq J_s \setminus J'_s$ be the set of jobs with their scheduling window intersecting the last $2p$ time units. We first schedule all the jobs in $J_s \setminus J'_s \cup J''_s$ in t machines as in the optimal solution. For each job $j \in J'_s$ that was initially scheduled on the rth, $r \in \{1, 2, \ldots, t\}$, machine in the optimal solution from time $[a, a + p(j))$, we schedule it in the $(t + r)$th machine from time $[a + 2p, a + 2p + p(j))$. First, clearly, $[a + 2p, a + 2p + p(j)) \cap [r(j), r(j) + 2p) = \emptyset$, since $a \geq r(j)$. Second, since $a < r(j) + 2p$ (by definition of J'_s), $p(j) \leq 2p$, we have $a + 2p + p(j) < r(j) + 6p$ but $d(j) \geq r(j) + 8p$. Hence, $a + 2p + p(j) \leq d(j) - 2p$. Therefore, $[a + 2p, a + 2p + p(j))$ is a valid interval for job j, and we have $r(j) + 2p \leq a + 2p < a + 2p + p(j) \leq d(j) - 2p$. Similarly, For each job $j \in J''_s$ that was initially scheduled on the rth, $r \in \{1, 2, \ldots, t\}$, machine in the optimal solution from time $[b, b + p(j))$, we schedule it in the $(t + 2r)$th machine from time $[b - 2p, b2p + p(j))$. Since $d(j) \geq b + p(j) \geq d(j) - 2p$, we have $d(j) - 2p \geq b - 2p + p(j) > b - 2p \geq d(j) - 6p \geq r(j) + 2p$. Therefore, $[b - 2p, b - 2p + p(j)]$ is a valid interval for job j, and we have $r(j) + 2p \leq b - 2p < b - 2p + p(j) \leq d(j) - 2p$. ◀

Define $r(j)_{new} = 2p\lceil r(j)/2p \rceil$, and $d(j)_{new} = 2p\lfloor d(j)/2p \rfloor$. From Lemma 12, we get the following corollary.

▶ **Corollary 13.** *If t denotes the minimum number of machines in which all slack jobs J_s can be scheduled, then there exists a schedule of J_s in $3t$ machines such that the scheduling window of any job $j \in J_s$ is contained in $[r(j)_{new}, d(j)_{new})$.*

Proof. From Lemma 12, we know there exists a schedule of jobs J_s in $3t$ machines such that, each job is scheduled after $2p$ time units of its release time $r(j)$, and hence after $r(j)_{new}$. Also, these jobs finish before $2p$ time units from their deadline and hence before $d(j)_{new}$, since $d(j) \geq 8p$. ◀

We can therefore assume without loss of generality that jobs are released and have deadlines at an integral multiples of $2p$.

We can further assume, with a loss of a factor of 2 in the approximation that the starting point of the scheduling interval of each job is an integral multiple of $2p$.

▶ **Lemma 14.** *If there exists a schedule of jobs in J_s with release time $r(j)_{new}$, deadline $d(j)_{new}$, and processing time $p(j) \in [p, 2p]$ in t' machines, then there exists a schedule of J_s in $2t'$ machines such that the starting point of the scheduling window of each job is an integral multiple of $2p$. Furthermore, for any $a \in \mathbb{Z}^+$, in the time window $[a * 2p, (a+1)2p)$ at most one job is scheduled in each machine.*

Proof. Consider an optimal schedule. We proceed in increasing order of time and modify the optimal schedule. Since the processing time of each job is in $[p, 2p]$, for each machine, the starting points of the scheduling intervals of at most two jobs j_1, j_2 can be contained in the time window $[0, 2p)$. If there are such j_1 and j_2, then, since "new" release times are integral multiples of $2p$, we must have $r(j_1)_{new} = r(j_2)_{new} = 0$. If j_1 and j_2 are scheduled on the rth machine, $r \in \{1, 2, \ldots, t'\}$, we schedule j_1 on the rth machine and j_2 on the $(r + t')$th machine respectively in the intervals $[0, p(j_1))$ and $[0, p(j_2))$. Clearly, the schedule in feasible and only one job is scheduled in each machine in the time window $[0, 2p)$. Now, by induction hypothesis, suppose, we can obtain a modified schedule for the jobs scheduled in $[0, a * 2p)$ in the optimal solution. Now, consider the time window $[a * 2p, (a + 1) * 2p))$. For the rth machine, again the starting points of the scheduling intervals of at most two jobs can be contained in $[a*2p, (a+1)*2p))$ in the optimal solution. We can safely schedule them respectively in the rth and $(r + t')$th machine starting from $a * 2p$, since the "new" release times of these two jobs cannot be after $a * 2p$. We thus have the proof by induction. ◀

Therefore, we strive to obtain a schedule with the following property: *We have a set of jobs J_s, with each job j having processing time in $[p, 2p]$, release time $r(j)_{new}$ that is an integral multiple of $2p$ and deadline $d(j)_{new}$. The goal is to use minimum number of machine and schedule all the jobs such that the scheduling interval of each job starts at integral multiple of $2p$.*

If we can obtain an online algorithm for the above stated scheduling problem with approximation factor γ, then from Lemma 12, Corollary 13 and Lemma 14, we get a schedule of all the slack jobs on 6γ times optimal number of machines. Indeed, we show, if OPT denotes the optimal number of machine for such schedules and we know OPT, then the following algorithm (EDF(OPT)) which schedules jobs based on the earliest deadline among all the released but unscheduled jobs at any time is optimal.

▶ **Lemma 15.** *Given there exists a schedule of jobs J_s in OPT machines where each job has release time and deadline at an integral multiple of $2p$, processing time in $[p, 2p]$ and must be scheduled at an integral multiple of $2p$,* EDF*(OPT) returns a feasible schedule in OPT machines.*

Algorithm 1 Earliest Deadline First Among Released yet Unscheduled Jobs (EDF)(s)

time $t = 0$,
while t is less than the latest deadline of the jobs in J_s **do**
  $a = 1$
  while there exists an unscheduled job with release time $r(j) \leq t$ and $a \leq s$ **do**
    Pick among these remaining jobs the one with the earliest deadline (breaking ties arbitrarily) and schedule it on the ath machine in the time window $[t, t + p(j))$.
    $a = (a + 1)$;
  end while
  $t = t + 2p$;
end while

Proof. An EDF schedule implies for any two jobs j_1 and j_2 scheduled on the same machine, if j_1 is scheduled earlier than j_2, then either j_1 has deadline before j_2, or j_2 is not released until j_1 finishes. We can convert any given optimal schedule to an EDF schedule. Given an optimal schedule, if it is not EDF, then consider the jobs in increasing order of the starting points of their scheduling intervals, and suppose the first violation from EDF happens at time $a * 2p$, for some $a \in \mathbb{Z}^+$. Note, since jobs are scheduled always at an integral multiples of $2p$, violations can also happen only at integral multiples of $2p$. Let j_1 be scheduled at $a * 2p$ and there exists a job j_2, which is scheduled at $a' * 2p$, $a' \in \mathbb{Z}^+, a' \geq a$, but $d_{new}(j_2) < d_{new}(j_1)$. If j_2 is not released at the time j_1 starts being scheduled, then since release times are integral multiples of $2p$, j_2 is not released until time $(a + 1)2p$. But, since processing time of j_1 is at most 2p, j_1 finishes by time $(a + 1)2p$. Therefore, this does not result in a violation. Hence, j_2 must have been released at or before time $a * 2p$. In that case, we can simply swap the positions of the scheduling intervals of j_1 and j_2, that is, we schedule j_2 starting from $a * 2p$ and j_1 starting from $a' * 2p$. This schedule is clearly valid for j_2. The schedule is also valid for j_1, since $d_{new}(j_1) > d_{new}(j_2)$, $d_{new}(j_2) \geq (a' + 1) * 2p$ and $p(j) \leq 2p$. We can continue doing such swaps until there is no violation from EDF at time $a * 2p$. This swaps do not affect jobs that are scheduled before $a * 2p$, and the maximum number of swaps that may be required is bounded by $|J_s|$. Therefore, at the end, we have an EDF schedule. ◀

Lemma 15 guarantees that when the number of machines required in an optimal solution is known, EDF(OPT) gives a constant competitive online algorithm for slack jobs with almost uniform processing time. Now suppose that the number of machines in an optimal schedule is not known. In that case, we proceed as follows. We make a guess of the optimal number of machines s'; suppose we set $s' = 1$. We start with $2s'$ machines. We follow the EDF algorithm using the first set of s' machines. But, at each time t, we also check whether the jobs that are released at time t or before but have not been scheduled yet, can be scheduled in s' machines using an offline algorithm for Machine Minimization. If so, in the second set of s' machines, we allocate the jobs that are scheduled in the time window $[t, t+2p)$ in the computed offline solution. Otherwise, we consider the jobs that are released strictly before time t (hence at or before time $t-2p$) and schedule all of them in s' machines using the offline solution computed at time $t - 2p$. We next consider the jobs that are released exactly at time t. We know they can be scheduled in at most OPT machines. We run an offline algorithm taking only these jobs and if the optimal solution uses s_1 machines, we allocate them in s_1 new machines and update our guessed optimal number of machines to $s' = max(s_1, 2s')$ and proceed to time $t + 2p$. We follow the same algorithm from time $t + 2p$ with new guessed value of s'.

We need a few notations to describe the algorithm.

Let $J(t) = \{\text{jobs released before time } t\}$. Let $S(t) = \{\text{jobs scheduled before time } t\}$. Let $R(t) = \{\text{jobs released at time } t\}$. We set $s' = 1$ and $t = 0$ and call the following algorithm EDF-Online(s', t).

Algorithm 2 EDF-Online(s', t)

1: Open $2s'$ new machines;
2: Schedule $\min(|J(t) \cup R(t) \setminus S(t)|, s')$ jobs with earliest deadline from $J(t) \cup R(t) \setminus S(t)$ in the first s' machines. Let $A(t)$ be the set of jobs scheduled. {this is according to EDF}
3: **if** Offline Machine Minimization algorithm can schedule $([J(t) \cup R(t)] \setminus [S(t) \cup A(t)])$ in s' machines **then**
4: Let $B(t)$ be the set of jobs scheduled by Offline Machine Minimization at time t on at most s' machines.
5: Schedule $B(t)$ in the second set of s' machines.
6: $J(t + 2p) = J(t) \cup R(t)$, $S(t + 2p) = S(t) \cup A(t) \cup B(t)$, $t = t + 2p$, GOTO 2.
7: **else**
8: Schedule $J(t) \setminus S(t)$ using offline Machine Minimization in the already available machines–the second set of s' machines.
9: Schedule $R(t)$ in new machines using offline Machine Minimization. Let it uses s_1 new machines.
10: $J(t + 2p) = J(t) \cup R(t)$, $S(t + 2p) = J(t + 2p)$;
11: EDF-Online$(\max(2s', s_1), t + 2p)$
12: **end if**

▶ **Lemma 16.** *If there exists a schedule of jobs J_s in OPT machines where each job has release time at an integral multiple of $2p$, processing time in $[p, 2p]$ and must be scheduled at an integral multiple of $2p$,* EDF-Online *returns a feasible schedule in $O(OPT)$ machines.*

Proof. First, at step 8, offline Machine Minimization can use at most s' machines. This is because, if $t = 0$ or if at time $(t - 2p)$ the event under **Else** happened, then $J(t) \setminus S(t) = \phi$. If at time $t = 2p$, event under **if** happened, then we know at time $t - 2p$, Machine Minimization returned a feasible schedule for jobs in $([J(t - 2p) \cup R(t - 2p)] \setminus [S(t - 2p) \cup A(t - 2p)])$ in s' machines and the schedule has jobs in $B(t - 2p)$ scheduled in the interval $[t, t + 2p)$. Since $J(t) \setminus S(t) \subseteq [J(t - 2p) \cup R(t - 2p)] \setminus [S(t - 2p) \cup A(t - 2p) \cup B(t - 2p)]$, Machine Minimization will use at most s' machines to schedule them. Second, at step 9 offline Machine Minimization can use at most OPT machines. This is because here we are considering a set of jobs which are just released and therefore, they can be allocated exactly as in the optimal original offline schedule.

Once our guessed value s' becomes at least OPT, it will never be increased (follows from Lemma 16). Therefore, at the end $s' < 2OPT$. Hence, the number of machines used is at most $2(s' + s'/2 + s'/4 + ... + 1) \leq 8OPT$. ◀

3.2 Scheduling Non-slack Jobs

We now consider the non-slack jobs, that is, the length of the time window in which each of them can be scheduled is at most $8p$. We consider time in blocks of length p. We know any job that is released in between $[a * p, (a + 1) * p)$, for some $a \in \mathbb{Z}^+$, it must be scheduled by $(a+9)*p$. We now divide the jobs based on their release time into 10 classes, $P_0, P_1, \ldots, P_9$:

$P_k = \{j | \lceil r(j)/p \rceil \bmod 10 = k$. We schedule jobs of each class separately in new machines, thus blowing up the approximation factor at most by 10. For the jobs of the same class, we have the following properties.

Property 1. For any two jobs $j_1, j_2 \in P_k$, $\forall k \in [0, 9]$, if $r(j_1) < r(j_2) + p$, then $d(j_1) < r(j_2)$.

Property 2. For all jobs in P_k, $k \in [0, 9]$, which have release times in a time window not more than p, at most 9 of them can be scheduled in a single machine.

Therefore, jobs of P_k, $k \in [0, 9]$, which have release times contained in some time window of length at most p, can be scheduled on a separate machine by using at most 9 times the optimal number of machines.

Overall, by blowing up the approximation factor by a constant, we can schedule all the non-slack jobs, which altogether using Lemma 15 implies a constant factor online algorithm for Machine Minimization for jobs with processing time in $[p, 2p]$. This immediately implies a $O(\log \frac{p_{max}}{p_{min}})$ online algorithm for the general Machine Minimization problem, where the bound can be achieved by simply classifying the jobs into $\lceil \log \frac{p_{max}}{p_{min}} \rceil$ classed based on their processing times, $[p = p_{min}, 2p], (2p, 4p], ...$, and applying the algorithm separately for each class.

▶ **Theorem 17.** *There exists an online algorithm for* Machine Minimization *with constant approximation factor where all the jobs have processing time in $[p, 2p]$, for some $p > 0$.*

▶ **Corollary 18.** *There exists an online algorithm for* Machine Minimization *that achieves an approximation factor of $O(\log \frac{p_{max}}{p_{min}})$.*

3.3 Online Algorithm for Rent Minimization

Recall the classification of jobs J into J_1, J_2, J_3^1, J_3^2 from Section 2. We consider each of these collection of jobs separately. We follow the same algorithm as in Section 2 for jobs in J_1, J_2 and J_3^2. We follow the online machine minimization algorithm separately for each $J_{3,r}^1$, $r = 0, 1, ..., s$.

- $J_1 = \{j \mid p(j) \geq \frac{D}{2}\}$. These are the big jobs. *Assign new machine to each job $j \in J_1$ and keep the machine open exactly for $\lceil \frac{p(j)}{D} \rceil$ machine intervals.*
- $J_2 = \{j \mid j \notin J_1, \exists a \in \mathbb{N}, r(j) < aD, r(j) + p(j) > aD, d(j) - p(j) < aD\}$. No matter in which valid intervals we schedule these jobs, their scheduling intervals always contain the time aD. *Assign a new machine to each job $j \in J_2$ and keep the machine open exactly for 1 machine interval.*
- $J_3^1 = \{j \in J_3 \mid \exists r \in \mathbb{N}, r(j) \in H_r, d(j) \in H_r \cup H_{r+1}\}$. $J_{3,r}^1 = J_3^1 \cap \{j \mid r(j) \in H_r\}$. *Apply online algorithm for* Machine Minimization *separately to each $J_{3,r}^1$.*
- $J_3^2 = \{j \in J_3 \mid \exists s \in \mathbb{N}, r(j) \in H_r, d(j) \in H_t, t \geq r + 2\}$. *Apply Algorithm A.*

▶ **Theorem 19.** *There exists an online algorithm for* Rent Minimization *that achieves an approximation factor of $O(\log \frac{p_{max}}{p_{min}})$.*

Proof. Follows from Lemma 4, Lemma 6, Lemma 8 and Corollary 18. ◀

3.4 Lower Bounds

The following establishes the lower bound of any deterministic online algorithm for Machine Minimization.

▶ **Theorem 20.** *No deterministic online algorithm for* Machine Minimization *can achieve an approximation factor better than $\log_3 \frac{p_{max}}{p_{min}}$.*

Proof. Consider any online algorithm O. At time $t = 0$, a job j_1 is released with processing time p_{max} and deadline $3p_{max}$. If O decides to schedule it in the window $[a, a + p_{max})$, then at time $t = a$, j_2 is released with processing time $p_{max}/3$ and deadline $a + p$. Clearly, algorithm O needs to open a new machine to schedule j_2. If O decides to schedule j_2, in the window $[b, b + p_{max}/3) \subset [a, a + p_{max})$, then at time $t = b$, job j_3 arrives with processing time $p_{max}/3^2$ and deadline $b + p_{max}$. Clearly, O cannot schedule j_3 in either of the first two machines and need a third machine to schedule it. We can proceed in this fashion for $\log_3 \frac{p_{max}}{p_{min}}$ steps. Algorithm O needs to open a new machine to schedule each of these jobs, while all of them can be scheduled in a single machine in an optimal offline solution. ◀

The following corollary is immediate.

▶ **Corollary 21.** *No deterministic online algorithm for* Rent Minimization *can achieve an approximation factor better than* $\log_3 \frac{p_{max}}{p_{min}}$.

Acknowledgements. The author would like to thank Howard Karloff for asking the question of Rent Minimization, for many helpful discussions throughout the work and helping to improve the writing.

References

1 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, September 2001.

2 Amotz Bar-Noy and Sudipto Guha. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.*, pages 331–352, 2001.

3 Yair Bartal, Francis Y. L. Chin, Marek Chrobak, Stanley P. Y. Fung, Wojciech Jawor, and Ron Lavi. Online competitive algorithms for maximizing weighted throughput of unit jobs. In *In Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS*, pages 187–198. Springer, 2004.

4 Marek Chrobak, Wojciech Jawor, Jiří Sgall, and Tomáš Tichỳ. Online scheduling of equal-length jobs: Randomization and restarts help. *SIAM J. Comput.*, 36(6):1709–1728, February 2007.

5 Julia Chuzhoy and Paolo Codenotti. Resource minimization job scheduling. In *APPROX-RANDOM*, pages 70–83, 2009.

6 Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. Machine minimization for scheduling jobs with interval constraints. In *FOCS*, pages 81–90, 2004.

7 Julia Chuzhoy and Joseph (Seffi) Naor. New hardness results for congestion minimization and machine scheduling. *J. ACM*, 53(5):707–721, September 2006.

8 Julia Chuzhoy, Rafail Ostrovsky, and Yuval Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Math. Oper. Res.*, pages 730–738, 2006.

9 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

Energy Efficient Scheduling and Routing via Randomized Rounding

Evripidis Bampis[*1], Alexander Kononov[†2], Dimitrios Letsios[*1,3], Giorgio Lucarelli[*1,3], and Maxim Sviridenko[‡4]

1 **LIP6, Université Pierre et Marie Curie, France**
{Evripidis.Bampis,Giorgio.Lucarelli}@lip6.fr
2 **Sobolev Institute of Mathematics, Novosibirsk, Russia**
alvenko@math.nsc.ru
3 **IBISC, Université d' Évry, France**
dimitris.letsios@ibisc.univ-evry.fr
4 **Department of Computer Science, University of Warwick, UK**
M.I.Sviridenko@warwick.ac.uk

Abstract

We propose a unifying framework based on configuration linear programs and randomized rounding, for different energy optimization problems in the dynamic speed-scaling setting. We apply our framework to various scheduling and routing problems in heterogeneous computing and networking environments. We first consider the energy minimization problem of scheduling a set of jobs on a set of parallel speed-scalable processors in a fully heterogeneous setting. For both the preemptive-non-migratory and the preemptive-migratory variants, our approach allows us to obtain solutions of almost the same quality as for the homogeneous environment. By exploiting the result for the preemptive-non-migratory variant, we are able to improve the best known approximation ratio for the single processor non-preemptive problem. Furthermore, we show that our approach allows to obtain a constant-factor approximation algorithm for the power-aware preemptive job shop scheduling problem. Finally, we consider the min-power routing problem where we are given a network modeled by an undirected graph and a set of uniform demands that have to be routed on integral routes from their sources to their destinations so that the energy consumption is minimized. We improve the best known approximation ratio for this problem.

1998 ACM Subject Classification F.2.2 [Nonnumerical Algorithms and Problems]: Sequencing and scheduling

Keywords and phrases randomized rounding, scheduling; approximation, energy-aware; configuration linear program

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.449

1 Introduction

We focus on energy minimization problems in heterogeneous computing and networking environments in the dynamic speed-scaling setting. For many years, the exponential increase of processors' frequencies followed Moore's law. This is no more possible because of physical

* Supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010, and by the project ALGONOW under the program THALES.
† Supported by the RFBR grant No 12-01-00184.
‡ Supported by EPSRC grants EP/J021814/1, EP/D063191/1, FP7 Marie Curie Career Integration Grant and Royal Society Wolfson Research Merit Award.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 449–460

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

(thermal) constraints. Today, for improving the performance of modern computing systems, designers use parallelism, i.e., multiple cores running at lower frequencies but offering better performances than a single core. These systems can be either homogeneous where an identical core is used many times, or heterogeneous combining general-purpose and special-purpose cores. Heterogeneity offers the possibility of further improving the performance of the system by executing each job on the most appropriate type of processors [9]. However in order to exploit the opportunities offered by the heterogeneous systems, it is essential to focus on the design of new efficient power-aware algorithms taking into account the heterogeneity of these architectures. In this direction, Gupta et al. in [12] have studied the impact of the heterogeneity on the difficulty of various power-aware scheduling problems.

In this paper, we show that rounding configuration linear programs helps in handling the heterogeneity of both the jobs and the processors. We adopt one of the main mechanisms for reducing the energy consumption in modern computer systems which is based on the use of speed scalable processors. Starting from the seminal paper of Yao et al. [15], many papers adopted the speed-scaling model in which if a processor runs at speed s, then the rate of the energy consumption, i.e., the power, is $P(s) = s^{\alpha}$ with α a constant close to 3 (new studies show that α is rather smaller: 1.11 for Intel PXA 270, 1.62 for Pentium M770 and 1.66 for a TCP offload engine [14]). Moreover, the energy consumption is the integral of the power over time. This model captures the intuitive idea that the faster a processor works the more energy it consumes.

We first consider a *fully heterogeneous environment* where both, the jobs' characteristics are processor-dependent and every processor has its own power function. Formally, we consider the following problem: we are given a set $\mathcal{J}$ of n jobs and a set $\mathcal{P}$ of m parallel processors. Every processor $i \in \mathcal{P}$ obeys to a different speed-to-power function, i.e., it is associated with a different $\alpha_i \geq 1$ and hence if a job runs at speed s on processor i, then the power is $P(s) = s^{\alpha_i}$. Each job $j \in \mathcal{J}$ has a different release date $r_{i,j}$, deadline $d_{i,j}$ and workload $w_{i,j}$ if job j is executed on processor $i \in \mathcal{P}$. The goal is to find a schedule of minimum energy respecting the release dates and the deadlines of the jobs.

In this paper we propose a unifying framework for minimizing energy in different heterogeneous computing and networking environments. We first consider two variants of the heterogeneous multiprocessor *preemptive* problem. In both cases, the execution of a job may be interrupted and resumed later. In the *non-migratory* case each job has to be entirely executed on a single processor. In the *migratory* case each job may be executed by more than one processors, without allowing parallel execution of a job. We also focus on the *non-preemptive* single processor case. Furthermore, we consider the energy minimization problem in an heterogeneous *job shop* environment where the jobs can be preempted. Finally, we consider the *min-power routing problem*, introduced in [4], where a set of uniform demands have to be routed on integral routes from their sources to their destinations so that the energy consumption to be minimized. We believe that our general techniques will find further applications in energy optimization.

1.1 Related Work

Yao et al. [15] proposed an optimal algorithm for finding a feasible preemptive schedule with minimum energy consumption when a single processor is available. The homogeneous multiprocessor case has been solved optimally in polynomial time when both the preemption and the migration of jobs are allowed [2, 5, 7, 8]. Albers et al. [3] considered the homogeneous multiprocessor preemptive problem, where the migration of the jobs is not allowed. They proved that the problem is $\mathcal{NP}$-hard even for instances with common release dates and

common deadlines. Greiner et al. [10] gave a generic reduction transforming an optimal schedule for the homogeneous multiprocessor problem with migration, to a $B_{\lceil\alpha\rceil}$-approximate solution for the homogeneous multiprocessor preemptive problem without migration, where $B_{\lceil\alpha\rceil}$ is the $\lceil\alpha\rceil$-th Bell number. Antoniadis and Huang [6] proved that the single processor non-preemptive problem is $\mathcal{NP}$-hard even for instances in which for any two jobs j and j' with $r_j \leq r_{j'}$ it holds that $d_j \geq d_{j'}$. They also proposed a $2^{5\alpha-4}$-approximation algorithm for general instances. Andrews et al. [4] studied the min-power routing problem and for uniform demands, i.e. for the case where all the demands have the same value, they proposed a γ-approximation algorithm, where $\gamma = \max\{1+\tau 2^{\alpha(\tau+1)\log e}, 2+\tau 2^{\alpha(\tau+1)}\}$, with $\tau = \lceil 2\log(\alpha+4)\rceil$. For non-uniform demands, they proposed a $O(\log^{\alpha-1} D)$-approximation algorithm, where D is the maximum value of the demands. For further results see [1].

1.2 Notation

Given a schedule $\mathcal{S}$ we denote by $E(\mathcal{S})$ the total energy consumed by $\mathcal{S}$. We denote by $\mathcal{S}^*$ an optimal schedule and by OPT the energy consumption of $\mathcal{S}^*$. For each job $j \in \mathcal{J}$, we say that j is *alive* on processor $i \in \mathcal{P}$ during the interval $[r_{i,j}, d_{i,j}]$. Let $\alpha = \max_{i\in\mathcal{P}}\{\alpha_i\}$. The Bell number, B_n, is defined for any integer $n \geq 0$ and corresponds to the number of partitions of a set of n items. It is well known that Bell numbers satisfy the following equality

$$B_n = \sum_{k=0}^{\infty} \frac{k^n e^{-1}}{k!}$$

known as Dobinski's formula. Another way to state this formula is that n-th Bell number is equal to the n-th moment of Poisson random variable with parameter (expected value) 1. That naturally leads to a more general definition. The generalized Bell number, denoted by $\tilde{B}_\alpha = \sum_{k=0}^{\infty} \frac{k^\alpha e^{-1}}{k!}$, is defined for any $\alpha \in \mathbb{R}^+$ and corresponds to the α-th (fractional) moment of Poisson random variable with parameter 1. Note that the ratios of our algorithms depend on the generalized Bell number. Due to space constraints, some results and proofs will be given in the full version of the paper.

1.3 Our Contribution

In this paper we formulate heterogeneous scheduling and routing problems using *configuration linear programs (LPs)* and we apply *randomized rounding*. In Section 3, we consider the heterogeneous multiprocessor speed-scaling problem without migrations and we propose an approximation algorithm of ratio $(1+\varepsilon)\tilde{B}_\alpha$. As this LP has an exponential number of variables, we give an alternative (compact) formulation of the problem using a polynomial number of variables and we prove the equivalence between the two LP relaxations. For real values of α our result improves the $B_{\lceil\alpha\rceil}$ approximation ratio of [10] for the homogeneous case to $(1+\varepsilon)\tilde{B}_\alpha$ for the fully heterogeneous environment that we consider here (see Table 1). In Section 4, using again a configuration LP formulation, we present an algorithm for the heterogeneous multiprocessor speed-scaling problem with migration. This algorithm returns a solution within an additive factor of ε far from the optimal solution and runs in time polynomial to the size of the instance and to $1/\varepsilon$. This result generalizes the results of [2, 5, 7, 8] from an homogeneous environment to a fully heterogeneous environment. In Section 5, we transform the single processor speed-scaling problem without preemptions to the heterogeneous multiprocessor problem without migrations and we give an approximation algorithm of ratio $2^{\alpha-1}(1+\varepsilon)\tilde{B}_\alpha$, improving upon the previous known $2^{5\alpha-4}$-approximation algorithm in [6] for any $\alpha < 114$ (see Table 1). In Section 6, we study the power-aware

Table 1 Comparison of our approximation ratios vs. better previous known ratios for: (i) the preemptive multiprocessor problem without migrations, (ii) the single processor non-preemptive problem, and (iii) the min-power routing problem.

Value of α	Preemptive without migrations		Non-preemptive single processor		Routing uniform demands	
	Homogeneous [10]	Heterogeneous This paper	[6]	This paper	[4]	This paper
1.11	2	$1.07(1+\varepsilon)$	2.93	$1.15(1+\varepsilon)$	375	1.07
1.62	2	$1.49(1+\varepsilon)$	17.15	$2.30(1+\varepsilon)$	2196	1.49
1.66	2	$1.54(1+\varepsilon)$	19.70	$2.43(1+\varepsilon)$	2522	1.54
2	2	$2(1+\varepsilon)$	64	$4(1+\varepsilon)$	8193	2
2.5	5	$3.08(1+\varepsilon)$	362	$8.72(1+\varepsilon)$	46342	3.08
3	5	$5(1+\varepsilon)$	2048	$20(1+\varepsilon)$	262145	5

preemptive job shop scheduling problem and we propose a $((1+\varepsilon)\tilde{B}_\alpha)$-approximation algorithm, where μ is the number of all the operations. Finally, in Section 7, we improve the analysis for the min-power routing problem with uniform demands given in [4], based on the randomized rounding analysis that we propose in this paper. Our approach gives an approximation ratio of $\tilde{B}_\alpha$ significantly improving the analysis given in [4] (see Table 1).

2 Technical Probabilistic Propositions

▶ **Proposition 1.** *Consider a set of real numbers $\{Y_1, Y_2, \ldots, Y_n\}$ such that $Y_i \in [0,1]$ for all $i \in \{1,\ldots,n\}$ and a set of non-negative constants $\{e_1, e_2, \ldots, e_n\}$. Assume that we split Y_n to $Y'_n \geq 0$ and $Y'_{n+1} \geq 0$ such that $Y_n = Y'_n + Y'_{n+1}$. Let $e_{n+1} = e_n$ and $Y'_j = Y_j$, $j \in \{1,2,\ldots,n-1\}$. It holds that*

$$\sum_{S\subseteq\{1,2,\ldots,n\}} |S|^{\alpha-1}\left(\sum_{j\in S} e_j\right)\prod_{j\in S} Y_j \prod_{j\notin S}(1-Y_j) \leq \sum_{S\subseteq\{1,2,\ldots,n+1\}} |S|^{\alpha-1}\left(\sum_{j\in S} e_j\right)\prod_{j\in S} Y'_j \prod_{j\notin S}(1-Y'_j)$$

▶ **Proposition 2.** *For any $\alpha \geq 1$, the function $f(x) = x^\alpha$ and parameter $a \in [0,1]$ we have*

$$\mathbb{E}[f(B_a)] \leq \mathbb{E}[f(P_a)]$$

where B_a is a sum of n independent Bernoulli random variables, $\mathbb{E}[B_a] = a$ and P_a is a Poisson random variable with parameter a.

▶ **Proposition 3.** *For any real $\alpha \geq 1$ and a Poisson random variable P_λ with parameter $\lambda \geq 0$, we have:*
(a) *If $0 \leq \lambda \leq 1$, then $\mathbb{E}[P_\lambda^\alpha] \leq \lambda \mathbb{E}[P_1^\alpha]$.*
(b) *If $\lambda > 1$, then $\mathbb{E}[P_\lambda^\alpha] \leq \lambda^\alpha \mathbb{E}[P_1^\alpha]$.*

3 Heterogeneous Multiprocessor without Migrations

In this section we consider the case where the migration of jobs is not permitted, but their preemption is allowed. The corresponding homogeneous problem is known to be $\mathcal{NP}$-hard even if all jobs have common release dates and deadlines [3]. We propose an approximation algorithm by formulating the problem as a configuration integer program (IP) with an

exponential number of variables and a polynomial number of constraints. Given an optimal solution for the configuration LP relaxation, we apply randomized rounding to get a feasible schedule for our problem. In order to get a polynomial-time algorithm, we can give another (compact) formulation of our problem with a polynomial number of variables and constraints and we can show that the relaxations of the two formulations are equivalent.

In order to formulate our problem as a configuration IP we need to discretize the time. In the following lemma we assume that the release dates and the deadlines of all jobs in all processors are integers.

▶ **Lemma 4.** *There is a feasible schedule with energy consumption at most* $((1+\frac{\varepsilon}{1-\varepsilon})(1+\frac{2}{n-2}))^{\alpha} \cdot OPT$ *in which each piece of each job* $j \in \mathcal{J}$ *(*j *is executed on processor* $i \in \mathcal{P}$*) starts and ends at a time point* $r_{i,j} + k\frac{\varepsilon}{n^3}(d_{i,j} - r_{i,j})$*, where* $k \geq 0$ *is an integer.*

Let $\mathcal{S}$ be a schedule that satisfies Lemma 4 and let $j \in \mathcal{J}$ be a job executed on the processor $i \in \mathcal{P}$ in $\mathcal{S}$. The above lemma implies that the interval $(r_{i,j}, d_{i,j}]$ can be partitioned into polynomial, with respect to n and $1/\varepsilon$, number of equal length slots. In each of these slots, j either is executed during the whole slot or is not executed at all. In what follows we consider schedules that satisfy Lemma 4.

3.1 Linear Programming Relaxation

A configuration c is a schedule for a single job on a single processor. Specifically, a configuration determines the slots, with respect to Lemma 4, during which one job is executed. Given a configuration c for a job $j \in \mathcal{J}$, we can define the execution time of j that is equal to the number of slots in c multiplied by the length of the slot. Due to the convexity of the speed-to-power function, in a minimum energy schedule that satisfies Lemma 4, the job j runs at a constant speed s_j. Hence, s_j is equal to the work of j over its execution time. Let $\mathcal{C}_{ij}$ be the set of all possible feasible configurations for all jobs in all processors.

In order to ensure the feasibility of our schedule we need to further partition the time, by merging the slots for all jobs. Given a processor $i \in \mathcal{P}$, consider the time points of all jobs of the form $r_{i,j} + k\frac{\varepsilon}{n^3}(d_{i,j} - r_{i,j})$ as introduced in Lemma 4. Let $t_{i,1}, t_{i,2}, \ldots, t_{i,\ell_i}$ be the ordered sequence of these time points. Consider now the intervals $(t_{i,p}, t_{i,p+1}]$, $1 \leq p \leq \ell_i - 1$. In a schedule that satisfies Lemma 4, in each such interval either there is exactly one job that is executed during the whole interval or the interval is idle. Note also that these intervals might not have the same length. Let $\mathcal{I}$ be the set of all these intervals for all processors.

We introduce the binary variable $x_{i,j,c}$ that is equal to one if the job $j \in \mathcal{J}$ is entirely executed on the processor $i \in \mathcal{P}$ according to the configuration c, and zero otherwise. Note that, given the configuration and the processor i where the job j is executed, we can compute the energy consumption $E_{i,j,c}$ for the execution of j. For ease of notation, we say $I \in (i,j,c)$ if the interval $I \in \mathcal{I}$ is included in the configuration c of processor $i \in \mathcal{P}$ for the job $j \in \mathcal{J}$, that is there is a slot $(r_{i,j} + k\frac{\varepsilon}{n^3}(d_{i,j} - r_{i,j}), r_{i,j} + (k+1)\frac{\varepsilon}{n^3}(d_{i,j} - r_{i,j})]$ in c that contains I.

$$\min \sum_{i,j,c} E_{i,j,c} \cdot x_{i,j,c}$$

$$\sum_{i,c} x_{i,j,c} \geq 1 \qquad \forall j \in \mathcal{J} \tag{1}$$

$$\sum_{(i,j,c): I \in (i,j,c)} x_{i,j,c} \leq 1 \qquad \forall I \in \mathcal{I} \tag{2}$$

$$x_{i,j,c} \in \{0,1\} \qquad \forall i \in \mathcal{P}, j \in \mathcal{J}, c \in \mathcal{C}_{ij} \tag{3}$$

Inequality (1) enforces that each job is entirely executed according to exactly one configuration. Inequality (2) ensures that at most one job is executed in each interval $(t_{i,p}, t_{i,p+1}]$, $1 \leq p \leq \ell_i - 1$. We next relax the constraints (3) such that $x_{i,j,c} \geq 0$. Since the structure of this LP is quite simple we can define an equivalent compact LP relaxation with polynomial number of constraints and variables. We describe how to do it in the full version of the paper. For now we assume that we can find an optimal solution of our configuration LP in polynomial time.

3.2 Randomized Rounding

In this section, we show how to apply randomized rounding to get an approximation algorithm for our problem. Our algorithm follows.

Algorithm 1

1: Solve the configuration LP relaxation.
2: For each job $j \in \mathcal{J}$, choose a configuration at random with probability $x_{i,j,c}$.
3: Let K_I be the number of configurations that contain the interval I. Scale the speeds during I by a factor of K_I.

▶ **Theorem 5.** *Assume that $\alpha_i \geq 1$ for all $i = 1, \ldots, m$. Algorithm 1 achieves an approximation ratio of $((1 + \frac{\varepsilon}{1-\varepsilon})(1 + \frac{2}{n-2}))^{\alpha} \tilde{B}_{\alpha}$ for the heterogeneous multiprocessor preemptive speed-scaling problem without migrations in time polynomial to n and to $1/\varepsilon$, where $\alpha = \max_{i \in \mathcal{P}} \alpha_i$.*

Proof. For each interval $I \in \mathcal{I}$, we estimate its expected energy consumption. By definition, an interval corresponds to a single processor $i \in \mathcal{P}$. Given a job $j \in \mathcal{J}$, let n_j be the number of the non-zero $x_{i,j,c}$ variables such that I belongs to configuration c. Moreover, let $X_{j,k}$ be the k-th, $1 \leq k \leq n_j$, of the above non-zero variables and $s_{j,k}$ be the corresponding speed.

Let Y_j be the probability that the job j is assigned to be processed in the interval I by the randomized rounding procedure, that is $Y_j = \sum_{k=1}^{n_j} X_{j,k}$. By the constraint (2), we know that $\sum_{j=1}^{n} Y_j \leq 1$. The expected energy that the job j consumes on the interval I under the condition that j is assigned to be processed in the interval I without considering the other jobs is $e_j = \frac{\sum_{k=1}^{n_j} |I| s_{j,k}^{\alpha_i} X_{j,k}}{Y_j}$.

The energy consumption in the interval I achieved by the optimal solution of the LP relaxation is $LP^* = \sum_{j=1}^{n} e_j Y_j$. If the randomized rounding assigns a set S of jobs to be processed during the interval I then we need to speed up the execution of all jobs in the intervals I by factor $|S|$. This means that the energy consumption increases by the factor $|S|^{\alpha_i - 1}$. Therefore, the expected energy consumption during the interval I in the final approximate schedule is

$$E_I = \sum_{S \subseteq \{1,2,\ldots,n\}} |S|^{\alpha_i - 1} \left(\sum_{j \in S} e_j \right) P_S(Y'_1, Y'_2, \ldots, Y'_n)$$

where $P_S(Y'_1, Y'_2, \ldots, Y'_n)$ is the probability that exactly the jobs in the set S are selected during I, and $Y'_1, Y'_2, \ldots, Y'_n$ are independent Bernoulli random variables with $Pr(Y'_j = 1) = Y_j$. Therefore, we have that

$$E_I = \sum_{S \subseteq \{1,2,\ldots,n\}} |S|^{\alpha_i - 1} \left(\sum_{j \in S} e_j \right) \prod_{j \in S} Y_j \prod_{j \in \mathcal{J} \setminus S} (1 - Y_j)$$

We can assume that there exists $Q \in \mathbb{N}$ such that $Y_j = \frac{q_j}{Q}$, $1 \leq j \leq n$, for some $q_j \in \mathbb{N}$ (we don't make any assumptions on the encoding length of these numbers, we use them only for analysis purposes) since these numbers come from solving an LP with rational coefficients. Hence we can chop each Y_j into q_j pieces $Y_{j,\ell} = \frac{1}{Q} = Y$. Let $q = \sum_{j=1}^{n} q_j$ be the number of all chopped pieces and $e_{j,\ell} = e_j$, $1 \leq j \leq n$ and $1 \leq \ell \leq q_j$. Note that, $q \leq Q$ since $\sum_{j=1}^{n} Y_j \leq 1$. For the ease of exposition we identify the set $\{1, 2, \ldots, q\}$ with the set of all pairs (j, ℓ) such that $1 \leq j \leq n$ and $1 \leq \ell \leq q_j$. Using Proposition 1 we get

$$\begin{aligned} E_I &\leq \sum_{S \subseteq \{1,2,\ldots,q\}} |S|^{\alpha_i - 1} \left(\sum_{(j,\ell) \in S} e_{j,\ell} \right) Y^{|S|} (1-Y)^{q-|S|} \\ &= \sum_{k=1}^{q} \sum_{S \subseteq \{1,2,\ldots,q\}, |S|=k} \left(\sum_{(j,\ell) \in S} e_{j,\ell} \right) k^{\alpha_i - 1} Y^k (1-Y)^{q-k} \end{aligned}$$

By changing the order of the sums in the above inequality we get

$$\begin{aligned} E_I &\leq \left(\sum_{j=1}^{n} \sum_{\ell=1}^{q_j} e_{j,\ell} \right) \sum_{k=1}^{q} \binom{q-1}{k-1} k^{\alpha_i - 1} Y^k (1-Y)^{q-k} \\ &= \left(\sum_{j=1}^{n} q_j e_j \right) \sum_{k=1}^{q} \binom{q}{k} \frac{\binom{q-1}{k-1}}{\binom{q}{k}} k^{\alpha_i - 1} Y^k (1-Y)^{q-k} \\ &= \left(\frac{\sum_{j=1}^{n} q_j e_j}{q} \right) \sum_{k=1}^{q} \binom{q}{k} k^{\alpha_i} Y^k (1-Y)^{q-k} = \frac{Q}{q} LP_I^* \sum_{k=1}^{q} \binom{q}{k} k^{\alpha_i} Y^k (1-Y)^{q-k} \\ &\leq \frac{Q}{q} LP_I^* \mathbb{E}[B_{q/Q}^{\alpha_i}] \end{aligned}$$

where $\binom{q-1}{k-1}$ is the number of sets of cardinality k that contain j. Moreover, $B_{q/Q}$ is a random variable with expectation $\frac{q}{Q}$ which corresponds to the sum of q independent Bernoulli random variables. Therefore,

$$E_I \leq \frac{Q}{q} LP_I^* \cdot \mathbb{E}[B_{q/Q}^{\alpha_i}] \leq \frac{Q}{q} LP_I^* \cdot \mathbb{E}[P_{q/Q}^{\alpha_i}] \leq \frac{Q}{q} LP_I^* \cdot \frac{q}{Q} \mathbb{E}[P_1^{\alpha_i}]$$

where the second inequality follows from Proposition 2 and the last inequality follows from Proposition 3(a). Therefore, by summing over all intervals and processors and as $\alpha = \max_{i \in \mathcal{P}} \alpha_i$, we get

$$E \leq LP^* \cdot \mathbb{E}[P_1^{\alpha}] = LP^* \cdot \tilde{B}_{\alpha}$$

and then the theorem follows. ◀

4 Heterogeneous Multiprocessor with Migrations

In this section we present an algorithm for the heterogeneous multiprocessor speed-scaling problem with preemptions and migrations. We assume that, if x units of work for the job j are executed on the processor i, then $x/w_{i,j}$ portion of j is accomplished by i. We first formulate the problem as a configuration LP with an exponential number of variables and a polynomial number of constraints. Then, we consider the dual LP and we show how to apply the Ellipsoid algorithm to it and obtain an $OPT + \varepsilon$ solution for the primal LP.

A configuration c is a one-to-one assignment of n_c, $0 \leq n_c \leq m$, jobs to the m processors as well as an assignment of a speed value for every processor. We denote by $\mathcal{C}$ the set of all possible configurations. A well defined schedule for our problem has to specify exactly one configuration at each time t. The cardinality of $\mathcal{C}$ is unbounded, since the processors' speeds may be real values. Hence, we have to discretize the possible speed values and consider only a finite number of speeds at which the processors can run.

▶ **Lemma 6.** *There is a feasible schedule of energy consumption at most $OPT + \varepsilon$ that uses a finite (exponential to the size of the instance and polynomial to $1/\varepsilon$) number of discrete processors' speeds.*

In what follows in this section, we deal with schedules that satisfy Lemma 6. Let, now, $t_0 < t_1 < \ldots < t_\ell$ be the time instants that correspond to release dates and deadlines of jobs so that there is a time t_i for every possible release date and deadline. We denote by $\mathcal{I}$ the set of all possible intervals of the form $(t_{i-1}, t_i]$, for $1 \leq i \leq \ell$. Let $|I|$ be the length of the interval I.

We introduce a variable $x_{I,c}$, for each $I \in \mathcal{I}$ and $c \in \mathcal{C}$, which corresponds to the total processing time during the interval $I \in \mathcal{I}$ that the processors run according to the configuration $c \in \mathcal{C}$. We denote by $E_{I,c}$ the instantaneous energy consumption of the processors if they run with respect to the configuration c during the interval I. Moreover, let $s_{j,c}$ be the speed of the job j according to the configuration c. For notational convenience, we denote by (I, c) the set of jobs which are alive during the interval I and which are executed on some processor by the configuration c. Finally, let $i(j, c)$ be the processor on which the job j is assigned into configuration c. We propose the following configuration LP:

$$\min \sum_{I \in \mathcal{I}, c \in \mathcal{C}} E_{I,c} \cdot x_{I,c}$$

$$\sum_{c \in \mathcal{C}} x_{I,c} \leq |I| \qquad \forall I \in \mathcal{I} \tag{4}$$

$$\sum_{I,c:\ j \in (I,c)} \frac{s_{j,c}}{w_{i(j,c),j}} x_{I,c} \geq 1 \qquad \forall j \in \mathcal{J} \tag{5}$$

$$x_{I,c} \geq 0 \qquad \forall I \in \mathcal{I},\ c \in \mathcal{C}$$

Consider the schedule for the interval I that occurs by an arbitrary order of the configurations assigned to I. This schedule is feasible, as the processing time of all configurations assigned to I is equal to the length of the interval. Hence, Inequality (4) ensures that for each interval I there is exactly one configuration for each time $t \in I$. Inequality (5) implies that each job j is entirely executed.

The above LP has an exponential number of variables. In order to handle this, we create the dual LP, which has an exponential number of constraints. In the full version of the paper we will show how to efficiently apply the Ellipsoid algorithm to it (see [11]). For this, we provide a separation oracle, i.e., we give a polynomial-time algorithm which given a solution for the dual LP decides if this solution is feasible or otherwise it identifies a violated constraint. As we can compute an optimal solution for the dual LP, we can also find an optimal solution for the primal LP by solving it with the variables corresponding to the constraints that were found to be violated during the run of the ellipsoid method and setting all other primal variables to be zero. The number of these violated constraints is polynomial to the size of the instance and to $1/\varepsilon$. Thus, we can solve the primal LP with a polynomial number of variables.

▶ **Theorem 7.** *A schedule for the heterogeneous multiprocessor speed-scaling problem with migrations of energy consumption $OPT + \varepsilon$ can be found in polynomial time with respect to the size of the instance and to $1/\varepsilon$.*

5 Single processor without Preemptions

In this section we present an approximation algorithm for the single processor speed-scaling problem when the preemption of jobs is not allowed. As a single processor is available, each job $j \in \mathcal{J}$ has a unique release date r_j, deadline d_j and amount of work w_j, while when the processor runs at speed s, it consumes energy with rate s^α. Due to the convexity of the speed-to-power function, j runs at a constant speed s_j in an optimal schedule $\mathcal{S}^*$. Antoniadis and Huang [6] proved that this problem is $\mathcal{NP}$-hard and gave a $2^{5\alpha-4}$-approximation algorithm.

The algorithm in [6] consists of a series of transformations of the initial instance. Our algorithm applies the first of these transformations. Then, we give a transformation to the heterogeneous multiprocessor speed-scaling problem without migrations. For completeness, we describe the first transformation given in [6]. We partition the time as follows: let t_1 be the smallest deadline of any job in $\mathcal{J}$, i.e., $t_1 = \min\{d_j : j \in \mathcal{J}\}$. Let $\mathcal{J}_1 \subseteq \mathcal{J}$ be the set of jobs which are released before t_1, i.e., $\mathcal{J}_1 = \{j \in \mathcal{J} : r_j \leq t_1\}$. Next, we set $t_2 = \min\{d_j : j \in \mathcal{J} \setminus \mathcal{J}_1\}$ and $\mathcal{J}_2 = \{j \in \mathcal{J} : t_1 < r_j \leq t_2\}$, and we continue this procedure until all jobs are assigned into a subset of jobs. Let k be the number of subsets of jobs that have been created. Moreover, let $t_0 = \min\{r_j : j \in \mathcal{J}\}$ and $t_{k+1} = \max\{d_j : j \in \mathcal{J}\}$.

Consider the intervals $(t_{i-1}, t_i]$, $1 \leq i \leq k+1$. Let $\mathcal{I}_j$ be the set of intervals in which the job $j \in \mathcal{J}$ is alive. In some of them j is alive during the whole interval, while in at most two of them it is alive during a part of the interval. Consider now the non-preemptive problem in which the execution of j should take place into exactly one interval $I \in \mathcal{I}_j$. Note that the execution of j should respect its release date and its deadline.

▶ **Proposition 8.** *Let $\mathcal{S}$ be an optimal non-preemptive schedule for the problem in which the execution of each job $j \in \mathcal{J}$ should take place into exactly one interval $I \in \mathcal{I}_j$. It holds that $E(\mathcal{S}) \leq 2^{\alpha-1} OPT$.*

Next, we describe how to pass from the transformed problem to the heterogeneous multiprocessor speed-scaling problem without migrations. For every interval $(t_{i-1}, t_i]$, $1 \leq i \leq k+1$, we create a processor i. For every job $j \in \mathcal{J}$ which is alive during a part or during the whole interval $(t_{i-1}, t_i]$, $1 \leq i \leq k+1$, we set: (i) $r_{i,j} = 0$ if $r_j \leq t_{i-1}$ or $r_{i,j} = r_j - t_{i-1}$ if $r_j > t_{i-1}$, (ii) $d_{i,j} = t_i - t_{i-1}$ if $d_j > t_i$ or $r_{i,j} = d_j - t_{i-1}$ if $d_j \leq t_i$, and (iii) $w_{i,j} = w_j$. For each processor i, $1 \leq i \leq k+1$, we set $\alpha_i = \alpha$.

We next apply the approximation algorithm presented in Section 3. This algorithm will create a preemptive schedule $\mathcal{S}$. However, we can transform S into a non-preemptive schedule $\mathcal{S}'$ of the same energy consumption. To see this, note that in each processor i, $1 \leq i \leq k+1$, each job $j \in \mathcal{J}$ has $r_{i,j} = 0$ or $d_{i,j} = t_i - t_{i-1}$. Hence, by applying the Earliest Deadline First policy to each processor separately we can get the non-preemptive schedule $\mathcal{S}'$.

▶ **Theorem 9.** *The single processor speed-scaling problem without preemptions can be approximated within a factor of $2^{\alpha-1}((1 + \frac{\varepsilon}{1-\varepsilon})(1 + \frac{2}{n-2}))^\alpha \tilde{B}_\alpha$.*

6 Job Shop Scheduling with Preemptions

In this section, we consider the energy minimization problem in a job shop environment. The instance of the problem consists of a set of jobs $\mathcal{J}$, where each job $j \in \mathcal{J}$ consists of μ_j

operations $O_{j,1}, O_{j,2}, \ldots, O_{j,\mu_j}$, which must be executed in this order. That is, $O_{k+1,j}$ can start only once the operation $O_{j,k}$ has finished. Let μ be the number of all the operations, i.e. $\mu = \sum_{j \in \mathcal{J}} \mu_j$. Each operation $O_{j,k}$ has an amount of work $w_{j,k}$. Moreover, we are given a set of m heterogeneous processors $\mathcal{P}$. Each operation $O_{j,k}$, $j \in \mathcal{J}$ and $1 \leq k \leq \mu_j$, is associated with a single processor $i \in \mathcal{P}$ on which it must be entirely executed. Note that more than one operations of the same job may have to be executed on the same processor. Furthermore, for each operation $O_{j,k}$, we are given a release date $r_{j,k}$ and a deadline $d_{j,k}$. For each $j \in \mathcal{J}$, we can assume that $r_{j,1} \leq r_{j,2} \leq \ldots \leq r_{j,\mu_j}$ as well as $d_{j,1} \leq d_{j,2} \leq \ldots \leq d_{j,\mu_j}$. Preemptions of operations are allowed. The objective is to find a feasible schedule of minimum energy consumption.

We propose a configuration IP for the job shop problem. A configuration is a schedule for a job, i.e., a schedule for all its operations. To define the configurations, we discretize the time into a number of length slots, which is polynomial to the size of the instance and to $1/\varepsilon$. As the configuration LP relaxation has an exponential number of variables and a polynomial number of constraints, we consider its dual and we propose a separation oracle for it. Thus, we can solve our problem by applying the Ellipsoid algorithm. Then we apply the same randomized rounding as in Section 3.2 and the following theorem holds (we will give a formal proof in the full version).

▶ **Theorem 10.** *There is an algorithm of complexity polynomial to μ and to $1/\varepsilon$ with approximation ratio $(1+\varepsilon)^\alpha (1+\frac{2}{\mu-2})^\alpha (1+\frac{\varepsilon}{1-\varepsilon})^\alpha \tilde{B}_\alpha$ for the preemptive job shop scheduling problem with the energy objective.*

7 Routing

We are given a directed graph $G = (V, E)$. We are also given a set of demands $\mathcal{D}$. The demand $i \in \mathcal{D}$ is associated with a source node s_i and a destination node t_i and it requests d_i integer units of bandwidth. We consider the special case where all the demands request the same bandwidth, i.e. $d_i = d$ for all $i \in \mathcal{D}$. Each edge $e \in E$ is associated with a constant α_e such that if f units of demand cross e, then there is an energy consumption equal to $c_e f^{\alpha_e}$. The objective is to route all the demands from their sources to their destinations so that the total energy consumption is minimized. We consider the unsplittable version of the problem where each demand has to be routed through a single path.

The min-power routing problem can be formulated as an integer convex program (see [4]). Specifically, we introduce a variable x_e, for all $e \in E$, which corresponds to the number of demands that cross the edge e and a binary variable $y_{i,e}$ which indicates if the demand $i \in \mathcal{D}$ crosses the edge e. Then, we obtain the following integer convex program suggested in [4].

$$\min \sum_{e \in E} c_e d^{\alpha_e} \max\{x_e, x_e^{\alpha_e}\}$$

$$x_e = \sum_i y_{i,e} \qquad \forall e \in E \tag{6}$$

$$\sum_{e \in \Gamma^+(u)} y_{i,e} - \sum_{e \in \Gamma^-(u)} y_{i,e} = 0 \qquad \forall i \in \mathcal{D}, u \in V \setminus \{s_i, t_i\} \tag{7}$$

$$\sum_{e \in \Gamma^+(s_i)} y_{i,e} = 1 \qquad \forall i \in \mathcal{D} \tag{8}$$

$$\sum_{e \in \Gamma^-(t_i)} y_{i,e} = 1 \qquad \forall i \in \mathcal{D} \tag{9}$$

$$y_{i,e} \in \{0,1\} \qquad \forall i \in \mathcal{D}, e \in E \tag{10}$$

The above integer convex program is a valid formulation for our problem. Note first that our original goal is to minimize the total energy consumption for all edges, i.e., $\sum_{e\in E} c_e d^{\alpha_e} x_e^{\alpha_e}$. Since in an optimal integral solution using this objective all variables x_e are integers, the above program has the same optimal integral solution as if we have used as objective the $\sum_{e\in E} c_e d^{\alpha_e} x_e^{\alpha_e}$. However, the use of this objective leads to an integer program with large integrality gap [4]. For this reason, we modify the objective to be $\sum_{e\in E} c_e d^{\alpha_e} \max\{x_e, x_e^{\alpha_e}\}$ obtaining a program with smaller integrality gap. Equation (6) relates the variables x_e and $y_{i,e}$, while Equations (7)-(9) ensure the flow conservation.

In order to obtain a feasible integral solution for our problem, we solve the relaxation of the above convex program, where the constraints $y_{i,e} \in \{0,1\}$ are relaxed so that $y_{i,e} \geq 0$, and we obtain a fractional solution. Then, we apply a randomized rounding procedure, introduced by Raghavan and Thompson [13], in order to select a path for each demand. Specifically, for each demand $i \in \mathcal{D}$, we consider the subgraph of G that contains only the edges with $y_{i,e} > 0$ and define the standard flow decomposition. We compute a (s_i, t_i)-path p on this graph and we set $z_{i,p} = \min_{e\in p}\{y_{i,e}\}$. Then, we subtract $z_{i,p}$ from the variables $y_{i,e}$ which correspond to the edges of the path p. We continue this procedure until there are no (s_i, t_i)-paths. Due to the flow conservation, at this point there are no edges with $y_{i,e} > 0$.

The randomized rounding algorithm chooses a path p for the demand i with probability $z_{i,p}$. Note that $\sum_p z_{i,p} = 1$.

▶ **Theorem 11.** *There is a $\tilde{B}_\alpha$-approximation algorithm for the min-power routing problem with uniform demands.*

Proof. Consider an edge $e \in E$ and let $\lambda_e = \sum_{i\in\mathcal{D}} y_{i,e}$ be the expected value of the number of demands that cross e. The expected energy consumption on the edge e is

$$E_e = c_e d^{\alpha_e} \sum_{S\subseteq\mathcal{D}} |S|^{\alpha_e} Pr(S)$$

where $Pr(S)$ is the probability that exactly the demands in S are routed through (cross) the edge e. Hence, we have

$$E_e = c_e d^{\alpha_e} \sum_{S\subseteq\mathcal{D}} |S|^{\alpha_e} \prod_{i\in S} y_{i,e} \prod_{i\notin S}(1 - y_{i,e}).$$

Since $y_{i,e}$ come from a mathematical programming solver we can assume that that there exist $N \in \mathbb{N}$ such that $y_{ie} = \lambda_e \cdot \frac{q_{i,e}}{N}$ for some $q_{i,e} \in \mathbb{N}$. Similarly with the proof of Theorem 5, we can chop each $y_{i,e}$ into $q_{i,e}$ pieces $z_{i,e,\ell} = \frac{\lambda_e}{N}$. Note that, $N = \sum_{i\in\mathcal{D}} q_{i,e}$ since $\sum_{i\in\mathcal{D}} \frac{y_{i,e}}{\lambda_e} = 1$. For the ease of exposition we identify the set $\{1, 2, \ldots, N\}$ with the set of all pairs $((i, e), \ell)$ such that $i \in \mathcal{D}$ and $1 \leq \ell \leq q_{i,e}$. Iteratively applying Proposition 1 we get

$$\begin{aligned}
E_e &\leq c_e d^{\alpha_e} \sum_{S\subseteq\{1,2,\ldots,N\}} |S|^{\alpha_e} \left(\frac{\lambda_e}{N}\right)^{|S|} \left(1 - \frac{\lambda_e}{N}\right)^{N-|S|} \\
&= c_e d^{\alpha_e} \sum_{k=0}^{N} \sum_{S\in\{1,2,\ldots,N\},|S|=k} k^{\alpha_e} \left(\frac{\lambda_e}{N}\right)^{k} \left(1 - \frac{\lambda_e}{N}\right)^{N-k} \\
&= c_e d^{\alpha_e} \sum_{k=0}^{N} k^{\alpha_e} \binom{N}{k} \left(\frac{\lambda_e}{N}\right)^{k} \left(1 - \frac{\lambda_e}{N}\right)^{N-k}
\end{aligned}$$

as there are $\binom{N}{k}$ subsets of $\{1, 2, \ldots, N\}$ with k elements. The sum in the last expression is the α_e-th moment of Binomial random variable (sum of independent Bernoulli trials) with

expectation λ_e and hence by using Propositions 2 and 3 we get

$$E_e \leq c_e d^{\alpha_e} \mathbb{E}[P_{\lambda_e}^{\alpha_e}] \leq c_e d^{\alpha_e} \max\{\lambda_e, \lambda_e^{\alpha_e}\} \mathbb{E}[P_1^{\alpha_e}] = LP_e^* \tilde{B}_{\alpha_e}$$

where P_{λ_e} is a Poisson random variable with parameter λ_e. By summing up over all edges and setting $\alpha = \max_{e \in E}\{\alpha_e\}$, the theorem follows. ◀

In Table 1, we show that our analysis for the algorithm presented in [4] leads to a significantly better approximation ratio.

Acknowledgements. We would like to thank Oleg Pikhurko for providing the original proof of Part (b) of the Proposition 3.

References

1 S. Albers. Algorithms for dynamic speed scaling. In *STACS*, LIPIcs, Vol. 9, pages 1–11, 2011. doi: 10.4230/LIPIcs.STACS.2011.1

2 S. Albers, A. Antoniadis, and G. Greiner. On multi-processor speed scaling with migration: extended abstract. In *SPAA*, pages 279–288. ACM, 2011.

3 S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. In *SPAA*, pages 289–298. ACM, 2007.

4 M. Andrews, A. F. Anta, L. Zhang, and W. Zhao. Routing for power minimization in the speed scaling model. *IEEE/ACM Trans. on Networking*, 20:285–294, 2012.

5 E. Angel, E. Bampis, F. Kacem, and D. Letsios. Speed scaling on parallel processors with migration. In *Euro-Par*, volume 7484 of *LNCS*, pages 128–140, 2012.

6 A. Antoniadis and C.-C. Huang. Non-preemptive speed scaling. In *SWAT*, volume 7357 of *LNCS*, pages 249–260. Springer, 2012.

7 E. Bampis, D. Letsios, and G. Lucarelli. Green scheduling, flows and matchings. In *ISAAC*, volume 7676 of *LNCS*, pages 106–115. Springer, 2012.

8 B. D. Bingham and M. R. Greenstreet. Energy optimal scheduling on multiprocessors with migration. In *ISPA*, pages 153–161. IEEE, 2008.

9 A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli. State-of-the-art in heterogeneous computing. *Sci. Program.*, 18:1–33, 2010.

10 G. Greiner, T. Nonner, and A. Souza. The bell is ringing in speed-scaled multiprocessor scheduling. In *SPAA*, pages 11–18. ACM, 2009.

11 M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimizations, 2nd corrected edition.* Springer-Verlag, 1993.

12 A. Gupta, S. Im, R. Krishnaswamy, B. Moseley, and K. Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *SODA*, pages 1242–1253, 2012.

13 P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Comninatorica*, 7:365–374, 1991.

14 A. Wierman, L. L. H. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems. In *INFOCOM*, pages 2007–2015, 2009.

15 F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.

PTAS for Ordered Instances of Resource Allocation Problems *

Kamyar Khodamoradi[1], Ramesh Krishnamurti[1], Arash Rafiey[1], and Georgios Stamoulis[2]

1 **Simon Fraser University, Burnaby, Canada**
{kka50,ramesh,arashr}@sfu.ca
2 **IDSIA/USI/SUPSI, Manno-Lugano, Switzerland**
georgios@idsia.ch

Abstract

We consider the problem of fair allocation of indivisible goods where we are given a set I of m indivisible resources (items) and a set P of n customers (players) competing for the resources. Each resource $j \in I$ has a same value $v_j > 0$ for a subset of customers interested in j and it has no value for other customers. The goal is to find a feasible allocation of the resources to the interested customers such that in the Max-Min scenario (also known as *Santa Claus problem*) the minimum utility (sum of the resources) received by each of the customers is as high as possible and in the Min-Max case (also known as $R||C_{\max}$ problem), the maximum utility is as low as possible.

In this paper we are interested in instances of the problem that admit a PTAS. These instances are not only of theoretical interest but also have practical applications. For the Max-Min allocation problem, we start with instances of the problem that can be viewed as a *convex bipartite graph*; there exists an ordering of the resources such that each customer is interested (has positive evaluation) in a set of *consecutive* resources and we demonstrate a PTAS. For the Min-Max allocation problem, we obtain a PTAS for instances in which there is an ordering of the customers (machines) and each resource (job) is adjacent to a consecutive set of customers (machines). Next we show that our method for the Max-Min scenario, can be extended to a broader class of bipartite graphs where the resources can be viewed as a tree and each customer is interested in a sub-tree of a bounded number of leaves of this tree (e.g. a sub-path).

1998 ACM Subject Classification G.2.2 Graph Theory, G.1.2 Approximation

Keywords and phrases Approximation Algorithms, Convex Bipartite Graphs, Resource Allocation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.461

1 Introduction and Problem Definition

A bipartite graph $H = (P, I)$ with white vertices P and black vertices I is *convex*, if there is an ordering π of the vertices in I such that the neighborhood of each vertex in P consists of consecutive vertices, i.e., the neighborhood of each vertex in P forms an interval. Convex bipartite graphs are well known for their nice structures and both theoretical and practical properties. Many hard (i.e. **NP**-complete) optimization problems become polynomial-time solvable or even linear-time solvable in convex bipartite graphs while remaining hard for general bipartite graphs [6].

* Fourth author supported by the Swiss National Foundation project 200020-122110/1

We consider the problem of allocating indivisible items (resources) to a set of players (customers) in a convex bipartite graph below.

Problem Description. We are given a convex bipartite graph $H = (P, I)$ together with an ordering π of the vertices of I, where P is a set of n players and I is a set of m items. We consider the problem of allocating the indivisible items from I to the set P. Each player $p \in P$ has a utility function $f_p(j) = v_j > 0$ for each item $j \in [m]$ (v_j is a positive integer). This represents the value of item j for player p. If p is adjacent to item j then its value for p is v_j, otherwise its value is zero. The *goal* is to find a maximum t and a partition $I_1 \cup I_2 \cup \cdots \cup I_n = I$ of the items such that for every $1 \leq j \leq n$, I_j is a subset of items adjacent to player p_j and the items in I_j have a total value at least t in the max-min case and a total value at most t in the min-max case.

The interval case arises naturally in energy production applications where resources (energy) can be assigned and used within a number of successive time steps (i.e. the energy produced at some time step is available only for a limited amount of time corresponding to an interval of time steps) and the goal is a fair allocation of the resources over time, i.e. an allocation that maximizes the minimum accumulated resource we collect at each time step. In other words, we would like to have an allocation that guarantees the energy we collect at each time step is at least t, a pre-specified threshold. See also [19] for some applications in on-line scheduling.

Related work. For the general Max-Min fair allocation problem, where a given item does not necessarily have the same value for each player, no "good" approximation algorithm is known. In [5], by using similar ideas as in [13], an additive ratio of $\max_{i,j} v_{ij}$ is obtained, which can be arbitrarily bad. A stronger LP formulation, the *configuration LP*, is used to obtain a solution at least opt/n in [3]. Subsequently, [2] provided a rounding scheme for this LP to obtain an objective function value no worse than $\mathcal{O}(\frac{\text{opt}}{\sqrt{n}(\log^3 n)})$. In [17], an $\mathcal{O}(\sqrt{\frac{\log\log n}{n \log n}})$ approximation factor, close to the integrality gap of the configuration LP, was shown. In the *restricted* case, where $v_{ij} \in \{0, v_j\}$ for $i \in [n]$ and $j \in [m]$, there is an $\mathcal{O}(\frac{\log\log\log n}{\log\log n})$ factor approximation algorithm [3] for the Max-Min allocation problem. Furthermore, there is a simple $\frac{1}{2}$ inapproximability result for both the restricted case, as well as the general case (where an item does not necessarily have the same value for each player) [5]. Feige proved that the integrality gap of the configuration LP is a constant [8]. In [1] an integrality gap of $\frac{1}{5}$ was shown for the same LP which was later improved to $\frac{1}{4}$. The authors provide a local search heuristic with an approximation guarantee of $\frac{1}{4}$ which is not known to run in polynomial. Later, it was shown in [16] that the local search can be done in $n^{\mathcal{O}(\log n)}$ time. In [10] the authors provided a constructive version of Feige's original nonconstructive argument based on Lovász Local Lemma, thus providing the first constant factor approximation for the restricted Max-Min fair allocation problem. They provide an α-approximation algorithm for some *constant* α where an explicit value of α is not provided. Thus there is still a gap between the $\frac{1}{2}$ inapproximability result and the constant α approximability result in [10].

Several special cases of the Max-Min fair allocation problem have been studied. The case where $v_{ij} \in \{0, 1, \infty\}$ is shown to be hard in [12] and a trade off between running time and approximation guarantee is established. In [4] the authors consider the case in which each item has positive utility for a bounded number of players D, and prove that the problem is as hard as the general case for $D \leq 3$. They also provide a $\frac{1}{2}$ inapproximability result and a $\frac{1}{4}$ approximation algorithm for the *asymmetric* case when $D \leq 2$. The authors also provide a simpler LP formulation for the general problem and devise a polylogarithmic approximation

algorithm that runs in quasipolynomial time. The same result has been obtained in [7], which includes a $\frac{1}{2}$ approximation when $D \leq 2$, thus matching the bound proved in [4]. In [21], the author provides a PTAS for a (very) special case of the problem considered in this paper, namely, when the instance graph of the problem is a complete bipartite graph. In [14] a $\frac{1}{2}$-approximation algorithm was developed for a subclass of instances considered in this paper. See also [18], [15] for other special cases that our results generalize.

The $R||C_{\max}$ problem, as it is known in standard scheduling notation, is an important class of resource allocation problems. In this problem, we have machines (the players) and jobs (the items). Each job can be executed on any machine that belongs to a subset of machines (the subset depends on the job). Furthermore, the time required to process the job depends on the machine it executes on. We seek an assignment of jobs to machines such that the makespan is minimized. For the $R||C_{\max}$ problem, a 2-approximation algorithm based on a characterization of the extreme point solutions of a linear programming relaxation of the problem is given [13]. The authors also provide a $\frac{3}{2}$ inapproximability result. So far, all efforts to improve either of the bounds have failed. In a very recent result [20], it is shown that the restricted version of $R||C_{\max}$ admits an α approximation guarantee for α strictly less than 2. This result is an *estimation* result i.e. it estimates the (optimal) makespan of the restricted $R||C_{\max}$ within a factor of $\alpha = \frac{33}{17} + \epsilon$ for some arbitrary small positive ϵ. In this paper we consider the restricted case of the $R||C_{\max}$ problem where the processing time of each job for the subset of machines is the same.

Outline Of Our Results. Our results can be summarized as follows:

1. We present a PTAS for the restricted Max-Min fair allocation problem when the instance of the problem is a convex bipartite graph (each player sees an interval of items). Notice that this instance of the problem is (strongly) **NP**-complete, as it contains complete bipartite graphs as a special case (each player is adjacent to all the items), which is known to be strongly **NP**-complete [9].
2. We modify our approach for the Min-Max allocation problem to obtain a PTAS for the $R||C_{\max}$ problem when the machines a job can run on are consecutive in some ordering (form an interval).
3. In the Max-Min fair allocation, we show how our techniques can be extended to a bigger class of bipartite graphs. In a convex bipartite graph the items adjacent to a player form an interval or, equivalently, a *path*. In this extension, we consider the case where the items are the vertices of a tree and each player is interested in (has positive evaluation for) items that lie in a *sub-tree with bounded number of leaves* of the tree. We show that our algorithm can be modified to obtain a PTAS, though the run time increases as a polynomial of the number of leaves.

To obtain the PTAS for the instances considered in this paper, we first use scaling to classify the items into small and big items. Because the items adjacent to a player are consecutive, we can construct a solution comprising small items efficiently. We then add the big items to the solution efficiently to construct the total solution.

2 Preprocessing the Input

Consider the convex bipartite graph $H = (P, I)$ together with an ordering π of the vertices in I. For every vertex $p \in P$ let $[\ell_p, r_p]$ be the interval of the items adjacent to p. Based on the ordering π, we define the following ordering on the vertices in P:

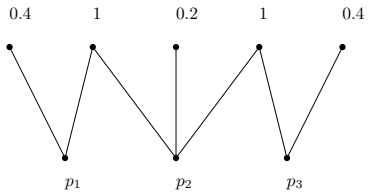


Figure 1 An example of an instance in which Hall's condition is satisfied for $t = 1$ but the optimal solution value is not greater than 0.4.

p is ordered before q whenever $\ell_p < \ell_q$, or $\ell_p = \ell_q$ and $r_p \leq r_q$ (breaking ties arbitrarily). According to ordering π, if $p \in P$ is adjacent to $i \in I$ and $q \in P$ is adjacent to $j \in I$ with $p < q$ and $j < i$ then p is also adjacent to j.

By a *feasible* assignment we mean an assignment such that each item is assigned to exactly one player that has non-zero evaluation for that item.

▶ **Definition 1** (t-assignment)**.** A t-assignment, $t \geq 0$, is a feasible assignment such that every player p receives a set of items $I_p \subseteq [\ell_p, r_p]$ with total value at least t.

Given a particular instance $H = (P, I)$ of the problem, we perform some steps that simplifies the input instance. For a positive integer t, we may assume that the value of each item is at most t. If item j has value $v_j > t$ then we set v_j to t without loss of generality. By a proper scaling, i.e. dividing each value by t, we may assume that the value of each item is in $[0, 1]$. Observe that a t-assignment becomes a 1-assignment. We do a binary search to find the largest value of t for which each player receives a set of items with total value at least t. The binary search is carried out in the interval $[0, \frac{1}{n}\sum_{j\in I} v_j]$ where 0 is an absolute lower bound, and $\frac{1}{n}\sum_{j\in I} v_j$ is an absolute upper bound of the optimal solution respectively.

For a subset $P' \subseteq P$ of players, let $N(P')$ be the union of the set of all neighbors of the players in P'. For an interval $[i, j]$ of the items, let $P[i, j]$ be the set of players whose *entire* neighborhood lies in $[i, j]$: $P[i, j] = \{p \in P : N(p) = [\ell_p, r_p] \subseteq [i, j]\}$. For a subset $I' \subseteq I$ of items, let $v(I')$ denote the sum of the values of all the items in I'. By *private neighborhood* of a player p we mean all the items that are adjacent only to p. We note that in every 1-assignment, for every subset $P' \subseteq P$ of players, the value of the items in its neighborhood should be at least $|P'|$. In other words, $\forall P' \subseteq P : \; v(N(P')) \geq |P'|$. If the value of each item is 1 then this condition is the well known Hall's condition [11], a condition sufficient and necessary for a bipartite graph to have a perfect matching. From now on we refer to the above condition as Hall's condition. Lemma 2 shows that in order to check Hall's condition for H it suffices to check it for every interval of items, and so Hall's condition in our setting becomes Condition (1) below:

$$\forall \;\; [\ell, r] \subseteq [1, m] : \qquad v([\ell, r]) \geq |P[\ell, r]|. \tag{1}$$

▶ **Lemma 2.** *In order to check Hall's condition for H it suffices to verify Condition (1). In other words, it suffices to check Hall's condition for every set of players $P[\ell, r]$, $[\ell, r] \subseteq [1, m]$.*

Note that the value $\frac{v([1,m])}{n}$ is an upper bound on the optimal value. In Figure 1 Hall's condition is satisfied but the optimal value is 0.4. This shows the integrality gap of the ILP formulation for the problem is more than 2. Thus, a different approach is required to get even a $\frac{1}{2}$ approximate solution.

For any integer $k \geq 3$, we let $\frac{1}{k}$ be the error parameter. For each instance for which there is an optimal 1-assignment, we seek an assignment such that each player receives a set of items with total value at least $1 - \frac{1}{k}$, $k \geq 3$. We call an item *small* if its value is less than $\frac{1}{k}$, otherwise it is considered a *big* item. We further round the values of the big items as follows. If v_j (the value of item j) is in the interval $[\frac{1}{k}(1+\frac{1}{k})^i, \frac{1}{k}(1+\frac{1}{k})^{i+1})$ then it is replaced by $\frac{1}{k}(1+\frac{1}{k})^{i+1}$. After the rounding, there are at most $K = \lceil \frac{\log k}{\log(1+\frac{1}{k})} \rceil$ distinct values more than $\frac{1}{k}$. Using straightforward calculus, one can show that K is no more than $k^{1.4}$. For i, $1 \leq i \leq K$ let $q_{i+1} = \frac{1}{k}(1+\frac{1}{k})^{i+1}$. For subset I' of I let $v_s(I')$ denote the value of the small items in I'.

In what follows let $p_1, p_2, \ldots, p_n$ be the ordering of the players and let m be the number of items in H. We also assume the following because it is a necessary condition for having an optimal 1-assignment.

Assumption: *A 1-assignment (an optimal 1-assignment) assigns to each player p_i a set of big items with total value $1 - w_i$ for some "deficit" w_i, $0 \leq w_i \leq 1$, and produces an instance H' of the problem for which Hall's condition is satisfied, i.e. for every interval $[\ell, r]$ of items, $v_s([\ell, r]) \geq \sum_{p_i \in P[\ell,r]} w_i$, that is, the deficit w_i of player p_i must be compensated for with small items.*

3 Structural Properties and the Algorithm

We start with a crucial lemma that will constitute the core of our algorithms. Intuitively, the lemma says that if a 1-assignment exists, then there exists another "almost" 1-assignment with a very particular structure.

▶ **Lemma 3.** *Suppose there exists an optimal 1-assignment for H in which player p_n (last player) receives a set S of items from $N(p_n)$, containing α_i, $1 \leq i \leq K$ big items with value q_i and a set of small items with total value at least $\frac{\alpha_0}{k}$ and less than $\frac{\alpha_0+1}{k}$ such that $v(S) \geq 1$. Then we obtain (in polynomial time) an assignment such that:*

1. *for every $i \geq 1$ the items with value q_i are the rightmost ones in the neighborhood of p_n.*
2. *p_n gets a set of consecutive small items from right to left (in the ordering) of the interval $N(p_n)$ with value at least $\frac{\alpha_0-1}{k}$.*
3. *The existence of a 1-assignment for the rest of H is preserved.*

Proof. *Proof of 1.* Suppose there are two big items x_1, x_2, $x_1 < x_2$, with the same value in the neighborhood of p_n such that $x_1 \in S$ and $x_2 \notin S$. Then item x_2 is either assigned to some player $p_i < p_n$ by the optimal solution or it is not assigned to any player. If x_2 is not assigned to any player by the optimal 1-assignment then we can include it instead of x_1. If x_2 is assigned to p_i in the optimal 1-assignment then x_1 is also adjacent to p_i by the ordering property and we can assign x_1 to p_i and x_2 to p_n.

Proof of 2. We note that we may look at the optimal 1-assignment as follows. The optimal 1-assignment assigns a set of big items to each player p_i with total value $1 - w_i$, $0 \leq w_i \leq 1$ in the first step. After this step, we have an instance of the fair allocation problem where each player p_i, $1 \leq i \leq n$ is allocated a set of small items with total value at least w_i. Because the solution consists of only small items we have $\frac{d_i}{k} \leq w_i \leq \frac{d_i+1}{k}$ for some integer d_i, $0 \leq d_i \leq k-1$. Since there is an optimal 1-assignment, Hall's condition is satisfied for each set of players. Also by Lemma 2, Hall's condition needs to be verified only for each interval of items. For every interval $[\ell, r]$ of the items, we have Condition (2) below:

$$v([\ell, r]) \geq \sum_{p_i \in P[\ell,r]} w_i \tag{2}$$

Let $S(p_n)$ be the set of items obtained as follows. Start from r_{p_n}, the *last* item in the neighborhood of p_n, and add the small items one by one from right to left to set $S(p_n)$, as long as $v(S(p_n)) < w_n - \frac{1}{k}$. Then, we add the next rightmost small item to the set $S(p_n)$ as well (so $v(S(p_n)) \geq w_n - \frac{1}{k}$). Let $\ell_s(p_n)$ be the index (according to the ordering) of the leftmost item added to $S(p_n)$. Note that we may need to add all of the small items to $S(p_n)$. Observe that $w_n - \frac{1}{k} \leq v(S(p_n)) < w_n$ since the last item added to $S(p_n)$ has value less than $\frac{1}{k}$. By assigning $S(p_n)$ to p_n and removing it from H, Condition (2) is still satisfied for each interval of items in the rest of the graph. Observe that since Hall's condition is satisfied, $v_s(N(p_n)) \geq w_n$. On the other hand, $v(S(p_n)) < w_n$. We assign $S(p_n)$ to p_n, and we observe that the items in $S(p_n)$ are consecutive. We will show that for the rest of the players and items, Hall's condition is still satisfied.

Proof of 3. Let $H' = H \setminus (S(p_n) \cup \{p_n\})$ be the reduced instance we derive after assigning items in $S(p_n)$ to player p_n. Note that the neighborhood of each player in H' is an interval. Consider an interval $[\ell, r]_{H'}$ in H' such that $P_{H'}[\ell, r] \neq \emptyset$ in H'. If $[\ell, r]_{H'} \cup S(p_n)$ is not an interval in H then Hall's condition is satisfied for $[\ell, r]_{H'}$ as otherwise $[\ell, r]_{H'} = [\ell, r]_H$ and Hall's condition would not be satisfied in H. So we assume $[\ell, r]_{H'} \cup S(p_n)$ forms an interval in H. Consider the set of items $[\ell, r]_{H'} \cup S(p_n)$ in H (an interval in H). We note that $S(p_n)$ corresponds to interval $[\ell_s(p_n), r_{p_n}]$ in H. First we notice that $\ell \leq \ell_s(p_n)$ (i.e. ℓ is to the left of $\ell_s(p_n)$). This follows from the ordering of the players based on the left end points of their intervals. Thus we have $[\ell, r]_{H'} \cup S(p_n) = [\ell, r]$. Moreover $P[\ell, r] = P[[\ell, r]_{H'}] \cup \{p_n\}$. Therefore we have $v([\ell, r]_{H'}) + v(S(p_n)) = v[\ell, r] \geq \sum_{p_i \in P[\ell,r]} w_i$. Since $v(S(p_n)) < w_n$, we have $v([\ell, r]_{H'}) \geq \sum_{p_i \in P[[\ell,r]_{H'}]} w_i$. ◀

The Algorithm: We first observe that if an optimal 1-assignment assigns a set of items containing α_i items with value q_i to player p_n then by Lemma 3 we may assume that these α_i big items are the rightmost big items of value q_i in the neighborhood of p_n.

Before we proceed, we need the following definition of the *right-most vectors* (intuitively vectors that satisfy the conditions of Lemma 3).

▶ **Definition 4** (Right-most Ordering)**.** Let $V = (\alpha_0, \alpha_1, \alpha_2, ..., \alpha_K)$ be a vector of non-negative integers. For a given interval of items $[\ell, r] \subseteq [1, m]$ let $\mathcal{S}([\ell, r])$ be all the sets of items $S \subseteq [\ell, r]$ that are consistent with V i.e. $S \in \mathcal{S}([\ell, r])$ if the vector of items that represents S is exactly V. There might be several different sets S in $[\ell, r]$ consistent with V. We say that S is a **right-most** set of items in $[\ell, r]$ if

- for each i, $1 \leq i \leq K$, S contains the rightmost α_i big items with value q_i from $[\ell, r]$.
- the small items in S are the rightmost consecutive small items from $[\ell, r]$.

Observe that such a set S in our setting is **unique**. Moreover, when we say that a vector of non-negative integers V is the right-most for a given interval $[\ell, r]$, we interpret it as the unique $S \in \mathcal{S}([\ell, r])$ with the properties listed above.

At each step i, $1 \leq i \leq n$ of the algorithm we keep track of the right-most vectors of items assigned to the players $p_{n-i+1}, p_{n-i+2}, \ldots, p_n$ as well as the subgraph left for the rest of the players. We call such a vector an *assigned vector* and there might be several such assigned vectors at step i. Each assigned vector $A_i = (\beta_0, \beta_1, \ldots, \beta_K)$ at step i indicates that all together β_j items of value q_j, $1 \leq j \leq K$ and a set of S' of small items with value $\frac{\beta_0}{k}$ can be assigned to players $p_{n-i+1}, p_{n-i+2}, \ldots, p_n$, i.e. A_i represents an assignment to the players $p_{n-i+1}, p_{n-i+2}, \ldots, p_n$.

In order to keep track of the right-most assigned vectors and subgraphs we construct an $n \times d$ matrix M. Here $d = (K+1)m^{K+1}$ is the number of all possible assigned vectors

that arise from the initial vector $V = (z_0, z_1, z_2, ..., z_K)$ representing all the items (recall that $m = |I|$). Each entry of M contains one bit (which is either 0 or 1) and an $n \times m$ adjacency matrix. In particular, $M[i, j] = 1$ if assigning some right-most vector indexed by j to player p_{n-i+1} makes this vector "active" in the next round (i.e., it potentially can lead to a valid assignment for all players, therefore should be considered). Moreover, with abuse of notation, we say that $M[i, j] = H' \subseteq H$ where H' is the subgraph that arises by ignoring the items from the newly assigned vector j and players after p_{n-i}. Once we consider player p_{n-i+1} we consider a right-most vector $V_i = (\alpha_0, \alpha_1, \ldots, \alpha_K)$ (in the neighborhood of p_{n-i+1}) with value at least $1 - \frac{1}{k}$ that includes all the items in the private neighborhood of p_{n-i+1} (as otherwise they will not be used later). Then we look at an entry $M[i-1, j'] = 1$, where j' represents the right-most vector $(\beta_0, \beta_1, \ldots, \beta_K)$ and we set $M[i, j] = 1$, where j corresponds to vector $V_i = (\alpha_0 + \beta_0, \alpha_1 + \beta_1, \ldots, \alpha_k + \beta_K)$. Moreover $M[i, j] = H' \subseteq H$ where H' is the subgraph that arises by ignoring the items from the current assigned vector and ignoring the players $p_{n-i+1}, p_{n-i+2}, \ldots, p_n$. Note that several possible configurations may set an entry to one. This subgraph is obtained from the set of items corresponding to V_i and the subgraph from $M[i-1, j']$.

Algorithm for Convex Case

At step i (at the beginning $i = 1$):

1. The current player for consideration is p_{n-i+1}. Let $\mathcal{A}_i$ be the current set of all right-most assigned vectors i.e. $\mathcal{A}_i = \{j \in [d] : M[i-1, j] = 1\}$ ($\mathcal{A}_1 = \emptyset$). For each $A \in \mathcal{A}_i$ do:
   - **a.** Consider all the *minimal* right-most vectors $V_i = (\alpha_0, \alpha_1, \alpha_2, ..., \alpha_k)$ representing items from $N(p_{n-i+1})$ in the subgraph induced by the *current* assigned vector A (this subgraph can be simply found by consulting the corresponding entry in the matrix M) such that:
     - $1 - \frac{1}{k} \leq \frac{\alpha_0}{k} + \sum_{j=1}^{j=K} \alpha_j q_j$
     - all the items corresponding to this vector are in the neighborhood of p_{n-i+1}
     - V_i includes all the private neighbors of p_{n-i+1} (items adjacent only to p_{n+1-i})
     - If the value of the items in the private neighborhood of p_{n-i+1} is at least $1 - \frac{1}{k}$ then let V_i be the vector of all the items in the private neighborhood of p_{n-i+1}

     Observe that each vector V_i represents a *unique* set of items since it is a right-most vector.
   - **b.** If there is no such V_i, report NO assignment and exit.
   - **c.** For every such vector V_i (at step i) we consider the assigned vector $\overline{V_i} = A_i + V_i$ (observe that, given V_i, the assigned vector $\overline{V_i}$ is uniquely defined):
     - Set $M[i, j] = 1$ where j is the column corresponding to this assigned vector $\overline{V_i}$.
     - Update the entry in $M[i, j]$ corresponding to the subgraph induced by players $p_1, p_2, \ldots, p_{n-i}$ by using the previous entry of the corresponding subgraph at step $i-1$ and the current set of items in vector V_i (at Step 1 we use the adjacency matrix of H).

     Step (c) above keeps track of the remaining subgraph for the rest of the players.
2. Set $i = i + 1$ and go to (1).
3. Assign the items in the neighborhood of p_1 (corresponding to one of the subgraph remained containing p_1) and trace back M to obtain an assignment for the rest of the players.

In order to retrieve an actual assignment (last step of the algorithm) we proceed as follows: at step n when we consider player p_1 there should be at least one vector of items with value $1 - \frac{1}{k}$ in the neighborhood of p_1). We assign the items that are uniquely defined

by such a vector to player p_1. To continue with the rest of the players we may find useful to include the index j at a step i that caused a particular assigned vector at the next step $i+1$ be set to 1. With this, when we allocate a particular vector of items to player p_{n-i} we know how to trace back a feasible assignment. In other words, whenever we set $M[i,j]=1$ in the body of the algorithm we also store which assigned vector j' from row $i-1$ is responsible for setting $M[i,j]=1$ at step i.

▶ **Lemma 5.** *Let H be an instance of the problem with n players and m items. Suppose there is an optimal 1-assignment for H. Then the Algorithm assigns (in polynomial time) to each player a set of items with value at least $1-\frac{2}{k}$.*

Proof. First by definition of right-most vector, the value of set S corresponding to vector V_i is at least $1-\frac{2}{k}$. This is true because $1-\frac{1}{k} \leq \frac{\alpha_0}{k} + \sum_{j=1}^{j=K} \alpha_j q_j$ and the value of the small items in S is at least $\frac{\alpha_0-1}{k}$.

Second we need to show that the number of assigned vectors A_i at step i is at most $(K+1)m^{K+1}$. According to Item (2) of Lemma 3 we can take the small items consecutively from right to left. This allows us to look at the small items as a number of blocks of size $\frac{1}{k}$ when they are considered from right to left. Therefore we may assume there are $K+1$ types of items resulting in at most m^{K+1} different possible assigned vectors. When the graph induced by players $p_{n-i+1}, p_{n-i+2}, \ldots, p_n$ and their neighborhood is a complete bipartite graph then the number of possible assignments (number of 1's in the row i of M) is bounded by $(K+1)m^{K+2}$. Moreover, since each right-most assigned vector *uniquely* defines a set of items S, this means that at each step the entry of $M[i,j]$ that corresponds to the subgraph induced for the rest of the players $(p_1, \ldots, p_{n-i})$ is unique. So, the size of the matrix M is $\mathcal{O}(nm^{K+2})$ and each entry of M contains an $m \times n$ adjacency matrix.

Note that each assigned vector at step i represents at least one assignment to the players $p_{n-i+1}, p_{n-i+2}, \ldots, p_n$ such that each of them receives at least $1-\frac{2}{k}$. We claim that if we keep track of at most $(K+1)m^{K+2}$ different possible ways of assigning the items to the players $p_n, p_{n-1}, ..., p_{n-i+1}$ then according to Lemma 3 we guarantee the existing of a 1-assignment for the players $p_1, p_2, \ldots, p_{n-i}$ using the remaining items.

Suppose there exists i, $1 \leq i \leq n$, such that there is no vector V_i in Step 1.b. Then we show that there is no optimal 1-assignment. We use induction on i. Note that i is more than 1 as otherwise there are not enough items in the neighborhood of p_n and clearly there is no optimal 1-assignment. We show that $i > 2$. If $i = 2$ then according to the selection of the items in Step (1) for player p_n we include all the private neighbors of p_n, and all the possible vectors V_1 considered for player p_n are right-most. Hence by Lemma 3 the existence of the 1-assignment should be preserved for the rest of the players, a contradiction.

Let $i \geq 3$. At step $i-1$ the algorithm considers a vector V_{i-1} from $N(p_{n-i+2})$, and together with an assigned vector A_{i-2} from row $i-2$ of M, it creates a new entry for row $i-1$. If the algorithm should have recorded some other assignment different from the ones in the entry of M at row $i-1$ then it means some big item x (of value q_j) and not in the items represented by $A_{i-2}+V_{i-1}$ (or a set X of small items with total value $\frac{\beta}{k}$) is assigned to a player p_t, $n-i+2 \leq t$ and some item x' (of value q_j) from the of items represented by $A_{i-2}+V_{i-1}$ (or a set Y of small items with total value $\frac{\beta}{k}$ represented by $A_{i-2}+V_{i-1}$) is assigned to p_{n-i+1}. Note that $x < x'$. However, since p_{n-i+1} is adjacent to x', it is also adjacent to x and hence we can exchange x and x'. In other words, as far as player p_{n-i+1} is concerned, the items from the right-most assigned vector are the ones that can be assigned to the players $p_{n-i+2}, p_{n-i+3}, \ldots, p_n$. ◀

▶ **Theorem 6.** *Let H be an instance of the problem with n players and m items. Then for $k \geq 3$ there exists a $(1 - \frac{3}{k+1})$-approximation algorithm with running time $O(n^2 m^{K+2})$.*

Proof. According to Lemma 5, each player receives a set of items with value at least $1 - \frac{2}{k}$, once we round the value of the items. Because of the rounding, this value should be divided by $1 + \frac{1}{k}$. Therefore each player receives a set of items with value at least $1 - \frac{3}{k+1}$. The size of the matrix M in Lemma 5 is $O(nm^{K+1})$ and each entry of M contains an $m \times n$ adjacency matrix. Therefore the running time of the algorithm is $O(n^2 m^{K+2})$. ◀

4 Min-Max Allocation Problem ($R||C_{\max}$)

Problem Description: We are given a set M of identical machines and a set J of jobs. Each job j has a same processing time p_j on a subset of machines and it has processing time ∞ on the rest of the machines. The goal is to find an assignment of the jobs to the machines, such that the maximum load among all the machines is minimized. Formally, we have a bipartite graph $H = (M, J, E)$ where M is a set of machines and J is a set of jobs, and E denotes the edge set. There is an edge in E between a machine and a job if the job can be executed on that machine. We consider the case where each job can be executed on an interval of machines:

Assumption: *We have an ordering $M_1, M_2, ..., M_n$ of machines such that each job can be executed on consecutive machines (an interval of machines).*

We denote the interval of job J_i by $[\ell_i, r_i]$. We assume that J_i is before J_j, $i < j$ whenever $\ell_i < \ell_j$ or $\ell_i = \ell_j$, $r_i \leq r_j$. We denote this ordering by π. The ordering π has the following property: if M_i is adjacent to J_r, and M_j for $j > i$ is adjacent to J_s, $s < r$ then M_i is also adjacent to J_s. By scaling down the value of the processing time, we may assume that $0 \leq p_i \leq 1$.

Consider the error parameter $\frac{1}{k}$ for an integer $k \geq 2$. The goal is to find an assignment such that each machine receives a set of jobs with total processing time at most $1 + \frac{1}{k}$, $k \geq 2$, when there exists an optimal 1-assignment. We say a job is *small* if its value is less than $\frac{1}{k}$, otherwise it is called a *big* job. Now we further round the values of the jobs as follows. If v_j (the value of item j) is in the interval $[\frac{1}{k}(1 + \frac{1}{k})^i, \frac{1}{k}(1 + \frac{1}{k})^{i+1})$ then it is replaced by $\frac{1}{k}(1 + \frac{1}{k})^i$. Using this method, we obtain at most $K = \lceil \frac{\log k}{\log(1+\frac{1}{k})} \rceil$ distinct values more than $\frac{1}{k}$. For $1 \leq i \leq K$ let $q_i = \frac{1}{k}(1 + \frac{1}{k})^i$.

We use the usual classification of the jobs into big and small, together with rounding step as in the case of Max-Min allocation. For subset J' of jobs let $w(J')$ ($w_s(J')$) be the sum of the processing times of all the jobs (small jobs) in J'. For every subset M' of machines let $\mathcal{J}[M']$ be the set of jobs whose entire neighborhood lies in set M'. A necessary condition for having a maximum load at most 1 is that for every subset M' of machines $w(\mathcal{J}[M']) \leq |M'|$. In order to check this condition, we need to check it for every interval of machines. For interval $[i, j]$ of machines $M_i, M_{i+1}, ..., M_j$, we look at all the jobs that are executed only on machines $M_i, M_{i+1}, \ldots, M_j$ and if the sum of the processing time of all these jobs is greater than $j - i + 1$ then the condition is violated. For interval $[\ell, r]$, $\ell \leq r$, let $\mathcal{J}[\ell, r]$ be the set of jobs that can be executed only on a subset of the machines in this interval. By argument similar to that used in the proof of Lemma 2, Condition 3 is given below. For simplicity we refer to the condition $\forall \ \ [\ell, r] \subseteq [1, n] : \ \ w(\mathcal{J}[\ell, r]) \leq r - \ell + 1$ as Hall's condition.

Assumption: *A 1-assignment (an optimal 1-assignment) is an assignment that assigns to each machine M_i a set of big jobs with total value $1 - w_i$, $0 \leq w_i \leq 1$ and it produces an*

instance H' of the problem for which the Hall's condition (with respect to the small jobs) is satisfied, i.e. for every interval $[\ell, r]$ of machines, $v_s(\mathcal{J}[\ell, r]) \leq \sum_{i=\ell}^{i=r} w_i$.

Let $N_0[\ell, c]$ be an ordered set of small jobs in the neighborhood of M_ℓ, obtained as follows. We first add all the jobs in $\mathcal{J}[\ell, \ell]$ one by one from left to right (according to ordering π). In step j, $1 \leq j \leq c$, we add to $N_0[\ell, c]$ all the small jobs from $\mathcal{J}[\ell, \ell + j] \setminus \mathcal{J}[\ell, \ell + j - 1]$ one by one from left to right.

Let $N_i[\ell, c]$, $i \geq 1$ be an ordered set of jobs with value q_i obtained as follows. We first add to $N_i[\ell, c]$ all the jobs with value q_i from $\mathcal{J}[\ell, \ell]$ one by one from left to right. In step j, $1 \leq j \leq c$, we add to $N_i[\ell, c]$ all the jobs with value q_i from $\mathcal{J}[\ell, \ell + j] \setminus \mathcal{J}[\ell, \ell + j - 1]$ from left to right.

▶ **Lemma 7.** *Suppose there exists an optimal 1-assignment for H in which machine M_1 (the first machine in the ordering) receives a set S of jobs from $N(M_1)$, containing α_i, $1 \leq i \leq K$, big jobs with value q_i, and a set of small jobs with total value at least $\frac{\alpha_0}{k}$ and less than $\frac{\alpha_0+1}{k}$, such that $v(S) \leq 1$. Then we obtain (in polynomial time) an assignment such that:*

1. *for every $i \geq 1$, the jobs with value q_i are the first α_i's jobs in $N_i[1, c]$ for some $c > 1$.*
2. *M_1 gets a set of consecutive small jobs from $N_0[1, c]$ with value less than $\frac{\alpha_0+2}{k}$ and the existence of a 1-assignment for the rest of H is preserved.*

Let $N(M[\ell, r])$ (a set of jobs) denote the neighborhood of machines $M_\ell, M_{\ell+1}, \ldots, M_r$.

▶ **Definition 8** (Left-most Ordering-$R||C_{\max}$)**.** Let $V = (\alpha_0, \alpha_1, \alpha_2, ..., \alpha_K)$ be a vector of non-negative integers. Let V represent a set S of jobs from $N(M[\ell, r])$ containing α_i, $1 \leq i \leq K$ big jobs of value q_i and a set of small jobs with total value at least $\frac{\alpha_0}{k}$ and at most $\frac{\alpha_0+1}{k}$. We say vector V is a *left-most* vector if:

- for each i, $1 \leq i \leq K$, S contains the first α_i jobs with value q_i from $N_i[\ell, r]$.
- the small jobs in S are the first set of consecutive small jobs from $N_0[\ell, r]$.

Let n be the number of machines and m be the number of jobs and set $d = (K+1)m^{K+1}$. Identical with the Max-Min case, we consider a matrix M with the same properties. The algorithm is similar to the one in the Max-Min case of Section 3 (with the necessary adjustments).

▶ **Lemma 9.** *Suppose there exists an optimal 1-assignment for H. Then there exists a polynomial time assignment that assign all the jobs to the machines without exceeding the maximum load $1 + \frac{2}{k}$.*

▶ **Theorem 10.** *Let H be an instance of the problem with n machines and m jobs. Then for $k \geq 3$ there exists an $(1 + \frac{3}{k+1} + \frac{2}{k^2})$-approximation algorithm with running time $O(nm^{K+2})$.*

5 Max-Min problem when the Items are in a Tree

In this section we consider instances of the problem when the items are the vertices of a tree T and each player is interested in a sub-tree of T with at most d leaves for some constant d. This class of instances contain, the class of convex bipartite instance, as a special case. We notice that if the items are vertices of a tree T and each player is interested in a sub-tree of T then we get the general instances of the problem. To see this we just need to assume T is a star.

Problem Description: We are given a bipartite graph $H = (P, T)$ where P is a set of n players and T is a tree where each node of T is an item. We consider the problem of allocating the indivisible items from I to the set P. Each player $p \in P$ has a utility function $f_p(j) = v_j > 0$ for each item $j \in [m]$ (v_j is a positive integer). This represents the value of item j for player p. If p is adjacent to item j then its value for p is v_j, otherwise its value is zero. For each players p the set of items adjacent to p forms a sub-tree of T with at most d leaves (d is a constant number). The *goal* is to find a maximum t and a partition $T_1 \cup T_2 \cup \cdots \cup T_n = T$ of the items (on T) such that for every j, $1 \leq j \leq n$, T_j is a subset of items adjacent to player p_j and the items in T_j have a total value at least t.

Indexing the tree: The *spine* of T is a longest path in T. Let $SP = v_1, v_2, \ldots, v_q$ be a spine of T. The index of a vertex x in T is the smallest i such that v_i is the closest vertex to x. For two vertices x, y of T we say x is before y, (we write $x \prec y$) if the index of x is less than the index of y. When x, y have the same index i, then $x \prec y$ if x is closer to v_i than y, and no other vertex z in the (x, y)-path is closer to v_i than x (note that the (x, y)-path is unique since T is a tree). In all other cases the order between x and y is arbitrary. The index of subtree P is the index of the vertex with the smallest index among the vertices in P. We say subtree P is before subtree Q and we write $P \prec Q$ if the index of P is less than the index of Q and if P and Q have the same index then the last vertex of P in the ordering $\prec$ lies inside Q.

Ordering the players: We order the players based on their sub-trees, i.e. p is before q if $P \prec Q$ where P, Q are the sub-trees corresponding to p, q.

For subtree T' of T let $P[T']$ denote the set of players whose entire neighborhood lies in T'. Let $p_1, p_2, ..., p_n$ be an ordering of the players. For player p_n, let $\ell_1(p_n), \ell_2(p_n), \ldots, \ell_t(p_n)$, $t \leq d$ be the leaves of $N(p_n)$ where $\ell_i(p_n) \prec \ell_j(p_n)$, $1 \leq i < j \leq t$. Let $x \in N(p_n)$ be the item with the smallest index.

▶ **Definition 11.**
- Let S be a subset of items in $N(p_n)$ with value q_i for an $1 \leq i \leq K$. We say S is *good* if there exist $\beta_1, \beta_2, \ldots, \beta_t$ such that $\sum_{j=1}^{j=t} \beta_j = |S|$ and S comprises of the *last* β_j, (for every $1 \leq j \leq t$) items with value q_i on the path from x to $\ell_j(p_n)$ in the sub-tree $N(p_n)$.
- Let S be a subset of small items in $N(p_n)$. We say S is *good* if there exist items $\ell_1, \ell_2, \ldots, \ell_t$, $\ell_j \preceq \ell_j(p_n)$, $1 \leq j \leq t$ such that S comprises of all the small items on the path from $\ell_j + 1$ to $\ell_j(p_n)$ in the sub-tree $N(p_n)$.

Analogous to Lemma 3 and Theorem 5 we have the Lemma 12 and the Theorem 13 below.

▶ **Lemma 12.** *Suppose there exists an optimal* 1*-assignment for* H *such that player* p_n *receives a set* S *of items from* $N(p_n)$ *where* S *contains* α_i, $1 \leq i \leq K$ *big items with value* q_i *and some small items with total value at least* $\frac{\alpha_0}{k}$ *and less than* $\frac{\alpha_0+1}{k}$ *such that* $v(S) \geq 1$. *Then there exists an assignment in which* p_n *gets a set* S' *of items such that for every* $1 \leq i \leq K$, *there are exactly* α_i *big items with value* q_i *in* S' *forming a good set and the small items in* S' *form a good set with total value at least* $\frac{\alpha_0-1}{k}$. *Moreover, the existence of a* 1*-assignment for the rest of* H *is preserved.*

▶ **Theorem 13.** *Let* H *be an instance of the problem with* n *players and* m *items. Then for* $k \geq 3$ *there exists an* $(1 - \frac{3}{k+1})$*-approximation algorithm with running time* $O(nm^{d \cdot K+2})$.

6 Conclusion and Future Work

In all instances of the problem considered in this paper, a proper ordering has played an important role. However we do not know a dichotomy classification for the instances of the problem that admit a PTAS. We ask for a dichotomy of the following form:

> If H belongs to class X of bipartite graphs then there is a PTAS for Max-Min allocation problem otherwise there is no PTAS.

Acknowledgments. We would like to thank Monaldo Mastrolilli for many useful discussions and for proposing this problem to us.

References

1 A. Asadpour, U. Feige, and A. Saberi. Santa claus meets hypergraph matchings. In *APPROX-RANDOM*, pages 10–20, 2008.
2 A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *STOC*, pp. 14–121. ACM, 2007.
3 N. Bansal and M. Sviridenko. The santa claus problem. In *STOC*, pages 31–40. ACM, 2006.
4 M. Bateni, M. Charikar, and V. Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *STOC*. ACM, 2009.
5 I. Bezáková and V. Dani. Allocating indivisible goods. *SIGecom Exchanges*, 5(3):11–18, 2005.
6 A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph classes: a survey*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA, 1999.
7 D. Chakrabarty, J. Chuzhoy, and S. Khanna. On allocating goods to maximize fairness. In *FOCS*, pages 107–116, 2009.
8 U. Feige. On allocations that maximize fairness. In *SODA*, pp. 287–293. ACM-SIAM, 2008.
9 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
10 B. Haeupler, B. Saha, and A. Srinivasan. New constructive aspects of the lovász local lemma. *J. ACM*, 58(6):28, 2011.
11 P. R. Halmos and H. E. Vaughan. The marriage problem. *American Journal of Mathematics*, pages 214–215, 1950.
12 S. Khot and A. K. Ponnuswami. Approximation algorithms for the max-min allocation problem. In *APPROX-RANDOM*, pages 204–217, 2007.
13 J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *FOCS*, pages 217–224. IEEE, 1987.
14 M. Mastrolilli and G. Stamoulis. Restricted max-min fair allocations with inclusion-free intervals. In J. Gudmundsson, J. Mestre, and T. Viglas, editors, *COCOON*, volume 7434 of *Lecture Notes in Computer Science*, pages 98–108. Springer, 2012.
15 G. Muratore, U. M. Schwarz, and G. J. Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Oper. Res. Lett.*, 38(1):47–50, 2010.
16 L. Polacek and O. Svensson. Quasi-polynomial local search for restricted max-min fair allocation. In *ICALP*, 2012.
17 B. Saha and A. Srinivasan. A new approximation technique for resource-allocation problems. In *ICS*, pages 342–357. Tsinghua University Press, 2010.
18 U. M. Schwarz. A PTAS for scheduling with tree assignment restrictions. *CoRR*, abs/1009.4529, 2010.

19 J. Sgall. Randomized on-line scheduling of parallel jobs. *J. Algorithms*, 21(1):149–175, 1996.

20 O. Svensson. Santa claus schedules jobs on unrelated machines. In *STOC*, pp. 617–626, 2011.

21 G. Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operation Research Letters*, 20(4):149–154, 1997.

On the Pseudoperiodic Extension of $u^\ell = v^m w^n$ *

Florin Manea, Mike Müller, and Dirk Nowotka

Christian-Albrechts-Universität zu Kiel, Kiel, Germany
{flm,mimu,dn}@informatik.uni-kiel.de

Abstract

We investigate the solution set of the pseudoperiodic extension of the classical Lyndon and Schützenberger word equations. Consider $u_1 \cdots u_\ell = v_1 \cdots v_m w_1 \cdots w_n$, where $u_i \in \{u, \theta(u)\}$ for all $1 \leq i \leq \ell$, $v_j \in \{v, \theta(v)\}$ for all $1 \leq j \leq m$, $w_k \in \{w, \theta(w)\}$ for all $1 \leq k \leq n$ and u, v and w are variables, and θ is an antimorphic involution. A solution is called pseudoperiodic, if $u, v, w \in \{t, \theta(t)\}^+$ for a word t. Czeizler et al. (2011) established that for small values of ℓ, m, and n non-periodic solutions exist, and that for large enough values all solutions are pseudoperiodic. However, they leave a gap between those bounds which we close for a number of cases. Namely, we show that for $\ell = 3$ and either $m, n \geq 12$ or $m, n \geq 5$ and either m and n are not both even or not all u_i's are equal, all solutions are pseudoperiodic.

1998 ACM Subject Classification F.4.m Mathematical Logic and Formal Languages – Misc.

Keywords and phrases Word equations, Pseudoperiodicity, Lyndon-Schützenberger equation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.475

1 Introduction

The study of the classical word equations $u^\ell = v^m w^n$ dates back to 1962. Lyndon and Schützenberger [6] showed that for $l, m, n \geq 2$, in all solutions of this equation in a free group, u, v, w are necessarily powers of a common element. Their result holds canonically if u, v and w are elements of a free semigroup, but, for this case, simpler proofs exist [5].

Czeizler et al. [1] introduced a generalisation of Lyndon and Schützenberger's equations of the form $u_1 \cdots u_\ell = v_1 \cdots v_m w_1 \cdots w_n$, where $u_i \in \{u, \theta(u)\}$ for all $1 \leq i \leq \ell$, $v_j \in \{v, \theta(v)\}$ for all $1 \leq j \leq m$, and $w_k \in \{w, \theta(w)\}$ for all $1 \leq k \leq n$, and studied under which conditions $u, v, w \in \{t, \theta(t)\}^+$ for some word t. In other words, they studied the case when u, v, w are generalised powers (more precisely, θ-powers). Here, θ is a function on the letters of the alphabet, which acts as an antimorphism (i.e., $\theta(uv) = \theta(v)\theta(u)$ for all words u, v) and as an involution (i.e., $\theta(\theta(u)) = u$ for all words u). These so called *antimorphic involutions* are commonly used to formally model the Watson-Crick complement occurring in DNA structures; this connection sparked the interest towards studying the combinatorial properties of words that can be expressed as catenation of factors and their image under such antimorphic involutions (see, [1]). Apart from this initial bio-inspired motivation, there is a strong intrinsic mathematical motivation in studying such words. Indeed, one of the simplest and most studied operations on words is mirroring, the very basic antimorphic involution. It is, thus, natural to study equations on words in which not only powers of variables, but also repeated concatenations of a variable and its mirror image appear.

The results obtained in [1, 4] are summarised in Table 1. One can notice easily from this table that the more interesting cases in this generalised setting are those in which $\ell, m, n \geq 3$. Moreover, when $\ell = 3$ only several negative results were found. That is, there is a series

* Supported by DFG grants 596676 (F. Manea), 582014 (M. Müller), and 590179 (D. Nowotka).

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 475–486

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

Table 1 The results known so far about the equation $u_1 \cdots u_\ell = v_1 \cdots v_m w_1 \cdots w_n$.

ℓ	m	n	$u, v, w \in \{t, \theta(t)\}^+$?
≥ 4	≥ 3	≥ 3	Yes
3	≥ 5	≥ 5	Open
3	4	≥ 3 and odd	Open
3	4	≥ 4 and even	No
3	3	≥ 3	No
	one of $\{\ell, m, n\}$ equals 2		No

of equations which have non-periodic solutions, but very little is known about those cases of such equations where the pseudoperiodicity of the solutions is forced, similarly to the classical Lyndon-Schützenberger equations (the only exception was the particular Lemma 12, see Prop. 51 in [4]). Finally, the case $\ell = 3$ seems to be especially intricate and interesting, because it separates the cases when the equation has only θ-powers as solutions ($\ell \geq 4$) from the cases when it may have solutions which are not θ-powers ($\ell \leq 2$). Accordingly, our work aims to add some relevant results regarding equations with $\ell = 3$, and solves some of the open cases presented in Table 1.

We show as a main result that for $\ell = 3$ and $m, n \geq 12$ the solutions of the equations $u_1 \cdots u_\ell = v_1 \cdots v_m w_1 \cdots w_n$ must be θ-powers of a common word. The same holds if $m, n \geq 5$ and not both of the values are even. To the same end, we show that if the words u_1, u_2 and u_3 are not all equal, or if $|v_1 \cdots v_m| \geq 2|u|$, then the solutions of the aforementioned equation are, again, θ-powers of a common word. Our results show the surprising fact that the case of $\ell = 3$ is the only one when we have both general equations $u_1 \cdots u_\ell = v_1 \cdots v_m w_1 \cdots w_n$ that have only solutions which are θ-powers, and general equations that may have solutions which are not θ-powers.

As expected (see the final remarks of [1]), we applied some arguments that have not been used in this context before, but an exhaustive case analysis on the alignments of parts of the equation seems unavoidable and these arguments must be adapted to every case separately. Due to space restrictions, some of the proofs (or parts thereof) had to be omitted.

■ **Basic concepts.** For more detailed definitions we refer to [5]. For a finite alphabet Σ, we denote by Σ^* and Σ^+ the set of all words and the set of all non-empty words over Σ, respectively. The empty word is denoted by ε and the length of a word w is denoted by $|w|$. For a word $w = uvz$ we say that u is a *prefix* of w, v is a *factor* of w, and z is a *suffix* of w. We denote that by $u \leq_p w$, $v \leq_f w$, and $v \leq_s w$, respectively. If $vz \neq \varepsilon$ we call u a *proper prefix*, and we denote that by $u <_p w$, and symmetrically for suffixes. Similarly, v is called a *proper factor* of w, denoted by $v <_f w$, if $u \neq \varepsilon, z \neq \varepsilon$. A word w is called *primitive*, if $w = u^k$ implies $k = 1$ and $u = w$; otherwise, w is called *power* or repetition. For a word w, we define the word w^ω as the infinite word whose prefix of length $n|w|$ is w^n, for all $n \in \mathbb{N}$. Primitive words are characterised as follows:

▶ **Proposition 1.** *If w is primitive and $ww = xwy$, then either $x = \varepsilon$ or $y = \varepsilon$.*

A word w is called *θ-primitive*, if $w = u_1 \cdots u_k$ with $u_i \in \{u, \theta(u)\}$ for all $1 \leq i \leq k$ implies $k = 1$ and $u = w$. A θ-primitive word is primitive, but the converse does not hold, as $w = abba$ is primitive but $w = ab\theta(ab)$, for θ being the mirror image. A word w is a *θ-palindrome* if $w = \theta(w)$. A word that is not θ-primitive is called a *θ-power*. Kari et al. [4] characterised θ-primitive words similarly to Proposition 1:

▶ **Lemma 1.** *For a θ-primitive word $x \in \Sigma^+$, neither $x\theta(x)$ nor $\theta(x)x$ can be a proper factor of a word in $\{x, \theta(x)\}^3$.*

The results of Proposition 2 and Theorem 2 are well known (see, e.g., [5]):

▶ **Proposition 2.** *If $xz = zy$ for some words $x, y, z \in \Sigma^*$, then there exist $p, q \in \Sigma^*$, such that $x = pq, y = qp$, and $z = (pq)^i p$ for some $i \geq 0$.*

The words x, y from Proposition 2 are called *conjugates*, denoted by $x \sim y$.

▶ **Theorem 2.** *If $\alpha \in u\{u, v\}^*$ and $\beta \in v\{u, v\}^*$ have a common prefix of length at least $|u| + |v| - \gcd(|u|, |v|)$, then $u, v \in \{t\}^+$ for some word t.*

Czeizler et al. [2] established the following two generalisations of Theorem 2:

▶ **Theorem 3.** *Let $u, v \in \Sigma^+$ with $|u| \geq |v|$. If $\alpha \in \{u, \theta(u)\}^+$ and $\beta \in \{v, \theta(v)\}^+$ have a common prefix of length at least $2|u| + |v| - \gcd(|u|, |v|)$, then $u, v \in t\{t, \theta(t)\}^+$ for some θ-primitive word $t \in \Sigma^+$.*

▶ **Theorem 4.** *Let $u, v \in \Sigma^+$ with $|u| \geq |v|$. If $\alpha \in \{u, \theta(u)\}^+$ and $\beta \in \{v, \theta(v)\}^+$ have a common prefix of length at least $\operatorname{lcm}(|u|, |v|)$, then $u, v \in t\{t, \theta(t)\}^+$ for some θ-primitive word $t \in \Sigma^+$.*

Harju and Nowotka [3] investigated equations that are similar to the ones by Lyndon and Schützenberger with the following result, which we use in our proofs:

▶ **Theorem 5.** *Let $n \geq 2$ and $x, z_i \in \Sigma^*$ with $|x| \neq |z_i|$ and $k, k_i \geq 3$, for all $1 \leq i \leq n$. If $x^k = z_1^{k_1} z_2^{k_2} \cdots z_n^{k_n}$ and $n \leq k$, then $x, z_i \in \{t\}^*$ for some word $t \in \Sigma^*$ and all $1 \leq i \leq n$.*

2 Overview

As mentioned in the Introduction, we are interested in solutions of the equation

$$u_1 u_2 u_3 = v_1 \cdots v_m w_1 \cdots w_n, \tag{1}$$

where $m, n \geq 5$, $u_1, u_2, u_3 \in \{u, \theta(u)\}$, $v_j \in \{v, \theta(v)\}$ for all $1 \leq j \leq m$ and $w_k \in \{w, \theta(w)\}$ for all $1 \leq k \leq n$.

Our main results are the following theorems. The first one shows that (1) has only pseudoperiodic solutions when the sequence $v_1 \cdots v_m$ is long enough.

▶ **Theorem 6.** *If $m|v| \geq 2|u|$, then (1) implies that $u, v, w \in \{t, \theta(t)\}^+$ for some word t.*

A similar result is obtained when not all of u_1, u_2, and u_3 are the same.

▶ **Theorem 7.** *If $\{u_1, u_2, u_3\} = \{u, \theta(u)\}$, then (1) implies that $u, v, w \in \{t, \theta(t)\}^+$ for some word t.*

Finally, if m and n are large enough, or at least one of these values is odd, (1) has only solutions which are θ-powers of the same word, with no additional restrictions on u, v or w:

▶ **Theorem 8.** *If $m, n \geq 12$, then (1) implies that $u, v, w \in \{t, \theta(t)\}^+$ for some word t.*

▶ **Theorem 9.** *If m or n is odd, then (1) implies that $u, v, w \in \{t, \theta(t)\}^+$ for some word t.*

The proofs of these theorems follow a common pattern. We first note that it is enough to prove the statements for the case when v and w are θ-primitive. Then we assume for the sake of a contradiction that $\theta(v) \neq w \neq v \neq \theta(w)$. In this setting, $|v| = |w|$ leads easily to a contradiction, so we assume that $|v| \neq |w|$. Further, working under the particular assumptions of each of the above theorems, we try to find a long enough factor of $u_1u_2u_3$ that reflects an alignment between some v factors and some w factors, allowing us to apply periodicity results like Theorems 2 or 3. In some cases, this is already enough in order to reach a contradiction: the longer word appears as a θ-power. However, sometimes we only get that a (well determined) conjugate of the longer word is a θ-power of the shorter one. As a final step in our proofs we show that such a situation leads to a contradiction, as well. While the first steps of these proofs are based on a deep (and, maybe, finer compared to [1,4]) analysis of the alignments between the v's and w's and their consequences on the form of these words, several length-related arithmetic and combinatorial arguments (that enrich the toolbox developed in [1,4]) were needed to conclude them.

One of the drawbacks of our proofs is that, although they are based on the same strategy, we were not able to reorganise them as a collection of shorter general lemmas from which the final result of each case follows. Mainly, this was because each of the cases we analyse below leads to significantly different alignments between the v and w factors and using them to obtain the final result in the way described above requires some particular technicalities.

Note that this paper does not address the case of equations with $\ell = 3, m = 4$, and odd $n \geq 3$, left open in [1,4] (see Table 1). We conjecture, though, that our results and proofs can be adapted to that case as well. A general result in the line of Theorem 8 remains, however, to be found both for the case when $m, n \geq 6$ such that both m and n are even and at least one of them is less than 12, as well as for the case $m = 4$ and odd $n \geq 3$.

3 The Proofs

We always assume that $v_1 = v$ and often we assume that both v and w are θ-primitive. Otherwise, if for instance $v \in \{v', \theta(v')\}^+$ for some word v', we consider the equation $u_1u_2u_3 = v'_1 \cdots v'_{m'} w_1 \cdots w_n$ instead, where $v'_i \in \{v', \theta(v')\}^+$, for all $1 \leq i \leq m'$, with $m' > m$, and similarly if $w \in \{w', \theta(w')\}^+$ for some word w'. Moreover, if (1) holds and two of u, v, w are in $\{t, \theta(t)\}^+$ for some word t, then so is the third.

We split the discussion into different sections depending on the length of $v_1 \cdots v_m$. One particularly easy case follows from Theorem 3.

▶ **Lemma 10.** *If $m|v| \geq 2|u| + |v|$ and (1) holds, then $u, v, w \in \{t, \theta(t)\}^+$ for some word t.*

Proof. By Theorem 3, we instantly get that $u, v \in \{t, \theta(t)\}^+$ for some θ-primitive word t. From this, one can get easily that $w \in \{t, \theta(t)\}^+$, as well. ◀

3.1 The case $2|u| < m|v| < 2|u| + |v|$: Proof of Theorem 6.

In this section, we frequently use the following results from [1].

▶ **Proposition 3** (Prop. 20 and 21 in [1]). *Let $u, v \in \Sigma^+$ so that v is θ-primitive, $u_1, u_2, u_3 \in \{u, \theta(u)\}$ and $v_1, \ldots, v_m \in \{v, \theta(v)\}$ for $m \geq 3$. Assume that $v_1 \cdots v_m <_p u_1u_2u_3$ and $2|u| < m|v| < 2|u| + |v|$. If m is odd, then $u_2 \neq u_1$ and $v_1 = \cdots = v_m$.*
If m is even, then one of the following holds:
1. *$u_1 \neq u_2$ and $v_1 = \cdots = v_m$, or*
2. *$u_1 = u_2$, $v_1 = \cdots = v_{\frac{m}{2}}$ and $v_{\frac{m}{2}+1} = \cdots = v_m = \theta(v_1)$.*

We split the discussion further according to every valuation of u_1, u_2 and u_3.

■ **Equations of the form** $u\theta(u)u = v_1 \cdots v_m w_1 \cdots w_n$**.** The following holds:

▶ **Lemma 11.** *If* $2|u| < m|v| < 2|u| + |v|$ *and* $u_1u_2u_3 = u\theta(u)u$ *and (1) holds, then* $u, v, w \in \{t, \theta(t)\}^+$ *for some word* t*.*

Proof. By Proposition 3 we get $v_1 \cdots v_m = v^m$. By the explanations given above, we assume that v and w are θ-primitive.

If $2n|w| < 2|v| + |w|$, we get that $|w| < \frac{2|v|}{2n-1} < \frac{|v|}{4}$. Now, if $m = 5$, we see that $u = v^2y$ with $|y| < \frac{|v|}{2}$ and $y\theta(y) \leq_p v$. Furthermore, the part of v_5 that overlaps with u_3 is of length $|v| - 2|y|$. Hence, $v = (y\theta(y))^k v'$ for some $k \geq 1$ and $v' \leq_p y\theta(y)$. From the length of this overlap we also get that $|w_1 \cdots w_n| = (2|v| + |y|) - (|v| - 2|y|) = |v| + 3|y|$ and, as $2n|w| < 2|v| + |w|$, we have $2(|v| + 3|y|) < 2|v| + |w|$. It follows that $6|y| < |w|$, and thus $|v| > 4|w| > 24|y|$. Therefore we actually have $v = (y\theta(y))^k v'$ with $k \geq 12$. As a consequence, $\theta(y)(y\theta(y))^{k-1}v'$ is a prefix of $\theta(u)$. As $\theta(w_n) \cdots \theta(w_1)$ also is a prefix of $\theta(u)$, it has a common prefix with $\theta(y)(y\theta(y))^{k-1}v'$ of length at least $\frac{|v|}{2} + |y| > 2|w| + |y|$. So we can apply Theorem 3, and get that w is not θ-primitive, a contradiction. In the case $m \geq 6$ we see that $|u| \geq \frac{5|v|}{2}$ must hold, so $|w_1 \cdots w_n| \geq |u| - |v| \geq \frac{3|v|}{2}$, and thus $2n|w| \geq 3|v| > 2|v| + |w|$, a contradiction.

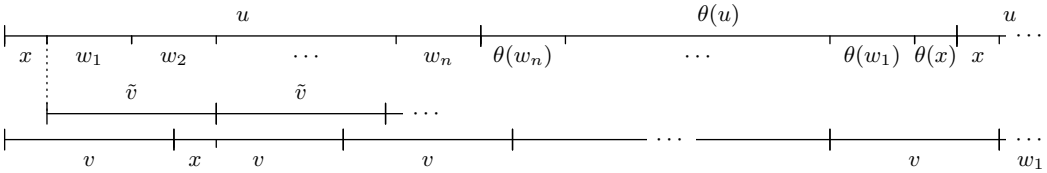


Figure 1 The alignment of $\tilde{v}^{m-1}$ with $w_1 \cdots w_n\theta(w_n) \cdots \theta(w_1)$.

Consequently, we have $2n|w| \geq 2|v| + |w|$. Then we can apply Theorem 3 to the factor $w_1w_2 \cdots w_n\theta(w_n)\theta(w_{n-1}) \cdots \theta(w_1)$, centred on the border between u and $\theta(u)$, and the factor $\tilde{v}^{m-1}$, where $\tilde{v} \sim v$ and $\tilde{v}$ appears after the prefix of length $|u| - n|w|$ in u. We get that $\tilde{v} \in \{w, \theta(w)\}^+$, as we assumed w to be θ-primitive. Clearly, $|w|$ divides $|v|$.

As $\tilde{v} \sim v$, it follows that the prefix of length $|u| - n|w|$ of v^m has the form $x\{w, \theta(w)\}^*$, with $|x| < |w|$. So u has the same form and furthermore the factor $v' \sim v$ occurring in u after the prefix x is in $\{w, \theta(w)\}^+$ as well (note that in Figure 1, we have $\tilde{v} = v'$, but this is so just to simplify the figure, and not the case in general, when v' is obtained as explained). Moreover, exchanging w and $\theta(w)$ if necessary, we can assume that $u \in xw\{w, \theta(w)\}^+$. If $x = \varepsilon$, we have $u \in \{w, \theta(w)\}^+$ and $v \in \{w, \theta(w)\}^+$ and since we assumed that v is θ-primitive, it follows that $v \in \{w, \theta(w)\}$, and the statement holds with $t = w$. Thus, assume $|x| > 0$. Since $|w|$ divides $3|u| \equiv 3|x| \mod 3|w|$, it follows that $|w|$ divides $3|x|$. But $|x| < |w|$, so either $3|x| = 2|w|$ or $3|x| = |w|$. In both cases, 3 divides $|w|$.

If $|w| = 3|x|$, we have $u\theta(u) \in x\{w, \theta(w)\}^+\theta(x)$. Since $m|v| > |u\theta(u)|$ and $3|x|$ divides $|v|$ we get that $m|v| = 2|u| + \ell|w| + |x|$, for some integer $\ell \geq 0$. Thus, a prefix of length $2|x|$ of w or of $\theta(w)$ occurs after the prefix of length $2|u| - |x|$ in $u\theta(u)u$. We have $w, \theta(w) \notin \{x, \theta(x)\}^3$, as w is θ-primitive. Hence, if $\theta(x)x \leq_p w$ then $w = \theta(x)xy$ for some word y with $|y| = |x|$ and $y \notin \{x, \theta(x)\}$, and if $\theta(x)x \leq_p \theta(w)$ then $w = \theta(y)\theta(x)x$ with y as above.

Further, we analyse what values $m|v|$ might have. For $\ell = 0$, i.e., $m|v| = 2|u| + |x|$, we have that $v^m = u\theta(u)x$. As $\theta(u)$ ends with $\theta(x)$, we have $\theta(x)x \leq_s v$. Thus, $\theta(x)xx \leq_s v'$, and it follows that $w \in \{x, \theta(x)\}^3$, a contradiction. So, $m|v| > 2|u| + |x|$. However, because

one of w or $\theta(w)$ occurs as a factor of v after the prefix of length $2|u| - |x|$ in $u\theta(u)u$, we get that the factor of length $|x|$ starting after a prefix of length $2|u|+|x|$ in $u\theta(u)u$ is neither x nor $\theta(x)$. Thus, it can only be y. Now, one of w or $\theta(w)$ occurs after the prefix of length $2|u|+|x|$ in $u\theta(u)u$ as well, by the fact that there exists a sequence of w's and $\theta(w)$'s that starts there and is a suffix of $u\theta(u)u$. In both cases, unless $x \in \{y, \theta(y)\}$, it follows immediately that $y = \theta(y)$ and $y\theta(x)x$ appears after a prefix of length $2|u| + |x|$ in $u\theta(u)u$. However, the prefix of length $2|u| + |w| + |x|$ of $u\theta(u)u$ ends with $\theta(x)x$. By the same reasoning as above we get that v^m cannot end here and so $m|v| > 2|u| + |w| + |x|$. We repeat this reasoning to see that, actually, $m|v| \neq 2|u| + \ell|w| + |x|$ for all $\ell \geq 0$. However, $m|v|$ should have this form. Therefore, we reached a contradiction in this case.

The reasoning for the case $3|x| = 2|w|$ is similar and omitted here. ◀

■ **Equations of the form** $uu\theta(u) = v_1 \cdots v_m w_1 \cdots w_n$**.** These are the only equations of the form (1) with $\ell = 3$ that were already investigated (see [4]), with the following result:

▶ **Lemma 12** (Proposition 51 in [4])**.** *If* $2|u| < m|v| < 2|u| + |v|$ *and* $u_1u_2u_3 = uu\theta(u)$ *and (1) holds, then* $u, v, w \in \{t, \theta(t)\}^+$ *for some word* t*.*

■ **Equations of the form** $u\theta(u)\theta(u) = v_1 \cdots v_m w_1 \cdots w_n$**.** In this case we were able to establish the following result.

▶ **Lemma 13.** *If* $2|u| < m|v| < 2|u| + |v|$ *and* $u_1u_2u_3 = u\theta(u)\theta(u)$ *and (1) holds, then* $u, v, w \in \{t, \theta(t)\}^+$ *for some word* t*.*

Proof. By Proposition 3, we know that $v_1 = \ldots = v_m = v$. We assume that v and w are θ-primitive. We analyse first the case of m being even.

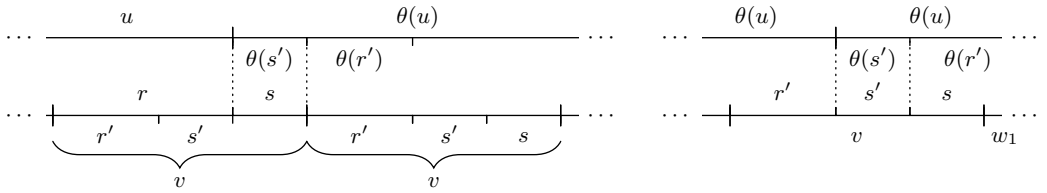


Figure 2 The situation at the two borders u_1u_2 and u_2u_3.

Case $m = 6$**.** We have $u_1 = vvr$, where $v = rs$ and $|r| \geq |s|$. As v is θ-primitive we can assume that $|r| > |s|$, as otherwise, we had $v = r\theta(r)$. Hence, let r' be the prefix of length $|r| - |s|$ of r and s' be the suffix of r such that $r = r's'$ (see Figure 2).

From the border between $u_1 = u$ and $u_2 = \theta(u)$, we can see that $r's'sr' = r'\theta(s)\theta(s')\theta(r')$. It follows that $\theta(s) = s'$ and $\theta(r') = r'$. Now, looking at the border between $u_2 = \theta(u)$ and $u_3 = \theta(u)$, we get that $\theta(s')r' \leq_p \theta(u)$ and $s's \leq_p \theta(u)$. It follows that $s = \theta(s) = s'$.

Subcase $|r'| < |s|$**.** In this case $r' \leq_p s$. Therefore, $s = \theta(s)$ ends with $\theta(r') = r'$. As a consequence, we get that $w_1 \cdots w_n = r'sr'ssr'$, and so $u_3 = ssr'sr'ssr'$. However, also $u_3 = sr'ssr'ssr'$ holds, thus $r's = sr'$ and v is not primitive.

Subcase $|r'| \geq |s|$**.** In this case, let $r' = sp$. As v is primitive, $|p| > 0$ must hold. We have $w_1 \cdots w_n = ps^3ps^3p$. Hence, $ps^3p = w_1 \cdots w_k w' = w''w_{n-k+1} \cdots w_n$, with $k \geq 2$. By Lemma 1 we get that $w_1 = \ldots = w_k$ and $w_{n-k+1} = \ldots = w_n$. Since w_1 is primitive, $w_1 \neq w_{n-k+1}$ must hold, thus $w_n = \theta(w_1)$.

If $|p| \leq |w|$, we get that $p \leq_p w_1$, so $\theta(p) \leq_s w_n = \theta(w_1)$. However, $p \leq_s w_n$ holds as well, and so $p = \theta(p)$. Yet, we have $sp = r' = \theta(r') = \theta(p)\theta(s) = ps$. This shows that $p, s \in \{t\}^+$ for some word t, so $v \in t\{t\}^+$, a contradiction.

If $|p| > |w|$ we can apply Theorem 3 to ps^3ps^3p and $w_1 \cdots w_n$ and obtain that $ps^3, p \in \{w, \theta(w)\}^+$. It follows that $s^3 \in \{w, \theta(w)\}^+$. It is not hard to see that this leads again to a contradiction with the primitivity of v or of w.

Case $m \geq 8$**.** We follow the exact same steps as above before splitting the discussion into the cases $|r'| < |s|$ and $|r'| \geq |s|$.

If $|r'| < |s|$, we get that $w_1 \cdots w_n = r'sr'(ssr')^k$ with $k \geq 2$. As $r'sr'$ is a suffix of ssr' of length at least $\frac{|ssr'|}{2}$ and $n \geq 5$, we can apply Theorem 3 and obtain that $ssr' \in \{w, \theta(w)\}^+$ and thus also $r'sr' \in \{w, \theta(w)\}^+$. As $|r'| < |s|$, the word ssr' is not θ-primitive, but $ssr' = \theta(v)$, a contradiction. If $|r'| \geq |s|$, we get that $w_1 \cdots w_n = p(s^3p)^k$ with $k \geq 3$. By Theorem 3, it follows that $s^3p \in \{w, \theta(w)\}^+$ and $p \in \{w, \theta(w)\}^+$, thus s^3p is not θ-primitive. However, $s^3p = ssr' = \theta(v)$, again a contradiction.

This concludes the analysis for even m. The case when m is odd had to be omitted due to the space restrictions. ◀

■ **Equations of the form** $uuu = v_1 \cdots v_m w_1 \cdots w_n$**.** We show the following:

▶ **Lemma 14.** *If* $2|u| < m|v| < 2|u| + |v|$ *and* $u_1u_2u_3 = uuu$ *and (1) holds, then* $u, v, w \in \{t, \theta(t)\}^+$ *for some word* t*.*

Proof. Assume that v and w are θ-primitive. By Proposition 3, m is even, $v_1 = \ldots = v_k = v$, and $v_{k+1} = \ldots = v_m = \theta(v)$, where k is such that $(k-1)|v| < |u| < k|v|$. Furthermore, as $m|v| < 2|u| + |v|$, the prefix of v_k occurring as a suffix of u is longer than $\frac{|v|}{2}$. From the border between u_1 and u_2 we get that $v = rpr$ with $r = \theta(r)$ and $pr = r\theta(p)$. The solutions of this equation are, clearly, $r = (\alpha\beta)^i\alpha, p = (\alpha\beta)^j, \theta(p) = (\beta\alpha)^j$ for some θ-palindromes α and β, $i \geq 0, j \geq 1$, and $\alpha\beta$ primitive.

Furthermore, if $w = \theta(w)$, then (1) is actually $u^3 = v^{\frac{m}{2}}\theta(v)^{\frac{m}{2}}w^n$. As $m, n \geq 5$ and m is even, we can apply Theorem 5 to get that $u, v, \theta(v), w \in \{t\}^+$ for some word t, and the statement of this lemma holds. Therefore, we also assume $w \neq \theta(w)$ in the following.

We have $u_1 = v^{\frac{m}{2}-1}rp$ and $u_2 = r\theta(v)^{\frac{m}{2}-1}v'$ where $v' \leq_p \theta(v)$ and $|v'| = |p|$. Since $v = rpr$, the suffix of $v_m = \theta(v)$ that is a prefix of u_3 is of length $|rr|$. Furthermore, since $u_3 = u$ starts with $v = rpr = rr\theta(p)$ (as $pr = r\theta(p)$), we get that $w_1 \cdots w_n = \tilde{v}^{\frac{m}{2}-2}\theta(p)rp$, where $\tilde{v} = \theta(p)rr \sim v$. We will show that for $m \geq 8$, this equation leads to a contradiction:

First of all, $\theta(p)rp \leq_p \theta(p)rpr = \theta(p)rr\theta(p)$ and thus $w_1 \cdots w_n \leq_p \tilde{v}^{\frac{m}{2}}$.

If $|r| < |p|$, then $\tilde{v}^{\frac{m}{2}-2}\theta(p)rp = \tilde{v}^{\frac{m}{2}-1}p'$ for some $p' \leq_s p$ with $|p'| < |p|$. Since $m \geq 8$, this word is of length at least $3|v|$. Thus, if $|v| \geq |w|$, Theorem 3 is applicable and we get $\tilde{v}, w \in \{t, \theta(t)\}^+$ for some word t. On the other hand, if $|v| < |w|$, then as $|p'| < |p| < |v|$, the word $w_1 \cdots w_{n-1}$ is a prefix of $\tilde{v}^\omega$. As $m \geq 5$, this prefix is of length at least $2|w| + |v|$, and by Theorem 3, we get $\tilde{v}, w \in \{t, \theta(t)\}^+$ in this case as well. Since w is θ-primitive, $\tilde{v} \in \{w, \theta(w)\}^+$ must hold. By the assumption that $|r| < |p|$, and because $pr = r\theta(p)$, we can write $p = rs$ for some word s. Then, since $\theta(p)rr \in \{w, \theta(w)\}^+$ and $w_1 \cdots w_n = (\theta(p)rr)^{\frac{m}{2}-2}\theta(p)rp$, also $\theta(p)rrs = \theta(p)rp \in \{w, \theta(w)\}^+$ holds. Combining these last two results, we see that $s \in \{w, \theta(w)\}^+$ and thus also $\theta(s) \in \{w, \theta(w)\}^+$. However, as $\theta(p)rr = \theta(s)rrr \in \{w, \theta(w)\}^+$, by Theorem 4, also $r \in \{w, \theta(w)\}^+$. As a consequence, $p = rs \in \{w, \theta(w)\}^+$, and so $v = rpr \in \{w, \theta(w)\}^+$, contradicting the θ-primitivity of v.

If $|r| \geq |p|$, then $\theta(p)rp \leq_p \theta(p)rr$, so the words $w_1 \cdots w_n$ and $\tilde{v}^\omega$ have a common prefix of length at least $\max\{(\frac{m}{2} - 1)|v|, 5|w|\}$. If $m \geq 10$, this is at least $\max\{3|v|, 5|w|\}$ which is always long enough to apply Theorem 3 to get that $\tilde{v}, w \in \{w, \theta(w)\}^+$. In the case $m = 8$, we have $w_1 \cdots w_n = \tilde{v}^2\theta(p)rp$. If $|w| > |\theta(p)rp|$, then $n|w| > |\tilde{v}^2\theta(p)rp|$, as $|\theta(p)rp| > \frac{|v|}{2}$, a contradiction. Thus, $|w| \leq |\theta(p)rp|$, and so we have a common prefix of $\tilde{v}^\omega$ and $w_1 \cdots w_n$ of length $2|v| + |w|$. By Theorem 3, once again, we get $\tilde{v} \in \{w, \tilde{w}\}^+$, as w is

θ-primitive. Now, dually to the previous case, we write $r = ps'$. As $\theta(p)rr = \theta(p)rps'$ and $\theta(p)rp$ are both in $\{w, \theta(w)\}^+$, so is s'. Furthermore, as $\theta(p)rp = \theta(p)ps'p \in \{w, \theta(w)\}^+$, if $\theta(p)p \in \{w, \theta(w)\}^+$, then by Theorem 4, also $p \in \{w, \theta(w)\}^+$, and so $r = ps' \in \{w, \theta(w)\}^+$. This is a contradiction, since $v = rpr$ is θ-primitive. Therefore, $p\theta(p) \notin \{w, \theta(w)\}^+$, which means that $s' \in \{w, \theta(w)\}^+$ is a proper factor of some word in $\{w, \theta(w)\}^+$. By Lemma 1, we must have that $s' \in \{w\}^+$ or $s' \in \{\theta(w)\}^+$, as w is θ-primitive. However, $pps' = pr = r\theta(p) = ps'\theta(p)$, so $ps' = s'\theta(p)$, and we saw before that this means that s' is a θ-palindrome. In conclusion, $w = \theta(w)$ in both cases, and we get a contradiction.

Therefore, as m must be even, the only case left is when $m = 6$, in which (1) is of the form $uuu = vvv\theta(v)\theta(v)\theta(v)w_1 \cdots w_n$.

We shift our attention to the factor $w_1 \cdots w_n$. As $m = 6$, we know that $u = rpr^2pr^2p = r^2\theta(p)rpr^2p$ and $w_1 \cdots w_n$ starts after a prefix of length $2|r|$ in u, so $w_1 \cdots w_n = \theta(p)rpr^2p = (\beta\alpha)^j(\alpha\beta)^i\alpha(\alpha\beta)^{i+j}\alpha\alpha(\beta\alpha)^i(\alpha\beta)^j$. Since α and β are θ-palindromes, so is $w_1 \cdots w_n$.

If n is odd, from $w_1 \cdots w_n = \theta(w_1 \cdots w_n)$ we immediately get that $w_{\frac{n+1}{2}} = \theta(w_{\frac{n+1}{2}})$. It follows that $w = \theta(w)$, which contradicts the assumption $w \neq \theta(w)$ we made at the beginning.

So we can further assume n to be even and so $n \geq 6$.

If $(\beta\alpha)^j(\alpha\beta)^i\alpha(\alpha\beta)^j \in \{w, \theta(w)\}^+$, then also $(\beta\alpha)^j(\alpha\beta)^i\alpha(\alpha\beta)^i\alpha \in \{w, \theta(w)\}^+$. Thus, if $i \geq j$, then $(\alpha\beta)^{i-j}\alpha \in \{w, \theta(w)\}^+$, and if $i < j$, then $(\beta\alpha)^{j-i-1}\beta \in \{w, \theta(w)\}^+$. In both cases, those words are θ-palindromes, so since $w \neq \theta(w)$, either $w\theta(w)$ or $\theta(w)w$ occurs as a factor in them.

If $i \geq j$, the factor $(\alpha\beta)^{i-j}\alpha$ appears in $w_1 \cdots w_n$ after the prefix $(\beta\alpha)^j$. By Lemma 1, we must have $(\beta\alpha)^j \in \{w, \theta(w)\}^+$ and by Theorem 4 thus $\beta\alpha \in \{w, \theta(w)\}^+$. Together with $(\alpha\beta)^{i-j}\alpha \in \{w, \theta(w)\}^+$, this leads to $\alpha, \beta \in \{w, \theta(w)\}^+$, which contradicts the θ-primitivity of v.

If $j > i$ and $i > 0$, then $(\beta\alpha)^{j-i-1}\beta$ appears inside the factor $(\beta\alpha)^{i+j}$ both as a prefix and after the prefix $\beta\alpha$. Thus, in this case $\beta\alpha \in \{w, \theta(w)\}^+$ as well, which again leads to $\alpha, \beta \in \{w, \theta(w)\}^+$. If $j > i$ and $i = 0$, then $(\beta\alpha)^j(\alpha\beta)^i\alpha(\alpha\beta)^j = (\beta\alpha)^j\alpha(\alpha\beta)^j$, and $(\beta\alpha)^j(\alpha\beta)^i\alpha(\alpha\beta)^i\alpha = (\beta\alpha)^j\alpha\alpha$. So we immediately get that $(\beta\alpha)^j \in \{w, \theta(w)\}^+$, which leads to the same contradiction as above.

By the previous paragraphs, we can assume that $(\beta\alpha)^j(\alpha\beta)^i\alpha(\alpha\beta)^j = w_1 \cdots w_\ell w'$ for some ℓ, and some nonempty $w' \leq_p w_{\ell+1}$. As $(\beta\alpha)^j(\alpha\beta)^i\alpha(\alpha\beta)^j$ appears also as a suffix of $w_1 \cdots w_n$, we have $w_1 = \cdots = w_\ell$ by Lemma 1. Without loss of generality, let $w_1 = w$. Since $|(\beta\alpha)^j(\alpha\beta)^i\alpha| = \frac{n}{3}|w|$, and $n \geq 6$, we get that $ww \leq_p (\beta\alpha)^j(\alpha\beta)^i\alpha$.

Now, if $i \geq j$, we can write $w^\ell w' = (\beta\alpha)^j(\alpha\beta)^j(\alpha\beta)^{i-j}\alpha(\alpha\beta)^j$. We observe that $w \leq_p (\beta\alpha)^j(\alpha\beta)^j$ must hold: Assume towards a contradiction, that $|w| > 2j|\alpha\beta|$. Then, the second w of the prefix ww of $(\beta\alpha)^j(\alpha\beta)^i\alpha$ begins inside the factor $(\alpha\beta)^{i-j}\alpha$. Since w starts with $\beta\alpha$ and this is primitive, we deduce that $w = (\beta\alpha)^j(\alpha\beta)^j(\alpha\beta)^k$ for some k. However, then the second occurrence of w that follows immediately afterwards is a prefix of $(\beta\alpha)^{i-j-k}(\alpha\beta)^j$. This is only possible if $\alpha\beta = \beta\alpha$, which is a contradiction to the primitivity of $\alpha\beta$. Thus we can safely assume that $w \leq_p (\beta\alpha)^j(\alpha\beta)^j$. This word $(\beta\alpha)^j(\alpha\beta)^j$ is a suffix of $w^\ell w' = (\beta\alpha)^j(\alpha\beta)^{i-j}(\alpha\beta)^j\alpha(\alpha\beta)^j$. Since w is assumed to be primitive, by Lemma 1, we must have $(\beta\alpha)^j\alpha(\alpha\beta)^{i-j} \in \{w\}^+$. Let $y = (\beta\alpha)^j\alpha(\alpha\beta)^{i-j}$. Then $w_1 \ldots w_n = yy(\beta\alpha)^{2j}(\alpha\beta)^j\theta(y)$, from which we conclude that $(\beta\alpha)^{2j}(\alpha\beta)^j \in \{w, \theta(w)\}^+$. Applying Lemma 4 now gives us $\alpha\beta \in \{w, \theta(w)\}^+$, from which we deduce the contradiction $\alpha, \beta \in \{w, \theta(w)\}^+$ as before.

On the other hand, if $i < j$, then $|w| < |(\beta\alpha)^j|$, since $n \geq 6$. Furthermore $w_1 \cdots w_\ell w' = (\beta\alpha)^j(\alpha\beta)^i\alpha(\alpha\beta)^j$ is then the rest of $w_1 \cdots w_n$, so $\ell \geq \frac{n}{2}$. Therefore, we got $w_1 \cdots w_n = w^{\frac{n}{2}}\theta(w)^{\frac{n}{2}}$. If $|w| < |(\beta\alpha)^{j-1}|$, we would have $|w|$ occurring as a prefix of $w_1 \cdots w_n$ and after the prefix $\beta\alpha$. Thus $w = \beta\alpha$ by Lemma 1. However, then $w_{j+1} = w = \alpha\beta$, contradicting the

primitivity of $\alpha\beta$. Hence, $|(\beta\alpha)^{j-1}| < |w| < |(\beta\alpha)^j|$.

If $j \geq 2$, then $(\beta\alpha)^{j-1} < |w|$ and $i < j$ imply that $|(\beta\alpha)^j(\alpha\beta)^i\alpha| < 3|w|$. Therefore $n < 9$, and as n is even, either $n = 8$ or $n = 6$. If $n = 8$, then w_4w_5 must be a factor of $(\alpha\beta)^{i+j}\alpha$, and since $j \geq 2$, the word $\beta\alpha$ is a prefix of $w_4 = w$. Using Lemma 1, this $\beta\alpha$ must be aligned with some $\beta\alpha$ inside $(\alpha\beta)^{i+j}\alpha$. This allows us to deduce that $j = i + 1$, and that $w_4 = w = (\beta\alpha)^{j-1}\beta'$, where $\beta = \beta'\theta(\beta')$. Then, $w_2 = w \leq_p \theta(\beta')\alpha(\alpha\beta)^{j-1}$. Now if $j \geq 3$, the factor $\alpha\beta$ appears as a proper factor inside $(\alpha\beta)^2$, unless $\beta = \theta(\beta')\alpha$. However, if $\beta = \theta(\beta')\alpha$, then $\alpha = \theta(\beta')$, and thus $\alpha\beta$ is not θ-primitive. Therefore $j = 2$ must hold, in which case we get that $\beta\alpha\beta' \leq_p \theta(\beta')\alpha\alpha\beta$. From this it immediately follows that α is not primitive, and furthermore that $\alpha \in \{\theta(\beta')\}^+$, again a contradiction to $\alpha\beta$ being θ-primitive. If $n = 6$, then $w_1w_2 = w^2 = (\beta\alpha)^j(\alpha\beta)^i\alpha$ and $w_3w_4 = w\theta(w) = (\alpha\beta)^{i+j}\alpha$. As $|w| \geq |\beta\alpha|$, we get the contradiction $\beta\alpha = \alpha\beta$.

Thus the only possibility that remains is $j = 1$ and thus $i = 0$. This means that $w^{\frac{n}{2}}\theta(w)^{\frac{n}{2}} = \beta\alpha^3\beta\alpha^3\beta$. By concatenating α^3 to the left on both sides, we get $\alpha^3w^{\frac{n}{2}}\theta(w)^{\frac{n}{2}} = (\alpha^3\beta)^3$, to which we can apply Theorem 5 to get $w = \theta(w)$, a contradiction. ◀

All the other valuations of $u_1u_2u_3$ follow from the ones considered in the last three paragraphs by the fact that θ is an involution. Hence, Theorem 6 follows.

3.2 The case $m|v| < 2|u|$: Proofs of Theorems 7, 8 and 9.

We continue with the case when the border between v_m and w_1 lies inside u_2.

■ **Equations of the form $uu_2\theta(u) = v_1 \cdots v_m w_1 \cdots w_n$, with $u_2 \in \{u, \theta(u)\}$.** For both possible values of u_2 the following lemma holds:

▶ **Lemma 15.** *If $u_1 = u$, $u_3 = \theta(u)$ and (1) holds, then $u, v, w \in \{t, \theta(t)\}^+$ for some word t.*

Proof. We can assume that $|v| \geq |w|$, otherwise we just change the roles of v and w in the following reasoning. Actually, if $|v| = |w|$, we get that $v_1 = \theta(w_n)$, and that $v, w \in \{v, \theta(v)\}$, so in this case the statement holds.

Therefore we can assume that $|v| > |w|$. Now, if $|u| \geq 3|v|$, then we have $u \leq_p v_1 \cdots v_m$ and $u \leq_p \theta(w_n) \cdots \theta(w_1)$, and $|u| \geq 2|v| + |w|$, so by Theorem 3 we get that $v, w \in \{t, \theta(t)\}^+$ for some word t, so the statement also holds in this case.

Since $m|v| < 2|u|$ and $m \geq 5$, it follows that $m = 5$ and $u = v_1v_2r$ for $v_3 = rs$. Furthermore, again from the facts that $m|v| < 2|u|$ and $m \geq 5$, it follows immediately that $|r| > |s|$. If $|w| \leq |r|$, then u would still be a prefix of $v_1 \cdots v_m$ and $\theta(w_n) \cdots \theta(w_1)$, long enough to apply Theorem 3, so we assume $|w| > |r|$.

As $|u| = 2|v| + |r| = 3|r| + 2|s|$, we get that $|w_1 \cdots w_n| = 3|u| - 5|v| = 3(3|r| + 2|s|) - 5(|r| + |s|) = 4|r| + |s|$, and so as $|r|, |s| < |w|$, this contradicts the fact that $n \geq 5$. ◀

■ **Equations of the form $u\theta(u)u = v_1 \cdots v_m w_1 \cdots w_n$.** We start this paragraph with two simple lemmas, that we use in our proofs. Their proofs are left out here.

▶ **Lemma 16.** *Let p and r be θ-palindromes such that $r \leq_p p$ and $\frac{|p|}{2} < |r| < |p|$. If p and r are not primitive, then neither is pr.*

▶ **Lemma 17.** *If $u\theta(u)u = v^m w_1 \cdots w_n$, $|w| > |v|$, $|u| > 3|w|$, and $|u| < m|v| < 2|u|$, then w is not θ-primitive.*

We analyse (1) for all possible relations between $|v|$ and $|w|$:

▶ **Lemma 18.** *If $u\theta(u)u = v_1 \cdots v_m w_1 \cdots w_n$, $|w| > |v|$ and $\frac{3}{2}|u| \leq n|w| < 2|u|$, then either v or w is not θ-primitive.*

Proof. We show this by contradiction, and assume that v and w are θ-primitive. By the length-restrictions, we have $u\theta(u) = v_1 \cdots v_m w_1 \cdots w_{i-1} s$, where w_i is the word overlapping with the border between $\theta(u)$ and the second u. We have two cases, depending on the position of w_i on the border between $\theta(u)$ and u.

Case $u = \theta(s)rw_{i+1} \cdots w_n$ and $w_i = s\theta(s)r$. We can assume $r \neq \varepsilon$, as otherwise w would not be θ-primitive, so $|s| < \frac{|w|}{2}$. Hence $i \geq 3$, as otherwise $n|w| \geq \frac{3}{2}|u|$ would not hold.

Furthermore, we can see that $w_{j-1}w_j <_f \theta(w_{2i-j+1})\theta(w_{2i-j})\theta(w_{2i-j-1})$ for all j with $i \geq j \geq 2$. Therefore, by Lemma 1 we have that $w_1 = \ldots = w_i$. By the same arguments and the fact that $u\theta(u)$ is a θ-palindrome, we also get that $w_i = \ldots = w_{2i-2}$ and that $s\theta(s) <_p w_{2i-1}$. Let $N = |w_1 \cdots w_{i-1}s| = (i-1)|w| + |s|$ and $M = N \mod |v|$. That is, M is the difference between the length of $w_1 \cdots w_{i-1}s$ and the longest θ-power of v that occurs as a suffix thereof. Now let $\tilde{w}$ be the conjugate of w_i occurring in $w_1 \cdots w_{i-1}s$ after the prefix of length M. The length of the prefix of $\tilde{w}^\omega$ that starts there is at least $(2i-2)|w| - M + 2|s| \geq 4|w| - M + 2|s| \geq 2|w| + |v|$. Therefore, we can apply Theorem 3 to this prefix and the θ-power of v, that occurs there and is at least as long, to get that $\tilde{w} \in \{v, \theta(v)\}^+$. Thus, $|v|$ divides $|w|$. Since we assume that $|v|$ does not divide $|u|$ (as otherwise the statement would trivially hold), and we have that $|v|$ divides $m|v|+n|w| = 3|u|$, it follows that $|v| = 3d$ for some d with $d \mid |u|$. We let k be such that $(k-1)|v| < |u| < k|v|$, and write $v_k = x_1x_2x_3$. By our previous divisibility reasoning, we have that $|u| = 3(k-1)d+d$ or $|u| = 3(k-1)d + 2d$. We only treat the first case explicitly here. In this case we get that $x_1 \leq_s u$ and $x_2x_3 \leq_p \theta(u)$, so $\theta(x_1) = x_2$. If $v_{k-1} = \theta(v_k)$, then $\theta(x_2) = x_3$ holds, and so v is not θ-primitive. Therefore, $v_{k-1} = v_k$ and, by the same reasoning, $v_{k+1} = v_k$. Repeating this process we get that $v_k = v_{k+1} = \ldots = v_m$ and that $x_1x_2 \leq_p w_1$. Hence, $x_1\theta(x_1) \leq_p w_1$. As $M = 2$, it follows that $x_3 \leq_s w_1$, as otherwise v would not be θ-primitive. Now $x_3x_1 \leq_s u$ and so if $w_n = w_1$, we have that $x_1 = x_3$, a contradiction to the θ-primitivity of v. Similarly, if $w_n = \theta(w_1)$, we have $x_3x_1 = \theta(x_2)\theta(x_1) = x_1\theta(x_1)$, so $x_1 = x_3$ and v is again not θ-primitive, a contradiction. The other case, $|u| = 3(k-1)d+2d$, leads to the same result in an identical fashion, and is left to the reader. Therefore, when $u = \theta(s)rw_{i+1} \cdots w_n$ and $w_i = s\theta(s)r$, one of v, w is not θ-primitive.

Case $u = \theta(s)w_{i+1} \cdots w_n$ and $w_i = rs\theta(s)$. This case can be analysed in a somewhat similar manner. However, due to the page limit, we have to omit this. ◀

The next case follows in a similar way.

▶ **Lemma 19.** *If $u\theta(u)u = v_1 \cdots v_m w_1 \cdots w_n$, $|w| \leq |v|$ and $\frac{3}{2}|u| \leq n|w| < 2|u|$, then either it is the case that v or w is not θ-primitive or $v \in \{w, \theta(w)\}$.*

As a consequence of the previous two lemmas we get the main result of this paragraph, which, together with Theorem 6 and Lemma 15, proves Theorem 7:

▶ **Lemma 20.** *If $|u| < m|v| < 2|u|$ and $u_1u_2u_3 = u\theta(u)u$ and (1) holds, then $u, v, w \in \{t, \theta(t)\}^+$ for some word t.*

■ **Equations of the form $uuu = v_1 \cdots v_m w_1 \cdots w_n$.**

▶ **Lemma 21.** *If $uuu = v_1 \cdots v_m w_1 \cdots w_n$, $|u| < m|v| < 2|u|$, at least one of m and n is odd, and (1) holds, then $u, v, w \in \{t, \theta(t)\}^+$ for some word t.*

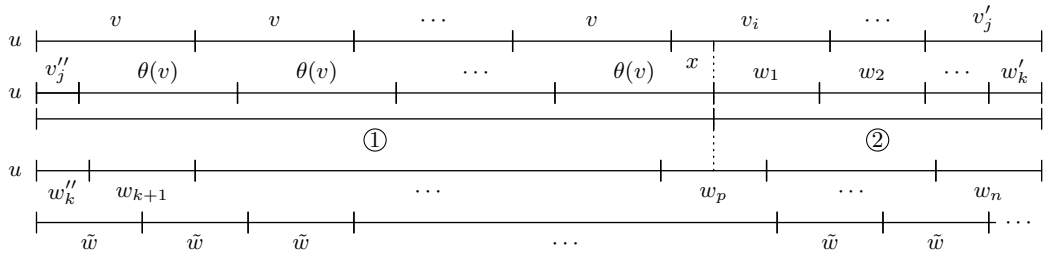


Figure 3 The situation in the case $uuu = v_1 \cdots v_m w_1 \cdots w_n$ with $|u| < m|v| < 2|u|$.

Proof. The situation of this case is depicted in Figure 3 (with $v_j = v'_j v''_j$ and $w_k = w'_k w''_k$).

As $m, n \geq 5$, either v_m or w_1 has to be a proper factor of u_2. We assume without loss of generality, that v_m is a factor of u_2, therefore $m > j$ (see Figure 3). We now show that the factor ① in Figure 3 is a θ-palindrome; this part of the proof holds also for the case when both m and n are even. To streamline the presentation we assume v to be θ-primitive. Otherwise $v \in \{v', \theta(v')\}^+$ for some θ-primitive word v', and we apply the reasoning below to v', reaching the same conclusion. Therefore, if $m - j \geq 2$, we have $v_1 = v_2 = \ldots = v_{i-1}$ and $v_{j+1} = \ldots = v_m = \theta(v)$ by Lemma 1. On the other hand, if $m = j + 1$, we use another result by Kari et al. [4], stating that if $x_1 x_2 y = z x_3 x_4$ holds, where $x_i \in \{t, \theta(t)\}$ for all $1 \leq i \leq 4$ with t a θ-primitive word, then $x_2 \neq x_3$. Applying this to $x_1 = v_j, x_2 = v_m, x_3 = v_1, x_4 = v_2$, and y and z chosen accordingly, we get that $v_m = \theta(v_1)$. If $v_i = \theta(v_j)$, then ① is clearly a θ-palindrome. If $v_i = v_j = v$, then x (from Figure 3) is a prefix of v and we see that $x = \theta(x)$, and thus $\theta(v^{i-1}x) = x\theta(v)^{i-1} = v^{i-1}x$. The same reasoning applies if $v_i = v_j = \theta(v)$.

Furthermore, the factor ② is a θ-palindrome by the same arguments. Here, there is another case to be considered, though, namely when ② is shorter than $|w|$. If $w_1 = \theta(w_n)$, then ② is obviously a θ-palindrome. If $w_1 = w_n$ we get that $w_1 = x'y$, where ② $= x'$ is the suffix of u_2 and $w_n = zx'$. As ① is a θ-palindrome, it holds that $z = \theta(y)$. Thus $x'y = \theta(y)x'$, and the solution of this equation is given by $y = (\alpha\beta)^i, \theta(y) = (\beta\alpha)^i, x' = (\beta\alpha)^j\beta$ for some $i \geq 1$, $j \geq 0$ and θ-palindromes α and β. Consequently, $x' =$ ② is a θ-palindrome.

As the factors ① and ② are θ-palindromes, so are $v_1 \cdots v_m$ and $w_1 \cdots w_n$. Now, if m is odd, we get that $v_{\frac{m+1}{2}} = \theta(v_{\frac{m+1}{2}})$ and therefore $v = \theta(v)$. Similarly, $w = \theta(w)$ if n is odd.

Hence, if both m and n are odd, we have the equation $u^3 = v^m w^n$, and as $m, n \geq 5$, we get that $u, v, w \in \{t\}^+$ for some word t by Lyndon and Schützenberger's original result.

Therefore, assume that only n is odd, while m is even (the other case works analogously). Thus we have the equation $u^3 = v_1 \cdots v_m w^n$, with $m \geq 6$ and $w = \theta(w)$. Furthermore, we can assume v to be θ-primitive in this case, as otherwise we would consider the same equation with v replaced by its θ-primitive root, and as m is even, this would not change the parity.

First we show the statement for $|v| > |w|$. Since $v_1 \cdots v_m$ has a common prefix with $\tilde{w}^\omega$ (where $\tilde{w} \sim w$, see Figure 3) of length $|u|$, and $|u| \geq 2|v| + |w|$ (if this was not true, we had $6|v| + 3|w| > 3|u|$, a contradiction), we can apply Theorem 3 and get that $v, \tilde{w} \in \{t, \theta(t)\}^+$ for some t. If $|v| > |w|$ then v is not θ-primitive, a contradiction. Thus, we assume $|v| \leq |w|$ in the following. Also, if $|②| \geq |w|$, we can apply Theorem 2 to get that $u, w \in \{t\}^+$ for some t; as $w = \theta(w)$ we get that $t = \theta(t)$ and the conclusion follows easily.

Therefore we can assume $|v| \leq |w|$ and also $|②| < |w|$. As $n \geq 5$, we have $4|w| < |u|$ and consequently $|w| < \frac{|u|}{4}$ and also $|v| < \frac{|u|}{4}$. It follows that the length of $v_1 \cdots v_{i-1} = v^{i-1}$ is at least $|u| - |②| - |v|$, thus $|v^{i-1}| \geq \frac{|u|}{2} \geq |v| + |w|$. Thus we can apply Theorem 2 to v^{i-1} and $\tilde{w}^\omega$, to get that $v, \tilde{w} \in \{t\}^+$ for some word t. As we assumed v to be θ-primitive and thus primitive, we get $\tilde{w} \in \{v\}^+$. Therefore, as u_1 is completely covered by $\tilde{w}^\omega$, u_1 must

be of the form $v^{j-1}x$, where $|x| < |v|$ and x is a prefix of v_j. As $|v| \leq |w| < \frac{|u|}{4}$, we have the equation $u^3 = v^{j-1}v_j\theta(v)^{m-j}w^n$ where both $j-1 \geq 3$ and $m-j \geq 3$. Hence, whatever value $v_j \in \{v, \theta(v)\}$ has, we can always apply Theorem 5 to get the claimed result. ◀

Theorem 9 now follows directly by combining Lemma 15, 18, 19 and 21.

The techniques developed so far allow us to establish two lemmas, which formalise, for the current case, the two meta-steps of our general approach, described in Section 2. These technical results are used in the proof of Lemma 24.

▶ **Lemma 22.** *In the setting of (1), assume that v is θ-primitive and $|v|$ does not divide $|u|$. Let j be so that $(j-1)|v| < |u| < j|v|$. Then $v_{j+1} = \ldots = v_m = \theta(v_1)$ and $v_1 = \ldots = v_{m-j}$. If $j|v| - |u| \geq \frac{|v|}{2}$, then $v_1 = v_{m-j+1}$ and $v_j = \theta(v_1)$ also.*

▶ **Lemma 23.** *In the setting of (1), assume that $|u| < m|v| < 2|u|$, w is θ-primitive, $|v| > |w|$, and there exists a word $\tilde{v} \sim v$, such that $\tilde{v}$ occurs in vv after the prefix of length $i = |u| \bmod |w|$, and $\tilde{v} \in \{w, \theta(w)\}^+$. Then $v = \tilde{v}$.*

The following lemma states the final result of this section. Alongside Theorems 6 and 7, it establishes the central result of our paper, namely Theorem 8.

▶ **Lemma 24.** *If $|u| < m|v| < 2|u|$, $u_1u_2u_3 = uuu$, and $m, n \geq 12$, then (1) implies that $u, v, w \in \{t, \theta(t)\}^+$ for some word t.*

Proof. We refer again to the notation used in Figure 3 and, as usual, we assume that v and w are θ-primitive.

First of all, without loss of generality, we assume that $|①| \geq |②|$. Then $|①| \geq 4|v|$, otherwise $|②| > 4|v|$ had to hold, but $|①| \geq |②|$. By the same reasoning $|①| \geq 4|w|$, so $p \geq k+5$ in Figure 3.

If $|w| \geq |v|$, then ① is long enough to apply Theorem 3 and we obtain $\tilde{v} \in \{w, \theta(w)\}^+$. On the other hand, if $|w| < |v|$, then $w'' \leq_p v_1$ and $w''w_{k+1} \leq_p v_1v_2$. Thus, $|①| - |w''_k| \geq 2|v| + |w|$ and as $x \leq_p v$, we can again apply Theorem 3 to get $\tilde{v} \in \{w, \theta(w)\}^+$. Now, using Lemma 23 we get that $v = \tilde{v} \in \{w, \theta(w)\}^+$, a contradiction. ◀

Acknowledgements. We thank the referees for their valuable remarks that improved the presentation significantly. In particular, we acknowledge the simplifications of our proofs for Lemma 14 and Lemma 24, that were suggested in one of the reviews.

References

1 Elena Czeizler, Eugen Czeizler, Lila Kari, and Shinnosuke Seki. An extension of the Lyndon-Schützenberger result to pseudoperiodic words. *Inf. Comput.*, 209(4):717–730, 2011.

2 Elena Czeizler, Lila Kari, and Shinnosuke Seki. On a special class of primitive words. *Theor. Comput. Sci.*, 411(3):617–630, 2010.

3 Tero Harju and Dirk Nowotka. On the equation $x^k = z_1^{k_1}z_2^{k_2}\cdots z_n^{k_n}$ in a free semigroup. *Theor. Comput. Sci.*, 330(1):117–121, 2005.

4 Lila Kari, Benoît Masson, and Shinnosuke Seki. Properties of pseudo-primitive words and their applications. *Int. J. Found. Comput. Sci.*, 22(2):447–471, 2011.

5 M. Lothaire. *Combinatorics on words.* Cambridge University Press, 1997.

6 Roger Conant Lyndon and Marcel-Paul Schützenberger. The equation $a^m = b^nc^p$ in a free group. *Michigan Math. J*, 9(4):289–298, 1962.

Solvency Markov Decision Processes with Interest

Tomáš Brázdil*[1], Taolue Chen[2], Vojtěch Forejt†[3], Petr Novotný[1], and Aistis Simaitis[3]

1 **Faculty of Informatics, Masaryk University, Czech Republic**
2 **Department of Computer Science, Middlesex University London, UK**
3 **Department of Computer Science, University of Oxford, UK**

Abstract

Solvency games, introduced by Berger et al., provide an abstract framework for modelling decisions of a risk-averse investor, whose goal is to avoid ever going broke. We study a new variant of this model, where, in addition to stochastic environment and fixed increments and decrements to the investor's wealth, we introduce interest, which is earned or paid on the current level of savings or debt, respectively.

We study problems related to the minimum initial wealth sufficient to avoid bankruptcy (i.e. steady decrease of the wealth) with probability at least p. We present an exponential time algorithm which approximates this minimum initial wealth, and show that a polynomial time approximation is not possible unless P = NP. For the qualitative case, i.e. $p = 1$, we show that the problem whether a given number is larger than or equal to the minimum initial wealth belongs to NP ∩ coNP, and show that a polynomial time algorithm would yield a polynomial time algorithm for mean-payoff games, existence of which is a longstanding open problem. We also identify some classes of solvency MDPs for which this problem is in P. In all above cases the algorithms also give corresponding bankruptcy avoiding strategies.

1998 ACM Subject Classification G.3 Probability and statistics

Keywords and phrases Markov decision processes, algorithms, complexity, market models

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.487

1 Introduction

Markov decision processes (MDP) are a standard model of complex decision-making where results of decisions may be random. An MDP has a set of *states*, where each state is assigned a set of *enabled actions*. Every action determines a distribution on the set of successor states. A run starts in a state; in every step, a *controller* chooses an enabled action and the process moves to a new state chosen randomly according to the distribution assigned to the action. The functions that describe decisions of the controller are called *strategies*. They may depend on the whole history of the computation and the choice of actions may be randomized.

MDPs form a natural model of decision-making in the financial world. To model nuances of financial markets, various MDP-based models have been developed (see e.g. [16, 2, 3]). A common property of these models is that actions correspond to investment choices and result in (typically random) payoffs for the controller. One of the common aims in this area is to find a *risk-averse* controller (investor) who strives to avoid undesirable events [13, 14].

In this paper we consider a model based on standard reward structures for MDPs, which is closely related to solvency games studied in [3]. The model is designed so that it captures

* Tomáš Brázdil is supported by the Czech Science Foundation, grant No P202/12/P612.
† Vojtěch Forejt is also affiliated with FI MU, Brno, Czech Republic.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 487–499

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

essential properties of risk-averse investments. We assume finite-state MDPs and assign a (real) reward to every action which is collected whenever the action is chosen. The states of the MDP capture the global situation on the market, prices of assets, etc. Note that it is usually plausible to model the prices by a finite-state stochastic process (see e.g. [16]). Rewards model money received (positive rewards) and money spent (negative rewards) by the controller. Controllers are then compared w.r.t. their ability to collect the reward over finite or infinite runs.

Standard objectives such as the *total reward*, or the *long-run average reward* are not suitable for modelling the behaviour of a risk-averse investor as they allow temporary loss of an arbitrary amount of money (i.e., a long sequence of negative rewards), which is undesirable, because normally the controller's access to credit is limited. The authors of [3] consider a "bankruptcy-avoiding" objective defined as follows: Starting with an initial amount of wealth W_0, in the n-th step, the current wealth W_n is computed from W_{n-1} by adding the reward collected in the n-th step. The goal is to find a controller which maximizes the probability of having $W_n > 0$ for all n.

Although the model of [3] captures basic behaviour of a risk-averse investor, it lacks one crucial aspect usually present in the financial environment, i.e., the *interest*. Interests model the value that is received from holding a certain amount of cash, or conversely, the cost of having a negative balance. To accommodate interests, we propose the following extension of the bankruptcy-avoiding objective: Fix an interest rate $\varrho > 1$.[1] Starting with an initial wealth W_0, in the n-th step, compute the current wealth W_n from W_{n-1} by adding not only the collected reward but also the interest $(\varrho - 1)W_{n-1}$. The economical motivation for such a model is that the controller can earn additional amount of wealth by lending its assets for a fixed interest, and conversely, when the controller is in debt, it has to pay interest to its creditors (for the clarity of presentation, we suppose the interest earned from positive wealth is the same as the interest paid on debts).

Hence, the objective is to "manage" the wealth so that it stays above some threshold and does not keep decreasing to negative infinity. More precisely, we want to maximize the probability of having $\liminf_{n\to\infty} W_n > -\infty$. Intuitively, $\liminf_{n\to\infty} W_n \geq 0$ means that the controller ultimately does not need to borrow money, and $-\infty < \liminf_{n\to\infty} W_n < 0$ means that the controller is able to sustain interest payments from its income. If $\liminf_{n\to\infty} W_n = -\infty$, then the controller cannot sustain interest payments and bankrupts.

An important observation is that this objective is closely related to another well-studied objective concerning the *discounted total reward*. Concretely, given a discount factor $0 < \beta < 1$, the discounted total reward T accumulated on a run is defined to be the weighted sum of rewards of all actions on the run where the weight of the n-th action is β^n. In particular, the *threshold problem* asks to maximize the probability of $T \geq t$ for a given threshold t. This problem has been considered in, e.g., [17, 11, 18, 19]. A variant of the threshold problem is the *value-at-risk* problem [4] which asks, for a given probability p, what is the infimum threshold, such that maximal probability of discounted reward surpassing the threshold is at least p? We show that for every controller, the probability of $T \geq t$ with discount factor β is equal to the probability of $\liminf_{n\to\infty} W_n > -\infty$ with $W_0 = -t$ for the interest rate $\varrho := \frac{1}{\beta}$. This effectively shows interreducibility of these problems. Note that the interpretation of the discount factor as the inverse of the interest is natural in financial mathematics.

Contribution. We introduce a model of solvency MDPs with interests (referred to as *solvency MDPs* for brevity), which allows to capture the complex dynamics of wealth management

[1] For notational convenience, we define the interest rate to be the number $1 + r$, where $r > 0$ is the usual interest rate, i.e. the percentage of money paid/received over a unit of time.

under uncertainty. We show that for every solvency MDP there is a bound on wealth such that above this bound the bankruptcy is surely avoided (no matter what the controller is doing), and another bound on wealth below which the bankruptcy is inevitable. Nevertheless, we also show that there still might be infinitely many reachable values of wealth between these two bounds.

The main results of our paper concentrate on the complexity of computing minimal wealth with which the controller can stay away from bankruptcy. Let $\mathbf{W}(s_0, p)$ be the *infimum* of all initial wealths W_0 such that starting in the state s_0 with W_0 the controller can avoid bankruptcy (i.e., $\liminf_{n\to\infty} W_n > -\infty$) with probability at least p. Our overall goal is to compute this number $\mathbf{W}(s_0, p)$. Solution to this problem is important for a risk-averse investor, whose aim is to keep the risk of bankruptcy below some acceptable level.

First we consider the *qualitative case*, i.e. $\mathbf{W}(s_0, 1)$. For this case we show a connection with two-player (non-stochastic) games with discounted total reward objectives. Then, using the results of [20] we show that there is an *oblivious* strategy (i.e., the one that looks only at the current state but is independent of the wealth accumulated so far) which starting in some state s_0 with wealth $\mathbf{W}(s_0, 1)$ avoids bankruptcy with probability one. The problem whether $W \geq \mathbf{W}(s_0, 1)$ for a given W (encoded in binary) is in $NP \cap coNP$ (we also obtain a reduction from discounted total reward games, showing that improving this complexity bound might be difficult). In addition, the number $\mathbf{W}(s_0, 1)$ can be computed in pseudo-polynomial time. Further it follows that for a restricted class of solvency Markov chains (i.e. when there is only one enabled action in every state) the value $\mathbf{W}(s_0, 1)$ can be computed in polynomial time.

The main part of our paper concerns the *quantitative case*, i.e. $\mathbf{W}(s_0, p)$ for an arbitrary probability bound p.

- We give an exponential-time algorithm that approximates $\mathbf{W}(s_0, p)$ up to a given absolute error $\varepsilon > 0$. We actually show that the algorithm runs in time polynomial in the number of control states and exponential in $\log(1/(\varrho - 1))$, $\log(1/\varepsilon)$ and $\log(r_{\max})$, where ϱ is the interest rate and $r_{\max}$ is the maximal $|r|$ where r is a reward associated to some action.
- Employing a reduction from the Knapsack problem, we show that the above complexity cannot be lowered to polynomial in either $\log(1/\varepsilon)$ or $\log(\varrho - 1) + \log(r_{\max})$ unless P=NP.
- We give an exponential-time algorithm that for a given $\varepsilon > 0$ and initial wealth W_0 computes v such that if the initial wealth is increased by ε, then the probability of avoiding bankruptcy is at least v (i.e. $W_0 + \varepsilon \geq \mathbf{W}(s_0, v)$) and $v \geq \sup\{v' \mid W_0 \in \mathbf{W}(s_0, v')\}$.

Moreover, via the aforementioned interreducibility between discounted and solvency MDPs we establish new complexity bounds for value-at-risk approximation in discounted MDPs.

We note that the aforementioned algorithms employ a careful rounding of numbers representing the current wealth W_n. Choosing the right precision for this rounding is quite an intricate step, since a naive choice would only yield a doubly-exponential algorithm.

The paper is organized as follows: after introducing necessary definitions and clarifying the relation with the discounted MDPs in Section 2 we summarise the results for qualitative problem in Section 3. In Section 4 we give the contributions for the quantitative problem. Proofs omitted due to the space constraints can be found in [7].

Related work. Processes involving interests and their formal models naturally emerge in the field of financial mathematics. An MDP-based model of a financial market is presented, e.g., in Chapter 3 of [2]. There, in every step the investor has to allocate his current wealth between riskless bonds, on which he receives an interest according to some fixed interest rate, and several risky stocks, whose price is subject to random fluctuations. Optimization of the investor's portfolio with respect to various utility measures was studied. However, this portfolio optimization problem was considered only in the finite-horizon case, where

the trading stops after some fixed number of steps. In contrast, we concentrate on the long-term stability of the investor's wealth. Also, the model in [2] was analysed mainly from the mathematical perspective (e.g., characterizing the form of optimal portfolios), while we focus on an efficient algorithmic computation of the optimal investor's behaviour.

The issues of a long-term stability and algorithms were considered for other related models, all of which concern total accumulated reward properties. Our model is especially close to *solvency games* [3], which are in fact MDPs with a single control state, where the investor aims to keep the total accumulated reward non-negative. In *energy games* (see e.g. [8, 9, 10]), there are two competing players, but no stochastic behaviour. In *one-counter* MDPs [6], the counter can be seen as a storage for the current value of wealth. All these models differ from the topic studied in this paper in that they do not consider interest on wealth. This makes them fundamentally different in terms of their properties, e.g. in our setting the set of all wealths reachable from a given initial wealth can have nontrivial limit points. Also, in all the three aforementioned models, the objective is to stay in the positive wealth. Here we focus on a different objective to capture the idea that it is admissible to be in debt as long as it is possible to maintain the debt above some limit.

As mentioned before, our work is also related to the threshold discounted total reward objectives, which were considered in [17, 11, 18, 19], where the authors studied finite- and infinite-horizon cases. In the finite-horizon case, in particular [19] gave an algorithm to compute the probability, but a careful analysis shows that their algorithm has a doubly-exponential worst-case complexity when the planning horizon (i.e., the number of steps after which the process halts) is encoded in binary. In [5] they proposed to approximate the probability through the discretisation of wealth, but in the worst the error of approximation is 1, no matter how small discretisation step is taken. In [19], the optimality equation characterising optimal probabilities has been provided for the infinite-horizon case, but no algorithm was proposed. Moreover, [4] considered the "value-at-risk" problem, but again only for the finite-horizon case, giving a doubly-exponential approximation algorithm. Although we consider only infinite-horizon MDPs, the exponential-time upper bound for the $\mathbf{W}(s,p)$ approximation and the NP-hardness lower bound can be easily carried over to the finite-horizon case. Thus, we establish new complexity bounds for value-at-risk approximation in both finite and infinite-horizon discounted MDPs. We also mention [12] which introduced the percentile performance criteria where the controller aims to find a strategy achieving a specified value of the long-run limit average reward at a specified probability level (percentile).

2 Preliminaries

We denote by $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ the sets of all natural, integer, rational and real numbers, respectively. For an index set I, its member i and vector $\mathbf{V} \in \mathbb{R}^I$ we denote by $\mathbf{V}(i)$ the i-component of $\mathbf{V}$. The encoding size of an object B is denoted by $||B||$. We use $\log x$ to refer to the binary logarithm of x. We assume that all numbers are represented in binary and that rational numbers are represented as fractions of binary-encoded integers.

We assume familiarity with basic notions of probability theory. Given an at most countable set X, we use $dist(X)$ to denote all probability distributions on X.

▶ **Definition 1** (MDP). A *Markov decision process* (MDP) is a tuple $M = (V, A, T)$ where V is at most countable set of *vertices*, A is a finite set of *actions*, and $T : V \times A \to dist(V)$ is a partial *transition function*. We assume that for every $v \in V$ the set $A(v)$ of all actions available at v (i.e., the set off all actions a s.t. $T(v,a)$ is defined) is nonempty.

We denote by $Succ(v,a) = \{u \mid T(v,a)(u) > 0\}$ the *support* of $T(v,a)$.

An *infinite path* (or *run*) is a sequence $v_0a_1v_1a_2v_2\cdots \in (V\times A)^\omega$ such that $a_{i+1}\in A(v_i)$ and $v_{i+1}\in Succ(v_i,a_{i+1})$ for all i. A *finite path* (or *history*) is a prefix of a run ending with a vertex, i.e. a word of the form $(V\times A)^*V$. We refer to the set of all runs as Runs_M and to the set of all histories as Hist_M. For a finite or infinite path $\omega = v_0a_1v_1a_2v_2\ldots$ and $i\in\mathbb{N}$ we denote by ω_i the finite path $v_0a_1\cdots a_iv_i$.

A *strategy* in M is a function that to every history w assigns a distribution on actions available in the last vertex of w. A strategy is *deterministic* if it always assigns distributions that choose some action with probability 1, and *memoryless* if it only depends on the last vertex of history. We use Σ_M (or just Σ) for the set of all strategies of M.

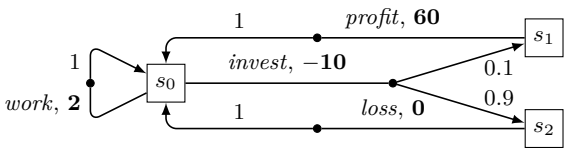
Each history $w\in\mathsf{Hist}_M$ determines the set $\mathsf{Cone}(w)$ consisting of all runs having w as a prefix. To an MDP M, its vertex v and strategy σ we associate the probability space $(\mathsf{Runs}_M,\mathcal{F},\mathbb{P}^\sigma_{M,v})$, where $\mathcal{F}$ is the σ-field generated by all $\mathsf{Cone}(w)$, and $\mathbb{P}^\sigma_{M,v}$ is the unique probability measure such that for every history $w = v_0a_1\ldots a_kv_k$ we have $\mathbb{P}^\sigma_{M,v}(\mathsf{Cone}(w)) = \mu(v_0)\cdot\prod_{i=1}^k x_i$, where $\mu(v_0)$ is 1 if $v_0 = v$ and 0 otherwise, and where $x_i = \sigma(w_{i-1})(a_i)\cdot T(v_{i-1},a_i)(v_i)$ for all $1\le i\le k$ (the empty product is equal to 1). We drop M from the subscript when the MDP is clear from the context.

▶ **Definition 2** (Solvency MDP)**.** A *solvency Markov decision process* is a tuple (S,A,T,F,ϱ) where S is a finite set of *states*, A and T are such that (S,A,T) is an MDP, $F: S\times A\to\mathbb{Q}$ is a partial *gain function* and $\varrho\in\mathbb{Q}\cap(1,\infty)$ is an *interest rate*.

We stipulate that for every $(s,a)\in S\times A$ the value $F(s,a)$ is defined iff $a\in A(s)$. A *solvency Markov chain* is a solvency MDP with one action per state, i.e. $|A(s)| = 1$ for all $s\in S$. A *configuration* of a solvency MDP $M = (S,A,T,F,\varrho)$ is represented as a state-wealth pair (s,x) where $s\in S$ and $x\in\mathbb{Q}$. The semantics of M is given by an infinite-state MDP $M_\varrho = (S\times\mathbb{Q},A,T_\varrho)$ where for every $(s,x)\in S\times\mathbb{Q}$ and $a\in A(s)$ we define $T_\varrho((s,x),a)(s',\varrho\cdot x+F(s,a)) = p$ whenever $T(s,a)(s') = p$. We sometimes do not distinguish between M and M_ϱ and refer to strategies or runs of M where strategies or runs of M_ϱ are intended. A strategy σ for M_ϱ is *oblivious* if it is memoryless and does not make its decision based on the current wealth, i.e. for all $w\cdot(s,x)$ and (s,x') we have $\sigma(w\cdot(s,x)) = \sigma((s,x'))$.

Objectives. Given an solvency MDP M and its initial configuration (s_0,x_0), we are interested in the set of runs in which the wealth always stays above some finite bound, denoted by $Win = \mathsf{Runs}_M\setminus\{(s_0,x_0)a_1(s_1,x_1)\cdots\in\mathsf{Runs}_M \mid \liminf_{n\to\infty}x_n = -\infty\}$. Intuitively, this objective models the ability of the investor not to go bankrupt, i.e. to compensate for the incurred interest by obtaining sufficient gains. We denote $Val_M(s_0,x_0) = \sup_\sigma\mathbb{P}^\sigma_{M,(s_0,x_0)}(Win)$ the maximal probability of winning with a given wealth, and $\mathbf{W}_M(s,p) = \inf\{x \mid Val_M(s,x)\ge p\}$ the infimum of wealth sufficient for winning with probability p. In this paper we are mainly interested in the problems of computing or approximating the values of $\mathbf{W}_M(s,p)$. We also address the problem of computing a convenient risk-averse strategy for an investor with a given initial wealth x_0. A precise definition of what we mean by a convenient strategy is given in Section 4 (Theorem 11). We say that a strategy is p-winning (in an initial configuration (s_0,x_0)) if $\mathbb{P}^\sigma_{M,(s_0,x_0)}(Win)\ge p$. A 1-winning strategy is called *almost surely winning*, and strategy σ with $\mathbb{P}^\sigma_{M,(s_0,x_0)}(Win) = 0$ is called *almost surely losing*.

▶ **Example 3.** Consider the following solvency MDP $M = (S,A,T,F,\varrho)$:



Here $S = \{s_0, s_1, s_2\}$, $A = \{work, invest, profit, loss\}$, T is depicted by the arrows in the figure, for example $T(s_0, invest) = [s_1 \mapsto 0.1, s_2 \mapsto 0.9]$, the function F is given by the bold numbers next to the actions, e.g. $F(s, work) = 2$, and $\varrho = 2$ (we take this extremely large value to keep the example computations simpler). The MDP models the choices of a person who can either work, which ensures certain but relatively small income, or can invest a larger amount of money but take a significant risk. Starting in the configuration $(s_0, -10)$ (i.e. in debt), an example strategy σ is the strategy which always chooses *work* in s_0, but as can be easily seen, we get $\mathbb{P}^{\sigma}_{M,(s_0,-10)}(Win) = 0$ since the constant gains are not high enough to cover the interest incurred by the debt. An optimal strategy here is to pick *work* only in histories ending with a configuration (s_0, x) for $x \geq -2$, and to pick *invest* otherwise. Such strategy shows that $Val_M(s_0, -10) = 0.1$. Now suppose that the investor wants to find out what is the wealth needed to make sure the probability of winning is at least 0.7, i.e. wants to compute $\mathbf{W}_M(s_0, 0.7)$. This number is equal to -2. To see this, observe that for any configuration (s_0, y) where $y < -2$ the optimal strategy must pick *invest*, which with probability 0.9 results in a debt from which it is impossible to recover. Finally, observe that $Val_M(s_0, -2) = 1$ since a strategy that always chooses *work* is 1-winning in $(s_0, -2)$. This demonstrates that the function $Val(s, \cdot)$ for a given state s may not be continuous.

Relationship with discounted MDPs. The problems we study for solvency MDPs are closely related to another risk-averse decision making model, so called *discounted MDPs with threshold objectives*. A discounted MDP is a tuple $D = (S, A, T, F, \beta)$, where the first four components are as in a solvency MDP and $0 < \beta < 1$ is a *discount factor*. The semantics of a discounted MDP is given by a finite-state MDP $D^{\beta} = (S, A, T)$ and a reward function $disc(\cdot)$ which to every run $\omega = s_0 a_1 s_1 a_2 \dots$ in D^{β} assigns its *total discounted reward* $disc(\omega) = \sum_{i=1}^{\infty} F(s_{i-1}, a_i) \cdot \beta^i$. The threshold objective asks the controller to maximize, for a given threshold $t \in \mathbb{Q}$, the probability of the event $Thr(t) = \{\omega \in Run(D^{\beta}) \mid disc(\omega) \geq t\}$.

Now consider a solvency MDP $M = (S, A, T, F, \varrho)$ with an initial configuration (s_0, x_0) and a discounted MDP $D = (S, A, T, F, 1/\varrho)$ with a threshold objective $Thr(-x_0)$. Note that once an initial configuration $(s_0, x_0) \in S \times \mathbb{Q}$ is fixed, there is a natural one-to-one correspondence between runs in M_{ϱ} initiated in (s_0, x_0) and runs in $D^{1/\varrho}$ initiated in s: we identify a run $(s_0, x_0)a_1(s_1, x_1)a_2 \dots$ in M_{ϱ} with a run $s_0 a_1 s_1 a_2 \dots$ in $D^{1/\varrho}$. This correspondence naturally extends to strategies in both MDPs, so we assume that these MDPs have identical sets of runs and strategies.

▶ **Proposition 4.** *Let M, D be as above. Then $\mathbb{P}^{\sigma}_{M,(s,x)}(Win) = \mathbb{P}^{\sigma}_{D,s}(Thr(-x))$ for all $\sigma \in \Sigma$.*

Proof. It suffices to show that for every run ω we have $\omega \in Win \Leftrightarrow disc(\omega) \geq -x$. Fix a run $\omega = (s_0, x_0)a_1(s_1, x_1)a_2 \dots$, and define, for every $n \geq 0$, $disc_n(\omega) \stackrel{\text{def}}{=} \sum_{i=1}^{n} F(s_{i-1}, a_i) \cdot \frac{1}{\varrho^i}$ (an empty sum is assumed to be equal to 0). Obviously, for every $n \geq 0$ we have $x_n = \varrho^n \cdot (disc_n(\omega) + x_0)$. Thus, if $disc(\omega) = \lim_{n\to\infty} disc_n(\omega) > -x_0$, then $\lim_{n\to\infty} x_n$ exists and it is equal to $+\infty$. Similarly, if $disc(\omega) < -x_0$, then $\lim_{n\to\infty} x_n = -\infty$. If $disc(\omega) = -x_0$, the infimum wealth x_n along ω is finite (see [7]), and so $\omega \in Win$. ◀

It follows that many natural problems for solvency MDPs (value computation etc.) are polynomially equivalent to similar natural problems for discounted MDPs with threshold objectives. In particular, our problem of computing/approximating $\mathbf{W}_M(s_0, p)$ is interreducible with the *value-at-risk* problem in discounted MDPs, where the aim is to compute/approximate the supremum threshold t such that under suitable strategy the probability (risk) of the discounted reward being $\leq t$ is at most $1 - p$.

3 Qualitative Case

In this section we establish a connection between the qualitative problem for solvency MDPs (i.e., determining whether $x \geq \mathbf{W}_M(s,1)$ for a given state s and number x) and the problem of determining the winner in non-stochastic discounted games.

▶ **Definition 5** (Discounted game). A finite *discounted game* is a tuple $G = (S_1, S_2, s_0, T, R, \beta)$ where S_1 and S_2 are sets of player 1 and 2 states, respectively; $s_0 \in S_1$ is the initial state; $T \subseteq (S_1 \times S_2) \cup (S_2 \times S_1)$ is a transition relation; $R : (S_1 \cup S_2) \to \mathbb{R}$ is a reward function; and $0 < \beta < 1$ is a discount factor.

A strategy for player $i \in \{1,2\}$ in a discounted game is a function $\zeta_i : (S_1 \cup S_2)^* \cdot S_i \to (S_1 \cup S_2)$ such that $(s, \zeta_i(ws)) \in T$ for every s and w. A strategy is *memoryless* if it only depends on the last state. A pair of strategies ζ_1 and ζ_2 for players 1 and 2 yields a unique run $run(\zeta_1, \zeta_2) = s_0 s_1 \ldots$ in the game, given by $s_j = \zeta_i(s_0 \ldots s_{j-1})$ where i is 1 or 2 depending on whether $s_{j-1} \in S_1$ or $s_{j-1} \in S_2$. The discounted total reward of the run is defined to be $disc(s_0 s_1 \ldots) := \sum_{i=0}^{\infty} \beta^{i+1} R(s_i)$. The *discounted game* problem asks, given a game G and a value x, whether there is a strategy ζ_1 for player 1 such that for all strategies ζ_2 of player 2 we have $disc(run(\zeta_1, \zeta_2)) \geq x$. Such a strategy ζ_1 is then called *winning*.

By Proposition 4 the problem of determining whether $x \geq \mathbf{W}_M(s,1)$ for a state s of a solvency MDP M is interreducible (in polynomial time) with the problem of determining whether there is $\sigma \in \Sigma_D$ such that $\mathbb{P}^{\sigma}_{D,s}(Thr(-x)) = 1$ in the corresponding discounted MDP D. We show that the latter is interreducible[2] with the discounted game problem.

Let us first fix a discounted MDP $D = (S, A, T, F, \beta)$. We say that a run $\omega = s_0 a_1 s_1 \ldots$ of D is *realisable* under a strategy σ if $\sigma(s_0 a_1 \ldots s_n)(a_{n+1}) > 0$, and $T(s_n, a_{n+1})(s_{n+1}) > 0$ for all n. The idea of the reduction relies on the following lemma, which is proved in [7].

▶ **Lemma 6.** *If $\sigma \in \Sigma_D$ satisfies $\mathbb{P}^{\sigma}_{D,s}(Thr(x)) = 1$, then* all *runs realisable under σ are in $Thr(x)$.*

Using the lemma above we can construct a game G from D by stipulating that the results of actions are chosen by player 2 instead of being chosen randomly, and vice versa. The technical details of the reduction are presented in [7]. The next theorem follows from the reduction and the fact that memoryless (deterministic) strategies suffice in discounted games.

▶ **Theorem 7.** *For every solvency MDP M there exists an oblivious deterministic strategy which is almost-surely winning in every configuration (s,x) with $x \geq \mathbf{W}_M(s,1)$.*

The discounted game problem is in $NP \cap coNP$ and there exists a pseudopolynomial algorithm computing the optimal value [20]. Also, when one of the players controls no states in a game, the problem can be solved in polynomial time [20]. Hence, we get the following theorem.

▶ **Theorem 8.** *The qualitative problem for solvency MDPs is in $NP \cap coNP$. Moreover, there is a pseudopolynomial algorithm that computes $\mathbf{W}_M(s,1)$ for every state s of M. For the restricted class of solvency Markov chains, to compute $\mathbf{W}_M(s,1)$ and to decide the qualitative problem can be done in polynomial time.*

[2] Actually, we use slightly different variants of the discounted game problem in reductions *from* and *to* the discounted MDPs problem, respectively. Nevertheless, they establish the desired complexity bounds.

Note that the existence of a reduction from mean-payoff games to discounted games [1] suggests that improving the above complexity to polynomial-time is difficult, since a polynomial-time algorithm for solvency MDPs would give a polynomial-time algorithm for mean-payoff games, existence of which is a longstanding open problem in the area of graph games.

4 Quantitative Case

This section formulates results on quantitative questions for solvency MDPs. We start with a proposition showing that we can restrict our attention to some subset of $S \times \mathbb{Q}$, since for every state there are two values below and above which all strategies are almost-surely winning or losing, respectively. Intuitively, these values represent wealth (positive or negative) for which losses/gains from the interest dominate gains/losses from the gain function F. An important consequence of the proposition, when combined with [15], is that deterministic strategies suffice to maximize the probability of winning. Therefore, in the rest of this section we consider only deterministic strategies. The proposition is proved in [7].

▶ **Proposition 9.** *For every state s of the solvency MDP M there are rational numbers*

$$U(M,s) \stackrel{\text{def}}{=} \operatorname*{arg\,inf}_{x\in\mathbb{R}} \forall\sigma \,.\, \mathbb{P}^{\sigma}_{M,(s,x)}(\mathit{Win}) = 1 \quad \text{and} \quad L(M,s) \stackrel{\text{def}}{=} \operatorname*{arg\,sup}_{x\in\mathbb{R}} \forall\sigma \,.\, \mathbb{P}^{\sigma}_{M,(s,x)}(\mathit{Win}) = 0,$$

of encoding size polynomial in $||M||$, and they can be computed in polynomial time using linear programming techniques. Moreover, we have $\mathbb{P}^{\sigma}_{M,(s,U(M,s))}(\mathit{Win}) = 1$ for every strategy σ.

To illustrate the proposition, we return to Example 3 and note that $U(M,s_0) = \frac{20}{3}$ and $L(M,s_0) = -\frac{40}{3}$. Obviously, for every s we have $K \geq U(M,s) \geq L(M,s) \geq -K$ where $K = \max_{(s,a)\in S\times A} \frac{|F(s,a)|}{\varrho-1}$, but as Example 3 shows, using $U(M,s)$ and $L(M,s)$ we can restrict the set of interesting configurations more than with the trivial bounds K and $-K$.

We also define the global versions of the bounds, i.e., $L(M) \stackrel{\text{def}}{=} \min_{s\in S} L(M,s)$ and $U(M) \stackrel{\text{def}}{=} \max_{s\in S} U(M,s)$. In accordance with the economic interpretation of our model, we call any configuration of the form (s,x) with $x \geq U(M,s)$ a *rentier configuration*. From Proposition 9 it follows that every run which visits a rentier configuration belongs to *Win*.

Note that although Proposition 9 suggests that we can restrict our analysis to the configurations (s,x) where $L(M,s) \leq x \leq U(M,s)$, the set of reachable configurations between these bounds is still infinite in general as the following example shows.

▶ **Example 10.** Consider a solvency MDP $M = (\{s\},\{a,b\},T,F,\frac{3}{2})$ with $T(s,a) = T(s,b) = s$, and $F(s,a) = \frac{1}{2}$ and $F(s,b) = -\frac{1}{2}$. We have $L(M) = -1$ and $U(M) = 1$. We will show that for any $n \in \mathbb{N}$ there is a configuration (s,x_n) where $x_n = k/2^n$ that is reachable in exactly n steps from an initial configuration $(s,\frac{1}{2})$ and satisfies $k \in \mathbb{N}_0$, $0 \leq k < 2^n$, $2 \nmid k$. Hence the reachable state space from $(s,\frac{1}{2})$ is infinite as the numbers x_n are pairwise different.

We set $x_0 = \frac{1}{2}$, and let (s,x_n) be a reachable configuration where x_n is of the form $k/2^n$ satisfying the above conditions. In one step we can reach configurations (s,x') where $x' = \varrho x_n \pm \frac{1}{2} = \frac{3k\pm 2^n}{2^{n+1}}$. Clearly $2 \nmid 3k \pm 2^n$; otherwise we would have $2 \mid 3k$ and thus $2 \mid k$ which contradicts the definition of x_n. It remains to show that one of the values of x' again satisfies the above conditions; this is a simple exercise that is carried out in [7].

Note that if the interest ϱ is restricted to be an integer, the reachable configuration space between $L(M)$ and $U(M)$ is finite, because for the initial configuration (s,x) it holds $x = \frac{p}{q}$ where $p,q \in \mathbb{Z}$, and $\varrho \cdot x + y = \frac{\varrho\cdot p + y\cdot q}{q}$. Hence, any reachable wealth is a multiple of $\frac{1}{q}$, and there are only finitely many such numbers between $L(M)$ and $U(M)$. This means that one

can use off-the-shelf algorithms for finite-state MDPs, i.e., minimising the probability to reach configuration with (s, x), where $x < L(M, s)$. However, for the general case, this is not possible and we need to devise new techniques.

4.1 Approximation Algorithms

In this subsection we show how to approximate $\mathbf{W}(s, p)$. Our algorithm depends on the following theorem, which allows us, in a certain sense that will be explained soon, to approximate the function $Val_M(s_0, \cdot)$.

▶ **Theorem 11.** *There is an algorithm that computes, for a solvency MDP M with initial configuration (s_0, x_0) and a given $\varepsilon > 0$, a rational number v and a strategy σ such that:*
1. $v \geq Val_M(s_0, x_0)$.
2. *Strategy σ is v-winning from configuration $(s_0, x_0 + \varepsilon)$.*
The running time of the algorithm is polynomial in $|S| \cdot |A| \cdot \log\left(p_{\min}^{-1}\right)$ where $p_{\min} = \min_{(s,s',a)\in S^2\times A} T(s, a)(s')$, and exponential in $\log(|r_{\max}|/(\varrho-1))$ and $\log(1/\varepsilon)$ where $r_{\max} = \max_{(s,a)\in S\times A} |F(s, a)|$.

We will prove Theorem 11 later, but first we argue that the theorem is important in its own right. Consider the following scenario. Suppose that an investor starts with wealth x_0. It is plausible to assume that this initial wealth is not strictly fixed. Instead, one can assume that the investor is willing to acquire some small additional amount of wealth (represented by ε), in exchange for some substantial benefit. Here, the benefit consists of the fact that the small difference in the initial wealth allows the investor to compute and execute a strategy, under which the risk of bankruptcy is provably no greater than the lowest risk achievable with the original wealth. Note that the strategy σ may not be $Val_M(s_0, x_0+\varepsilon)$-winning from $(s_0, x_0 + \varepsilon)$. We now proceed with the theorem providing the approximation of $\mathbf{W}(s, x)$.

▶ **Theorem 12.** *For a given solvency MDP M, its state s and rational numbers $\delta > 0$, $p \in [0, 1]$, it is possible to approximate $\mathbf{W}(s, p)$ up to the absolute error δ in time polynomial in $(|S| \cdot |A|)^{\mathcal{O}(1)} \cdot \log\left(p_{\min}^{-1}\right)$, and exponential in $\log(|r_{\max}|/(\varrho - 1))$ and $\log(1/\delta)$, where $p_{\min}$ an $r_{\max}$ are as in Theorem 11.*

Proof. Suppose that we already know that $a \leq \mathbf{W}(s, p) \leq b$, for some a, b. We can use the algorithm of Theorem 11 for $s_0 = s$, $x_0 = a + (b - a)/2$ and $\varepsilon = (b - a)/4$. If the algorithm returns $v \leq p$, we know that $a + (b - a)/2 \leq \mathbf{W}(s, p) \leq b$, otherwise we can conclude that $a \leq \mathbf{W}(s, p) \leq a + 3(b - a)/4$. Initially we know that $L(M) \leq \mathbf{W}(s, p) \leq U(M)$, so in order to approximate $\mathbf{W}(s, p)$ with absolute error δ it suffices to perform $\mathcal{O}(\log((U(M)-L(M))/\delta))$ iterations of this procedure, finishing when $\varepsilon \leq \delta/4$. ◀

Later we will show that the time complexity of the algorithm cannot be improved to polynomial in either $\log(|r_{\max}|/(\varrho - 1))$ or $\log(1/\delta)$ unless P=NP.

Proof of Theorem 11. For the rest of this section we fix a solvency MDP $M = (S, A, T, F, \varrho)$ and its initial configuration (s_0, x_0). First we establish the existence of a strategy that, given a small additional amount of wealth, reaches a rentier configuration in at most exponential number of steps with probability at least $Val_M(s_0, x_0)$. Then, we will show how to compute such a strategy in exponential time.

To establish the proof of the following proposition, we use a suitable Bellman functional whose unique fixed point is equal to $\mathbf{W}$. The proof can be found in [7].

▶ **Proposition 13.** *For every initial configuration (s,x) and every $\varepsilon > 0$ there is a strategy σ_ε such that starting in $(s, x+\varepsilon/2)$, σ_ε ensures hitting of a rentier configuration in at most $n = \left\lceil \frac{\log(U(M)-L(M))+\log\varepsilon^{-1}+2}{\log\varrho} \right\rceil$ steps with probability at least $Val_M(s,x)$. In particular, $\mathbb{P}^{\sigma_\varepsilon}_{(s,x+\varepsilon/2)}(Win) \geq Val_M(s,x)$.*

The previous proposition shows that the number v and strategy σ of Theorem 11 can be computed by examining the possible behaviours of M during the first n steps. However, since $\log\varrho \approx \varrho - 1$ for ϱ close to 1, the number n can be exponential in $||M||$. Thus, the trivial algorithm, that unfolds the MDP from the initial configuration $(s_0, x_0+\varepsilon/2)$ into a tree of depth n, and on this tree computes a strategy maximising the probability of reaching a rentier configuration, has a doubly-exponential complexity. The key idea allowing to reduce this complexity to singly-exponential is to round the numbers representing the wealth in the configurations of M to numbers of polynomial size. If the size is chosen carefully, the error introduced by the rounding is not large enough to thwart the computation. In the following we assume that $\log\varrho < \log(U(M)-L(M)) + \log(\varepsilon^{-1}) + 2$, since otherwise $n = 1$ and we can compute the strategy σ and number v by computing an action that maximizes the one-step probability of reaching a rentier configuration from $(s_0, x_0+\varepsilon/2)$.

We now formalise the notion of rounding the numbers appearing in configurations of M. Let λ be a rational number. We say that two configurations (s,x), (s',x') are λ-*equivalent*, denoted by $(s,x) \sim_\lambda (s',x')$, if $s = s'$ and one of the following conditions holds:

- both x and x' are greater than $U(M,s)$ or less than or equal to $L(M,s)$; or
- $L(M,s) < x, x' \leq U(M,s)$ and there is $k \in \mathbb{Z}$ such that both $x, x' \in (k\lambda, (k+1)\lambda]$.

Clearly, $\sim_\lambda$ is indeed an equivalence on the set $S \times \mathbb{Q}$, and every member of the quotient set $(S\times\mathbb{Q})/\!\sim_\lambda$ is a tuple of the form (s,D), with $s \in S$ and D being either a half-open interval of length at most λ or one of the intervals $(U(M,s),+\infty)$, $(-\infty, L(M,s)]$. For such D, we denote by w_D the maximal element of D (putting $w_{(U(M,s),+\infty)} = +\infty$). We also denote by $[s,x]_\lambda$ the equivalence class of (s,x).

Now let n be as in Proposition 13. We define an MDP $M_{\lambda,n}$ representing an unfolding of M into a DAG of depth n, in which the current wealth w is always rounded up to the least integer multiple of λ greater than w, with configurations exceeding the upper or dropping below the lower threshold of Proposition 9 being immediately recognized as winning or losing. The unfolded MDP $M_{\lambda,n}$ is formally defined as follows.

▶ **Definition 14.** [Unfolded MDP] Let $M = (S, A, T, F, \varrho)$ be an solvency MDP, and $n > 0$ and $\lambda > 0$ two numbers. We define an MDP $M_{\lambda,n} = (S', A, T')$ where S' is $((S\times\mathbb{Q})/\!\sim_\lambda) \times \{0,1,\ldots,n\}$, and the transition function T' is the unique function satisfying the following:

- for all $(s,D,i) \in S'$ and $a \in A$ where $i < n$ and D is a bounded interval, the distribution $T'((s,D,i),a)$ is defined iff $a \in A(s)$, and assigns $T(s,a)(s')$ to $([s', \varrho\cdot w_D + F(s,a)]_\lambda, i+1)$
- for every other vertex $(s,D,i) \in S'$ there is only a self loop on this vertex under every action, i.e., $T'((s,D,i),a)$ is given by $[(s,D,i) \mapsto 1]$ for every action $a \in A$.

The size of $M_{\lambda,n}$ as well as the time needed to construct it is $(|S|\cdot|A|\cdot\log(p_{\min}^{-1})\cdot n\cdot\lambda^{-1})^{\mathcal{O}(1)}$.

Now we denote by *Hit* the set of all runs in $M_{\lambda,n}$ that contain a vertex of the form $(t,(U(M,t),\infty),i)$, and by $Ar(z)$ (for "almost rentier") the set of all runs in M that hit a configuration of the form (t,y) with $y \geq U(M,t) - z$ in at most n steps. In particular, $Ar(0)$ is the event of hitting a rentier configuration in at most n steps. The following lemma (proved in [7]) shows that $M_{\lambda,n}$ adequately approximates the behaviour of M.

▶ **Lemma 15.** *Let (s,y) be an arbitrary configuration of M. Then the following holds:*

1. *For every $\sigma \in \Sigma_M$ there is $\pi \in \Sigma_{M_{\lambda,n}}$ such that $\mathbb{P}^{\pi}_{M_{\lambda,n},([s,y]_\lambda,0)}(Hit) \geq \mathbb{P}^{\sigma}_{M,(s,y)}(Ar(0))$.*

2. *There is* $\sigma \in \Sigma_M$ *such that* $\mathbb{P}^{\sigma}_{M,(s,y)}(Ar(n \cdot \lambda \cdot \varrho^n)) \geq \sup_{\pi} \mathbb{P}^{\pi}_{M_{\lambda,n},([s,y]_\lambda,0)}(Hit) \stackrel{\text{def}}{=} v$*, where the supremum is taken over* $\Sigma_{M_{\lambda,n}}$*. Moreover, the number* v *and a finite representation of the strategy* σ *can be computed in time* $||M_{\lambda,n}||^{\mathcal{O}(1)}$*.*

We can now finish the proof of Theorem 11. Let us put $\lambda = \lceil (64 \cdot n \cdot (U(M) - L(M))^2)/\varepsilon^3 \rceil^{-1}$. An easy computation (see [7]) proves that $n \cdot \lambda \cdot \varrho^n \leq \frac{\varepsilon}{2}$ thanks to our assumption that $\log \varrho < \log(U(M) - L(M)) + \log(\varepsilon^{-1}) + 2$.

By Proposition 13 there is a strategy σ_ε in M with $\mathbb{P}^{\sigma}_{M,(s_0,x_0+\varepsilon/2)}(Ar(0)) \geq Val_M(s_0, x_0)$, and so from Lemma 15 (1.) we get $\sup_\pi \mathbb{P}^{\pi}_{M_{\lambda,n},([s_0,x_0+\varepsilon/2]_\lambda,0)}(Hit) \geq Val_M(s_0, x_0)$. By part (2.) of the same lemma we can compute, in time $||M_{\lambda,n}||^{\mathcal{O}(1)}$, a strategy σ in M and a number v such that $\mathbb{P}^{\sigma}_{M,(s_0,x_0+\varepsilon/2)}(Ar(\varepsilon/2)) \geq v \geq Val_M(s_0, x_0)$. In other words, from $(s_0, x_0 + \varepsilon/2)$ the strategy σ reaches with probability at least v a configuration that is only $\varepsilon/2$ units of wealth away from being rentier. Note that once an initial configuration is fixed, any strategy can be viewed as being *wealth-independent*, i.e. being only a function of a sequence of states and actions in the history, since the current wealth can be inferred from this sequence and the initial wealth. Suppose now that we fix the initial configuration $(s_0, x_0 + \varepsilon)$ instead of $(s_0, x_0 + \varepsilon/2)$, keeping the same strategy σ (i.e., we use a strategy that selects the same action as σ after observing the same sequence of states and actions). It is then obvious that we reach a rentier configuration with probability at least v, i.e., $\mathbb{P}^{\sigma}_{(s,x+\varepsilon)}(Win) \geq v$ as required.

The complexity analysis of the reduction is a mere technicality and it is shown in [7]. ◀(Thm. 11)

4.2 Lower Bounds

Now we complement the positive results given above with lower complexity bounds.

▶ **Theorem 16.** *The problem of deciding whether* $\mathbf{W}(s, p) \leq x$ *for a given* x *is NP-hard. Furthermore, existence of any of the following algorithms is not possible unless P=NP:*

1. *An algorithm approximating* $\mathbf{W}(s, p)$ *up to the absolute error* δ *in time polynomial in* $|S| \cdot |A| \cdot \log\left(p_{\min}^{-1}\right)$ *and* $\log(|r_{\max}|/(\varrho - 1))$ *and exponential in* $\log(1/\delta)$*.*
2. *An algorithm approximating* $\mathbf{W}(s, p)$ *up to the absolute error* δ *in time polynomial in* $|S| \cdot |A| \cdot \log\left(p_{\min}^{-1}\right)$ *and* $\log(1/\delta)$ *and exponential in* $\log(|r_{\max}|/(\varrho - 1))$*.*

Above, the numbers $r_{\max}$ *and* $p_{\min}$ *are as in Theorem 11.*

Proof sketch. In [7] we show how to construct, for a given instance of the Knapsack problem, a solvency MDP M in which the item values are suitably encoded into probabilities of certain transitions, while the item weights are encoded as rewards associated to some actions. We then show that the instance of Knapsack has a solution if and only if for a certain state s of M and a certain number p (which can be computed from the instance) it holds that $\mathbf{W}(s, p) \leq 0$. We also show that in order to decide this inequality it suffices (for the constructed MDP M) to approximate $\mathbf{W}(s, p)$ up to the absolute error $\frac{1}{4}$. (Intuitively, this corresponds to the well-known fact that no polynomial approximation algorithm for Knapsack can achieve a constant absolute error.) To get part (2.) we use a slight modification of the same approach.

A crucial component of these reductions is the fact that $Val_M(t, \cdot)$ may not be a continuous function (see example 3). Intuitively, this allows us to recognise whether the current wealth, which in M always encodes weight of some set of items, surpasses some threshold. ◀

Note that thanks to the interreducibility from Proposition 4, the (suitably rephrased) results of Theorems 12 and 16 hold also for the value-at-risk approximation in discounted MDPs.

5 Conclusions

We have introduced solvency MDPs, a model apt for analysis of systems where interest is paid or received for the accumulated wealth. We have analysed the complexity of fundamental problems, and proposed algorithms that approximate the minimum wealth needed to win with a given probability and compute a strategy that achieves the goal. As a by-product, we obtained new results for the *value-at-risk* problem in discounted MDPs.

There are several important directions of future study. One question deserving attention is to find an algorithm computing or approximating $Val(s, x)$. The usual approaches of discretising the state space do not work in this case since the function $Val(s, \cdot)$ is not continuous and thus it is difficult to bound the error introduced by the discretisation. Another direction is the implementation of the algorithms and their evaluation on case-studies.

References

1 D. Andersson and P. Bro Miltersen. The Complexity of Solving Stochastic Games on Graphs. In *Proceedings of the ISAAC'09*, pages 112–121, Berlin, Heidelberg, 2009. Springer.

2 N. Bäuerle and U. Rieder. *Markov Decision Processes with Applications to Finance*. Springer, 2011.

3 N. Berger, N. Kapur, L.J. Schulman, and V. Vazirani. Solvency Games. In *Proceedings of FST&TCS 2008*, volume 2 of *LIPIcs*, pages 61–72. Schloss Dagstuhl, 2008.

4 Kang Boda and Jerzy A. Filar. Time Consistent Dynamic Risk Measures. *Math. Meth. of OR*, 63(1):169–186, 2006.

5 Kang Boda, Jerzy A. Filar, Yuanlie Lin, and Lieneke Spanjers. Stochastic target hitting time and the problem of early retirement. *IEEE Trans. Aut. Contr.*, 49(3):409–419, 2004.

6 T. Brázdil, V. Brožek, K. Etessami, and A. Kučera. Approximating the termination value of one-counter MDPs and stochastic games. *Inf. Comput.*, 222:121–138, 2013.

7 T. Brázdil, T. Chen, V. Forejt, P. Novotný, and A. Simaitis. Solvency Markov Decision Processes with Interest. *CoRR*, abs/1310.3119, 2013.

8 A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource Interfaces. In *Proc. of EMSOFT 2003*, volume 2855 of *LNCS*, pages 117–133, Heidelberg, 2003. Springer.

9 K. Chatterjee and L. Doyen. Energy Parity Games. In *Proceedings of ICALP 2010, Part II*, volume 6199 of *LNCS*, pages 599–610. Springer, 2010.

10 K. Chatterjee and L. Doyen. Energy and Mean-Payoff Parity Markov Decision Processes. In *Proceedings of MFCS 2011*, volume 6907 of *LNCS*, pages 206–218. Springer, 2011.

11 K.J. Chung and M. J. Sobel. Discounted MDPs: Distribution functions and exponential utility maximization. *SIAM J. Contr. Optim.*, 25:49–62, 1987.

12 J.A. Filar, D. Krass, and K.W. Ross. Percentile performance criteria for limiting average Markov decision processes. *IEEE Trans. Automat. Contr.*, 40(1):2–10, 1995.

13 J. D. Hamilton. *Time series analysis*, volume 2. Cambridge Univ Press, 1994.

14 John Hull. *Options, futures, and other derivatives*. Pearson, 2009.

15 D.A. Martin. The Determinacy of Blackwell Games. *J. of Symb. Logic*, 63(4):1565–1581, 1998.

16 M. Schäl. Markov Decision Processes in Finance and Dynamic Options. *International Series in Operations Research & Management Science*, 40:461–487, 2002.

17 M. J. Sobel. The variance of discounted Markov decision processes. *J. Appl. Probab.*, 19:794–802, 1982.

18 D. J. White. Minimizing a threshold probability in discounted Markov decision processes. *J. Math. Anal. Appl.*, 173:634–646, 1993.

19 C. Wu and Y. Lin. Minimizing Risk Models in Markov Decision Processes with Policies Depending on Target Values. *J. Math. Anal. Appl.*, 231(1):47–67, 1999.

20 U. Zwick and M. Paterson. The Complexity of Mean Payoff Games on Graphs. *TCS*, 158(1–2):343–359, 1996.

Parameterized Verification of Many Identical Probabilistic Timed Processes

Nathalie Bertrand[1] and Paulin Fournier[2]

1 **Inria Rennes, France**
`nathalie.bertrand@inria.fr`
2 **ENS Cachan Antenne de Bretagne, France**
`paulin.fournier@eleves.bretagne.ens-cachan.fr`

Abstract

Parameterized verification aims at validating a system's model irrespective of the value of a parameter. We introduce a model for networks of identical probabilistic timed processes, where the number of processes is a parameter. Each process is a probabilistic single-clock timed automaton and communicates with the others by broadcasting. The number of processes either is constant (static case), or evolves over time through random disappearances and creations (dynamic case). An example of relevant parameterized verification problem for these systems is whether, independently of the number of processes, a configuration where one process is in a target state is reached almost-surely under all scheduling policies. On the one hand, most parameterized verification problems turn out to be undecidable in the static case (even for untimed processes). On the other hand, we prove their decidability in the dynamic case.

1998 ACM Subject Classification D.2.4 Software/Program verification, F.3.1 Specifying and Verifying and Reasoning about Programs, G.3 Probabilities and Statistics

Keywords and phrases model checking, Markov decision processes, parameterized verification

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.501

1 Introduction

Automated verification of reactive programs started in the 80's with simple models, and specifically finite state automata. It was successfully applied to hardware validation. Later, the focus on software verification called the need for techniques tackling infinite-state systems. The infinite nature is mainly caused by two reasons: either the program handles data structures over infinite domains (*e.g.* counters, dense-time clocks, queues or stacks), or it runs on a network of an arbitrary number of processes. In the latter case, *parameterized verification* aims at verifying the system independently of its actual instantiation, that is, independently of the number of processes involved. Our contribution falls in these two categories: we investigate the parameterized verification of a class of programs over infinite domains.

Already in the 80's, Apt and Kozen showed the undecidability of very general parameterized model-checking problems [7]. For more specific models and properties, one can however achieve some decidability, see *e.g.* [12] where networks of identical finite-state automata are model-checked against regular properties. This framework was later extended to networks of identical timed automata for which safety properties are decidable if the timed automata have a single clock [5] and undecidable otherwise [4]. Broadcasts protocols form a model with an unbounded number of processes that communicate by rendez-vous or broadcast. For broadcast protocols, safety properties are decidable, and liveness properties are undecidable [15]. More recently, a series of papers, initiated with [14], investigates the parameterized

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 501–513

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

verification of a model of networks, *e.g.* suitable for the representation of ad-hoc networks. The nodes in the network are modeled by finite automata that communicate through multiple broadcasts. The decidability status of reachability and coverability problems depends on the topology and its evolution [14]. Later, the positive results have been extended to networks of single-clock timed automata [3].

Simultaneous to the extension of verification techniques to infinite-state models, and with the common objective to verify more complex systems, the models and problems moved from boolean to quantitative. A prominent class of quantitative systems is the one of probabilistic models. Finite-state probabilistic models have been extensively studied and mature tools perform their verification (see [10, Chapter 10] and references therein). However, to the best of our knowledge, the parameterized verification of probabilistic systems hasn't been investigated yet. The model-checking of PCTL formulas against Markov chains with parametric coefficients was first investigated in [13], and the recent work [17] studies the verification of probabilistic programs with parameters. Both use parameters to model unknown probabilities and differ from our setting of parameterized verification where the parameter is the number of processes in a network.

We introduce a modeling formalism that combines infinite-state space, due to an unknown number of processes in the network as well as data structures with infinite domains, and probabilistic behaviors. *Probabilistic timed networks* are formed of many identical probabilistic timed automata [18] with a single clock, and interaction between the processes is modeled by message broadcasting. Moreover, in order to represent some mobility in the network, we further introduce *dynamic* probabilistic timed networks, where processes can disappear and be created, according to fixed probability laws.

A potential application domain for our model is the one of wireless sensor networks (WSN) that consist of a large number of nodes measuring and transmitting data. The number of nodes is a significant parameter while setting up the network, since it affects the performance by influencing the risks of collisions in communications. Most protocols for WSN include probabilistic choices and timing constraints. Also, in many cases the number of nodes in the network evolves over time due to nodes breaking down, or nodes refilling their battery using *e.g.* solar energy. Moreover, in some applications, the placement of sensors and their exact number is unknown. We therefore advocate that (dynamic) probabilistic timed networks with a parametric number of processes make a quite suitable model for WSN protocols. So far, the automated verification of such protocols has been performed for a fixed, and rather small, number of nodes, as in [16] where Prism [18] is used to verify the contention resolution protocol in IEEE standard 802.15.4. In comparison, the parameterized verification of probabilistic timed networks would provide answers for an arbitrary number of processes.

Given a probabilistic timed network, we consider relevant *parameterized verification problems*, such as the following. For a target state of the process model, can a configuration with some process in the target state be reached almost surely, whatever the initial number of processes and for every scheduling policy? Equivalently, the problem is whether independently of the number of processes, the minimum probability to reach a target configuration is 1. Beyond this particular problem, we consider all variants of qualitative questions, *i.e.* where the minimum or maximum probabilities are compared to the thresholds 0 or 1.

On the one hand we prove that most qualitative verification problems are undecidable when the topology is static, that is if the number of processes is constant (but unknown). These undecidability results already hold in the untimed case, *i.e.* when the individual processes are Markov decision processes (MDP) rather than probabilistic timed automata. To establish the undecidability results, we explain how a probabilistic network can simulate

a 2-counter machine, using the processes to encode the counter values. On the other hand, in the dynamic case, we provide a decision procedure for all the parameterized verification problems of interest. In each case, the termination of the algorithm is ensured by a dedicated well-quasi-ordering on configurations of the network, and the correctness of the algorithm relies on a finite attractor property in our model. We also establish a complexity lower-bound by reducing the reachability problem in lossy channel systems: the qualitative parameterized verification problems in the dynamic case are non-primitive recursive.

The rest of the paper is organized as follows. We define the model of dynamic probabilistic timed networks in Section 2 together with the parameterized verification problems we consider. Section 3 is devoted to showing the undecidability in the static case. In Section 4 we establish the decidability results in the dynamic case. We conclude by mentioning open questions and future work.

2 Modeling probabilistic protocols

Given E, an at most countable set, we write $\mathsf{Dist}(E)$ for the set of discrete probability distributions over E, that is, functions $\delta : E \to [0,1]$ such that $\sum_{e\in E} \delta(e) = 1$.

For an arbitrary set E, we write $\mathbb{M}(E)$ for the set of multisets over E, *i.e.* the set of multiplicity functions $f : E \to \mathbb{N}$. We also write $f = \langle x, x, x, y\rangle$ for the multiset f defined as $f(x) = 3$, $f(y) = 1$ and $\forall z \in E \setminus \{x, y\}, f(z) = 0$. We now introduce simple operations on multisets. For $f \in \mathbb{M}(E)$ and $x \in E$ we write $f + x$ for the multiset f' defined by $f'(x) = f(x) + 1$ and $f'(y) = f(y)$ otherwise. Symmetrically, and assuming $f(x) > 0$, $f - x$ is a notation for f' such that $f' + x = f$.

Given a clock $\mathtt{x}$, we write $\mathcal{G}(\mathtt{x})$ for the set of guards over $\mathtt{x}$, *i.e.* conjunctions of atomic constraints of the form $\mathtt{x} \sim k$ for $\sim \in \{<, \leq, >, \geq, =\}$ and $k \in \mathbb{N}$. The trivial guard, consisting of no constraints, is written true. A clock is said to *satisfy* a guard, denoted $\mathtt{x} \models g$ if it's value satisfies all the constraints of the guard. The set of possible updates for clock $\mathtt{x}$ is $\mathsf{Up} = \{\mathtt{x} := 0, \emptyset\}$: $\mathtt{x}$ is either reset or left unchanged. We write $\mathsf{up}(\mathtt{x})$ the result on $\mathtt{x}$ of the update $\mathsf{up} \in \mathsf{Up}$.

2.1 Probabilistic timed networks

▶ **Definition 1** (Probabilistic one-clock timed protocol)**.** A *probabilistic one-clock timed protocol* (or for short *probabilistic timed protocol*) is a tuple $\mathcal{P} = (Q, q_0, \mathtt{x}, \Sigma, \Delta)$ where

- Q is a finite set of control states, and $q_0 \in Q$ is initial,
- $\mathtt{x}$ is a clock,
- Σ is a finite message alphabet with a subset Σ_ε of internal labels,
- Δ is the finite probabilistic edge relation, partitioned into
  - internal actions: $\Delta_{\mathsf{i}} \subseteq Q \times \mathcal{G}(\mathtt{x}) \times \Sigma_\varepsilon \times \mathsf{Dist}(\mathsf{Up} \times Q)$,
  - broadcasts: $\Delta_{\mathsf{b}} \subseteq Q \times \mathcal{G}(\mathtt{x}) \times !(\Sigma \setminus \Sigma_\varepsilon) \times \mathsf{Up} \times Q$,
  - receptions: $\Delta_{\mathsf{r}} \subseteq Q \times \mathcal{G}(\mathtt{x}) \times ?(\Sigma \setminus \Sigma_\varepsilon) \times \mathsf{Up} \times Q$.

A simple example of probabilistic timed protocol modelling mutual exclusion over two resources is given in Figure 1. Internal actions are $\{\alpha^i \mid i \in [\![0, 2]\!]\}$, and broadcast actions are $\{\beta^i, \beta^{i'} \mid i \in \{1, 2\}\}$. This example will be further developed in the sequel.

▶ Remark. In our model, a control state can be the source of several internal actions, each giving rise to a probability distribution for the successor state, whereas broadcasts and receptions are deterministic. This is not a real restriction, since systems with nondeterministic and probabilistic choices for broadcast and receptions can be encoded in our model by introducing intermediary states and additional internal actions.

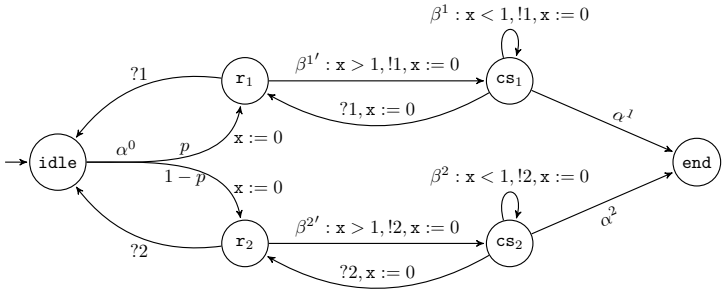


Figure 1 A probabilistic timed protocol modeling mutual exclusion over two resources.

Probabilistic timed protocols are probabilistic timed automata [18] with a single clock. We introduced a new terminology to highlight this particularity, and more importantly, to emphasize the communicative nature of probabilistic timed protocols.

▶ **Definition 2** (Probabilistic timed network)**.** A probabilistic timed network $\mathcal{P}^N$, is composed of $N \in \mathbb{N}_{>0}$ copies, called *processes*, of a probabilistic timed protocol $\mathcal{P}$.

The intuitive interpretation of a probabilistic timed network $\mathcal{P}^N$ is that N processes arranged in a clique execute the probabilistic timed protocol $\mathcal{P}$ simultaneously.

▶ **Example 3.** On the simple example from Figure 1 modeling mutual exclusion over two resources, each process starts in state `idle` and can at any time request a resource. The choice of the allocated resource is probabilistic. The requesting process must then stay in the corresponding request state ($\mathtt{r}_i$) at least one time unit before moving to the critical section state ($\mathtt{cs}_i$) representing usage of resource i. If a requesting process receives a message indicating that the resource is already used, it moves back to the initial state `idle`. Hence, when granted the access to a resource, a process may use it for at least 1 time unit, and can inform the others by broadcast that he is still using it to be granted an other time unit.

Let us detail the semantics of the probabilistic timed network $\mathcal{P}^N$. A *configuration* is a finite multiset $\gamma \in \mathbb{M}(Q \times \mathbb{R}_+)$ over the set of pairs composed of a control state and a real value for the clock. Intuitively, if configuration γ contains exactly two occurrences of $(q, 0.5)$, written $\gamma(q, 0.5) = 2$, then two processes have current state q and current clock value $\mathtt{x} = 0.5$. Moreover, since N processes are involved in the network, $\sum_{(q,x)|\gamma(q,x)>0} \gamma(q, x) = N$. In the sequel, the set of configurations with N processes is written Conf^N.

The semantics of $\mathcal{P}^N$ is given in terms of a timed Markov decision process $[\![\mathcal{P}^N]\!] = (\mathsf{Conf}^N, \gamma_0^N, \mathsf{T}^N, \mathsf{Act} \cup \mathbb{R}_+)$, where γ_0^N is the initial configuration, defined by $\gamma_0^N(q_0, 0) = N$ and $\gamma_0^N(q, x) = 0$ otherwise; T^N is the set of transitions (defined in the sequel); and actions are partitioned into time elapsing $\mathbb{R}_+$, and discrete actions $\mathsf{Act} = Q \times \mathbb{R}_+ \times \Delta \cup \{\mathsf{unlock}\}$ where unlock is a special action. From any configuration $\gamma \in \mathsf{Conf}^N$, the set of possible discrete actions depends on the control state and clock value of the processes in γ, which explains the typing of discrete actions: $Q \times \mathbb{R}_+ \times \Delta$. Informally, when performing a discrete action from configuration γ, first a process is selected nondeterministically, and second, a (communication or internal) transition enabled for that process is performed. However, since we do not distinguish processes with same control state and same clock value, the intuitive semantics of discrete steps is rather to select a pair (q, x) with $\gamma(q, x) > 0$ and then to fire for some process in state (q, x) an enabled action. Messages are broadcast to all other processes, whereas internal actions only affect the chosen process. Moreover, in order to forbid finite

runs, we use a special action unlock that is enabled only when no other discrete action is enabled, even after some time elapsing.

We now define formally the transition function T^N and exemplify it on the example protocol from Figure 1 for the configuration $\gamma_{\mathsf{ex}} = \langle(\mathtt{idle}, 2), (\mathtt{r}_1, 0.3), (\mathtt{r}_1, 0.4), (\mathtt{cs}_1, 0.8)\rangle$. T^N is composed of the following transitions:

- time elapse: For every delay $\tau \in \mathbb{R}_+$ and every configuration $\gamma \in \mathsf{Conf}^N$, there exists a deterministic transition in the Markov decision process $[\![\mathcal{P}^N]\!]$ from γ to γ' defined by $\gamma'(q, x+\tau) = \gamma(q, x)$ (denoted $\gamma' = \gamma + \tau$), for all states $q \in Q$ and clock values $x \in \mathbb{R}_+$. In such a case we write $\gamma \xrightarrow{\tau} \gamma'$.
  E.g.: $\gamma_{\mathsf{ex}} \xrightarrow{0.1} \langle(\mathtt{idle}, 2.1), (\mathtt{r}_1, 0.4), (\mathtt{r}_1, 0.5), (\mathtt{cs}_1, 0.9)\rangle$
- internal: For every internal action $\alpha = (q, g, \varepsilon, \delta) \in \Delta_{\mathsf{i}}$, every $\gamma \in \mathsf{Conf}^N$ and every clock value $x \in \mathbb{R}_+$, such that $\gamma(q, x) > 0$ and $x \models g$, α_x is *enabled* from γ in the Markov decision process $[\![\mathcal{P}^N]\!]$ and yields a distribution μ defined by $\mu(\gamma'|\gamma, \alpha_x) = \delta(\mathsf{up}, q')$ where $\gamma' = \gamma - (q, x) + (q', \mathsf{up}(x))$. Note that several α actions can be available in γ, for various clock values x; this is why in $[\![\mathcal{P}^N]\!]$, we attach the subscript x to α. Here, we write $\gamma \xrightarrow{\alpha_x, \mu(\gamma'|\gamma, \alpha_x)} \gamma'$.
  E.g.: $\gamma_{\mathsf{ex}} \xrightarrow{\alpha^0_2, 1-p} \langle(\mathtt{r}_2, 0), (\mathtt{r}_1, 0.3), (\mathtt{r}_1, 0.4), (\mathtt{cs}_1, 0.8)\rangle$
- communication: For every broadcast $\beta = (q, g, !a, \mathsf{up}, q') \in \Delta_{\mathsf{b}}$, every $\gamma \in \mathsf{Conf}^N$ and every clock value $x \in \mathbb{R}_+$ such that $\gamma(q, x) > 0$ and $x \models g$, β_x is *enabled* from γ and is a deterministic transition from γ to γ' in the Markov decision process $[\![\mathcal{P}^N]\!]$, where γ' is obtained in three steps
  - $\gamma_1 = \gamma - (q, x)$ (the process responsible for the broadcast is treated separately)
  - $\gamma_2(r', y') = \sum\{\gamma_1(r, y) \mid \exists (r, g', ?a, \mathsf{up}', r') \in \Delta_{\mathsf{r}} \wedge y \models g' \wedge \mathsf{up}'(y) = y'\}$ (all other processes receive the message)
  - $\gamma' = \gamma_2 + (q', \mathsf{up}(x))$.

  For such communications, we write $\gamma \xrightarrow{\beta_x} \gamma'$.
  E.g.: $\gamma_{\mathsf{ex}} \xrightarrow{\beta^1_{0.8}} \langle(\mathtt{idle}, 2), (\mathtt{idle}, 0.3), (\mathtt{idle}, 0.4), (\mathtt{cs}_1, 0)\rangle$
- deadlock: Any configuration $\gamma \in \mathsf{Conf}^N$ such that for every $\tau \in \mathbb{R}_+$ there is no enabled actions from $\gamma + \tau$, is called a *deadlock* configuration. Action unlock is then *enabled* from γ, and does not change the configuration. Here, we write $\gamma \xrightarrow{\mathsf{unlock}} \gamma$.
  E.g.:$\langle(\mathtt{end}, 2), (\mathtt{end}, 0.3), (\mathtt{end}, 1.4)\rangle \xrightarrow{\mathsf{unlock}} \langle(\mathtt{end}, 2), (\mathtt{end}, 0.3), (\mathtt{end}, 1.4)\rangle$

2.2 Dynamic probabilistic timed networks

Probabilistic timed networks from Definition 2 are networks where the number of processes is constant. We now discuss a reasonable way to introduce dynamism in the network. In the application to wireless sensor networks, the number of nodes can change during the execution of the system, since nodes can break down or run out of battery, but also, can be newly inserted or refill their battery. We therefore propose a model of probabilistic timed networks, in which the number of processes evolves over time, and in which, disappearances and creations of processes are independent random events following given probability distributions. Abstracting dynamism by random events seems a good trade-off between simplicity and realism of the model.

▶ **Definition 4** (Dynamic probabilistic timed network)**.** A *dynamic probabilistic timed network*, written $\mathcal{P}^{N_0}_{\Lambda}$ is composed of initially N_0 copies of $\mathcal{P}$, together with a pair of disappearance and creation rates $\Lambda = (\lambda_-, \lambda_+) \in (0, 1)^2$.

The rates λ_+ and λ_- represent dynamic creation and disappearance of processes according to fixed probabilistic laws: after each discrete action , each process disappears with probability λ_-, followed by the creation of k processes (in control state q_0 with clock value 0) with probability $\lambda_+^k(1-\lambda_+)$, for every integer $k \in \mathbb{N}$. Obviously if $\lambda_+ = \lambda_- = 0$, the number of processes is constant and we recover the model of (static) probabilistic timed network from Definition 2.

The formal semantics of a dynamic probabilistic timed network $\mathcal{P}_\Lambda^{N_0}$ is obtained from the ones of $\mathcal{P}^N$ for each $N \in \mathbb{N}$. More precisely $[\![\mathcal{P}_\Lambda^{N_0}]\!]$ is a timed Markov decision process $(\mathsf{Conf}, \gamma_0^{N_0}, \mathsf{T}, \mathsf{Act} \cup \mathbb{R}_+)$, where the transition function T is defined based on the T^N's and the rates Λ ((see below)). The set of configurations is $\mathsf{Conf} = \bigcup_N \mathsf{Conf}^N$, and the set of discrete actions in the MDP is still $\mathsf{Act} = Q \times \mathbb{R}_+ \times \Delta \cup \{\mathsf{unlock}\}$. The initial configuration is still $\gamma_0^{N_0}$ defined by $\gamma_0(q_0, 0) = N_0$ and $\gamma_0(q, x) = 0$ otherwise. Note that the only difference between two instances of the dynamic probabilistic timed network, $[\![\mathcal{P}_\Lambda^{N_0}]\!]$ and $[\![\mathcal{P}_\Lambda^{N_0'}]\!]$, lies in their respective initial state.

We now detail the transition function T. For every $\gamma \in \mathsf{Conf}$, $\alpha \in \mathsf{Act} \cup \mathbb{R}_+$ and every transition $\gamma \xrightarrow{\alpha,p} \gamma'$ (in T^N, for N the size of γ), there are transitions $\gamma \xrightarrow{\alpha,p\cdot p'} \gamma''$ in T for every configuration γ'' that can be obtained from γ' through disappearance followed by creation of processes, with the appropriate probability. Most likely, γ'' contains a different number of processes than γ and γ'. Let us explain how the probability p' is defined. We write $\mathbf{p}_-(\gamma_1, \gamma_2)$ for the probability to obtain γ_2 from γ_1 when processes disappear; recall that each process can disappear with probability λ_-. Similarly, $\mathbf{p}_+(\gamma_1, \gamma_2)$ is the probability to obtain γ_2 from γ_1 when processes are created; recall that for every integer k, k processes are created (in $(q_0, 0)$) with probability $\lambda_+^k(1-\lambda_+)$. Using these notations, the probability to move from γ' to γ'' by disappearance and creation of processes is $p' = \sum_{\gamma_1} \mathbf{p}_-(\gamma', \gamma_1) \cdot \mathbf{p}_+(\gamma_1, \gamma'')$.

An *execution* in $[\![\mathcal{P}_\Lambda^{N_0}]\!]$ is a finite or infinite sequence $\rho = \gamma_0 \to \gamma_1 \to \gamma_2 \cdots$, where the transitions correspond to time elapsing, internal actions, communications or the special action unlock.

2.3 Schedulers

Schedulers (also called strategies or policies) resolve the non-determinism in Markov decision processes. We restrict to stationary deterministic schedulers (sometimes also called pure memoryless schedulers), that make their decision based on the current configuration only.

▶ **Definition 5** (Scheduler). A *scheduler* for the timed Markov decision process $[\![\mathcal{P}_\Lambda^{N_0}]\!] = (\mathsf{Conf}, \gamma_0^{N_0}, \mathsf{T}, \mathsf{Act} \cup \mathbb{R}_+)$ is a function $\sigma : \mathsf{Conf} \to \mathbb{R}_+ \times \mathsf{Act}$, specifying in each configuration the delay and discrete action to perform, and such that

(i) whenever $\sigma(\gamma) = (\tau, (q, x, \alpha))$, then $(\gamma + \tau)(q, x) > 0$ and α is enabled in (q, x), and

(ii) whenever $\sigma(\gamma) = (\tau, \mathsf{unlock})$, γ is a deadlock configuration.

Recall that $\mathsf{Act} = (Q \times \mathbb{R}_+ \times \Delta) \cup \{\mathsf{unlock}\}$. In words, a scheduler resolves the non-determinism by choosing in each configuration a delay τ, followed by some discrete action. Condition (i) ensures that σ only chooses enabled discrete actions; moreover, thanks to condition (ii), if the system is deadlock, σ chooses the special action unlock.

The dynamic probabilistic timed network $\mathcal{P}_\Lambda^{N_0}$ together with a fixed scheduler σ give rise to a Markov chain with state space Conf, and in which the probability measure over executions of $\mathcal{P}_\Lambda^{N_0}$, defined in a standard way, is written $\mathbb{P}_\sigma$.

2.4 Problem formulation

We are now in a position to introduce relevant verification questions for dynamic probabilistic timed networks. In this paper, we focus on qualitative reachability problems.

Let $q_f \in Q$, and ρ an execution of $\mathcal{P}_\Lambda^{N_0}$. Execution ρ satisfies $\Diamond q_f$, denoted $\rho \models \Diamond q_f$, if there exists a configuration γ along ρ with $\gamma(q_f, x) > 0$ for some arbitrary clock value x. Given a scheduler σ for $\mathcal{P}_\Lambda^{N_0}$, we write $\mathbb{P}_\sigma(\mathcal{P}_\Lambda^{N_0} \models \Diamond q_f)$ for the probability under σ of the set of executions ρ with $\rho \models \Diamond q_f$. Further, $\min_\sigma \mathbb{P}_\sigma(\mathcal{P}_\Lambda^{N_0} \models \Diamond q_f)$ (resp. $\max_\sigma \mathbb{P}_\sigma(\mathcal{P}_\Lambda^{N_0} \models \Diamond q_f)$) denotes the minimum (resp. maximum) of these values among all possible schedulers.

We define the following family of decision problems, for $\texttt{opt} \in \{\min, \max\}$, $\texttt{b} \in \{0, 1\}$ and $\sim \in \{<, =, >\}$

$\textsc{Reach}^{\sim \texttt{b}}_{\texttt{opt}}$
Input: A probabilistic timed protocol $\mathcal{P}$, a rate pair Λ, and a control state $q_f \in Q$.
Question: Does there exist $N_0 \in \mathbb{N}_{>0}$ such that $\texttt{opt}_\sigma \, \mathbb{P}_\sigma(\mathcal{P}_\Lambda^{N_0} \models \Diamond q_f) \sim \texttt{b}$?

Note that we state the decision problems in an existential way; however negating the condition $\sim \texttt{b}$, we equivalently deal with universal quantification over network sizes.

3 Decidability status in the static case

In this section, we consider the static case, i.e., when $\lambda_- = \lambda_+ = 0$, and start by establishing simple decidability results.

▶ **Proposition 6.** *The problems* $\textsc{Reach}^{=0}_{\max}$, $\textsc{Reach}^{<1}_{\max}$, *and* $\textsc{Reach}^{>0}_{\max}$ *are decidable for static probabilistic timed networks.*

Proof. First, remark that $\max_\sigma \mathbb{P}_\sigma(\mathcal{P}^N \models \Diamond q_f) \leq \max_\sigma \mathbb{P}_\sigma(\mathcal{P}^{N+1} \models \Diamond q_f)$. Indeed in the network $\mathcal{P}^{N+1}$, some schedulers only involve N processes. Hence, there exists N such that $\max_\sigma \mathbb{P}_\sigma(\mathcal{P}^N \models \Diamond q_f) = 0$ if and only if $\max_\sigma \mathbb{P}_\sigma(\mathcal{P}^1 \models \Diamond q_f) = 0$. And the same holds for the case < 1. The decidability of $\textsc{Reach}^{=0}_{\max}$, and $\textsc{Reach}^{<1}_{\max}$ thus derive from the decidability of $\max_\sigma \mathbb{P}_\sigma(M \models \Diamond q_f) = 0$ (resp. < 1) for M a finite-state Markov decision process [18, 10].

For $\textsc{Reach}^{>0}_{\max}$, observe that, for $N \in \mathbb{N}$, $\max_\sigma \mathbb{P}_\sigma(\mathcal{P}^N \models \Diamond q_f) > 0$ if and only if there exists a scheduler σ with $\mathbb{P}_\sigma(\mathcal{P}^N \models \Diamond q_f) > 0$ if and only if there exists an execution ρ in $\mathcal{P}^N$ with $\rho \models \Diamond q_f$. The decidability of $\textsc{Reach}^{>0}_{\max}$ is therefore a consequence of the decidability of the parameterized reachability problem in the non-probabilistic case, established in [3]. ◀

We now consider the remaining cases, and prove their undecidability, already for the restricted class of untimed probabilistic networks.

▶ **Theorem 7.** *The problems* $\textsc{Reach}^{=1}_{\max}$, $\textsc{Reach}^{=0}_{\min}$, $\textsc{Reach}^{>0}_{\min}$, $\textsc{Reach}^{<1}_{\min}$ *and* $\textsc{Reach}^{=1}_{\min}$ *are undecidable for static probabilistic (timed) networks.*

Proof sketch. Let us explain the key ideas of the undecidability proof for $\textsc{Reach}^{<1}_{\min}$, which is inspired by techniques from [1]. We reduce from the halting problem of a deterministic infinitely testing 2-counter machine $\mathcal{M}$, which is known to be undecidable [19]. From $\mathcal{M}$, we build a probabilistic protocol $\mathcal{P}$, such that, for every $N \in \mathbb{N}_{>0}$, the network $\mathcal{P}^N$ weakly simulates $\mathcal{M}$: each execution either faithfully simulates $\mathcal{M}$ or a simulation error is detected, and some process is in an error state.

First, one process is selected to play the role of the *controller*, that keeps track of the control state in $\mathcal{M}$. The other processes will serve to encode the values of the counters, and are grouped in state `idle`. The increment of counter $\texttt{c}_i$ is represented by moving a

process from $\texttt{idle}$ to state $\texttt{c}_i$ where the counter value is encoded. This can be done by two communications: one from the controller to the processes in $\texttt{idle}$, followed by one from one "counter" process to the controller. For the *test to zero decrement* operation of counter $\texttt{c}_i$, the controller randomly guesses whether the counter value is zero or not. If it guesses zero while $\texttt{c}_i > 0$, all processes in $\texttt{c}_i$ move to an error state $\texttt{err}_z$. Symmetrically, if it guesses non-zero while $\texttt{c}_i = 0$, the infeasibility of the decrement will force the controller to move to an other error state $\texttt{err}$. If the guess is correct, the simulation continues, and no process are in error states. The only way to avoid error states is thus to faithfully simulate $\mathcal{M}$. Indeed, in the probabilistic protocol $\mathcal{P}$, the only blocking state for the controller is $\texttt{k}_{acc}$, representing the accepting state of $\mathcal{M}$, and from all other states it can reach state $\texttt{err}$. As a consequence, all maximal executions reach $\texttt{err}$ except for the ones which faithfully simulate $\mathcal{M}$ and end in $\texttt{k}_{acc}$. Moreover, for N large enough there exists an execution ρ in $\mathcal{P}^N$ simulating π the unique maximal execution of $\mathcal{M}$. Assuming $\mathcal{M}$ terminates, π is finite, and thus ρ has a positive probability under some scheduler σ. We derive that $\min_\sigma \mathbb{P}_\sigma(\mathcal{P}^N \models \Diamond(\texttt{err} \cup \texttt{err}_z)) < 1$.

Assume now that $\mathcal{M}$ does not terminate, and thus its unique execution π contains infinitely many zero tests, with a non-zero counter value. Under any scheduler σ, the probability of executions simulating faithfully π is then zero. As a consequence reaching an error state is almost-sure, for all schedulers.

The undecidability of $\textsc{Reach}^{<1}_{\min}$ derives from the observation that this construction ensures: $\exists N \in \mathbb{N}_{>0}\ \min_\sigma \mathbb{P}_\sigma(\mathcal{P}^N \models \Diamond(\texttt{err}_z \cup \texttt{err})) < 1 \iff \mathcal{M}$ terminates. ◀

4 Decidability status in the dynamic case

We now turn to dynamic probabilistic timed networks, and will see that the decidability of the qualitative parameterized verification problems is recovered thanks to probabilistic disappearance and creation of processes. To establish this result, we first abstract the Markov decision process $[\![\mathcal{P}^{N_0}_\Lambda]\!]$ into a discrete MDP, using an ad-hoc region abstraction which preserves the extremal probabilities. Then, we prove that the so-called region MDP enjoys the finite attractor property, and that it can be equipped with a well-quasi-ordering on its configurations. These two properties entail the decidability of the qualitative verification questions in the region MDP that are equivalent to the initial parameterized verification problems in the network.

4.1 Region abstraction

The classical region abstraction for timed automata, presented in the seminal paper [6], is based on the observation that the relevant information in clock valuations consists of the integer part of each clock (up to the maximal constant appearing in guards) and the ordering of their fractional parts. In our context, since the number of processes is unbounded (hence the number of clocks is unbounded), the region abstraction cannot be used directly. Still, based on classical regions, we present an equivalence relation over configurations.

For $x \in \mathbb{R}_+$ a non-negative real, we denote $\lfloor x \rfloor$ its integer part and $\{x\}$ its fractional part. Note that $x = \lfloor x \rfloor + \{x\}$.

▶ **Definition 8** (Region equivalence)**.**
Let $b \in \mathbb{N}$. Two configurations $\gamma_1 = \langle (q_1, x_1), \dots, (q_n, x_n) \rangle$ and $\gamma_2 = \langle (p_1, y_1), \dots, (p_m, y_m) \rangle$ are *region equivalent*, denoted $\gamma_1 \approx_b \gamma_2$ if there exists a bijection $h : [\![1, n]\!] \to [\![1, m]\!]$ such that the following conditions hold, $\forall i, j \in [\![1, n]\!]$:

(i) $q_i = p_{h(i)}$: states of processes agree,
(ii) $(\lfloor x_i \rfloor \leq b) \vee (\lfloor y_{h(i)} \rfloor \leq b) \Rightarrow \lfloor y_{h(i)} \rfloor = \lfloor x_i \rfloor$: integer part of clocks agree up to b,

(iii) $(\{x_i\} = 0) \Leftrightarrow (\{y_{h(i)}\} = 0)$: clocks with integer value agree,
(iv) for $\sim \in \{<, =, >\}$, $(\{x_i\} \sim \{x_j\}) \Leftrightarrow (\{y_{h(i)}\} \sim \{y_{h(j)}\})$: the two orderings of fractional parts coincide.

In Definition 8, the region equivalence is indexed by a bound b. For a given protocol $\mathcal{P}$ this bound is set as the maximal constant appearing in guards. For simplicity, we omit it in what follows and simply write $\approx$. Given $\gamma \in \mathsf{Conf}$ a configuration, $[\gamma]$ denotes its equivalence class for $\approx$, and is called a *region-configuration*. As an example with $b = 1$, $\langle (q_1, 0), (q_2, 2.1)\rangle \approx \langle (q_1, 0), (q_2, 4.5)\rangle \not\approx \langle (q_1, 0.8), (q_2, 4.5)\rangle$.

Similarly to classical regions in timed automata, two region-equivalent configurations exhibit similar future behaviors in $\mathcal{P}_\Lambda^{N_0}$. This is formalized in the following proposition:

▶ **Proposition 9.** *Let γ_1 and γ_2 be two configurations. If $\gamma_1 \approx \gamma_2$, then γ_1 and γ_2 are time-abstract bisimilar,* i.e.*:*

- $\forall \tau_1 \in \mathbb{R}_+$, $\forall \gamma_1 \xrightarrow{\tau_1} \gamma_1'$, $\exists \tau_2 \in \mathbb{R}_+$, $\exists \gamma_2' \in [\gamma_1']$ *such that* $\gamma_2 \xrightarrow{\tau_2} \gamma_2'$*;*
- $\forall \alpha \in \mathsf{Act}$, $\forall \gamma_1 \xrightarrow{\alpha,p} \gamma_1'$, $\exists \gamma_2' \in [\gamma_1']$ *such that* $\gamma_2 \xrightarrow{\alpha,p} \gamma_2'$.

Thanks to Proposition 9, the MDP $[\![\mathcal{P}_\Lambda^{N_0}]\!]$ can be abstracted into its quotient by $\approx$, a countable MDP, formally defined as follows:

▶ **Definition 10** (Region MDP)**.** The *region MDP* of a dynamic network $\mathcal{P}_\Lambda^{N_0}$ is $\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!]) = (\mathsf{Conf}_\approx, [\gamma_0^{N_0}], \mathsf{T}_\approx, \mathsf{Act}_\approx)$ defined from $[\![\mathcal{P}_\Lambda^{N_0}]\!] = (\mathsf{Conf}, \gamma_0^{N_0}, \mathsf{T}_\Lambda, \mathsf{Act})$ by $\mathsf{Conf}_\approx = \{[\gamma] \mid \gamma \in \mathsf{Conf}\}$, $\mathsf{Act}_\approx = \mathsf{Act} \cup \{\mathtt{TSucc}\}$, and $\mathsf{T}_\approx \subseteq \mathsf{Conf}_\approx \times \mathsf{Act}_\approx \times \mathsf{Dist}(\mathsf{Conf}_\approx)$ is such that

- $[\gamma_1] \xrightarrow{\mathtt{TSucc}} [\gamma_2] \in \mathsf{T}_\approx$ as soon as $\exists \tau \in \mathbb{R}_+$, $\exists \gamma_1 \xrightarrow{\tau} \gamma_2 \in \mathsf{T}_\Lambda$ such that $\gamma_1 \not\approx \gamma_2$ and $\forall \tau' \leq \tau$, $\gamma_1 + \tau' \in [\gamma_1] \cup [\gamma_2]$;
- $[\gamma_1] \xrightarrow{\alpha,p} [\gamma_2] \in \mathsf{T}_\approx$ as soon as $\exists x$, $\exists \gamma_1 \xrightarrow{\alpha_x,p'} \gamma_2 \in \mathsf{T}_\Lambda$, and $p = \sum_{\gamma_2' \approx \gamma_2} \{p' \mid \gamma_1 \xrightarrow{\alpha_x,p'} \gamma_2'\}$,

Remark that the region MDP $\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!])$ is well-defined, thanks to Proposition 9. First, the successor by `TSucc` is well-defined, since the next time-successor is uniform inside an equivalence class for $\approx$. Second, the existence of successors by discrete actions are also uniform inside an equivalence class, and the sum $\sum_{\gamma_2' \approx \gamma_2} \{p' \mid \gamma_1 \xrightarrow{\alpha_x,p'} \gamma_2'\}$ does not depend on γ_1 but only on $[\gamma_1]$. Last, the above sum is well-defined since there are only finitely many γ_2' such that $\gamma_2' \approx \gamma_2$ and $\gamma_1 \xrightarrow{\alpha_x,p'} \gamma_2'$.

A *scheduler* for $\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!])$ is a mapping $\mathcal{U} : \mathsf{Conf}_\approx \to \mathsf{Act}_\approx$ resolving the non-determinism and such that $\mathcal{U}([\gamma])$ is enabled in $[\gamma]$. Given $q_f \in Q$ a state of the protocol $\mathcal{P}$, we write $\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!]) \models \Diamond q_f$ for the set of executions in $\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!])$ that eventually visit some $[\gamma]$ with $\gamma(q_f, x) > 0$ for some $x \in \mathbb{R}_+$. When a scheduler $\mathcal{U}$ is fixed, $\mathbb{P}_\mathcal{U}(\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!]) \models \Diamond q_f)$ is the probability in $\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!])$ under $\mathcal{U}$ of reaching q_f. As intended, the region MDP $\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!])$ is equivalent to $[\![\mathcal{P}_\Lambda^{N_0}]\!]$ in the following sense:

▶ **Proposition 11.** *Let $\sim \in \{<, =, >\}$ and $\mathtt{b} \in \{0, 1\}$.*

$$\exists \sigma\ \mathbb{P}_\sigma(\mathcal{P}_\Lambda^{N_0} \models \Diamond q_f) \sim \mathtt{b} \iff \exists \mathcal{U}\ \mathbb{P}_\mathcal{U}(\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!]) \models \Diamond q_f) \sim \mathtt{b}\ .$$

The right-to-left implication of the equivalence is the easiest: From a scheduler $\mathcal{U}$ in the region MDP, one can define a scheduler σ for the original network by mimicking the discrete actions, and transforming the abstract delays (`TSucc`) into concrete ones. The other implication is more subtle since a scheduler in the network may well take different decisions for equivalent configurations. Yet, we can prove that for qualitative properties, one can always consider schedulers that are region-uniform.

A consequence of Proposition 11 is that for $\mathtt{opt} \in \{\min, \max\}$,

$$\mathtt{opt}_\sigma \mathbb{P}_\sigma(\mathcal{P}_\Lambda^{N_0} \models \Diamond q_f) \sim \mathtt{b} \quad \Longleftrightarrow \quad \mathtt{opt}_\mathcal{U} \mathbb{P}_\mathcal{U}(\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!]) \models \Diamond q_f) \sim \mathtt{b} \ .$$

Therefore the parameterized verification problems are equivalent in the dynamic probabilistic timed network and its region MDP.

4.2 Deciding parameterized problems on the region MDP

Using Proposition 11, the qualitative reachability problems are equivalent in $\mathcal{P}_\Lambda^{N_0}$ and in $\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!])$. We now expose how to decide them in the region MDP. The decidability relies on two key arguments: first, $\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!])$ admits a finite attractor, and second, in $\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!])$, the predecessor operator is effective and preserves upward-closure for some well-quasi-ordering. The decidability can then be derived by applying similar techniques as for nondeterministic and probabilistic lossy channel systems [9, 11].

The finite attractor property was introduced originally for probabilistic lossy channel systems (pLCS) and states that Markov chains induced by pLCS admit a finite recurrent set [2, 8]. Roughly said, some results for finite Markov chains extend to infinite Markov chains with a finite attractor. A Markov chain is said to have a *finite attractor*, if there exists a finite set of states that is visited infinitely often almost-surely (*i.e.*, with probability 1). Further, a Markov decision process has a finite attractor if there exists a finite set of states which is an attractor under all possible schedulers.

▶ **Proposition 12.** *The set* $\{[\emptyset]\} \subseteq 2^{\mathsf{Conf}_\approx}$ *is an attractor for* $\mathcal{R}([\![\mathcal{P}_\Lambda^{N_0}]\!])$.

▶ Remark. In the region MDP, the empty region-configuration $[\emptyset]$, with no process, forms an attractor. Since any initial region-configuration $[\gamma_0^{N_0}] = [\langle (q_0, 0)^{N_0} \rangle]$ can be reached, with positive probability, from the empty region-configuration, this entails that along infinite executions, almost-surely, all possible initial region-configurations $[\gamma_0^{N_0}]$ are visited infinitely often. A consequence is that the validity of a qualitative property does not depend on the initial number of processes N_0. Moreover the extremal probabilities of reaching a configuration with some process in q_f is either 0 or 1.

These observations can be thought as drawbacks of the dynamic probabilistic timed network model. Yet, we argue that the setting and the decidability results to come can easily be adapted to the case where a fixed number of processes cannot disappear. This would lead to a realistic model which, for example, can represent a system with fixed antennas and a parametric number of wireless devices that disappear and are created following probabilistic laws. More importantly, it would yield a system still with a finite attractor (the finite set of possible states for the antennas), but in which the reachability probabilities can be different from 0 and 1. The fact that the empty region-configuration forms a finite attractor, should thus not be seen as an unrealistic feature.

Now that the existence of a finite attractor has been established, we define an appropriate partial order $\preceq$ on $\mathsf{Conf}_\approx$. Intuitively, region-configuration c is smaller than c', if we can remove some processes of a configuration of $\gamma' \in c'$ and obtain a configuration $\gamma \in c$. We also add a side-condition on the clocks with integer value, which is necessary to obtain a good property on the predecessor operator. Formally:

▶ **Definition 13** (Ordering on region-configurations)**.** The order $\preceq \subseteq \mathsf{Conf}_\approx \times \mathsf{Conf}_\approx$ is defined as follows: for $c, c' \in \mathsf{Conf}_\approx$, $c \preceq c'$ if there exists $\gamma \in c$ and $\gamma' \in c'$ such that:

(i) $\forall q \in Q,\ \forall x \in \mathbb{R}_+,\ \gamma(q, x) \leq \gamma'(q, x)$; and
(ii) $\sum_{q \in Q} \sum_{n \in \mathbb{N}} \gamma(q, n) = 0 \quad \Longrightarrow \quad \sum_{q \in Q} \sum_{n \in \mathbb{N}} \gamma'(q, n) = 0.$

Adapting the proof of Higman's lemma, one obtains:

▶ **Proposition 14.** *The partial order $\preceq$ is a well-quasi-ordering.*

We now consider the upward-closure operator $\uparrow$ w.r.t. $\preceq$: given $C \subseteq \mathsf{Conf}_{\approx}$,

$$\uparrow C = \{c' \in \mathsf{Conf}_{\approx} \mid \exists c \in C \text{ such that } c \preceq c'\} \ .$$

A set $C \subseteq \mathsf{Conf}_{\approx}$ is said to be *upward-closed* whenever $C = \uparrow C$. Since $\preceq$ is a well-quasi-ordering, any non-decreasing sequence of upward-closed sets eventually stabilizes. This property will be useful in the sequel.

Last we define, in the region MDP, the predecessor operator: for $c \in \mathsf{Conf}_{\approx}$, $\mathsf{Pre}(c)$ denotes the set of region-configurations $c' \in \mathsf{Conf}_{\approx}$ that can reach c in one step. It happens that for any upward-closed set $C \subseteq \mathsf{Conf}_{\approx}$, the set $\mathsf{Pre}(C)$ can be computed, and is itself upward closed.

▶ **Proposition 15.** Pre *preserves upward-closure, is effective and satisfies* $\mathsf{Pre}(C) = \mathsf{Pre}(\uparrow C)$.

Propositions 12, 14 and 15 are decisive to obtain the decidability of parameterized verification problems in the region MDP $\mathcal{R}([\![\mathcal{P}_{\Lambda}^{N_0}]\!])$. Combined with Proposition 11 we obtain the main contribution of this paper:

▶ **Theorem 16.** *The problems* $\textsc{Reach}^{\sim \mathsf{b}}_{\mathsf{opt}}$ *are decidable for dynamic probabilistic timed networks, and are non-primitive recursive.*

Proof sketch. As an example, we explain how decidability is obtained for $\textsc{Reach}^{>0}_{\max}$. As explained above, by Proposition 11, we can consider the same decision problem in the region MDP. It thus suffices to show the decidability of whether there exists $N_0 \in \mathbb{N}$ and a scheduler $\mathcal{U}$ such that $\mathbb{P}_{\mathcal{U}}(\mathcal{R}([\![\mathcal{P}_{\Lambda}^{N_0}]\!]) \models \Diamond q_f) > 0$. We have the series of equivalences:

$$\begin{aligned} \exists N_0,\ \exists \mathcal{U}\ \mathbb{P}_{\mathcal{U}}(\mathcal{R}([\![\mathcal{P}_{\Lambda}^{N_0}]\!]) \models \Diamond q_f) > 0 \quad &\iff \quad \exists \mathcal{U}\ \mathbb{P}_{\mathcal{U}}(\mathcal{R}([\![\mathcal{P}_{\Lambda}^{0}]\!]) \models \Diamond q_f) > 0 \\ &\iff \quad [\emptyset] \in \mathsf{Pre}^*(q_f) \ . \end{aligned}$$

The first equivalence uses Remark 4.2, and more specifically that the qualitative reachability does not depend on N_0 because of the finite attractor property. Now, recall that q_f represents the set where some process is in state q_f, and is thus upward-closed. Since the Pre operator preserves upward-closure and is effective, and because $\preceq$ is a well-quasi-ordering, the set $\mathsf{Pre}^*(q_f)$ can be computed effectively by successive iterations of Pre. It then suffices to test whether the region-configuration $[\emptyset]$ belongs to $\mathsf{Pre}^*(q_f)$ to decide $\textsc{Reach}^{>0}_{\max}$.

For the lower-bound, we perform a reduction from the reachability problem in lossy channel systems, which is known to be non-primitive recursive. ◀

5 Conclusion

We studied qualitative parameterized verification problems for a model of network of many identical probabilistic timed processes. Interesting qualitative questions turn out to be undecidable in the static case, and become decidable under the assumption that processes can be created and disappear. The complexity of the decision algorithms for parameterized reachability questions in dynamic probabilistic timed networks is theoretically high, but it would be worth implementing our decision algorithms into a prototype to demonstrate whether they can be used in practice on academic case studies.

Interesting research directions for future work, are to consider distributed schedulers, where the choice of each process is independent of the state of the other processes, and to tackle quantitative verification problems. For the latter, adapting existing techniques for Markov chains to Markov decision processes could allow one to approximate maximum expected time to reachability, or to compute optimal values of the parameter.

References

1 P. A. Abdulla, N. Ben Henda, and R. Mayr. Decisive Markov chains. *Logical Methods in Computer Science*, 3(4), 2007.

2 P. A. Abdulla, N. Bertrand, A. Rabinovich, and Ph. Schnoebelen. Verification of probabilistic systems with faulty communication. *Information and Computation*, 202(2):141–165, 2005.

3 P. A. Abdulla, G. Delzanno, O. Rezine, A. Sangnier, and R. Traverso. On the verification of timed ad hoc networks. In *Proc. 9th Int. Conference on Formal Modeling and Analysis of Timed Systems (ForMATS'11)*, volume 6919 of *LNCS*, pages 256–270. Springer, 2011.

4 P. A. Abdulla, J. Deneux, and P. Mahata. Multi-clock timed networks. In *Proc. 19th IEEE Symposium on Logic in Computer Science (LICS'04)*, pp. 345–354. IEEE CS, 2004.

5 P. A. Abdulla and B. Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–263, 2003.

6 R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

7 K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22:307–309, 1986.

8 C. Baier, N. Bertrand, and Ph. Schnoebelen. A note on the attractor-property of infinite-state Markov chains. *Information Processing Letters*, 97(2):58–63, 2006.

9 C. Baier, N. Bertrand, and Ph. Schnoebelen. Verifying nondeterministic probabilistic channel systems against ω-regular linear-time properties. *ACM Transactions on Computational Logic*, 9(1), 2007.

10 C. Baier and J.-P. Katoen. *Principles of model checking.* MIT Press, 2008.

11 N. Bertrand and Ph. Schnoebelen. Computable fixpoints in well-structured symbolic model checking. *Formal Methods in System Design*, 2013. To appear.

12 E. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. In *Proc. 6th Int. Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *LNCS*, pages 395–407. Springer, 1995.

13 C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *Proc. of 1st International Colloquium on Theoretical Aspects of Computing (ICTAC'04)*, volume 3407 of *Lecture Notes in Computer Science*, pages 280–294. Springer, 2004.

14 G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *Proc. 21th Int. Conference on Concurrency Theory (CONCUR'10)*, volume 6269 of *LNCS*, pages 313–327. Springer, 2010.

15 J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. of 14th Annual IEEE Symposium on Logic in Computer Science (LICS'99)*, pages 352–359. IEEE Computer Society, 1999.

16 M. Fruth. Probabilistic model checking of contention resolution in the IEEE 802.15.4 low-rate wireless personal area network protocol. In *Proc. 2nd Int. Symposium on Leveraging Applications of Formal Methods (IsoLA'06)*, pages 290–297. IEEE, 2006.

17 F. Gretz, J.-P. Katoen, and A. McIver. Operational versus weakest precondition semantics for the probabilistic guarded command language. In *Proc. 9th Int. Conference on Quantitative Evaluation of SysTems (QEST'12)*, 2012.

18 M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. *Modeling and Verification of Real-Time Systems: Formalisms and Software Tools*, chapter Verification of Real-Time Probabilistic Systems, pages 249–288. John Wiley & Sons, 2008.

19 M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall International, 1967.

Simulation Over One-counter Nets is PSPACE-Complete*

Piotr Hofman[1], Sławomir Lasota[1], Richard Mayr[2], and Patrick Totzke[2]

1 **University of Warsaw, Poland**
2 **University of Edinburgh, UK**

Abstract

One-counter nets (OCN) are Petri nets with exactly one unbounded place. They are equivalent to a subclass of one-counter automata with just a weak test for zero. Unlike many other semantic equivalences, strong and weak simulation preorder are decidable for OCN, but the computational complexity was an open problem. We show that both strong and weak simulation preorder on OCN are PSPACE-complete.

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

Keywords and phrases Simulation preorder, one-counter nets, complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.515

1 Introduction

The model. One-counter automata (OCA) are Minsky counter automata with only one counter, and they can also be seen as a subclass of pushdown automata with just one stack symbol (plus a bottom symbol). One-counter nets (OCN) are Petri nets with exactly one unbounded place, and they correspond to a subclass of OCA where the counter cannot be fully tested for zero, because transitions enabled at counter value zero are also enabled at nonzero values. OCN are arguably the simplest model of discrete infinite-state systems, except for those that do not have a global finite control.

Previous results on semantic equivalence checking. Notions of behavioral semantic equivalences have been classified in Van Glabbeek's linear time - branching time spectrum [3]. The most common ones are, in order from finer to coarser, bisimulation, simulation and trace equivalence. Each of these have their standard (called strong) variant, and a weak variant that abstracts from arbitrarily long sequences of internal actions.

For OCA/OCN, strong bisimulation is PSPACE-complete [2], while weak bisimulation is undecidable [9]. Strong trace inclusion is undecidable for OCA [11], and even for OCN [5], and this trivially carries over to weak trace inclusion.

The picture is more complicated for simulation preorders. While strong and weak simulation are undecidable for OCA [8], they are decidable for OCN. Decidability of strong simulation on OCN was first proven in [1], by establishing that the simulation relation follows a certain regular pattern. This idea was made more graphically explicit in later proofs [7, 6],

* This work was partially supported by Polish NCN grant 2012/05/NST6/03226 and Polish MNiSW grant N N206 567840.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 515–526

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

which established the so-called *Belt Theorem*, that states that the simulation preorder relation on OCN can be described by finitely many partitionings of the grid $\mathbb{N} \times \mathbb{N}$, each induced by two parallel lines. In particular, this implies that the simulation relation is semilinear. However, the proofs in [1, 7, 6] did not yield any upper complexity bounds, since the first was based on two semi-decision procedures and the later proof of the Belt Theorem was non-constructive. A PSPACE lower bound for strong simulation on OCN follows from [10].

Decidability of weak simulation on OCN was shown in [5], using a converging series of semilinear approximants. This proof used the decidability of strong simulation on OCN as an oracle, and thus did not immediately yield any upper complexity bound.

Our contribution. We provide a new constructive proof of the Belt Theorem and derive a PSPACE algorithm for checking strong simulation preorder on OCN. Together with the lower bound from [10], this shows PSPACE-completeness of the problem.

Via a technical adaption of the algorithm for weak simulation in [5], and the new PSPACE algorithm for strong simulation, we also obtain a PSPACE algorithm for weak simulation preorder on OCN. Thus even weak simulation preorder on OCN is PSPACE-complete.

	simulation	bisimulation	weak sim.	weak bis.	trace inclusion
OCN	**PSPACE**	PSPACE [2]	**PSPACE**	undecidable [9]	undecidable [11]
OCA	undecidable [8]	PSPACE [2]	undecidable [8]	undecidable [9]	undecidable [5]

2 Problem Statement

A labelled transition system (LTS) over a finite alphabet A of actions consists of a set of configurations and, for every action $a \in A$, a binary relation $\xrightarrow{a}$ between configurations.

Given two LTS S and S', a relation R between the configurations of S and S' is a *simulation* if for every pair of configurations $(c, c') \in R$ and every step $c \xrightarrow{a} d$ there exists a step $c' \xrightarrow{a} d'$ such that $(d, d') \in R$. Simulations are closed under union, so there exists a unique maximal simulation. If $S = S'$ then this maximal simulation is a preorder, called *simulation preorder*, and denoted by $\preccurlyeq$. If $c \preccurlyeq c'$ then one says that c' *simulates* c.

Simulation preorder can also be characterized by a *Simulation Game* as follows. The *positions* are all pairs (c, c') of configurations of S and S' respectively. The game is played by two players called *Spoiler* and *Duplicator* and proceeds in rounds. In every round, starting in a position (c, c'), Spoiler chooses some $a \in A$ and some configuration d with $c \xrightarrow{a} d$. Then Duplicator responds by choosing a configuration d' with $c' \xrightarrow{a} d'$, and the next round continues from position (d, d'). If one of the players cannot move then the other player wins, and Duplicator wins every infinite play. It is well known that the Simulation Game is determined: for every initial position (c, c'), exactly one of players has a winning strategy. Configuration c' simulates c iff Duplicator has a strategy to win the Simulation Game from position (c, c').

▶ **Definition 1** (One-Counter Nets). A *one-counter net* (OCN) is a triple $\mathcal{N} = (Q, A, \delta)$ given by finite sets of control-states Q, action labels A and transitions $\delta \subseteq Q \times A \times \{-1, 0, 1\} \times Q$. It induces an infinite-state labelled transition system over the state set $Q \times \mathbb{N}$, whose elements will be written as pm, where $pm \xrightarrow{a} qn$ iff $(p, a, d, q) \in \delta$ and $n = m + d \geq 0$.

We study the computational complexity of the following decision problem.

Simulation Checking for OCN

INPUT:	Two OCN $\mathcal{N}$ and $\mathcal{N}'$ together with configurations qn and $q'n'$ of $\mathcal{N}$ and $\mathcal{N}'$ respectively, where n and n' are given in binary.
QUESTION:	$qn \preccurlyeq q'n'$?

▶ **Theorem 2.** *The Simulation Checking Problem for OCN is in PSPACE.*

Combined with the PSPACE-hardness result of [10], this yields PSPACE-completeness of the problem.

▶ Remark. Our construction can also be used to compute the simulation relation as a semilinear set, but its description requires exponential space. However, checking a point instance $qn \preccurlyeq q'n'$ of the simulation problem can be done in polynomial space by stepwise guessing and verifying only a polynomialy bounded part of the relation; cf. Section 5.

Without restriction (see [1] for a justification) we assume that both OCN are *normalised*:

1. In Spoiler's net $\mathcal{N}$, every control-state has some outgoing transition with a non-negative change of counter value.
2. Duplicator's net $\mathcal{N}'$ is *complete*, i.e., every control-state has an outgoing transition for every action (though the change in counter value may be negative).

Thus Spoiler cannot get stuck and only loses the game if it is infinite. Moreover, Duplicator can only be stuck (and lose the game) when his counter equals zero.

Outline of the proof. One easily observes that the Simulation Game is monotone for both players. If Duplicator wins the Simulation Game from a position $(qn, q'n')$ then he also wins from $(qn, q'm)$ for $m > n'$. Similarly, if Spoiler wins from $(qn, q'n')$ then she also wins from $(qm, q'n')$ for $m > n$. For a fixed pair (q, q') of control-states, both players winning regions therefore split the grid $\mathbb{N} \times \mathbb{N}$ into two connected subsets. It is known [7, 6] that the *frontier* between these subsets is contained in a *belt*, i.e., it lays between two parallel lines with rational slope.

For the proof of our main result we analyse a symbolic *Slope Game*. This new game is similar to the Simulation Game but necessarily ends after a small number of rounds. We show that given sufficiently high excess of counter-values, both players can re-use winning strategies for the Slope Game also in the Simulation Game. As a by-product of this characterization, we obtain polynomial bounds on widths and slopes of the belts. Once the belt-coefficients are known, one can compute the frontiers exactly because every frontier necessarily adheres to a regular pattern.

3 Polynomially Bounded Belts

Let us fix two OCN $\mathcal{N}$ and $\mathcal{N}'$, with sets of control-states Q and Q', respectively. Following [6], we interpret $\preccurlyeq$ as 2-colouring of $K = |Q \times Q'|$ Euclidean planes, one for each pair of control-states $(q, q') \in Q \times Q'$.

The main combinatorial insight of [6] (this was also present in [1], albeit less explicitly) is the so-called *Belt Theorem*, that states that each such plane can be cut into segments by two parallel lines such that the colouring of $\preccurlyeq$ in the outer two segments is constant; see Figure 1. We provide a new constructive proof of this theorem, stated as Theorem 4 below, that allows us to derive polynomial bounds on the coefficients of all belts.

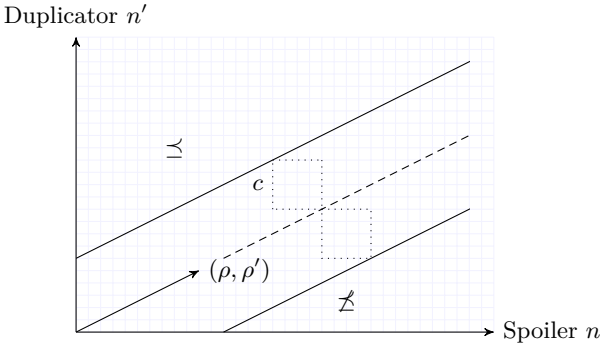


Figure 1 A belt with slope $\frac{\rho}{\rho'}$. The dashed half-line is the direction of (ρ, ρ').

▶ **Definition 3** (Positive vectors, direction, c-above, c-below). A vector $(\rho, \rho') \in \mathbb{Z} \times \mathbb{Z}$ of integers is called *positive* if $(\rho, \rho') \in \mathbb{N} \times \mathbb{N}$ and $(\rho, \rho') \neq (0, 0)$. Its *direction* is the half-line $\mathbb{R}^+ \cdot (\rho, \rho')$. For a positive vector (ρ, ρ') and a number $c \in \mathbb{N}$ we say that the point $(n, n') \in \mathbb{Z} \times \mathbb{Z}$ is *c-above* (ρ, ρ') iff there exists some point $(r, r') \in \mathbb{R}^+ \cdot (\rho, \rho')$ in the direction of (ρ, ρ') such that

$$n < r - c \qquad \text{and} \qquad n' > r' + c. \tag{1}$$

Symmetrically, (n, n') is *c-below* (ρ, ρ') if is a point $(r, r') \in \mathbb{R}^+ \cdot (\rho, \rho')$ with

$$n > r + c \qquad \text{and} \qquad n' < r' - c. \tag{2}$$

▶ **Theorem 4** (Belt Theorem). *For every two one-counter nets $\mathcal{N}$ and $\mathcal{N}'$ with sets of control-states Q and Q' respectively, there is a bound $c \in \mathbb{N}$ such that for every pair $(q, q') \in Q \times Q'$ of control-states there is a positive vector (ρ, ρ') such that*
1. *if (n, n') is c-above (ρ, ρ') then $qn \preccurlyeq q'n'$, and*
2. *if (n, n') is c-below (ρ, ρ') then $qn \not\preccurlyeq q'n'$.*
Moreover, c and all ρ, ρ' are bounded polynomially w.r.t. the sizes of $\mathcal{N}$ and $\mathcal{N}'$.

4 Proof of the Belt Theorem

We consider OCN $\mathcal{N}$ and $\mathcal{N}'$ with sets of control-states Q and Q', resp., and define the constant $K = |Q \times Q'|$. Abdulla and Cerans [1] showed that, above a certain level, the simulation relation has a regular structure. An important parameter for this structure is the *ratio* n/n' of the respective counter values n in Spoiler's configuration qn of $\mathcal{N}$ and n' in Duplicator's configuration $q'n'$ of $\mathcal{N}'$.

We further develop this intuition by defining a new finitary game (called the Slope Game; cf. Section 4.1) that is played directly on the control graphs of the nets, and in which the objective of the players is to minimize (resp. maximize) the ratio of the effects of recently observed minimal cycles. Then we show how to transform winning strategies in the Slope Game into winning strategies in the original simulation game. First we need to define some properties of vectors.

▶ **Definition 5** (Behind, Steeper). Let (ρ, ρ') be a positive and $(\alpha, \alpha') \in \mathbb{Z}^2$ an arbitrary vector. We place the two on the plane with a common starting point and consider the clockwise oriented angle from (ρ, ρ') to (α, α'). We say that (α, α') is *behind* (ρ, ρ') if the oriented angle is strictly between $0°$ and $180°$. See Figure 2 for an illustration.

Positive vectors may be naturally ordered: We will call (ρ, ρ') *steeper* than (α, α'), written $(\alpha, \alpha') \prec (\rho, \rho')$, if (α, α') is behind (ρ, ρ').

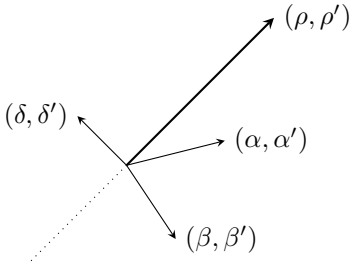


Figure 2 Vectors (α, α') and (β, β') are behind (ρ, ρ'), but (δ, δ') is not. Also, $(\alpha, \alpha') \prec (\rho, \rho')$.

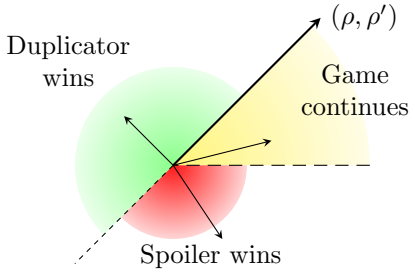


Figure 3 Evaluating the winning condition in position $(\pi, (\rho, \rho'))$ after a phase of the Slope Game.

Note that the property of one vector being behind another only depends on their directions. The following simple lemma will be useful in the sequel.

▶ **Lemma 6.** *Let (ρ, ρ') be a positive vector and $c, n, n' \in \mathbb{N}$.*

1. *If (n, n') is c-below (ρ, ρ') then $(n, n') + (\alpha, \alpha')$ is c-below (ρ, ρ') for any vector (α, α') which is behind (ρ, ρ').*
2. *If (n, n') is c-above (ρ, ρ') then $(n, n') + (\alpha, \alpha')$ is c-above (ρ, ρ') for any vector (α, α') which is not behind (ρ, ρ').*

4.1 Slope Game

▶ **Definition 7** (Product Control Graph, Lasso, Effect of a path)**.** Given two OCN $\mathcal{N} = (Q, A, \delta)$ and $\mathcal{N}' = (Q', A, \delta')$, their *product control graph* is the finite, edge-labelled graph with nodes $Q \times Q'$ and $(A \times \mathbb{N} \times \mathbb{N})$-labelled edges E given by

$$(p, p') \xrightarrow{a,d,d'} (q, q') \in E \text{ iff } p \xrightarrow{a,d} q \in \delta \text{ and } p' \xrightarrow{a,d'} q' \in \delta'. \tag{3}$$

A *path*

$$\pi = (q_0, q_0') \xrightarrow{a_0,d_0,d_0'} (q_1, q_1') \xrightarrow{a_1,d_1,d_1'} \dots \xrightarrow{a_{k-1},d_{k-1},d_{k-1}'} (q_k, q_k') \tag{4}$$

from (q_0, q_0') to (q_k, q_k') in this graph is called *lasso* if it contains a cycle while none of its strict prefixes does. That is, if there exist $i < k$ such that $(q_k, q_k') = (q_i, q_i')$ and for all $0 \leq i < j < k$, $(q_i, q_i') \neq (q_j, q_j')$. The lasso π splits into $\textsc{prefix}(\pi) = (q_0, q_0') \xrightarrow{a_0,d_0,d_0'} \dots \xrightarrow{a_{i-1},d_{i-1},d_{i-1}'} (q_i, q_i')$ and $\textsc{cycle}(\pi) = (q_i, q_i') \xrightarrow{a_i,d_i,d_i'} \dots \xrightarrow{a_{k-1},d_{k-1},d_{k-1}'} (q_k, q_k')$. The *effect* of a path is the cumulative sum of the effects of its transitions:

$$\Delta(\pi) = \sum_{i=0}^{k-1} (d_i, d_i') \in \mathbb{Z} \times \mathbb{Z}. \tag{5}$$

The effects of cycles will play a central role in our further construction. The intuition is that if a play of a Simulation Game describes a lasso then the players "agree" on the chosen cycle. Repeating this cycle will change the ratio of the counter values towards its effect.

To formalize this intuition, we define a finitary Slope Game which proceeds in phases. In each phase, the players alternatingly move on the control graphs of their original nets, ignoring the counter, and thereby determine the next lasso that occurs. After such a phase,

a winning condition is evaluated that compares the effect of the chosen lasso's cycle with that of previous phases. Now either one player immediately wins or the next phase starts, but then the steepness of the observed effect must have strictly decreased. The number of different effects of simple cycles thus bounds the maximal length of a game.

▶ **Definition 8** (Slope Game)**.** A *Slope Game* is a strictly alternating two player game played on a pair $\mathcal{N}, \mathcal{N}'$ of one-counter nets. The game positions are pairs $(\pi, (\rho, \rho'))$, where π is an acyclic path in the product control graph of $\mathcal{N}$ and $\mathcal{N}'$, and (ρ, ρ') is a positive vector which we call *slope*.

The game is divided into *phases*, each starting with a path $\pi = (q_0, q_0')$ of length 0. Until a phase ends, the game proceeds in rounds like a Simulation Game, but the players pick transition rules instead of transitions: in a position $(\pi, (\rho, \rho'))$ where π ends in states (q, q'), Spoiler chooses a transition rule $q \xrightarrow{a,d} p$, then Duplicator responds with a transition rule $q' \xrightarrow{a,d} p'$. If the extended path $\pi' = \pi \xrightarrow{a,d,d'} (p, p')$ is still not a lasso, the next round continues from the updated position $(\pi', (\rho, \rho'))$; otherwise the phase ends with *outcome* $(\pi', (\rho, \rho'))$. The slope (ρ, ρ') does not restrict the possible moves of either player, nor changes during a phase. We thus speak of *the slope of a phase*.

If a round ends in position $(\pi, (\rho, \rho'))$ where π is a lasso, then the winning condition is evaluated. We distinguish three non-intersecting cases depending on how the effect $\Delta(\text{CYCLE}(\pi)) = (\alpha, \alpha')$ of the lasso's cycle relates to (ρ, ρ'):

1. If (α, α') is not behind (ρ, ρ'), Duplicator wins immediately.
2. If (α, α') is behind (ρ, ρ') but not positive, Spoiler wins immediately.
3. If (α, α') is behind (ρ, ρ') and positive, the game continues with a new phase from position $(\pi', (\alpha, \alpha'))$, where π' is the path of length 0 consisting of the pair of ending states of π.

Figure 3 illustrates the winning condition. Note that if there is no immediate winner it is guaranteed that (α, α') is a positive vector.

The fundamental intuition for the connection between the Slope Game and the Simulation Game is as follows. The Slope Game from initial position $((q, q'), (\rho, \rho'))$ determines how the initial slope (ρ, ρ') relates to the belt in the plane for (q, q') in the simulation relation. Roughly speaking, if (ρ, ρ') is less steep than the belt then Spoiler wins; if (ρ, ρ') is steeper then Duplicator wins. Finally, when the initial slope (ρ, ρ') is exactly as steep as the belt, any player may win the Slope Game.

Consider a Simulation Game in which the ratio n/n' of the counter values of Spoiler and Duplicator is the same as the ratio ρ/ρ', i.e. suppose (n, n') is contained in the direction of (ρ, ρ'). Suppose also that the values (n, n') are sufficiently large. By monotonicity, we know that the steeper the slope (ρ, ρ'), the better for Duplicator. Hence if the effect (α, α') of some cycle is behind (ρ, ρ') and positive, then it is beneficial for Spoiler to repeat this cycle. With more and more repetitions, the ratio of the counter values will get arbitrarily close to (α, α'). On the other hand, if (α, α') is behind (ρ, ρ') but not positive then Spoiler wins by repeating the cycle until the Duplicator's counter decreases to 0. Finally, if the effect of the cycle is not behind (ρ, ρ') then repeating this cycle leads to Duplicator's win.

The next lemma follows from the observation that in Slope Games, the slope of a phase must be strictly less steep than those of all previous phases.

▶ **Lemma 9.** *For a fixed pair $\mathcal{N}, \mathcal{N}'$ of OCN,*

1. *any Slope Game ends after at most $(\mathrm{K}+1)^2$ phases, and*
2. *Slope Games are effectively solvable in PSPACE.*

Proof. After every phase, the slope (ρ, ρ') is equal to the effect of a simple cycle, which must be a positive vector. Thus the absolute values of both numbers ρ and ρ' are bounded by $\mathrm{K} = |Q \times Q'|$. It follows that the total number of different possible values for (ρ, ρ'), and therefore the maximal number of phases played, is at most $(\mathrm{K}+1)^2$. This proves the first part of the claim. Point 2 is a direct consequence as one can find and verify winning strategies by an exhaustive search. ◀

Strategies in Slope Games. Consider one phase of a Slope Game, starting from a position $(\pi, (\rho, \rho'))$. The phase ends with a lasso whose cycle effect (α, α') satisfies exactly one of three conditions, as examined by the evaluating function. Accordingly, depending on its initial position, every phase falls into exactly one of three disjoint cases:

1. Spoiler has a strategy to win the Slope Game immediately,
2. Duplicator has a strategy to win the Slope Game immediately or
3. neither Spoiler nor Duplicator have a strategy to win immediately.

In case 1. or 2. we call the phase *final*, and in case 3. we call it *non-final*. The non-final phases are the most interesting ones because in those, both players have a strategy that at least prevents an immediate loss.

Strategy Trees. Both in final and non-final phases, a strategy for Spoiler or Duplicator is a tree as described below. For the definition of strategy trees we need to consider, not only Spoiler's positions $(\pi, (\rho, \rho'))$ but also Duplicator's positions, the intermediate positions within a single round. These intermediate positions may be modelled as triples $(\pi, (\rho, \rho'), t)$ where t is a transition rule in $\mathcal{N}$ from the last state of π. Observe that the bipartite directed graph, with positions of a phase as vertices and edges determined by the single-move relation, is actually a tree, call it T. Thus a Spoiler-strategy, i.e. a subgraph of T containing exactly one successor of every Spoiler's position and all successors of every Duplicator's position, is a tree as well; and so is any strategy for Duplicator.

Such a strategy (tree) in the Slope Game naturally splits into *segments*, each segment being a strategy (tree) in one phase. The segments themselves are also arranged into a tree, which we call *segment tree*. Irrespectively which player wins a Slope Game, according to the above observations, this player's winning strategy contains segments of two kinds:

- non-leaf segments are strategies to either win immediately or continue the Slope Game (these are strategies for non-final phases);
- leaf segments are strategies to win the Slope Game immediately (these are strategies in final phases).

By the *segment depth* of a strategy we mean the depth of its segment tree. By Lemma 9, Point 1, we know that a Slope Game ends after at most $d_{\max} = (\mathrm{K}+1)^2$ phases. Consequently, the segment depths of strategies are at most $d_{\max}$ as well.

A value of $c = \mathrm{K} \cdot d_{\max}$ is sufficient for the claim of Theorem 4. The intuition behind this value is that for a winning player in the Slope Game, an excess of K per phase is sufficient to be able to safely "replay" a winning strategy in the Simulation Game. Formally, this is stated by the following two crucial lemmas, proofs of which can be found in [4], Appendix A.

▶ **Lemma 10.** *Suppose Spoiler has a winning strategy of segment depth d in the Slope Game from a position $((q, q'), (\rho, \rho'))$. Then Spoiler wins the Simulation Game from every position $(qn, q'n')$ which is $(\mathrm{K} \cdot d)$-below (ρ, ρ').*

▶ **Lemma 11.** *Suppose Duplicator has a winning strategy of segment depth d in the Slope Game from a position $((q,q'),(\rho,\rho'))$. Then Duplicator wins the Simulation Game from every position $(qn,q'n')$ which is $(\mathrm{K}\cdot d)$-above (ρ,ρ').*

4.2 Proof of Theorem 4

Let $c = \mathrm{K}\cdot d_{max}$. For any two states $q \in Q$ and $q' \in Q'$ of the nets $\mathcal{N}$ and $\mathcal{N}'$ we will determine the ratio (ρ,ρ') that, together with c, characterises the belt of the plane (q,q'). First observe the following monotonicity property of the Slope Game.

▶ **Lemma 12.** *If Spoiler wins the Slope Game from a position $((q,q'),(\rho,\rho'))$ and (σ,σ') is less steep than (ρ,ρ') then Spoiler also wins the Slope Game from $((q,q'),(\sigma,\sigma'))$.*

Proof. Assume that Spoiler wins the Slope Game from $((q,q'),(\rho,\rho'))$ while Duplicator wins from $((q,q'),(\sigma,\sigma'))$, for some $(\sigma,\sigma') \prec (\rho,\rho')$. Observe that in both cases, winning strategies of segment depth $\leq d_{\max}$ exist. As (σ,σ') is less steep than (ρ,ρ'), there is a point $(n,n') \in \mathbb{N}\times\mathbb{N}$ which is both c-above (σ,σ') and c-below (ρ,ρ'). Applying both Lemma 10 and 11 immediately yields a contradiction. ◀

Equivalently, if Duplicator wins the Slope Game from $((q,q'),(\rho,\rho'))$ and (σ,σ') is steeper than (ρ,ρ') then Duplicator also wins the Slope Game from $((q,q'),(\sigma,\sigma'))$. We conclude that for every pair (q,q') of states, there is a *boundary slope* (β,β') such that

1. Spoiler wins the Slope Game from $((q,q'),(\sigma,\sigma'))$ for every (σ,σ') less steep than (β,β');
2. Duplicator wins the Slope Game from $((q,q'),(\sigma,\sigma'))$ for every (σ,σ') steeper than (β,β').

Note that we claim nothing about the winner from the position $((q,q'),(\beta,\beta'))$ itself. Applying Lemmas 10 and 11 we see that this boundary slope (β,β') satisfies the claims 1 and 2 of Theorem 4. Indeed, consider a pair $(n,n') \in \mathbb{N}\times\mathbb{N}$ of counter values. If (n,n') is c-below (β,β'), then there is certainly a line $(\bar\beta,\bar\beta')$ less steep than (β,β') such that (n,n') is c-below $(\bar\beta,\bar\beta')$. By point 1 above, Spoiler wins the Slope Game from $((q,q'),(\bar\beta,\bar\beta'))$. By Lemma 10, Spoiler wins the Simulation Game from $(qn,q'n')$. Analogously, one can use point 2 above together with Lemma 11 to show Point 2 of Theorem 4.

It remains to show that the boundary slope (β,β') is polynomial in the sizes of $\mathcal{N}$ and $\mathcal{N}'$. We show that (β,β') must in fact be the effect of a simple cycle. Because such cycles are no longer than $K = |Q\times Q'|$ and because along a path of length K the counter values cannot change by more than K, we conclude that $-K \leq \beta,\beta' \leq K$.

▶ **Definition 13** (Equivalent vectors)**.** Consider all the non-zero effects (α,α') of all cycles together with their opposite vectors $(-\alpha,-\alpha')$ and denote the set of all these vectors by V. Call two positive vectors (ρ,ρ') and (σ,σ') *equivalent* if for all $(\alpha,\alpha') \in V$,

$$(\alpha,\alpha') \text{ is behind } (\rho,\rho') \iff (\alpha,\alpha') \text{ is behind } (\sigma,\sigma'). \tag{6}$$

In other words, equivalent vectors lie in the same angle determined by a pair of vectors from V that are neighbours angle-wise. We claim that equivalent slopes have the same winner in the Slope Game:

▶ **Lemma 14.** *If (ρ,ρ') and (σ,σ') are equivalent then the same player wins the Slope Game from $((q,q'),(\rho,\rho'))$ and $((q,q'),(\sigma,\sigma'))$.*

Proof. A winning strategy in the Slope Game from $((q,q'),(\rho,\rho'))$ may be literally used in the Slope Game from $((q,q'),(\sigma,\sigma'))$. This holds because the assumption that (ρ,ρ') and (σ,σ') are equivalent implies that all possible outcomes of the initial phase of the Slope Game are evaluated equally. ◀

Lemma 14 implies that the boundary slope is in V. This concludes the proof of Theorem 4. ◀

4.3 A Sharper Estimation

Theorem 4 provides a polynomial bound on the constant c and the slopes of all belts, with respect to the sizes of $\mathcal{N}$ and $\mathcal{N}'$. However, the proof of Theorem 4 reveals that a slightly stronger result actually holds, which will be useful in proving the complexity bound for weak simulation in Section 6. We can estimate a bound on c in terms of the following two parameters of the product control graph $\mathcal{N} \times \mathcal{N}'$:

- SCC, the size of the largest strongly connected component, and
- ACYC, the length of the longest acyclic path.

In particular, we claim that Theorem 4 still holds with the constant c bounded by

$$c \leq poly(\text{SCC}) + \text{ACYC}. \tag{7}$$

Intuitively, c is the excess of counter value needed to replay a Slope Game strategy in the Simulation Game. This directly corresponds to the maximal number of alternations in a play of the Slope Game. Every phase ends in a cycle, which must be contained in some strongly connected component and is thus no longer than SCC. So the segment depth of Slope Game strategies is bounded by $(\text{SCC}+1)^2$.

We can decompose plays of the Slope Game by separating subpaths that contain at least one cycle and stay in one strongly connected component, and the remaining subpaths. One can now show that in fact, a counter value of SCC suffices to enable subpaths of the first kind. The segment depth bounds the number of such subpaths in any play. Secondly, by definition, the subpaths of the second kind cannot share any points. The sum of their lengths is hence bounded by ACYC. We conclude that a value of $c = (\text{SCC}+1)^2 \cdot \text{SCC} + \text{ACYC}$ is sufficient.

5 Strong Simulation is PSPACE-complete

Using our stronger version of the Belt Theorem from Section 4, we derive an algorithm for checking simulation preorder, similarly as in [1, 7, 6].

As before we fix two OCN $\mathcal{N}$ and $\mathcal{N}'$, with sets of control-states Q and Q', respectively. By Lemma 9, Point 2, we can compute in PSPACE, for every pair $(q,q') \in Q \times Q'$, the positive vector (ρ,ρ') satisfying Theorem 4; we denote this vector by $\text{SLOPE}(q,q')$. We define $\text{BELT}(q,q')$ to be the set of points $(n,n') \in \mathbb{N}^2$ that are neither c-above nor c-below $\text{SLOPE}(q,q')$. As all vectors $\text{SLOPE}(q,q')$ and the widths of all belts are polynomially bounded (by Theorem 4), we observe that every two non-parallel belts are disjoint outside a polynomially bounded *initial rectangle*, denoted L_0, between corners $(0,0)$ and (l_0,l_0') (see Figure 4).

Recall that the simulation preorder on the configurations with the pair of control-states (q,q') is trivial outside of $\text{BELT}(q,q')$: it contains all pairs $(qn,q'n')$ s.t. (n,n') is c-above $\text{SLOPE}(q,q')$, and contains no pairs $(qn,q'n')$ s.t. (n,n') is c-below $\text{SLOPE}(q,q')$. We show that inside a belt, the points corresponding to configurations in simulation are ultimately periodic in the sense defined below.

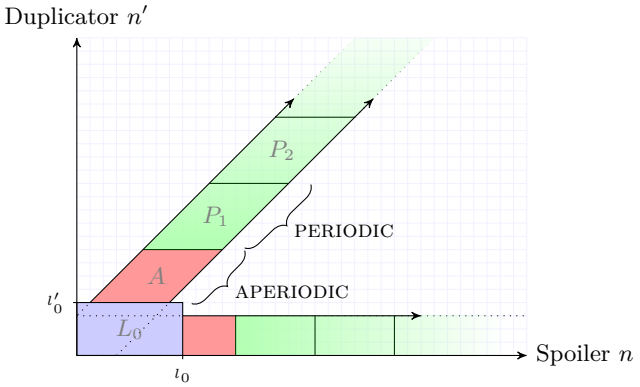


Figure 4 The initial rectangle L_0 (blue) and two belts. Outside L_0, the colouring of a belt consists of some exponentially bounded block (red), and another exponentially bounded non-trivial block (green) which repeats ad infinitum along the rest of the belt.

By the definition of belts, $(n,n') \in \textsc{belt}(q,q') \iff (n,n') + \textsc{slope}(q,q') \in \textsc{belt}(q,q')$, i.e., translation via the vector $\textsc{slope}(q,q')$ preserves membership in $\textsc{belt}(q,q')$. This is why we restrict our focus to multiples of vectors $\textsc{slope}(q,q')$. We write $\textsc{rect}(q,q',j)$ for the rectangle between corners $(0,0)$ and $(l_0,l'_0) + j \cdot \textsc{slope}(q,q')$.

▶ **Definition 15** (ultimately-periodic)**.** For a fixed pair $(q,q') \in Q \times Q'$ and $j,k \in \mathbb{N}$, a subset $R \subseteq \textsc{belt}(q,q')$ is called (j,k)-*ultimately-periodic* if for all $(n,n') \in \mathbb{N}^2 \setminus \textsc{rect}(q,q',j)$,

$$(n,n') \in R \iff (n,n') + k \cdot \textsc{slope}(q,q') \in R. \tag{8}$$

▶ Remark. Observe that for fixed q and q', every (j,k)-ultimately-periodic set R can be represented by the numbers j and k, and two sets

$$R \cap \textsc{rect}(q,q',j) \qquad \text{and} \qquad (R \setminus \textsc{rect}(q,q',j)) \cap \textsc{rect}(q,q',j+k).$$

The following lemma states a property which is crucial for our algorithm. It is actually a sharpening of the result of [6], with additional effective bounds on periods inside belts.

▶ **Lemma 16.** *For every pair $(q,q') \in Q \times Q'$, the set*

$$\preccurlyeq_{q,q'} \;=\; \{(n,n') \in \textsc{belt}(q,q') : qn \preccurlyeq q'n'\}$$

is (j,k)-ultimately periodic for some $j,k \in \mathbb{N}$ exponentially bounded w.r.t. the sizes of $\mathcal{N}$, $\mathcal{N}'$.

Thus, when searching for a simulation relation inside belts, we may safely restrict ourselves to (j,k)-ultimately-periodic relations, for exponentially bounded j and k. According to the remark above, every such simulation admits the EXPSPACE description that consists, for every pair of states (q,q'), of:

- a polynomially bounded vector $(\rho,\rho') = \textsc{slope}(q,q')$;
- a polynomially bounded relation $\textsc{init}(q,q') \subseteq L_0$ inside the initial rectangle L_0;
- exponentially bounded natural numbers $j_{q,q'}, k_{q,q'} \in \mathbb{N}$; and
- two exponentially bounded relations:

$$\begin{aligned} \textsc{aperiodic}(q,q') &\subseteq \textsc{belt}(q,q') \cap \textsc{rect}(q,q',j_{q,q'}) \\ \textsc{periodic}(q,q') &\subseteq (\textsc{belt}(q,q') \setminus \textsc{rect}(q,q',j_{q,q'})) \cap \textsc{rect}(q,q',j_{q,q'}+k_{q,q'}). \end{aligned}$$

The above characterization leads to the following naive decision procedure, which works in EXPSPACE: Guess the description of a candidate relation R for the simulation relation, verify that it is a simulation and check if it contains the input pair of configurations.

Checking whether the input pair is in the (semilinear) relation R is trivial. To verify that the relation R is a simulation, one needs to check the *simulation condition* for every pair of configurations $(qn, q'n')$ in R, i.e., Duplicator can ensure that after playing one round of the Simulation Game, the resulting pair of configurations is still in R.

The simulation condition is local in the sense that it refers only to positions with neighbouring counter values (plus/minus 1). This, together with the fact that belts are disjoint outside L_0, implies that the complete one-neighbourhoods of points in the periodic part repeats along the belt. It therefore suffices to examine those elements which are in the EXPSPACE description to check if the simulation condition holds.

A PSPACE procedure. The naive algorithm outlined above may easily be turned into a PSPACE algorithm by a standard shifting window trick. Instead of guessing the complete exponential-size description upfront, we start by guessing the polynomially bounded relation inside L_0 and verifying it locally. Next, the procedure stepwise guesses parts of the relations $\textsc{aperiodic}(q, q')$ and later $\textsc{periodic}(q, q')$, inside a polynomially bounded rectangle window through the belt and shifts this window along the belt, checking the simulation condition for all contained points on the way. Since the simulation condition is local, everything outside this window may be forgotten, save for the first repetitive window that is used as a certificate for successfully having guessed a consistent periodic set, once it repeats. Because this repetition needs to occur after an exponentially bounded number of shifts, polynomial space is sufficient to store a binary counter that counts the number of shifts and allows to terminate unsuccessfully once the limit is reached. ◀

6 Application to Weak Simulation Checking

A natural extension of simulation is *weak simulation*, that abstracts from internal steps.

▶ **Definition 17.** For a LTS over actions $A \cup \{\tau\}$ define *weak* step relations by $\stackrel{\tau}{\Longrightarrow} = \stackrel{\tau}{\longrightarrow}^*$ and $\stackrel{a}{\Longrightarrow} = \stackrel{\tau}{\longrightarrow}^* \stackrel{a}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^*$ for $a \neq \tau$. Weak simulation ($\preccurlyeq$) is now defined just like $\preccurlyeq$, using Simulation Games, in which Duplicator moves along weak steps.

For systems without τ-labelled transitions, $\stackrel{a}{\longrightarrow} = \stackrel{a}{\Longrightarrow}$ and therefore strong and weak simulation coincide. The PSPACE lower bound from [10] for checking strong simulation thus also holds for weak simulation checking over OCN.

Weak simulation has recently been shown to be decidable for OCN [5]. The main obstacle was that Duplicator's system is infinitely branching w.r.t. the weak $\stackrel{a}{\Longrightarrow}$ steps, which implies that non-simulation does not necessarily manifest itself locally.

In [5], this problem is resolved by constructing a monotone decreasing sequence of semilinear *approximant relations* that converges to weak simulation at a finite index. The approximant relations are derived from a symbolic characterization of Duplicator's infinitely-branching system. They can be computed inductively by characterizing them in terms of strong simulation over suitably modified OCN. The fact that one can effectively compute semilinear descriptions of $\preccurlyeq$ over OCN [6] allows to successively compute the approximant relations and to detect convergence of the sequence.

Here we show that the polynomial bounds from Theorem 4, together with the technique from [5], imply a PSPACE upper bound even for checking *weak* simulation on OCN. In

particular, we claim that the sizes of the "suitably modified OCN" mentioned above, which characterize the approximants, are in fact polynomial for every index $i \in \mathbb{N}$ in the sequence. A more detailed analysis can be found in [4], Appendix B.

▶ **Theorem 18.** *Checking weak simulation preorder on OCN is PSPACE-complete.*

7 Conclusion

We have shown that both strong and weak simulation preorder checking between two given OCN processes is PSPACE-complete. Moreover, it is possible to compute representations of the entire simulation relations as semilinear sets, but these require exponential space. One cannot expect polynomial-size representations of the relations as semilinear sets, because otherwise one could first guess the representation and then verify in $coNP^{NP}$ (for strong simulation) that there are no counterexamples to the local simulation condition. This would yield an algorithm in Σ_p^3 in the polynomial hierarchy, which (under standard assumptions in complexity theory) contradicts the PSPACE-hardness of the problem.

References

1 P.A. Abdulla and K. Cerans. Simulation is decidable for one-counter nets (extended abstract). In *CONCUR*, volume 1466 of *LNCS*, pages 253–268, 1998.

2 S. Böhm, S. Göller, and P. Jančar. Bisimilarity of one-counter processes is PSPACE-complete. In *CONCUR*, volume 6269 of *LNCS*, 2010.

3 R.J. van Glabbeek. The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001.

4 P. Hofman, S. Lasota, R. Mayr, and P. Totzke. Simulation over one-counter nets is PSPACE-complete. Technical Report EDI-INF-RR-1418, University of Edinburgh, 2013.

5 P. Hofman, R. Mayr, and P. Totzke. Decidability of weak simulation on one-counter nets. In *Proc. of LICS 2013*. IEEE, 2013.

6 P. Jančar, A. Kučera, and F. Moller. Simulation and bisimulation over one-counter processes. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *LNCS*, pages 334–345, London, UK, 2000. Springer-Verlag.

7 P. Jančar and F. Moller. Simulation of one-counter nets via colouring. Technical report, Uppsala Computing Science Research Report 159, February 1999.

8 P. Jančar, F. Moller, and Z. Sawa. Simulation problems for one-counter machines. In *SOFSEM*, volume 1725 of *LNCS*, pages 404–413, 1999.

9 R. Mayr. Undecidability of weak bisimulation equivalence for 1-counter processes. In *ICALP*, volume 2719 of *LNCS*, pages 570–583, 2003.

10 J. Srba. Beyond language equivalence on visibly pushdown automata. *Logical Methods in Computer Science*, 5(1):1–22, 2009.

11 L.G. Valiant. *Decision procedures for families of deterministic pushdown automata.* PhD thesis, Department of Computer Science, University of Warwick, Coventry, July 1973.

Optimal Constructions for Active Diagnosis *

Stefan Haar[1], Serge Haddad[1], Tarek Melliti[2], and Stefan Schwoon[1]

1 **LSV (CNRS & ENS Cachan) & INRIA, France**
2 **IBISC (Université d'Evry Val-d'Essonne), France**

Abstract

The task of *diagnosis* consists in detecting, without ambiguity, occurrence of faults in a partially observed system. Depending on the degree of observability, a discrete event system may be *diagnosable* or not. *Active* diagnosis aims at controlling the system in order to make it diagnosable. Solutions have already been proposed for the active diagnosis problem, but their complexity remains to be improved. We solve here the active diagnosability decision problem and the active diagnoser synthesis problem, proving that (1) our procedures are optimal w.r.t. to computational complexity, and (2) the memory required for the active diagnoser produced by the synthesis is minimal. We then focus on the delay between the occurrence of a fault and its detection by the diagnoser. We construct a memory-optimal diagnoser whose delay is at most twice the minimal delay, whereas the memory required for a diagnoser with optimal delay may be highly greater.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Diagnosis, Control theory, Automata theory, Games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.527

1 Introduction

In monitoring discrete event systems, one of the central tasks is that of *diagnosis*: Given a finite labeled transition system $\mathcal{A}$ (also called "plant") whose events are partially observable, our task is to decide – based on the stream of observation labels – whether or not particular unobservable events, called *faults*, have occurred. More precisely, the system is considered k-diagnosable iff at most k events after the occurrence of a fault, the observation is sufficient to detect that occurrence with certainty, i.e. all possible system runs compatible with the partial observation collected so far are faulty. The system $\mathcal{A}$ is *diagnosable* iff there exists $k \geq 1$ such that $\mathcal{A}$ is k-diagnosable. As the system may be insufficiently observable, or the observation not discriminating enough, *diagnosability* verification has received considerable attention since the seminal paper by Sampath et al [13]; see also [4, 3]. Those works construct a dedicated deterministic version of the original plant, a so-called *diagnoser*; the absence of indeterminate cycles in this auxiliary automaton is equivalent to diagnosability.

On the other hand, once a system has been shown to be undiagnosable – in a sense that we will formalize later – several actions can follow, such as complete redesign of the system, or adding further sensors to enhance observability. Sampath et al [12] have initiated a different approach, that of *active diagnosis*: if the given plant $\mathcal{A}$ is not diagnosable, synthesize a partial-observation controller $\mathcal{C}$ that forces $\mathcal{A}$ to stay within a diagnosable subset of its behaviors (or, equivalently, such that the controlled plant $\mathcal{A}_C$ is diagnosable). The pair consisting of the controller and the diagnoser is called an *active diagnoser*. Later, Chanthery and Pencolé [5] have proposed a planning-based approach via a twin plant construction.

* This work has been supported by project ImpRo ANR-2010-BLAN-0317 and the European Union Seventh Framework Programme [FP7/2007-2013] under grant agreement 257462 HYCON2 NOE.

Our contributions. We follow the approach of Sampath et al [12], but via a different method based on automata and game theory. This allows us to improve the construction of diagnosers and moreover establish complexity results, which were not treated in previous work:

1. We build a deterministic Büchi automaton that accepts the sublanguage of infinite *unambiguous* observable sequences, i.e. those that are either (i) triggered by a set of correct runs or (ii) triggered by a set of faulty runs. Its size is upper-bounded by $2^{\mathcal{O}(n)}$, where n is the number of states, which is better than all previous constructions. In addition we show the optimality of our construction proving that there is a family of systems for which any corresponding deterministic Büchi automaton must have a size in $2^{\Omega(n)}$.
2. We then design a Büchi game, where a winning strategy yields an active diagnoser for the system, and vice versa. We thus solve the active diagnosis problem by deciding whether there exists a winning strategy, and the synthesis problem by giving an active diagnoser associated with a positional strategy. The size of the active diagnoser is singly exponential w.r.t. the size of the system, while that of [12] is doubly exponential. We also prove that the decision procedure is EXPTIME-complete and that the synthesis procedure is optimal w.r.t. the number of states of the active diagnoser (still in $2^{\mathcal{O}(n)}$).
3. We then study the delay between a fault and its detection by an active diagnoser. We first present a family of systems for which a minimal-delay diagnoser must have $2^{\Omega(n \log(n))}$ states. However, refining our earlier construction yields an active diagnoser with size $2^{\mathcal{O}(n)}$, whose delay is at most twice the minimal possible delay. In addition, we sketch the construction of a minimal-delay active diagnoser with at most $2^{\mathcal{O}(n^2)}$ states.

Organization. Section 2 recalls notions related to diagnosis and active diagnosis. In Section 3, we establish the lower bounds related to the computational complexity, the memory requirements and the index. Section 4.1 presents the construction of the deterministic Büchi automaton. Then in Section 4.2, we solve the decision and the synthesis problems for active diagnosis. After that, Section 4.3 refines the synthesis problem w.r.t. the delay. Section 5 gives some perspectives of this work. A long version with all proofs is available [7].

2 The active diagnosis problem

Labeled transition systems

When dealing with discrete event systems (DES) diagnosis, systems are often modeled using labeled transition systems (LTS). So we define LTS, their properties and languages.

▶ **Definition 1.** A labeled transition system is a tuple $\mathcal{A} = \langle Q, q_0, \Sigma, T \rangle$ where:

- Q is a set of states with $q_0 \in Q$ the initial state;
- Σ is a finite set of events;
- $T \subseteq Q \times \Sigma \times Q$ is the set of transitions.

We note $q \xrightarrow{a} q'$ for $(q, a, q') \in T$; this transition is then said to be *enabled* in q. A *run* over the word $\sigma = a_1 a_2 \ldots \in \Sigma^\omega$ is a sequence of states $(q_i)_{i \geq 0}$ such that $q_i \xrightarrow{a_{i+1}} q_{i+1}$ for all $i \geq 0$, and we write $q_0 \overset{\sigma}{\Rightarrow}$ if such a run exists. A finite run over $w \in \Sigma^*$ is defined analogously, and we write $q \overset{w}{\Rightarrow} q'$ if such a run ends at state q'. A state q is *reachable* if there exists a run $q_0 \overset{w}{\Rightarrow} q$ for some w.

▶ **Definition 2** (Languages of an LTS). Let $\mathcal{A} = \langle Q, q_0, \Sigma, T\rangle$ be an LTS. The finite language $\mathcal{L}^*(\mathcal{A}) \subseteq \Sigma^*$ of $\mathcal{A}$ and the infinite language $\mathcal{L}^\omega(\mathcal{A}) \subseteq \Sigma^\omega$ of $\mathcal{A}$ are defined by:

$$\mathcal{L}^*(\mathcal{A}) = \{\, w \in \Sigma^* \mid \exists q : q_0 \stackrel{w}{\Rightarrow} q \,\} \qquad \mathcal{L}^\omega(\mathcal{A}) = \{\, \sigma \in \Sigma^\omega \mid q_0 \stackrel{\sigma}{\Rightarrow} \,\}$$

An LTS $\mathcal{A}$ is *live* if for any reachable state there exists a transition enabled in that state. An LTS $\mathcal{A}$ is *deterministic* if for every pair $q \in Q, a \in \Sigma$ there is at most one q' such that $q \xrightarrow{a} q'$. For a deterministic automaton we write $T(q,a) = q'$ if $q \xrightarrow{a} q'$.

Observations

In order to formalize problems related to diagnosis, we partition Σ into two disjoint sets Σ_o and Σ_{uo}, the sets of *observable* and of *unobservable events*, respectively. Moreover, we distinguish a special *fault* event $f \in \Sigma_{uo}$. Let σ be a finite word; its length is denoted $|\sigma|$. For $\Sigma' \subseteq \Sigma$, define $\mathcal{P}_{\Sigma'}(\sigma)$ inductively by: $\mathcal{P}_{\Sigma'}(\varepsilon) = \varepsilon$; for $a \in \Sigma'$, $\mathcal{P}_{\Sigma'}(\sigma a) = \mathcal{P}_{\Sigma'}(\sigma)a$; and $\mathcal{P}_{\Sigma'}(\sigma a) = \mathcal{P}_{\Sigma'}(\sigma)$ for $a \notin \Sigma'$. Write $|\sigma|_{\Sigma'}$ for $|\mathcal{P}_{\Sigma'}(\sigma)|$, and for $a \in \Sigma$, write $|\sigma|_a$ for $|\sigma|_{\{a\}}$. When σ is an infinite word, its projection is the limit of the projections of its finite prefixes. This projection can be either finite or infinite. As usual the projection is extended to languages. $\mathcal{P}_{\Sigma_o}$ will be more simply denoted by $\mathcal{P}$.

An LTS $\mathcal{A}$ is *convergent* if $\mathcal{L}^\omega(\mathcal{A}) \cap \Sigma^*\Sigma_{uo}^\omega = \emptyset$ (i.e. no infinite sequence of unobservable events from any reachable state). When $\mathcal{A}$ is convergent, then for all $\sigma \in \mathcal{L}^\omega(\mathcal{A})$, one has $\mathcal{P}(\sigma) \in \Sigma_o^\omega$. We shall assume that the system under diagnosis is live and convergent.

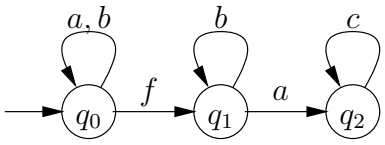


Figure 1 An LTS.

▶ **Example 3.** Figure 1 shows a live and convergent LTS with $\Sigma_o = \{a, b, c\}$ and $\Sigma_{uo} = \{f\}$.

Diagnosability

A finite (resp. infinite) sequence σ is *correct* if it belongs to $(\Sigma \setminus \{f\})^*$ (resp. $(\Sigma \setminus \{f\})^\omega$). Otherwise σ is called *faulty*. An observation sequence may be the projection of both a correct and a faulty sequence, hence ambiguous.

▶ **Definition 4** (ambiguous and surely faulty sequence). Let $\mathcal{A}$ be an LTS, $\sigma_1, \sigma_2 \in \mathcal{L}^\omega(\mathcal{A})$ be two sequences and $\sigma \in \Sigma_o^\omega$ be an observable sequence such that:

(1) $\mathcal{P}(\sigma_1) = \mathcal{P}(\sigma_2) = \sigma$, (2) σ_1 is correct and (3) σ_2 is faulty.

Then σ is called *ambiguous* and the pair (σ_1, σ_2) is a *witness* for the ambiguity of σ. Ambiguous finite observable sequences are defined analogously.
A sequence $\sigma' \in \mathcal{P}(\mathcal{L}^*(\mathcal{A}))$ is *surely faulty* iff $\mathcal{P}^{-1}(\sigma') \cap \mathcal{L}^*(\mathcal{A}) \subseteq \Sigma^* f \Sigma^*$.

▶ **Definition 5** (Diagnosability). Let $k \in \mathbb{N}$. An LTS $\mathcal{A}$ is k-diagnosable if:

$$\forall \sigma = \sigma' f \sigma'' \in \mathcal{L}^*(\mathcal{A})\ |\sigma''|_{\Sigma_o} \geq k \Rightarrow \mathcal{P}(\sigma) \text{ is a surely faulty sequence,}$$

Furthermore, $\mathcal{A}$ is *diagnosable* if there exists a k such that $\mathcal{A}$ is k-diagnosable.

Our definition of diagnosability is a slight variation of the one given in [13]. Indeed the number k above is related to observable events while in former works, it is related to any kind of events. However for finite-state convergent systems (which are the ones addressed by both works) the definitions of diagnosability coincide.

▶ **Example 6.** The LTS of Figure 1 is not diagnosable since the correct infinite trace b^ω and the faulty infinite trace fb^ω have the same projection.

Active diagnosability

We suppose that Σ_o is partitioned into subsets $\Sigma_c \subseteq \Sigma_o$ of *controllable* and $\Sigma_{uc} = \Sigma_o \backslash \Sigma_c$ of *uncontrollable* actions. Intuitively, a controller may forbid a subset of the controllable actions based on the observations made so far, thereby restricting the behaviour of $\mathcal{A}$.

▶ **Definition 7** (Controller)**.** Let $\mathcal{A}$ be an LTS. A *controller* for $\mathcal{A}$ is a mapping $cont : \mathcal{P}(\mathcal{L}^*(\mathcal{A})) \to 2^\Sigma$ such that for all σ, $\Sigma_{uc} \cup \Sigma_{uo} \subseteq cont(\sigma)$. The controlled LTS $\mathcal{A}_{cont} = \langle Q_{cont}, q_{0cont}, \Sigma, T_{cont} \rangle$ is defined by:

- Q_{cont} is the smallest subset of $\Sigma_o^* \times Q$ such that
  1. $(\varepsilon, q_0) \in Q_{cont}$;
  2. $(\sigma, q) \in Q_{cont} \wedge a \in cont(\sigma) \wedge q \xrightarrow{a} q'$ implies $(\mathcal{P}(\sigma a), q') \in Q_{cont}$.
- $q_{0cont} = (\varepsilon, q_0)$
- $((\sigma, q), a, (\sigma', q')) \in T_{cont}$ iff $q \xrightarrow{a} q' \wedge a \in cont(\sigma) \wedge \sigma' = \mathcal{P}(\sigma a)$

In the diagnosis framework, the goal of our controller is to make the system diagnosable, and to perfom diagnosis. However, one requires that the control cannot introduce deadlocks.

▶ **Definition 8** (Pilot and Active Diagnoser)**.** Let $\mathcal{A}$ be an LTS. We call $h = \langle cont, diag \rangle$ a *pilot* for $\mathcal{A}$ if *cont* is a controller and *diag* is a mapping from $\mathcal{P}(\mathcal{L}^*(\mathcal{A}_{cont}))$ to $\{\bot, \top\}$. Moreover, h is called an *active diagnoser* if:

1. $\mathcal{A}_{cont}$ is live;
2. $\mathcal{P}(\mathcal{L}^\omega(\mathcal{A}_{cont}))$ does not contain any ambiguous sequence;
3. $diag(\sigma) = \top$ if and only if σ is a surely faulty sequence for $\sigma \in \mathcal{P}(\mathcal{L}^*(\mathcal{A}_{cont}))$.

For $k \geq 1$, we say that h is a *k-active diagnoser*, if for all $\sigma = \sigma' f \sigma'' \in \mathcal{L}^*(\mathcal{A}_{cont})$ with $|\sigma''|_{\Sigma_o} \geq k$, $diag(\mathcal{P}(\sigma)) = \top$, i.e. every fault is diagnosed after at most k observations. The minimal k such that h is a k-active diagnoser is called the *delay* of h. We call $\mathcal{A}$ *(k-)actively diagnosable* if a (k-)active diagnoser exists, and the minimal such k the *index* of $\mathcal{A}$.

▶ **Example 9.** In the LTS of Figure 1, assume that $\Sigma_c = \{a, b\}$. Let $h_n = \langle cont_n, diag \rangle$, with $n \geq 1$, be the pilot defined by:

- $cont_n(\sigma b^n) = \{a, c, f\}$ for $\sigma \in \Sigma_c^*$ and $cont_n(\sigma) = \Sigma$ otherwise;
- $diag(\sigma) = \top$ iff $\sigma \in \Sigma_o^* c \Sigma_o^*$.

Then h_n is an active diagnoser with delay $n + 2$.

Notice that an active diagnoser does not necessarily have a finite delay. For instance, in Figure 1, there is an active diagnoser that admits the sequence $bab^2ab^3a\cdots$ and is not an k-active diagnoser for any k. However, we will see that if $\mathcal{A}$ is actively diagnosable, there does exist a k-active diagnoser (for some k). We come back to this point in Section 4.3.

We are now in a position to formally state the relevant problems for active diagnosis. Let $\mathcal{A}$ be a live and convergent LTS with finitely many states. We are interested in:

- the *active diagnosis decision problem*, i.e. decide whether $\mathcal{A}$ is actively diagnosable;
- the *synthesis problem*, i.e. decide whether $\mathcal{A}$ is actively diagnosable and in the positive case build an active diagnoser.
- the *minimal-delay synthesis problem*, i.e. decide whether $\mathcal{A}$ is actively diagnosable and in the positive case build an active diagnoser with minimal delay.

We introduce the notion of *state-based pilot* as finite representation of an active diagnoser.

▶ **Definition 10** (state-based pilot)**.** A *state-based pilot* $\mathcal{C} = \langle \mathcal{B}, cont_\mathcal{C}, diag_\mathcal{C} \rangle$ consists of a deterministic LTS $\mathcal{B} = \langle Q^c, q_0^c, \Sigma_o, T^c \rangle$ and labellings $cont_\mathcal{C}, diag_\mathcal{C} : Q^c \to 2^\Sigma \times \{\bot, \top\}$, such that for all $q \in Q^c$, $\Sigma_{uc} \cup \Sigma_{uo} \subseteq cont_\mathcal{C}(q)$. The pilot $h_\mathcal{C} = \langle cont, diag \rangle$ associated with $\mathcal{C}$ is

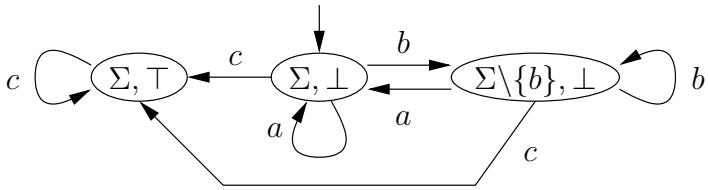


Figure 2 A state-based pilot.

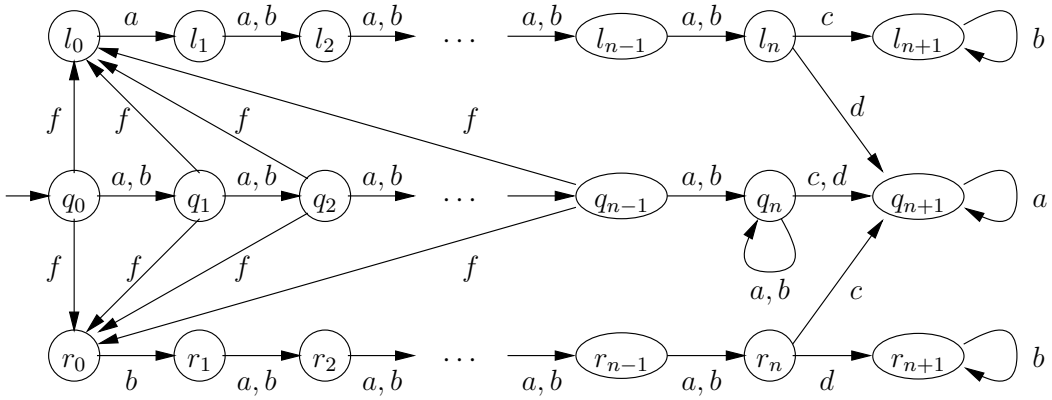


Figure 3 An LTS $\mathcal{A}_n$ with $\Sigma_o = \{a, b, c, d\}$, $\Sigma_c = \{c, d\}$ used in Theorem 14.

given by $cont(\sigma) = cont_{\mathcal{C}}(q)$ and $diag(\sigma) = diag_{\mathcal{C}}(q)$ for all $\sigma \in \mathcal{P}(\mathcal{L}^*(\mathcal{A}))$, where q is the unique state such that $q_0^c \stackrel{\sigma}{\Rightarrow} q$.

▶ **Example 11.** Figure 2 shows a state-based pilot for the LTS of Figure 1. Observe that there is an outgoing transition b from the rightmost state (to fulfil the language inclusion requirement) but b is disabled in this state (in order to implement the active diagnoser h_1).

3 Lower bounds

We first establish that the active diagnosis decision problem is EXPTIME-hard. The proof [7] relies on a reduction from safety games with imperfect information [1].

▶ **Theorem 12** (hardness). *The active diagnosis decision problem is EXPTIME-hard.*

The next theorems focus on the memory required for synthesis problems related to active diagnosis. We start with the language of unambiguous sequences of an LTS.

▶ **Definition 13** (Büchi automaton). A *Büchi automaton* over Σ is a tuple $\mathcal{B} = \langle \mathcal{B}', F \rangle$, where $\mathcal{B}' = \langle S, s_0, \Sigma, \delta \rangle$ is an LTS such that S is finite, and $F \subseteq S$ an *acceptance condition*. A *run* $(q_i)_{i \geq 0}$ is *accepting* if $q_i \in F$ for infinitely many values of i. The language $\mathcal{L}(\mathcal{B})$ consists of all words in $\mathcal{L}^\omega(\mathcal{B}')$ for which there exists an accepting run. A Büchi automaton is called deterministic (live) if its underlying LTS is.

▶ **Theorem 14** (lower bound for determinization). *There exists a family $(\mathcal{A}_n)_{n \geq 1}$ of LTS with the size of $\mathcal{A}_n$ in $\mathcal{O}(n)$ such that any deterministic Büchi automaton recognizing the unambiguous sequences of A_n has at least 2^n states.*

The family of LTS $(\mathcal{A}_n)_{n\geq 1}$ is depicted in Figure 3. During the n first steps a fault can occur leading to the upper (resp. lower) "branch" of the LTS when followed by a (resp. b). However the corresponding observable sequence becomes definitively ambiguous if n steps later the LTS performs d (resp. c). So any deterministic automaton should lead to different states when reading two different words of length n. With an appropriate choice of controllable events, this family also provides a lower bound for a state-based active diagnoser.

▶ **Theorem 15** (lower bound for pilots). *There exists a family $(\mathcal{A}_n)_{n\geq 1}$ of actively diagnosable LTS with the size of $\mathcal{A}_n$ in $\mathcal{O}(n)$ such that the LTS of any state-based pilot $\mathcal{C}$, where $h_{\mathcal{C}}$ is an active diagnoser for $\mathcal{A}$, has at least 2^n states.*

We shall now see that the lower bound is even higher when one tries to minimize the fault-detection delay. The LTS $\mathcal{A}_n$ of Figure 4 contains the observable (and uncontrollable) observation sequence $a_{\pi(1)} \dots a_{\pi(n)}$, where π is a permutation. Such a sequence is ambiguous since a fault may have occurred before any observable event. To remove ambiguity with minimal delay (i.e. $n+2$) an active diagnoser must disallow at time $n+i$ all events in B except $b_{\pi(i)}$ that forces the potential faulty sequence to reach state s where only c is possible. Thus, any active diagnoser for $\mathcal{A}_n$ must remember the permutation π.

▶ **Theorem 16** (minimal-delay diagnoser). *There exists a family $(A_n)_{n\geq 1}$ of $f(n)$-actively diagnosable LTS (for some function f) with $\mathcal{O}(n)$ states such that the LTS of any state-based pilot $\mathcal{C}$, where $h_{\mathcal{C}}$ is an $f(n)$-active diagnoser for $\mathcal{A}$, has at least $n!$ states.*

Note that in Figure 4 the alphabet size depends on n; however Theorem 16 also holds for a fixed-size alphabet [7]. While the previous examples exhibit an index linear w.r.t. the size of the LTS, this index may be exponential in the worst case (and no more as shown in the Section 4). An example for this is shown in [7].

▶ **Theorem 17** (lower bound for index). *There exists a family $(A_n)_{n\geq 1}$ of actively diagnosable LTS with $\mathcal{O}(n)$ states such that the index of $\mathcal{A}_n$ is at least 2^n.*

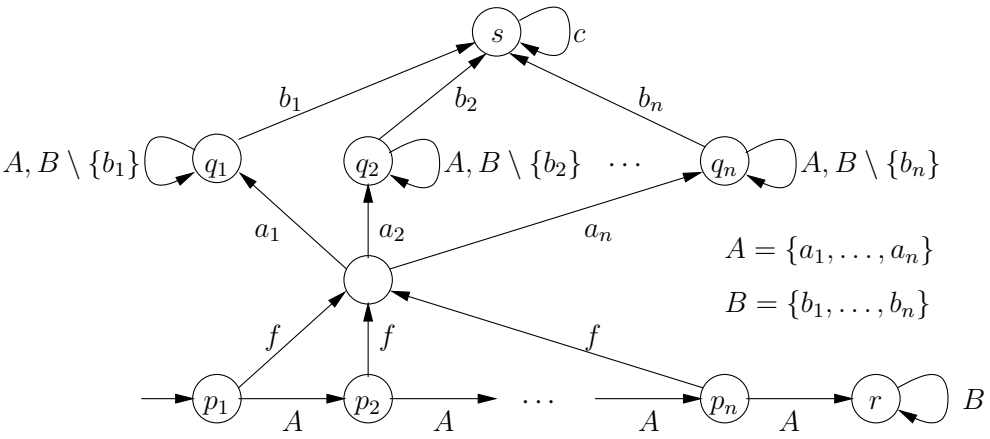


Figure 4 An LTS $\mathcal{A}_n$ with $\Sigma_o = A \cup B \cup \{c\}$, $\Sigma_c = B$ whose minimal-delay active diagnoser requires at least $\mathcal{O}(n!)$ states.

4 Size-Optimal Controller

4.1 Characterization of unambiguous sequences

In this section, we characterize the infinite unambiguous sequences in an efficient way. Fix a finite-state live, convergent LTS $\mathcal{A} = \langle Q, q_0, \Sigma, T\rangle$ for the rest of the section. We build a Büchi automaton $\mathcal{B} = (\mathcal{B}', F)$ that accepts the unambiguous observation sequences. Since $\mathcal{B}$ is the base of the active diagnoser constructed in Section 4.2, we want $\mathcal{B}$ to be deterministic.

A potential procedure for obtaining a deterministic automaton accepting unambiguous sequences is as follows: First, build a non-deterministic Büchi automaton that accepts observable sequences explainable by both a correct and a faulty sequence. This leads to a quadratic blow up w.r.t. the size of $\mathcal{A}$. Then, determinize it by the Safra procedure [11], yielding a deterministic Rabin automaton, and complement it so it accepts the unambiguous sequences. However, we now provide the construction of a simpler and smaller deterministic Büchi automaton. More precisely, the automaton that we build has the following properties:

- $\mathcal{B}'$ is deterministic;
- $\mathcal{B}'$ "reads" the observable sequences of $\mathcal{A}$, i.e. $\mathcal{L}^\omega(\mathcal{B}') = \mathcal{P}(\mathcal{L}^\omega(\mathcal{A}))$;
- $\mathcal{B}$ accepts exactly the unambiguous observation sequences.

We first give some intuition about the way $\mathcal{B}$ works. Its states are triples $\langle U, V, W\rangle$, where $U, V, W \subseteq Q$. The states in U represent states reachable by non-faulty traces in $\mathcal{A}$, whereas $V \cup W$ are states reachable by committing a fault. Let $\sigma = a_1 a_2 \ldots \in \Sigma_o^\omega$ be an observation sequence. An ambiguous prefix of σ will lead to a state in which both U and $V \cup W$ are non-empty, and if σ is ambiguous, then its run will eventually remain in such states forever. Unfortunately, the reverse implication is not true, as the example from Figure 1 shows: every finite prefix of the sequence a^ω is ambiguous, but a^ω is not. In order to distinguish ambiguous sequences from those that merely have infinitely many ambiguous prefixes, V and W assume different functions: W represents a "watchlist", initially empty. Suppose that the observation $a_1 \ldots a_j$, for some j, corresponds to some faulty execution. Then we put the state reachable by that faulty execution into W and trace its successor states there while making further observations. If W never becomes empty, then indeed there exists a faulty element of $\mathcal{P}(\sigma)$ in $\mathcal{L}^\omega(\mathcal{A})$. On the other hand, if some observation $a_{j'}$, for $j' > j$, is impossible in all states of W, then we can conclude that no fault has occurred before a_j. In the meantime, V serves as a "waiting room": it stores states that can be reached by faulty sequences where the fault has occurred between observations a_j and $a_{j'}$. When W becomes empty, those states are shifted from V to W to form the new watchlist.

Let $S' \subseteq S$, $a \in \Sigma_o$, and $\mathcal{L} \subseteq \Sigma_{uo}^*$ be a language of unobservable actions. We denote $\delta_{\mathcal{L}}(S', a) := \{\, q \in Q \mid \exists q' \in S', w \in \mathcal{L} : q' \xRightarrow{wa} q \,\}$, and introduce the abbreviations

- δ_n for $\mathcal{L} = (\Sigma_{uo} \setminus \{f\})^*$ (non-faulty executions),
- δ_f for $\mathcal{L} = \Sigma_{uo}^* f \Sigma_{uo}^*$ (faulty executions),
- and δ_* for $\mathcal{L} = \Sigma_{uo}^*$ (arbitrary executions).

We can now state the formal construction of $\mathcal{B} = \langle\langle S, s_0, \Sigma_o, \delta\rangle, F\rangle$ as follows:

- $S = 2^Q \times 2^Q \times 2^Q \setminus \{\langle\emptyset, \emptyset, \emptyset\rangle\}$ and $s_0 = \langle\{q_0\}, \emptyset, \emptyset\rangle$;
- $F = \{\, \langle\emptyset, S_1, S_2\rangle, \langle S_1, S_2, \emptyset\rangle \mid S_1, S_2 \subseteq Q \,\}$;
- for $s = \langle U, V, W\rangle \in S$ and $a \in \Sigma_o$ such that $\delta_*(U \cup V \cup W, a) \neq \emptyset$, let $\Delta := \delta_f(U, a) \cup \delta_*(V, a)$; then

$$\delta(s, a) = \begin{cases} \langle \delta_n(U, a), \emptyset, \Delta\rangle & \text{if } W = \emptyset; \\ \langle \delta_n(U, a), \Delta \setminus \delta_*(W, a), \delta_*(W, a)\rangle & \text{otherwise.} \end{cases}$$

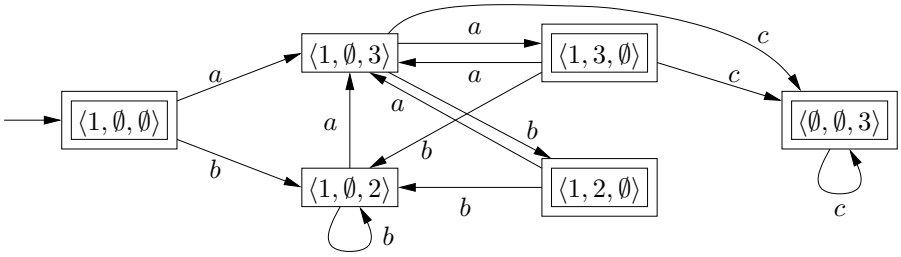


Figure 5 Büchi automaton resulting from Figure 1; accepting states have double frames.

Observe that disregarding the acceptance condition, the sequences read by $\mathcal{B}$ exactly correspond to observable sequences of $\mathcal{A}$, i.e. $\mathcal{P}(\mathcal{L}^\omega(\mathcal{A}))$.

▶ **Theorem 18.** *A sequence of observations $\sigma \in \Sigma_o^\omega$ is accepted by $\mathcal{B}$ iff it is unambiguous.*

▶ **Example 19.** Figure 5 shows the result of the construction on the system from Figure 1. Since all non-empty sets are singletons we have represented them by their item. Notice that any sequence ending in b^ω is ambiguous in Figure 1 and hence not accepted in Figure 5. On the other hand, e.g., sequence a^ω is accepted: while every prefix a^i, for $i \geq 1$, is ambiguous, we always know after $i+1$ observation that no fault has occurred before the i-th observation.

We briefly discuss the relationship of our determinization construction with other standard constructions in diagnosis and automata theory. In [16], diagnosability of an LTS $\mathcal{A}$ is decided by building two automata: one is a modification of $\mathcal{A}$ that accepts the projections all non-faulty sequences, the other accepts the projections of all faulty sequences, remembering whether a fault has occurred in the current state. The cross product of these two is a non-deterministic Büchi automaton of size $2n^2$ (for $|Q| = n$) that accepts all ambiguous sequences. A direct determinization [11] of that cross product would yield a Rabin automaton of size $2^{\mathcal{O}(n^2 \log n)}$. However, given that the cross product is *weak* in the sense that all its strongly connected components are either fully accepting or fully non-accepting, one could apply the *breakpoint construction* of Miyano and Hayashi [9] to obtain a deterministic Büchi automaton of its complement language, of size 3^{2n^2}. Our construction, while similar in spirit to that of [9], is more efficient than that: for a reachable Büchi state $\langle U, V, W\rangle \in S$, any LTS state $q \in Q$ may or may not appear in U, and it may appear in at most one of V or W, but not in both. Thus, the number of reachable states in $\mathcal{B}$ is bounded by $2^n \cdot 3^n = 6^n = 2^{\mathcal{O}(n)}$. Theorem 14 shows that an exponential blowup in n is unavoidable in general, i.e. our construction is optimal up to a constant factor in the exponent.

4.2 Synthesizing the controller

We simultaneously solve the decision and synthesis problems. As before, we fix an LTS $\mathcal{A} = \langle Q, q_0, \Sigma, T\rangle$. We shall try to construct a state-based pilot $\mathcal{C}$ such that $h_\mathcal{C}$ is an active diagnoser for $\mathcal{A}$. The construction succeeds iff $\mathcal{A}$ is actively diagnosable. According to Definition 8, the main challenges in building an active diagnoser are to ensure that (i) the controlled system remains live, (ii) the controller excludes the ambiguous sequences, and (iii) diagnosis information is provided. For this, we introduce Büchi games.

▶ **Definition 20** (game)**.** A game $\mathcal{G}$ (between two players called Control and Environment) is a tuple $\langle V_C, V_E, E, v_0, V_F\rangle$, where V_C, V_E are the vertices owned by Control and Environment, respectively; $V_\mathcal{G}$ denotes all vertices, and $v_0 \in V_C$ is an *initial vertex*. $E \subseteq V_\mathcal{G} \times V_\mathcal{G}$

are directed edges such that for all $v \in V_C$ there exists some $(v, w) \in E$, and $V_F \subseteq V_\mathcal{G}$ is a *winning condition*. A *play* is a function $\rho\colon \mathbb{N} \to V_\mathcal{G}$ such that $\rho(0) = v_0$ and $\langle \rho_i, \rho_{i+1} \rangle \in E$ for all $i \geq 0$; we call $\rho^k := \rho(0) \cdots \rho(k)$, for some $k \geq 0$, a *partial play* if $\rho(k) \in V_C$, and set $\mathit{state}(\rho^k) := \rho(k)$. We write $\mathit{Play}^*(\mathcal{G})$ for the set of partial plays of $\mathcal{G}$. A play ρ is called *winning* (for Control) if $\rho(i) \in V_F$ for infinitely many i.

▶ **Definition 21** (strategy). Let $\mathcal{G} = \langle V_C, V_E, E, v_0, V_F \rangle$ be a game. A *strategy* (for Control) is a function $\theta\colon \mathit{Play}^*(\mathcal{G}) \to V_\mathcal{G}$ such that $\langle \mathit{state}(\xi), \theta(\xi) \rangle \in E$ for all $\xi \in \mathit{Play}^*(\mathcal{G})$. A play ρ *adheres* to θ if $\rho(i) \in V_C$ implies $\rho(i+1) = \theta(\rho^i)$ for all $i \geq 0$. A strategy is called *winning* if every play ρ that adheres to θ is winning. A *positional strategy* is a function $\theta'\colon V_C \to V_\mathcal{G}$ such that $\langle v, \theta'(v) \rangle \in E$ for all $v \in V_C$; we call θ' *winning* if the strategy θ with $\theta(\xi) = \theta'(\mathit{state}(\xi))$ is winning.

In the game that we have defined, a play can only be stuck in a state of Environment. Thus we do not consider finite maximal plays for defining the winning strategies of Control. Let $\mathcal{B} = \langle \mathcal{B}', F \rangle$, with $\mathcal{B}' = \langle S, s_0, \Sigma_o, \delta \rangle$, be the deterministic Büchi automaton constructed from $\mathcal{A}$ in Section 4.1. We shall take $\mathcal{B}'$ as the LTS component of $\mathcal{C}$. To determine $\mathit{cont}_\mathcal{C}$, we construct a Büchi game based on $\mathcal{B}$. The objective of Control is to obtain an accepting run by suitably restricting the possible actions, and any winning strategy will be a suitable candidate for $\mathit{cont}_\mathcal{C}$. Intuitively, a round of the game is played as follows:

1. Control restricts the set of possible actions to Σ'.
2. Environment chooses an action $a \in \Sigma'$ to determine the next state of $\mathcal{B}$.

The choices of Control are subject to some restrictions. Indeed, each state $s = \langle U, V, W \rangle$ represents Control's knowledge about the current potential states of $\mathcal{A}$. To ensure that the controlled system remains live, Σ' must not cause deadlocks in any state reachable by unobservable events from $U \cup V \cup W$. Also, Control cannot prevent the uncontrollable events. So we define the admissible sets and the game as follows.

▶ **Definition 22** (admissible action set). Let $s = \langle U, V, W \rangle$ be a state of $\mathcal{B}$. We call $\Sigma' \subseteq \Sigma_o$ *admissible* for s if (i) $\Sigma_{uc} \subseteq \Sigma'$ and (ii) for all states q' of $\mathcal{A}$ with $q \overset{w}{\Rightarrow} q'$ for some $q \in U \cup V \cup W$ and $w \in \Sigma_{uo}^*$, there exists $a \in \Sigma'$ and $q'' \in Q$ with $q' \xrightarrow{a} q''$. The admissible sets for s are denoted $\mathit{adm}(s)$.

▶ **Definition 23** (controller-synthesis game). Let $\mathcal{B} = \langle \langle S, s_0, \Sigma_o, \delta \rangle, F \rangle$ be a Büchi automaton. We denote $\mathcal{G}(\mathcal{B})$ the game $\langle V_C, V_E, E, s_0, F \rangle$, where $V_C = S$, $V_E = (S \times 2^{\Sigma_o}) \cup (S \times \Sigma_o)$, and $E = E_1 \cup E_2 \cup E_3$, where

- $E_1 = \{ \langle s, \langle s, \Sigma' \rangle \rangle \mid s \in S,\ \Sigma' \in \mathit{adm}(s) \}$;
- $E_2 = \{ \langle \langle s, \Sigma' \rangle, \langle s, a \rangle \rangle \mid s \in S,\ a \in \Sigma' \}$;
- $E_3 = \{ \langle \langle s, a \rangle, s' \rangle \mid \delta(s, a) = s' \}$.

The set E_3 is only introduced to record the sequence of observable actions that occur during a play. Furthermore Environment can be stuck in a vertex of E_3 meaning that the action chosen by Environment does not correspond to a possible behavior of the system.

▶ **Example 24.** Figure 6 depicts an excerpt of the game for Example 1. In the initial state, there are three possible admissible sets, all including c, the uncontrollable observable action. $\{c\}$ is not an admissible set as it blocks the system. If Environment chooses action c, it immediately loses since c is not possible initially even after a fault.

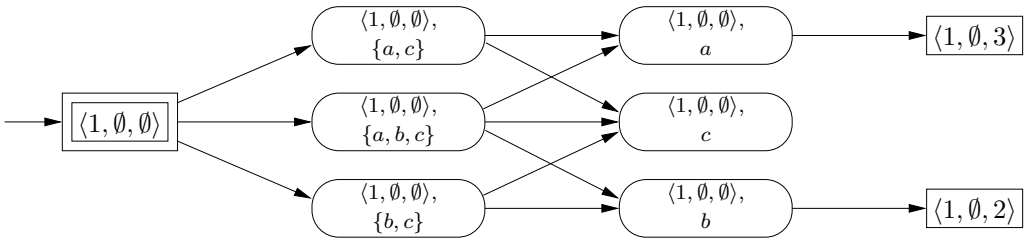


Figure 6 Excerpt of the Büchi game for Example 1.

We can now address the decision and synthesis problems. The next theorem is based on the following: (1) Büchi games can be solved in polynomial time, (2) a positional winning strategy can always be chosen for Control if it wins and (3) there is a tight correspondence between winning strategies and active diagnosers.

▶ **Theorem 25.** *Let $\mathcal{A}$ be an LTS with n states and m controllable actions. The active diagnosis decision and synthesis problems for $\mathcal{A}$ can be solved in $2^{\mathcal{O}(n+m)}$ time. Moreover, if $\mathcal{A}$ is actively diagnosable, then one can synthesize a state-based pilot $\mathcal{C}$ with at most 6^n states such that $h_{\mathcal{C}}$ is an active diagnoser for $\mathcal{A}$.*

We briefly discuss the relationship of our construction with that of [12]. There, an active diagnoser is built on the basis of a powerset construction that is similar to ours but without splitting the possibly faulty states into a 'watchlist' W and a 'waiting room' V. However, they then face the aforementioned problem of distinguishing sequences with infinitely many ambiguous prefixes (like a^ω in Example 1) from truly ambiguous sequences (like b^ω), which they resolve by examining each cycle of the automaton. Since the number of states in that automaton is 3^n,[1] and there can be exponentially many cycles, this procedure is doubly exponential in n. Our construction is only singly exponential in n.

Using Theorems 12 and 25, we get the following corollary.

▶ **Corollary 26.** *The active diagnosis decision problem is EXPTIME-complete.*

4.3 Index and waiting time

We assume that $\mathcal{A}$ is actively diagnosable and develop the construction of an active diagnoser with a delay close to the index of $\mathcal{A}$, and a computational complexity still in $2^{\mathcal{O}(n)}$. For simplicity, we denote the game $\mathcal{G}(\mathcal{B})$ by $\mathcal{G}$. Let $\mathcal{G}'$ be any game. Given a strategy θ for $\mathcal{G}'$, we denote by $Play^\omega_\theta(\mathcal{G}')$ the set of plays that adhere to strategy θ, and by $R(\theta)$ the subset of states of S that are visited by a play of $Play^\omega_\theta(\mathcal{G}')$. We are now in the position to introduce the main concept of this section, the *waiting time* of a strategy: the maximal number of states visited without encountering an accepting state.

▶ **Definition 27** (waiting time)**.** Let θ be a strategy for $\mathcal{G}$. Then the *waiting time* $K(\theta)$ is defined as $\sup(|\{k \mid i \leq k \leq j \wedge \rho(k) \in S\}| \mid \exists i, j\ \exists \rho \in Play^\omega_\theta(\mathcal{G})\ F \cap \{\rho(k)\}_{i \leq k \leq j} = \emptyset)$ with the convention $\sup(\emptyset) = 0$.

Observe that $K(\theta)$ may be infinite for a non-positional winning strategy. However, it is finite and strictly smaller than $|S|$ (since there is at least one accepting state) for a winning

[1] This is the result when only one fault type is considered; [12] actually provides for several fault types, which we omit here for sake of simplicity.

positional strategy. In fact, for θ a winning positional strategy, $K(\theta)$ can be computed in linear time (with appropriate data structures) w.r.t. the size of $\mathcal{G}$. In order to present it and for subsequent use, we introduce the following notation. Let s be a state of the Büchi automaton, $Out(s) := \{a \in \Sigma_o \mid \delta(s,a) \text{ is defined }\}$. First one computes, by increasing values, the minimal solution of the following equation system:

$$V_\theta(s) = \begin{cases} 0 & \text{if } s \in R(\theta) \cap F; \\ 1 + \max(V_\theta(\delta(s,a)) \mid a \in \Sigma' \cap Out(s) \text{ s.t. } (s,\Sigma') = \theta(s)) & \text{if } s \in R(\theta) \setminus F. \end{cases}$$

Then $K(\theta) = \max(V_\theta(s) \mid s \in R(\theta))$. Denote by $D(\theta)$ the delay of the active diagnoser related to strategy θ. Lemma 28 shows that $K(\theta)$ provides useful information about $D(\theta)$.

▶ **Lemma 28.** *Let θ be a strategy for game $\mathcal{G}$ with finite waiting time. Then:*

$$1 + K(\theta) \leq D(\theta) \leq 1 + 2K(\theta)$$

Intuitively, the upper bound is potentially due to a fault staying in the "waiting room" of $\mathcal{B}$ for at most $K(\theta)$ steps, then in the "watchlist" for at most $K(\theta)+1$ steps. The lower bound is due to the fact that along a subrun with a non-empty watchlist, a possible fault could have occurred before this subrun.

Define $K_\mathcal{A} = \min(K(\theta))$, where θ ranges over the winning strategies for $\mathcal{G}$. Since a positional such strategy exists, we know that $K_\mathcal{A}$ is finite and belongs to $2^{\mathcal{O}(n)}$. Let us note $D_\mathcal{A} = \min(D(\theta))$ the index of $\mathcal{A}$. The following corollary provides a tight frame for $D_\mathcal{A}$ and shows that the index is in $2^{\mathcal{O}(n)}$.

▶ **Corollary 29.** *Let $\mathcal{A}$ be actively diagnosable. Then:* $1 + K_\mathcal{A} \leq D_\mathcal{A} \leq 1 + 2K_\mathcal{A}$

Let us compute an active diagnoser or, equivalently, a strategy θ that achieves $K(\theta) = K_\mathcal{A}$. To this aim, we introduce a family of games $\{\mathcal{G}_i\}_{i\in\mathbb{N}}$ defined as follows. The set of vertices of $\mathcal{G}_i$ are: $V_{\mathcal{G}_i} = \{v^j \mid v \in V_\mathcal{G} \wedge 0 \leq j \leq i\} \cup \{lost\}$ where the subset of vertices owned by Control are $\{v^j \mid v \in V_C \wedge 0 \leq j \leq i\} \cup \{lost\}$, the initial vertex is s_0^0, and the set of accepting states are $\{s^0 \mid s \in F\}$. Its set of edges $E' = E'_1 \cup E'_2 \cup E'_3$ is defined by:

- for all $j \leq i$, $\langle v^j, w^j\rangle$ belongs to E'_1 iff $\langle v, w\rangle$ belongs to E_1;
- for all $j \leq i$, $\langle v^j, w^j\rangle$ belongs to E'_2 iff $\langle v, w\rangle$ belongs to E_2;
- for all $j \leq i$, $\langle \langle s,a\rangle^j, s'^0\rangle \in E'_3$ iff $\langle \langle s,a\rangle, s'\rangle \in E_3$ and $s' \in F$;
- for all $j < i$, $\langle \langle s,a\rangle^j, s'^{j+1}\rangle \in E'_3$ iff $\langle \langle s,a\rangle, s'\rangle \in E_3$ and $s' \notin F$;
- $\langle \langle s,a\rangle^i, lost\rangle$ belongs to E'_3 iff $\langle \langle s,a\rangle, s'\rangle$ belongs to E_3 and $s' \notin F$;
- $\langle lost, lost\rangle$ belongs to E'_3 and there is no other edge.

Game $\mathcal{G}_i$ has the following properties: an infinite play either ends up in *lost* or visits the accepting states infinitely often, with at most i visits of the set $\{v^j \mid v \in S \setminus F,\ 0 \leq j \leq i\}$ between two visits of accepting states. The following lemma relates strategies in $\mathcal{G}$ and $\mathcal{G}_i$. Based on it an efficient computation of an optimal strategy w.r.t. $K(\theta)$ can be performed.

▶ **Lemma 30.** *There is a winning strategy θ in $\mathcal{G}$ with $K(\theta) \leq i$ iff there is a winning strategy θ_i in $\mathcal{G}_i$. Moreover, in the positive case, θ can be chosen to be positional.*

▶ **Theorem 31.** *If $\mathcal{A}$ is actively diagnosable, there exists a positional strategy θ that fulfills $K(\theta) = K_\mathcal{A}$. Moreover, such a strategy can be computed in $2^{\mathcal{O}(n)}$.*

This construction represents a reasonable tradeoff, since, due to Theorem 16, an active diagnoser that realizes a delay equal to the index of $\mathcal{A}$ may need to be much larger, i.e. $2^{\Omega(n\log(n))}$. We sketch the construction of a controller with minimal delay once one knows

that the system is actively diagnosable. One iteratively builds a safety game $\mathcal{G}'_i$ parametrized by increasing values of i. A controller state of this game is defined by (U, d) where U is the set of states reached by a correct sequence while d associates with every state s reached by a faulty sequence a duration $d(s) \leq i + 1$ since the occurrence of the earliest fault that would lead to s. As in the previous games the controller selects a subset of observable actions letting the environment select an action among them. The aim of the controller is to avoid states with some $d(s) = i + 1$. The first i for which $\mathcal{G}'_i$ has a winning strategy is the index and the winning strategy yields an active diagnoser with minimal delay. Observe that since the index is bounded by $2^{\mathcal{O}(n)}$, in the worst case the final game has $2^{\mathcal{O}(n^2)}$ states.

5 Conclusion and Perspectives

We have developed an active-diagnosis method for finite-state systems and shown it to be optimal w.r.t. several criteria. For instance, our work allows to minimize the delay between the occurrence of a fault and its detection. In general, striving for a minimal delay may lead to an overly restrictive controller, e.g., in Figure 1 the controller could completely forbid the action b. We have therefore undertaken work to allow for *parametrized* active diagnosis, which constructs the most permissive controller that respects a user-specified delay. This work, not included here for space reasons, is contained in the long version [7].

Future work has some research leads to address. First, it remains to determine the precise memory requirements for the minimal-delay diagnoser since we showed that it lies between $2^{\Theta(n \log(n))}$ and $2^{\Theta(n^2)}$. Second, the control for active diagnosis could be refined into a *safe* control, i.e. one that does not "encourage" the faulty behaviours. Last we aim at addressing infinite-state systems or systems with quantitative features, as for passive diagnosability in pushdown systems [10], Petri nets [2], timed [17, 15] and probabilistic systems [14].

References

1 D. Berwanger and L. Doyen. On the power of imperfect information. In *Proc. FSTTCS*, volume 2 of *LIPICS*, Bangalore, India, 2008.

2 M.P. Cabasino, A. Giua, S. Lafortune, and C. Seatzu. Diagnosability analysis of unbounded Petri nets. In *CDC09: 48th IEEE Conf. on Decision and Control*, pages 1267–1272, 2009.

3 C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems – Second Edition*. Springer, 2008.

4 F. Cassez and S. Tripakis. Fault diagnosis with static and dynamic observers. *Fundamenta Informaticae*, 88:497–540, 2008.

5 E. Chanthery and Y. Pencolé. Monitoring and active diagnosis for discrete-event systems. In *Proc. SafeProcess'09*, 2009.

6 A. Cimatti, C. Pecheur, and R. Cavada. Formal verification of diagnosability via symbolic model checking. In *Proceedings of IJCAI*, pages 363–369. Morgan Kaufmann, 2003.

7 Stefan Haar, Serge Haddad, Tarek Melliti, and Stefan Schwoon. Optimal constructions for active diagnosis. Research Report LSV-13-12, ENS Cachan, September 2013.

8 R. Küsters. *Memoryless Determinacy of Parity Games*, pages 95–106. LNCS 2500. 2002.

9 S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

10 C. Morvan and S. Pinchinat. Diagnosability of pushdown systems. In *Proceedings of the Haifa Verification Conference*, LNCS 6405, 2009.

11 S. Safra. On the complexity of omega-automata. In *FOCS*, pages 319–327. IEEE, 1988.

12 M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control*, 43(7):908–929, July 1998.

13 M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Trans. Aut. Cont.*, 40(9):1555–1575, 1995.

14 D. Thorsley and D. Teneketzis. Diagnosability of stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 50(4):476–492, 2005.

15 S. Xu, S. Jiang, and R. Kumar. Diagnosis of dense-time systems using digital clocks. *IEEE Transactions on Automation Science and Engineering*, 7(4):870–878, 2010.

16 T-S. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Trans. Automat. Contr.*, 47(9):1491–1495, 2002.

17 S. Hashtrudi Zad, R.H. Kwong, and W.M. Wonham. Fault diagnosis in discrete-event systems: Incorporating timing information. *Trans. Aut. Cont.*, 50(7):1010–1015, 2005.