Ravi Kannan
K. Narayan Kumar (Eds.)

Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2009

29th Annual Conference
Kanpur, India

December 15 – 17, 2009
Proceedings

Indian Association for Research in Computing Science (IARCS)

Editors:
Ravi Kannan
K. Narayan Kumar (Eds.)

Published online by Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany
Leibniz International Proceedings in Informatics (LIPIcs) – Volume 4
Online ISSN 1868-8969
Online ISBN 978-3-939897-13-2

Copyrights are held by the respective authors under Creative Commons License-NC-ND

Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2341

Preface

This volume contains the proceedings of the 29th international conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009), organized under the auspices of the Indian Association for Research in Computing Science (IARCS).

This volume is part of the Leibniz International Proceedings in Informatics (LIPIcs), published online by Schloss Dagstuhl – Leibniz Zentrum für Informatik, Germany. The copyrights to the papers will reside with the respective authors. The copyright is governed by the Creative Commons attribution NC-ND.

This conference attracted 117 submissions. Each submission was reviewed by at least three independent referees. The final selection of the papers making up the programme was done through an electronic discussion on EasyChair, held during the first two weeks of September 2009.

The PC members devoted their valuable time to reviewing papers and coordinating with reviewers. Without their effort, the conference would not be possble. We express our deep appreciation to the PC and to all the reviewers.

We have five invited speakers this year: Anuj Dawar, Kim Larsen, Martin Odersky, R. Ravi and Avi Wigderson. We thank them for having readily accepted our invitation to talk at the conference and also for their contributions to the proceedings.

We thank the Organizing Committee for making the arrangements for the conference. This year, the conference is being held at the Indian Institute of Technology, Kanpur, as part of its golden jubilee celebrations. It is a great honour and privilege for the conference to be recognized and associated with the institute on this occasion.

We would also like to thank Madhavan Mukund, S P Suresh and V Vinay for their help in preparing the style files used in typesetting this volume.

December 2009

Ravi Kannan,
K. Narayan Kumar

Program Committee

Christel Baier (TU Dresden)
Nikhil Bansal (IBM)
Hubert Comon (AIST/ENS-Cachan)
Deepak D'Souza (IISc)
Uri Feige (Weizmann)
Alan Frieze (CMU)
Navin Goyal (MSR)
Erich Grädel (RWTH Aachen)
Anupam Gupta (CMU)
Peter Habermehl (U. of Paris 7)
Ravi Kannan (MSR) co-chair
Amit Kumar (IIT Delhi)
Piyush Kurur (IIT Kanpur)
Dietrich Kuske (U. of Leipzig)
Satya Lokam (MSR)
Meena Mahajan (IMSc)
Rupak Majumdar (UCLA)
Anca Muscholl (U. of Bordeaux)
K Narayan Kumar (CMI) co-chair
Aditya Nori (MSR)
Jaikumar Radhakrishnan (TIFR)
Uday S Reddy (Birmingham)
Luc Segoufin (INRIA/ENS-Cachan)
S P Suresh (CMI)
Shanghua Teng (USC)
Nisheeth Vishnoi (MSR/CNRS)
Mahesh Viswanathan (UIUC)
Thomas Wilke (U. of Kiel)
James Worrell (Oxford)
Andrew C Yao (Tsinghua)

Organizing Committee

Sanjeev Aggarwal (IIT Kanpur)
Manindra Agrawal (IIT Kanpur)
Surender Baswana (IIT Kanpur)
Somenath Biswas (IIT Kanpur)
Sumit Ganguly (IIT Kanpur) chair
Piyush Kurur (IIT Kanpur)

Referees

Samy Abbes
S. Akshay
Eric Allender
Krzysztof Apt
Nimar Arora
Vikraman Arvind
Mohamed Faouzi Atig

Amitabha Bagchi
Mohua Banerjee
Vince Bara
A Baskar
Surender Baswana
Raghav Bhaskar
Benedikt Bollig
Patricia Bouyer
Tomas Brazdil

Olivier Carton
Franck Cassez
Rohit Chadha
Deeparnab Chakrabarty
Jean-Marc Champarnaud
Eden Chlamtac
Tom Chothia
Bruno Codenotti
Thomas Colcombet

Samir Datta
Stephane Demri
Nikhil Devanur
Yevgeni Dodis
Cezara Dragoi
Sagarmoy Dutta

Michael Emmi

Guillaume Fertin
Azadeh Farzan
Bernd Finkbeiner
Matthias Fitzi
Carsten Fritz

Sibylle Frschle

Anna Gal
Didier Galmiche
Sumit Ganguly
Pierre Ganty
Naveen Garg
Wouter Gelade
Blaise Genest
Dan Ghica
Hugo Gimbert
Navin Goyal
Fabrizio Grandoni
Marcus Groesser
Hermann Gruber
Dimitar Guelev
Hari Gupta

Christoph Haase
Matthew Hague
Ramesh Hariharan
Jonathan Hayman
Thanh Minh Hoang
Florian Horn
Pavel Hrubes

Lucian Ilie
Neil Immerman
Radu Iosif

Florent Jacquemard
Mahdi Jaghouri
Daniel Johannsen
Jean-Pierre Jouannaud

Subrahmahyam Kalyanasundaram
Purushottam Kar
Joost-Pieter Katoen
Telikepalli Kavitha
Sascha Klueppelholz
Timo Koetzing
Vijay Anand Korthikanti

Sasha Kostochka
Yiannis Koutis
Robi Krauthgamer
Steve Kremer
Klaas Ole Kuertz
Orna Kupferman
Roman Kuznets

Martin Leucker
Kamal Lodaya
Christof Loeding
Markus Lohrey
Sylvain Lombardy
Anand Louis

Raj Mohan M.
Sebastian Maneth
Roman Manevich
Nicolas Markey
Richard Mayr
Tomasz Mazur
Ingmar Meinecke
Roland Meyer
Partha Mukhopadhyay
Madhavan Mukund
Filip Murlak
Muthu Muthukrishnan

Prasad Naldurg
Satyadev Nandakumar
Rolf Niedermeier
Gethin Norman
Adam O'Neill

Friedrich Otto
Joel Ouaknine

Hristina Palikareva
David Parker
Pawel Parys
Dominique Perrin
Paul Pettersson
Geevarghese Philip
Pavithra Prabhakar
Prakash Prabhu
Sanjiva Prasad
M. Praveen

Karin Quaas

Alexander Rabinovich
Venkatesh Raman
Vishwanath Raman
R Ramanujam
Eike Ritter

Indranil Saha
Chandan Saha
Ramprasad Saptharishi
Shubhangi Saraf
Jayalal M.N. Sarma
Atish Das Sarma
Saket Saurabh
Nitin Saxena
Benedikt Schmidt
Henning Schnoor
Warren Schudy
Pranab Sen
Olivier Serre
Robert Simmons
D. Sivakumar
Gregory Sorkin
Srikanth Srinivasan
David Steurer
Lutz Strassburger
Martin Strauss

Inbal Talgam
Hayo Thielecke
Richard Trefler

Viktor Vafeiadis
Daniele. Varacca
Kasturi Varadarajan
Kapil Vaswani
Ramesh Viswanathan
Mikhail Volkov

Magnus Wahlstrom
Glynn Winskel
Nicols Wolovick

Mihalis Yannakakis

Marc Zeitoun
Mariano Zelke

LIPIcs Leibniz International Proceedings in Informatics

Mediating for Reduction
(on Minimizing Alternating Büchi Automata)

Parosh A. Abdulla[1], Yu-Fang Chen[1], Lukáš Holík[2], Tomáš Vojnar[2]

[1] University of Uppsala, Sweden,
{parosh,yu-fang.chen}@it.uu.se

[2] FIT, Brno University of Technology, Czech Republic,
{holik,vojnar}@fit.vutbr.cz

ABSTRACT. We propose a new approach for minimizing alternating Büchi automata (ABA). The approach is based on the so called *mediated equivalence* on states of ABA, which is the maximal equivalence contained in the so called *mediated preorder*. Two states p and q can be related by the mediated preorder if there is a *mediator* (mediating state) which forward simulates p and backward simulates q. Under some further conditions, letting a computation on some word jump from q to p (due to they get collapsed) preserves the language as the automaton can anyway already accept the word without jumps by runs through the mediator. We further show how the mediated equivalence can be computed efficiently. Finally, we show that, compared to the standard forward simulation equivalence, the mediated equivalence can yield much more significant reductions when applied within the process of complementing Büchi automata where ABA are used as an intermediate model.

1 Introduction

Alternating Büchi automata (ABA) are succinct state-machine representations of ω-regular languages (regular sets of infinite sequences). They are widely used in the area of formal specification and verification of non-terminating systems. One of the most prominent examples of the use of ABA is the complementation of nondeterministic Büchi automata [9]. It is an essential step of the automata-theoretic approach to model checking when the specification is given as a positive Büchi automaton [12] and also learning based model checking for liveness properties [4]. The other important usage of ABA is as the intermediate data structure for translating a linear temporal logic (LTL) specification to an automaton [7].

However, because of the compactness of ABA*, usually the algorithms that work on them are of high complexity. For example, both the complementation and the LTL translation algorithms transform an intermediate ABA to an equivalent NBA. The transformation is exponential in the size of the input ABA. Hence, one may prefer to reduce the size of the ABA (with some relatively cheaper algorithm) before giving it to the exponential procedure.

In the study of Fritz and Wilke, simulation-based minimization is proven as a very effective tool for reducing the size of ABA [6]. However, they considered only *forward* simulation relations. Inspired by some previous works [1], we believe that *backward* simulation can be used for reducing the size of ABA as well. Unfortunately, as will be explained in Section 3, quotienting wrt. *backward* simulation (i.e., simplify the automaton by collapsing backward simulation equivalent states) does not preserve the language.

*ABA's are exponentially more succinct than the nondeterministic ones.

© Abdulla,Chen, Holík,Vojnar; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 1–12
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs/FSTTCS/2009/2302

In this paper, we develop an approach that uses backward simulation for simplifying ABA indirectly. Instead of looking for a suitable fragment of backward simulation that can be used to reduce the number of states of an ABA, we combine backward and forward simulation to form an even coarser relation called *mediated preorder* that can be used for minimization. The performance of minimizing ABA with *mediated preorder* is evaluated on a large set of experiments. In the experiments, we apply different simulation-based minimization approaches to improve the complementation algorithm of nondeterministic Büchi automata. The experimental results show that the minimization using mediated preorder significantly outperforms the minimization using forward simulation. To be more specific, in average, mediated minimization results in a 30% better reduction in the number of states and 50% better reduction in the number of transitions than forward minimization on the intermediate ABA. Moreover, in the complemented nondeterministic Büchi automata, mediated minimization results in a 100% better reduction in the number of states and 300% better reduction in the number of transitions than forward minimization.

2 Basic Definitions

Given a finite set X, we use X^* to denote the set of all finite words over X and X^ω for the set of all infinite words over X. The empty word is denoted ϵ and $X^+ = X^* \setminus \{\epsilon\}$. The concatenation of a finite word $u \in X^*$ and a finite or infinite word $v \in X^* \cup X^\omega$ is denoted by uv. For a word $w \in X^* \cup X^\omega$, $|w|$ is the length of w ($|w| = \infty$ if $w \in X^\omega$), w_i is the ith letter of w and w^i the ith prefix of w (the word u with $w = uv$ and $|u| = i$). $w^0 = \epsilon$. The concatenation of a finite word u and a set $S \subseteq X^* \cup X^\omega$ is defined as $uS = \{uv \mid v \in S\}$.

An *alternating Büchi automaton* is a tuple $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$ where Σ is a finite alphabet, Q is a finite set of states, $\iota \in Q$ is an initial state, $\alpha \subseteq Q$ is a set of accepting states, and $\delta : Q \times \Sigma \to 2^{2^Q}$ is a total transition function. A *transition* of $\mathcal{A}$ is of the form $p \xrightarrow{a} P$ where $P \in \delta(q, a)$.

A *tree T over Q* is a subset of Q^+ that contains all nonempty prefixes of each one of its elements (i.e., $T \cup \{\epsilon\}$ is prefix-closed). Furthermore, we require that T contains exactly one $r \in Q$, the *root of T*, denoted $root(T)$. We call the elements of Q^+ *paths*. For a path πq, we use $leaf(\pi q)$ to denote its last element q. Define the set $branches(T) \subseteq Q^+ \cup Q^\omega$ such that $\pi \in branches(T)$ iff T contains all prefixes of π and π is not a proper prefix of any path in T. In other words, a *branch of T* is either a maximal path of T, or it is a word from Q^ω such that T contains all its nonempty prefixes. We use $succ_T(\pi) = \{r \mid \pi r \in T\}$ to denote the set of successors of a path π in T, and $height(T)$ to denote the length of the longest branch of T. The tree U over Q is a *prefix of T* iff $U \subseteq T$ and for every $\pi \in U$, $succ_U(\pi) = succ_T(\pi)$ or $succ_U(\pi) = \emptyset$. The *suffix of T* defined by a path πq is the tree $T(\pi q) = \{q\psi \mid \pi q \psi \in T\}$.

Given a word $w \in \Sigma^\omega$, a tree T over Q is a *run of $\mathcal{A}$ on w*, if for every $\pi \in T$, $leaf(\pi) \xrightarrow{w_{|\pi|}} succ_T(\pi)$ is a transition of $\mathcal{A}$. Finite prefixes of T are called *partial runs on w*. A run T of $\mathcal{A}$ over w is *accepting* iff every infinite branch of T contains infinitely many accepting states. A word w is *accepted* by $\mathcal{A}$ from a state $q \in Q$ iff there exists an accepting run T of $\mathcal{A}$ over w with $root(T) = q$. The *language of a state $q \in Q$ in $\mathcal{A}$*, denoted $\mathcal{L}_\mathcal{A}(q)$, is the set of all words accepted by $\mathcal{A}$ from q. Then $\mathcal{L}(\mathcal{A}) = \mathcal{L}_\mathcal{A}(\iota)$ is the *language of $\mathcal{A}$*. For simplicity of presentation, we assume in the rest of the paper that δ never allows a transition of the form $p \xrightarrow{a} \emptyset$. This means that no run can contain a finite branch. Any automaton can be easily transformed into one without such transitions by adding a new accepting state q with $\delta(q, a) = \{\{q\}\}$ for every $a \in \Sigma$ and replacing every transition $p \xrightarrow{a} \emptyset$ by $p \xrightarrow{a} \{q\}$.

3 Simulation Relations

In this section, we give the definitions of forward and backward simulation over ABA and discuss some of their properties. The notion of backward simulation is inspired by a similar tree automata notion studied in [1, 3]—namely, the upward simulation parametrised by a downward simulation (the connection between tree automata and ABA follows from the fact that the runs of ABA are in fact trees).

For the rest of the section, we fix an ABA $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$. We define relations $\preceq_\alpha$ and $\preceq_\iota$ on Q s.t. $q \preceq_\alpha r$ iff $q \in \alpha \implies r \in \alpha$ and $q \preceq_\iota r$ iff $q = \iota \implies r = \iota$. For a binary relation $\preceq$ on a set X, the relation $\preceq^{\forall\exists}$ on subsets of X is defined as $Y \preceq^{\forall\exists} Z$ iff $\forall z \in Z.\, \exists y \in Y.\, y \preceq z$, i.e., iff the upward closure of Z wrt. $\preceq$ is a subset of the upward closure of Y wrt. $\preceq$.

Forward Simulation. A *forward simulation* on $\mathcal{A}$ is a relation $\preceq_F \subseteq Q \times Q$ such that $p \preceq_F r$ implies that (i) $p \preceq_\alpha r$ and (ii) for all $p \xrightarrow{a} P$, there exists a $r \xrightarrow{a} R$ such that $P \preceq_F^{\forall\exists} R$.

For the basic properties of forward simulation, we rely on the work [8] by Gurumurthy et al. In particular, (i) there exists a unique maximal forward simulation $\preceq_F$ on $\mathcal{A}$ which is reflexive and transitive, (ii) for any $q, r \in Q$ such that $q \preceq_F r$, it holds that $\mathcal{L}_\mathcal{A}(q) \subseteq \mathcal{L}_\mathcal{A}(r)$, and (iii) quotienting wrt. $\preceq_F \cap \preceq_F^{-1}$ preserves the language of $\mathcal{A}$.

Backward Simulation. Let $\preceq_F$ be a forward simulation on $\mathcal{A}$. A *backward simulation* on $\mathcal{A}$ parameterized by $\preceq_F$ is a relation $\preceq_B \subseteq Q \times Q$ such that $p \preceq_B r$ implies that (i) $p \preceq_\iota r$, (ii) $p \preceq_\alpha r$, and (iii) for all $q \xrightarrow{a} P \cup \{p\}, p \notin P$, there exists a $s \xrightarrow{a} R \cup \{r\}, r \notin R$ such that $q \preceq_B s$ and $P \preceq_F^{\forall\exists} R$. The below lemma describes some properties of backward simulation.

LEMMA 1. *For any reflexive and transitive forward simulation $\preceq_F$ on $\mathcal{A}$, there exists a unique maximal backward simulation $\preceq_B$ on $\mathcal{A}$ parameterized by $\preceq_F$ that is reflexive and transitive.*

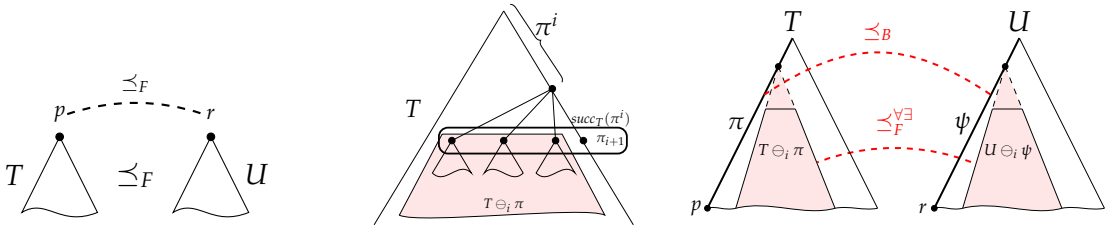
Backward simulation itself cannot be used for quotienting. In [2], we give an example of an automaton, where quotienting using backward simulation does not preserve language. However, in Section 4.1, we show how backward simulation can be used to define a new relation for reducing ABA.

Let $\preceq_F$ and $\preceq_B$ be forward and backward simulations on $\mathcal{A}$, which are both reflexive and transitive. For every $x \in \{B, F, \alpha\}$, we extend the relation $\preceq_x$ to $Q^+ \times Q^+$ such that for $\pi, \psi \in Q^+$, $\pi \preceq_x \psi$ iff $|\pi| = |\psi|$ and for all $1 \leq i \leq |\pi|$, $\pi_i \preceq_x \psi_i$. We say that ψ forward simulates π, ψ backward simulates π, or ψ is more accepting than π when $\pi \preceq_F \psi$, $\pi \preceq_B \psi$, or $\pi \preceq_\alpha \psi$, respectively. This notation is further extended to trees. For trees T, U over Q and for $x \in \{\alpha, F\}$, we write, $T \preceq_x U$ if $branches(T) \preceq_x^{\forall\exists} branches(U)$. Similarly, we say that U forward simulates T, or U is more accepting than T when $T \preceq_F U$, or $T \preceq_\alpha U$, respectively. Note that $\preceq_x$ is reflexive and transitive for all the variants of $x \in \{F, B, \alpha\}$ defined over states, paths, or trees (this follows from the assumption that the original relations $\preceq_F$ and $\preceq_B$ on states are reflexive and transitive). Moreover, $\preceq_B \subseteq \preceq_\alpha$, $\preceq_B \subseteq \preceq_\iota$, and $\preceq_F \subseteq \preceq_\alpha$.

The following two lemmas formulate properties of the simulation relations that we will use in the rest of the paper.

LEMMA 2. *For any $p, r \in Q$ with $p \preceq_F r$ and a partial run T of $\mathcal{A}$ on $w \in \Sigma^\omega$ with the root p, there is a partial run U of $\mathcal{A}$ on w with the root r such that $T \preceq_F U$.*

For a tree T over Q, $\pi \in T$, and $1 \leq i \leq |\pi|$, the set $T \ominus_i \pi$ is the union of branches of suffix trees $T(\pi^i q), q \in succ_T(\pi^i)$, with the branches of the suffix tree $T(\pi^{i+1})$ excluded.



(a) Lemma 2 (b) The notation $T \ominus_i \pi$ (c) Lemma 3

Figure 1: Illustration of the lemmas

Formally, let $Q^i = succ_T(\pi^i) \setminus \{\pi_{i+1}\}$ be the set of all successors of π^i in T without the successor continuing in π. Then $T \ominus_i \pi = \bigcup_{q \in Q^i} branches(T(\pi^i q))$ (notice that if $i = 0$, then $T \ominus_i \pi = \emptyset$).

LEMMA 3. *For any $p, r \in Q$ with $p \preceq_B r$, a partial run T of $\mathcal{A}$ on $w \in \Sigma^\omega$ and $\pi \in branches(T)$ with $leaf(\pi) = p$, there is a partial run U of $\mathcal{A}$ on w and $\psi \in branches(U)$ with $leaf(\psi) = r$ such that $\pi \preceq_B \psi$, and for all $1 \leq i \leq |\pi|$, $T \ominus_i \pi \preceq_F^{\forall\exists} U \ominus_i \psi$.*

4 Mediated Equivalence and Quotienting

Here we discuss the possibility of an indirect use of backward simulation for simplifying ABA via quotienting. We do not look for a suitable fragment of backward simulation only. Instead, we (1) combine backward and forward simulation to form an equivalence that subsumes both backward and forward simulation equivalence and (2) take a certain fragment of this equivalence, called *mediated equivalence*, that can be used for quotienting.

4.1 The Notion and Intuition of Mediated Equivalence

Collapsing states of an automaton wrt. some equivalence allows a run that arrives to some state to *jump* to another equivalent state and continue from there. Alternatively, this can be viewed as *extending* the source state of the jump by the outgoing transitions of the target state[†]. The equivalence must have the property that the language is not increased even when the jumps (or, alternatively, transition extensions) are allowed. This is what we aim at when introducing the *mediated equivalence* $\equiv_M$ based on a so called *mediated preorder* $\preceq_M$. The mediated preorder $\preceq_M$ will in particular be defined as a suitable transitive fragment of $\preceq_F \circ \preceq_B^{-1}$ in the following.

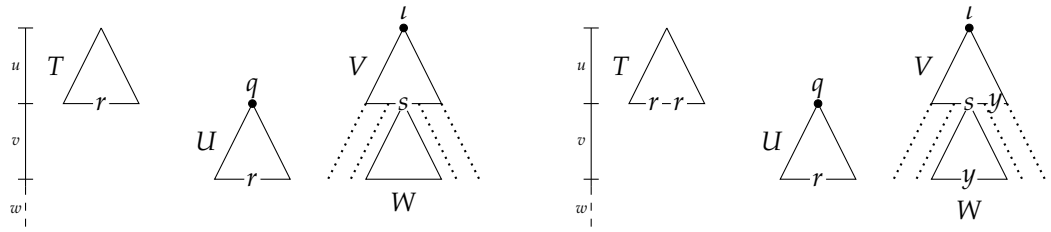
The intuition behind allowing a run to jump from a state r to a state q such that $q \preceq_F \circ \preceq_B^{-1} r$ is the existence of the so called *mediator*, i.e., a state s such that $q \preceq_F s \preceq_B^{-1} r$ (cf. Fig. 2(a)). The state s can be reached in the same way and in the same context[‡] as r, and, at the same time, the automaton can continue from s in the same way as from q. Hence, intuitively, the newly allowed run based on the jump from r to q does not add anything to the language because it can anyway be realized through s without jumps.

Unfortunately, the relation $\preceq_F \circ \preceq_B^{-1}$ cannot be directly used as it is not transitive, and taking its symmetric closure would thus not yield an equivalence. We thus have to take some of its *transitive fragments*. This is natural as if the automaton can safely jump from q_1 to q_2 and from q_2 to q_3, it should be able to safely jump from q_1 to q_3 too.

This is, however, still not enough. Not all of the transitive fragment of $\preceq_F \circ \preceq_B^{-1}$ can be used for quotienting. We can only take a fragment $\preceq_M$ that is *forward extensible*, meaning

[†]The first view is better when explaining the intuition whereas the other is easier to be used in proofs.

[‡]If a state s is a leaf of a partial run, then by a *context* of s we mean all the other leaves of the partial run.



(a) The Mediator (b) Potential Problems

Figure 2: Basic Intuition Behind Mediated Equivalence

that if $q_1 \preceq_M q_2 \preceq_F q_3$, then $q_1 \preceq_M q_3$. The intuitive meaning of this requirement is the following. When a run jumps from r to q, it may be the case that r is again reached later on or it appears in the context of itself (cf. Fig. 2(b)). If r is reached in the continuation of the run from q, the mediated preorder assures that there is some state y in the run continuing from the mediator s that forward simulates r. Similarly, if the context of r contains another occurrence of r, there is some state y in the context of s that forward simulates r. However, this forward simulation is in general guaranteed to hold only when no further jumps are allowed. In order to guarantee a possibility of further simulation, we require that if the computation is allowed to jump from r to q, it is allowed to jump from y to q too.

Finally, to make the mediated equivalence applicable, we must pose one more requirement. Namely, we require that the transitions of the given ABA are not $\preceq_F$*-ambiguous*, meaning that no two states on the right hand side of a transition are forward equivalent. Intuitively, allowing such transitions goes against the spirit of the backward simulation. For a mediator p to backward simulate a state r wrt. rules $\rho_1 : p' \xrightarrow{a} P \cup \{p\}, p \notin P$, and $\rho_2 : r' \xrightarrow{a} R \cup \{r\}, r \notin R$, it must be the case that each state x in the context P of p within ρ_1 is less restrictive (i.e., forward bigger) than some state y in the context R of r within ρ_2. The state r itself is not taken into account when looking for y because we aim at extending its behaviour by collapsing (and it could then become less restrictive than the appropriate x). In the case of $\preceq_F$-ambiguity, the spirit of this restriction is in a sense broken since the forward behaviour of r may still be taken into account when checking that the context of p is less restrictive than that of r. This is because the behaviour of r appears in R as the behaviour of some other state r'' too. Consequently, r and r'' may back up each other in a circular way when checking the restrictiveness of the contexts within the construction of the backward simulation. Both of them can then seem extensible, but once their behaviour gets extended, the restriction of their context based on their own original behaviour is lost, which may then increase the language (an example of such a scenario is given in [2]). However, in Section 5, we show that $\preceq_F$-ambiguity can be efficiently removed.

Mediated Preorder and Equivalence. Let $\preceq_F$ be a reflexive and transitive forward simulation on $\mathcal{A}$, and $\preceq_B$ a reflexive and transitive backward simulation on $\mathcal{A}$ parameterized by $\preceq_F$. A preorder $\preceq_M \subseteq \preceq_F \circ \preceq_B^{-1}$ such that for all $q, r, s \in Q$, $q \preceq_M r \preceq_F s$ implies $q \preceq_M s$, is a *mediated preorder* induced by $\preceq_F$ and $\preceq_B$. The relation $\equiv_M = \preceq_M \cap \preceq_M^{-1}$ is then a *mediated equivalence* induced by $\preceq_F$ and $\preceq_B$.

LEMMA 4.*[3] There is a unique maximal mediated preorder $\preceq_M$ induced by $\preceq_F$ and $\preceq_B$.*

4.2 Extending Automata According to Mediated Preorder Preserves Language

Quotient Automata versus Extended Automata. We first show that quotienting can be seen as a simpler operation of adding transitions and accepting states. Let $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$ be an ABA and let $\equiv$ be an equivalence on Q such that $\equiv = \preceq \cap \preceq^{-1}$ for some preorder $\preceq$. Let the automaton $\mathcal{A}/\equiv$ be the quotient of $\mathcal{A}$ wrt. $\equiv$ that arises by merging $\equiv$-equivalent states of $\mathcal{A}$, and let $\mathcal{A}^+$ be the automaton extended according to $\preceq$, that is created as follows: for every two states q, r of $\mathcal{A}$ with $q \preceq r$, (i) add all outgoing transitions of q to r, (ii) if $q \equiv r$ and q is final, make r final.

The automata $\mathcal{A}/\equiv$ and $\mathcal{A}^+$ are formally defined as follows. Let $Q/\equiv$ denote the quotient of Q wrt. $\equiv$, and let $[q]$ denote the equivalence class of $\equiv$ containing q. Then $\mathcal{A}/\equiv = (\Sigma, Q/\equiv, [\iota], \delta/\equiv, \{[q] \mid q \in \alpha\})$ and $\mathcal{A}^+ = (\Sigma, Q, \delta^+, \iota, \alpha^+)$, where $\alpha^+ = \{p \mid \exists q \in \alpha.\ q \equiv p\}$ and, for each $a \in \Sigma$, $q \in Q$, $\delta/\equiv([q], a) = \bigcup_{p \in [q]} \{\{[p'] \mid p' \in P\} \mid P \in \delta(p, a)\}$ and $\delta^+(q, a) = \bigcup_{p \in Q \wedge p \preceq q} \delta(p, a)$. It is not difficult to show that $\mathcal{L}(\mathcal{A}/\equiv) \subseteq \mathcal{L}(\mathcal{A}^+)$ [2] (Lemma 8 in [2]). Hence, if adding transitions and accepting states according to $\preceq$ preserves the language, then quotienting according to $\equiv$ preserves the language too.

Language Preservation by Mediated Equivalence. We now give a sketch of the proof that extending automata according to the mediated preorder preserves the language. The full proofs can be found in [2]. For the rest of the section, we fix an ABA $A = (\Sigma, Q, \iota, \delta, \alpha)$, a reflexive and transitive forward simulation $\preceq_F$ on $\mathcal{A}$ such that $\mathcal{A}$ is $\preceq_F$-unambiguous, and a reflexive and transitive backward simulation $\preceq_B$ on $\mathcal{A}$ parameterized by $\preceq_F$. Let $\preceq_M$ be a mediated preorder induced by $\preceq_F$ and $\preceq_B$, and let $\mathcal{A}^+$ be the automaton extended according to $\preceq_M$. Let $\equiv_M = \preceq_M \cap \preceq_M^{-1}$.

We want to prove that $\mathcal{L}(\mathcal{A}^+) = \mathcal{L}(\mathcal{A})$. The nontrivial part is showing that $\mathcal{L}(\mathcal{A}^+) \subseteq \mathcal{L}(\mathcal{A})$—the converse is obvious. To prove $\mathcal{L}(\mathcal{A}^+) \subseteq \mathcal{L}(\mathcal{A})$, we need to show that, for every accepting run of $\mathcal{A}^+$ on a word w, there is an accepting run of $\mathcal{A}$ on w. We proceed as follows. We first prove Lemma 5, which shows how partial runs of $\mathcal{A}$ with an increased power of their leaves (wrt. $\preceq_F$) can be built incrementally from other runs of $\mathcal{A}$, bridging the gap between $\mathcal{A}$ and $\mathcal{A}^+$. Then we prove Lemma 7 saying that, for every partial run on a word w of $\mathcal{A}^+$, there is a partial run of $\mathcal{A}$ on w that is more accepting (recall that partial runs are finite). By carry this result over to infinite runs we get the proof of Theorem 8.

Consider a partial run T of $\mathcal{A}$ on a word w, we choose for each leaf p of T an $\preceq_M$-smaller state p'. Suppose that we allow p to make one step using the transitions of p' or to become accepting if p' is accepting and $p' \equiv_M p$. (Thus, we give the leaves of T a part of the power they would have in $\mathcal{A}^+$). We will show that there exists a partial run U of $\mathcal{A}$ on w such that (1) it is more accepting than T, and (2) the leaves of U can mimic the next step of the leaves of T even if the leaves of T use their extended power.

The above is formalized in Lemma 5 using the following notation. For a partial run T of $\mathcal{A}$ on w, we define *ext* as an *extension function* that assigns to every branch π of T a state $ext(\pi)$ such that $ext(\pi) \preceq_M leaf(\pi)$.

Let U be a partial run of $\mathcal{A}$ on w. For two branches $\pi \in branches(T)$ and $\psi \in branches(U)$, we say that ψ *strongly covers* π wrt. *ext*, denoted $\pi \preceq_{ext} \psi$, iff $\pi \preceq_\alpha \psi$ and $ext(\pi) \preceq_F leaf(\psi)$. Similarly, we say that ψ *weakly covers* π wrt. *ext*, denoted $\pi \preceq_{\text{w-}ext} \psi$, iff $\pi \preceq_\alpha \psi$ and $ext(\pi) \preceq_M leaf(\psi)$. We extend the concept of covering to partial runs as follows. We write $T \preceq_{ext} U$ (U *strongly covers* T wrt. *ext*) iff $branches(T) \preceq_{ext}^{\forall\exists} branches(U)$ and $root(T) \preceq_B$

root(U). Likewise, we write $T \preceq_{\text{w-}ext} U$ (U *weakly covers* T wrt. *ext*) iff *branches*$(T) \preceq^{\forall\exists}_{\text{w-}ext}$ *branches*(U) and *root*$(T) \preceq_B$ *root*(U). Note that we have $\preceq_{ext} \subseteq \preceq_{\text{w-}ext}$ for branches as well for partial runs because $\preceq_F \subseteq \preceq_M$. So, the strong covering implies the weak one.

LEMMA 5. *For any partial run T of $\mathcal{A}$ on a word w with an extension function ext, there is a partial run U of $\mathcal{A}$ on w with $T \preceq_{ext} U$.*

Proving Lemma 5 is the most intricate part of the proof of Theorem 8. We introduce the concepts used within the proof of Lemma 5 and provide an overview of the proof.

If $T \preceq_{ext} T$, we are done as in the statement of the lemma, we can take T to be U. So, suppose that $T \not\preceq_{ext} T$. Observe that *root*$(T) \preceq_B$ *root*(T), and every branch of T weakly covers itself, which means that $T \preceq_{\text{w-}ext} T$. We will show how to reach U by a chain of partial runs derived from T. The partial runs within the chain will all weakly cover T. Runs further from T will in some sense cover T more strongly than the runs closer to T. The last partial run of the chain will cover T strongly. To do this, we need a suitable measure that, for a partial run V of $\mathcal{A}$ on w with $T \preceq_{\text{w-}ext} V$, tells us how strongly V covers T.

To define the measure, we concentrate on branches of V that cause that V does not cover T strongly. These are branches $\psi \in$ *branches*(V) for which there is no $\pi \in$ *branches*(T) with $\pi \preceq_{ext} \psi$ (there are only some $\pi \in$ *branches*(T) with $\pi \preceq_{\text{w-}ext} \psi$). We call them *strict weakly covering branches*. Let $\mathsf{sw}_T(V)$ denote the tree which is the subset of V containing prefixes of strict weakly covering branches of V wrt. T. Note that $T \preceq_{ext} V$ iff V contains no strict weakly covering branches, which is equivalent to $\mathsf{sw}_T(V) = \emptyset$. For a partial run W of $\mathcal{A}$ on w, we will define which of V and W cover T more strongly by comparing $\mathsf{sw}_T(V)$ and $\mathsf{sw}_T(W)$. For this, we need the following definitions.

Given a finite tree X over Q and $\tau \in Q^+$, we define the *tree decomposition* of X according to τ as the sequence of (finite) sets of paths $\langle \tau, X\rangle = X \ominus_1 \tau, X \ominus_2 \tau, \ldots, X \ominus_{|\tau|} \tau$. We also let $\langle \epsilon, X\rangle =$ *branches*(X), which is a sequence of length 1. Notice that under the condition that $\tau \notin$ *branches*(X), $\langle \tau, X\rangle = \emptyset \ldots \emptyset$ implies that $X = \emptyset$[§].

Let $\tau_V \in V \cup \{\epsilon\}$ and $\tau_W \in W \cup \{\epsilon\}$ be such that $\tau_V \notin$ *branches*$(\mathsf{sw}_T(V))$ and $\tau_W \notin$ *branches*$(\mathsf{sw}_T(W))$. We say that W *covers* T *more strongly* than V wrt. τ_V and τ_W, denoted $V \prec^T_{\tau_V,\tau_W} W$, iff *root*$(V) \preceq_B$ *root*(W) and $\langle \tau_V, \mathsf{sw}_T(V)\rangle \sqsubset \langle \tau_W, \mathsf{sw}_T(W)\rangle$, where $\sqsubset$ is a binary relation on sequences of sets of paths defined as follows.

For two sets of paths P and P', we use $P \prec^{\forall\exists}_F P'$ to denote that $P \preceq^{\forall\exists}_F P'$ but not $P' \preceq^{\forall\exists}_F P$. In other words, the upward closure of P' wrt. $\preceq_F$ is a proper subset of the upward closure of P wrt. $\preceq_F$. Then, for sequences of finite sets $S, S' \in (2^Q)^+$, $S \sqsubset S'$ iff there is some $k \in \mathbb{N}, k \leq \min\{|S|, |S'|\}$, such that $S_k \prec^{\forall\exists}_F S'_k$ and for all $1 \leq j < k$, $S_j \preceq^{\forall\exists}_F S'_j$. It is not hard to show that the relation $\sqsubset$ is a partial order. Observe that $\sqsubset$ does not allow infinite increasing chains of sequences where the length of the sequences is bounded by some constant (this follows from that $\preceq_F$ compares only paths of an equal length and therefore every increasing chain of finite sets of paths related by $\prec^{\forall\exists}_F$ is finite). Moreover, $S \sqsubset \emptyset \ldots \emptyset$ for every sequence of sets of paths $S \neq \emptyset \ldots \emptyset$.

[§]Note that if $\tau \in$ *branches*(X), $\langle \tau, X\rangle = \emptyset \ldots \emptyset$ does not imply $X = \emptyset$ as τ could be the only branch of X. This is important as for a partial run Y and $\tau' \in Y$, if $\tau' \notin$ *branches*(Y), the implications $\langle \tau', \mathsf{sw}_T(Y)\rangle = \emptyset \ldots \emptyset \implies \mathsf{sw}_T(Y) = \emptyset \implies T \preceq_{ext} Y$ hold. However, the first implication does not hold if $\tau' \in$ *branches*(Y).

LEMMA 6. *Given a partial run V of $\mathcal{A}$ on w s.t. $T \preceq_{\text{w-}ext} V$, $T \not\preceq_{ext} V$, and $\tau_V \in V \cup \{\epsilon\}$ with $\tau_V \notin branches(\mathsf{sw}_T(V))$, we can construct a partial run W of $\mathcal{A}$ on w with $T \preceq_{\text{w-}ext} W$ and a path $\tau_W \in W$ with $\tau_W \notin branches(\mathsf{sw}_T(W))$ such that $V \prec^T_{\tau_V,\tau_W} W$.*

PROOF. [Sketch] The proof of Lemma 6 relies on Lemma 3 and the definition of $\preceq_M$. We first choose a suitable branch π of $\mathsf{sw}_T(V)$ as follows. Let $1 \leq k \leq |\tau_V|$ be some index such that $\mathsf{sw}_T(V) \ominus_k \tau_V$ is nonempty. If $\tau_V = \epsilon$, then $k = 1$. We choose some $\pi' \in \mathsf{sw}_T(V) \ominus_k \tau_V$ which is minimal wrt. $\preceq_F$, meaning that there is no $\pi'' \in \mathsf{sw}_T(V) \ominus_k \tau_V$ different from π' such that $\pi'' \preceq_F \pi'$. We put $\pi = \tau_V^k \pi'$. We note that this is the place where we use the $\preceq_F$-unambiguity assumption. If $\mathcal{A}$ was $\preceq_F$-ambiguous, there need not be a k such that $\mathsf{sw}_T(V) \ominus_k \tau_V$ contains a minimal element wrt. $\preceq_F$.

From $ext(\pi) \preceq_M leaf(\pi)$, there is a mediator s with $ext(\pi) \preceq_F s \succeq_B leaf(\pi)$. We apply Lemma 3 to V, π, $leaf(\pi)$ and s, which give us a partial run W and $\psi \in branches(W)$ with $leaf(\psi) = s$ such that $\pi \preceq_B \psi$, and for all $1 \leq i \leq |\pi|$, $V \ominus_i \pi \preceq_F^{\forall\exists} W \ominus_i \psi$. Let $\tau_W = \psi$. The proof can be concluded by showing that (i) $T \preceq_{\text{w-}ext} W$, (ii) $\tau_W \notin branches(\mathsf{sw}_T(W))$, and (iii) $\langle \tau_V, \mathsf{sw}_T(V) \rangle \sqsubset \langle \tau_W, \mathsf{sw}_T(W) \rangle$, which implies $V \prec^T_{\tau_V,\tau_W} W$. ∎

Now we construct a run U strongly covering T as follows. Starting from T and ϵ, we can construct a chain $T \prec^T_{\epsilon,\tau_1} T_1 \prec^T_{\tau_1,\tau_2} T_2 \prec^T_{\tau_2,\tau_3} T_3 \ldots$ by successively applying Lemma 6 for each i, $\tau_i \in T_i$, $\tau_i \notin branches(\mathsf{sw}_T(T_i))$, and $T \preceq_{\text{w-}ext} T_i$. Observe that by the definition of stronger covering, we have that $\langle \epsilon, \mathsf{sw}_T(T) \rangle \sqsubset \langle \tau_1, \mathsf{sw}_T(T_1) \rangle \sqsubset \langle \tau_2, \mathsf{sw}_T(T_2) \rangle \sqsubset \langle \tau_3, \mathsf{sw}_T(T_3) \rangle \ldots$ Notice that, for each i, as $T \preceq_{\text{w-}ext} T_i$, $height(T_i) = height(T)$. Therefore the length of τ_i as well as the length of $\langle \tau_i, \mathsf{sw}_T(T_i) \rangle$ are bounded by $height(T)$.

Recall that (i) the relation $\sqsubset$ is a partial order, (ii) that $\sqsubset$ does not allow infinite increasing chains of sequences where the length of the sequences is bounded by some constant, and (iii) that $S \sqsubset \emptyset \ldots \emptyset$ for every sequence $S \neq \emptyset \ldots \emptyset$. This means that after a finite number of steps, this chain must arrive to its last T_k and τ_k with $\langle \tau_k, \mathsf{sw}_T(T_k) \rangle = \emptyset \ldots \emptyset$. This means that $\mathsf{sw}_T(T_k) = \emptyset$, which implies that $T \preceq_{ext} T_k$. We can put $U = T_k$ and Lemma 5 is proven.

Now we can use Lemma 5 to prove Lemma 7. It relates partial runs of $\mathcal{A}^+$ with partial runs of $\mathcal{A}$ by the relation $\preceq_{\alpha^+ \Rightarrow \alpha}$ defined as follows. For two states q and r, $q \preceq_{\alpha^+ \Rightarrow \alpha} r$ iff $q \in \alpha^+ \implies r \in \alpha$. For two paths $\pi, \psi \in Q^+$, $\pi \preceq_{\alpha^+ \Rightarrow \alpha} \psi$ iff $|\pi| = |\psi|$ and for all $1 \leq i \leq |\pi|$, $\pi_i \in \alpha^+ \implies \psi_i \in \alpha$. Finally, for finite trees T and U over Q, we use $T \preceq_{\alpha^+ \Rightarrow \alpha} U$ to denote that $branches(T) \preceq^{\forall\exists}_{\alpha^+ \Rightarrow \alpha} branches(U)$.

LEMMA 7. *For any partial run T of $\mathcal{A}^+$ on $w \in \Sigma^\omega$, there exists a partial run U of $\mathcal{A}$ on w such that $root(T) \preceq_B root(U)$ and $T \preceq_{\alpha^+ \Rightarrow \alpha} U$.*

The proof of Lemma 7 is done by induction on the structure of T, where the induction step employs Lemma 5 (which bridges the gap between $\mathcal{A}^+$ and $\mathcal{A}$ by showing that there is a partial run of $\mathcal{A}$ strongly covering T even when the power of its leaves is extended by transitions of some $\preceq_M$-smaller states). With Lemma 7 in hand, we can prove that for each accepting run of $\mathcal{A}^+$ on a word w, there is an accepting run of $\mathcal{A}$ on w. This requires to carry Lemma 7 from finite partial runs to full infinite runs[¶]. This results in Theorem 8, which together with the fact that $\mathcal{L}(\mathcal{A}/\equiv) \subseteq \mathcal{L}(\mathcal{A}^+)$ immediately gives Corollary 9.

[¶]For an accepting run T of $\mathcal{A}^+$ on a word w, Lemma 7 gives us for every $k \in \mathbb{N}$ and a prefix of T of the height k a partial run of U of the same height that is more accepting. From the infinite set of partial runs of $\mathcal{A}$ obtained this way, we can construct an accepting run of $\mathcal{A}$ on w. The details may be found in [2] and in [2].

THEOREM 8. $\mathcal{L}(\mathcal{A}^+) = \mathcal{L}(\mathcal{A})$.

COROLLARY 9. *Quotienting with mediated equivalence preserves the language.*

5 Algorithm for Computing Mediated Preorder

In this section, we describe an algorithm for computing mediated preorder on an ABA $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$. We first explain how to compute the maximal forward simulation $\preceq_F$ and backward simulation $\preceq_B$ of $\mathcal{A}$. Both $\preceq_F$ and $\preceq_B$ will be used as the input parameters for computing the mediated preorder $\preceq_M$. In the rest of the section, we will fix $\mathcal{A}$ as the input ABA, use n for the number of states in $\mathcal{A}$, and use m for the number of transitions in $\mathcal{A}$.

Forward Simulation. The algorithm for computing maximal forward simulation $\preceq_F$ on $\mathcal{A}$ can be found in Fritz and Wilke's work [5] (it is called direct simulation in their paper). They reduce the problem of computing maximal forward simulation to a simulation game. Although Fritz and Wilke use a slightly different definition of ABA, it is easy to translate $\mathcal{A}$ to an ABA under their definition with $O(n+m)$ states and $O(nm)$ transitions and then use their algorithm to compute $\preceq_F$. The time complexity of the above procedure is $O(nm^2)$.

Removing Ambiguity. As shown in Section 4.1, $\mathcal{A}$ needs to be $\preceq_F$-unambiguous for mediated minimization. Here we describe how to modify $\mathcal{A}$ to make it not $\preceq_F$-ambiguous. The modification does not change the the language of $\mathcal{A}$ and also the forward simulation relation $\preceq_F$, therefore we do not need to recompute forward simulation again for the modified automaton.

Here we describe the ambiguity removal procedure. For every transition $p \xrightarrow{a} P$ with $P = \{p_1, \ldots, p_k\}$ and for each $i \in \{1, \ldots, k\}$, we check if there exists some $i < j \leq k$ such that $p_j \preceq_F p_i$. If there is one, remove p_i from P. This procedure has time complexity $O(n^2 m)$.

Backward Simulation. We now show how to translate the problem of computing maximal backward simulation to a problem of computing maximal simulation on a labeled transition system.

Computing Simulation on Labeled Transition Systems. Let $T = (S, \mathcal{L}, \rightarrow)$ be a finite *labeled transition system (LTS)*, where S is a finite set of states, $\mathcal{L}$ is a finite set of labels, and $\rightarrow \subseteq S \times \mathcal{L} \times S$ is a transition relation. A *simulation* on T is a binary relation $\preceq_L$ on S such that if $q \preceq_L r$ and $(q, a, q') \in \rightarrow$, then there is an r' with $(r, a, r') \in \rightarrow$ and $q' \preceq_L r'$.

Here we describe the problem of computing the maximal simulation on an LTS. Given an LTS $T = (S, \mathcal{L}, \rightarrow)$ and an *initial* preorder $I \subseteq S \times S$, the task is to find out the unique maximal simulation on T included in I. An algorithm for computing maximal simulation $\preceq^I$ on the LTS T included in I with time complexity $O(|\mathcal{L}|.|S|^2 + |S|.|\rightarrow|)$ and space complexity $O(|\mathcal{L}|.|S|^2)$ can be found in [1].

Computing Backward Simulation via a Reduction to LTS. The problem of computing the maximal backward simulation on $\mathcal{A}$ can be reduced to the problem of computing simulation on an LTS. In order to simplify the explanation of the reduction, we first make the following definition. An *environment* is a tuple of the form $(p, a, P \setminus \{p'\})$ obtained by removing a state $p' \in P$ from the transition $p \xrightarrow{a} P$ of $\mathcal{A}$. Intuitively, an environment records the neighbors of the removed state p' in the transition $p \xrightarrow{a} P$. We denote the set of all environments of $\mathcal{A}$ by $Env(A)$. Formally, we define the LTS $A^{\odot} = (Q^{\odot}, \Sigma, \Delta^{\odot})$ as follows:

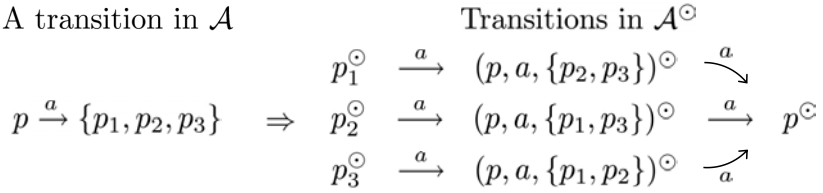


Figure 3: An example of the reduction from an ABA transition to LTS transitions

- $Q^{\odot} = \{q^{\odot} \mid q \in Q\} \cup \{(p, a, P)^{\odot} \mid (p, a, P) \in Env(A)\}$.
- $\Delta^{\odot} = \{(p, a, P \setminus \{p'\})^{\odot} \xrightarrow{a} p^{\odot}, p'^{\odot} \xrightarrow{a} (p, a, P \setminus \{p'\})^{\odot} \mid P \in \delta(p, a), p' \in P\}$.

An example of the reduction is given in Figure 3. The goal of this reduction is to obtain a simulation relation on $A^{\odot}$ with the following property: $p^{\odot}$ is simulated by $q^{\odot}$ in $A^{\odot}$ iff $p \preceq_B q$ in $\mathcal{A}$. However, the maximal simulation on $A^{\odot}$ is not sufficient to achieve this goal. Some essential conditions for backward simulation (e.g., $p \preceq_B q \implies p \preceq_\alpha q$) are missing in $A^{\odot}$. This can be fixed by defining a proper initial preorder I.

Formally, we define $I = \{(q_1^{\odot}, q_2^{\odot}) \mid q_1 \preceq_\iota q_2 \wedge q_1 \preceq_\alpha q_2\} \cup \{((p, a, P)^{\odot}, (r, a, R)^{\odot}) \mid P \preceq_F^{\forall\exists} R\}$. Observe that I is a preorder. Recall that according to the definition of the backward simulation, $p \preceq_B r$ implies that (1) $p \preceq_\iota r$, (2) $p \preceq_\alpha r$, and (3) for all transitions $q \xrightarrow{a} P \cup \{p\}, p \notin P$, there exists a transition $s \xrightarrow{a} R \cup \{r\}, r \notin R$ such that $q \preceq_B s$ and $P \preceq_F^{\forall\exists} R$. The set $\{(q_1^{\odot}, q_2^{\odot}) \mid q_1 \preceq_\iota q_2 \wedge q_1 \preceq_\alpha q_2\}$ encodes the conditions (1) and (2) required by the backward simulation, while the set $\{((p, a, P)^{\odot}, (r, a, R)^{\odot}) \mid P \preceq_F^{\forall\exists} R\}$ encodes the condition (3). A simulation relation $\preceq^I$ can be computed using the aforementioned procedure with LTS $A^{\odot}$ and the *initial* preorder I. The following theorems shows the correctness and complexity of computing backward simulation.

THEOREM 10. *For all $q, r \in Q$, we have $q \preceq_B r$ iff $q^{\odot} \preceq^I r^{\odot}$.*

THEOREM 11. *Computing backward simulation has both time and space complexity $O(nm^3)$.*

The complexity comes from three parts of the procedure: (1) compiling $\mathcal{A}$ into its corresponding LTS $A^{\odot}$, (2) computing the initial preorder I, and (3) running the algorithm for computing the LTS simulation relation. The LTS $A^{\odot}$ has at most $nm+n$ states and $2nm$ transitions. It follows that Part (3) has time complexity $O(|\Sigma|n^2m^2)$ and space complexity $O(|\Sigma|n^2m^2)$. In [2], we show that among the three parts, Part (3) has the highest time[‖] and space complexity and therefore computing backward simulation also has time complexity $O(|\Sigma|n^2m^2)$ and space complexity $O(|\Sigma|n^2m^2)$. Under our definition of ABA, every state has at least one outgoing transition for each symbol in Σ. It follows that $m \geq |\Sigma|n$. Therefore, we can also say that the procedure for computing maximal backward simulation has time complexity $O(nm^3)$ and space complexity $O(nm^3)$.

Mediated Preorder. Here we explain how to compute the mediated preorder $\preceq_M$ of $\mathcal{A}$ from $\preceq_F$ and $\preceq_B$. It is proved in [1] that $\preceq_M$ equals the maximal relation $R \subseteq \preceq_F \circ \preceq_B^{-1}$ satisfying $x \; R \; y \preceq_F z \implies x \; (\preceq_F \circ \preceq_B^{-1}) \; z$. Based on the result, we can obtain the mediated preorder by the following procedure. Initially, let $\preceq_M = \preceq_F \circ \preceq_B^{-1}$. For all $(p, q) \in \preceq_M$, if there exists some $(q, r) \in \preceq_F$ such that $(p, r) \notin \preceq_F \circ \preceq_B^{-1}$, remove (p, q) from $\preceq_M$. A naive implementation of this simple procedure has time complexity $O(n^3)$.

[‖] In [2] we will describe an efficient algorithm for computing I. It has time complexity $O(n^2m^2)$ and space complexity $O(n^2m^2)$.

6 Experimental Results

In this section, we evaluate the performance of mediated minimization by applying it to accelerate the algorithm proposed by Vardi and Kupferman [9] for complementing nondeterministic Büchi automata (NBA). In this algorithm, ABA's are used as intermediate notion for the complementation. To be more specific, the complementation algorithm has two steps: (1) it translates an NBA to an ABA that recognizes its complement language, and (2) it translates the ABA back to an equivalent NBA. The second step is an exponential procedure (exponential in the size of the ABA), hence reducing the size of the ABA before the second step usually pays off.

The experimentation is carried out as follows. Three sets of 100 random NBA's (of $|\Sigma| =$ 2,4, and 8, respectively) are generated by the GOAL [11] tool and then used as inputs of the complementation experiments. We compare results of experiments performed according to the following different options: (1) **Original:** keep the ABA as what it is, (2) **Mediated:** minimizing the ABA with mediated equivalence, and (3) **Forward:** minimizing the ABA with forward equivalence.

For each input NBA, we first translate it to an ABA that recognizes its complement language. The ABA is (1) processed according to one of the options described above and then (2) translated back to an equivalent NBA using an exponential procedure [**]. The results are given in Table 1 and Table 2. Table 1 is an overall comparison between the three different options and Table 2 is a more detailed comparison between **Mediated** and **Forward** minimization.

In Table 1, the columns "NBA" and "Complemented-NBA" are the average statistical data of the input NBA and the complemented NBA. The column "Time(ms)" is the average execution time in milliseconds. "Timeout" is the number of cases that cannot finish within the timeout period (10 min). Note that in the table, the cases that cannot finish within the timeout period are excluded from the average number. From this table, we can see that minimization by mediated equivalence can effectively speed up the complementation and also reduce the size of the complemented NBA's.

	$\|\Sigma\|$	NBA		Complemented-NBA		Time (ms)	Timeout (10 min)
		St.	Tr.	St.	Tr.		
Original	2	2.5	3.3	13.9	52.75	5500.9	0
Mediated				6.68	34.02	524.7	0
Forward				9.45	55.25	5443.7	1
Original	4	3.3	6.0	46.4	348.5	9298.6	6
Mediated				20.42	235.5	1985.4	6
Forward				26.88	325.6	1900.6	7
Original	8	4.7	11.9	127.1.3	1723.4	33429.4	24
Mediated				57.63	1738.3	12930.6	21
Forward				81.23	2349.2	22734.2	24

Table 1: Combining minimization with complementation.

In Table 2, we compare the performance between **Mediated** and **Forward** minimization in detail. The columns "Minimized-ABA" and "Complemented-NBA" are the average difference in the sizes of the ABA after minimization and the complemented BA. From the table, we observe that mediated minimization results in a much better reduction than forward minimization.

	$\|\Sigma\|$	Minimized-ABA		Complemented-NBA	
		St.	Tr.	St.	Tr.
Average Difference	2	33.54%	51.62%	63.3%	235.56%
	4	36.24%	51.44%	89.9%	298.99%
	8	27.94%	40.88%	152.3%	412.7%

Table 2: Comparison: **Mediated** v.s. **Forward**

[**]For the option "Original", we also use the optimization suggested in [9] that only takes consistent subset.

7 Conclusion and Future Work

We combined forward and backward simulation to form a coarser relation called mediated preorder and showed that quotienting wrt. mediated equivalence preserves the language of ABA. Moreover, we developed an efficient algorithm for computing mediated equivalence. Experimental results show that the mediated reduction of ABA significantly outperforms the reduction based on forward simulation. In the future, we would like to extend our experiments to other applications such as LTL to NBA translation. We would like to extend the mediated equivalence by building it on top of even coarser forward simulation relations, e.g., *delayed* or *fair* forward simulation [6]. Also, we intend to study possibilities of using mediated preorder to remove redundant transitions (in a similar way to [10]). We believe that the extensions described above can improve the performance of mediated reduction.

Acknowledgements. The work was supported by the CONNECT project, the UPMARC project, the Czech Science Foundation (projects 102/07/0322, 102/09/H042), the Barrande project MEB 020840, and the Czech institutional project MSM 0021630528.

References

[1] P. Abdulla, A. Bouajjani, L. Holík, L. Kaati, T. Vojnar. Computing Simulations over TA: Efficient Techniques for Reducing TA. In *Proc. of TACAS'08*, LNCS 4963, Springer, 2008.

[2] P. Abdulla, Y.-F. Chen, L. Holík, T. Vojnar. Mediating for Reduction (On Minimizing Alternating Büchi Automata). Tech. Rep. FIT-TR-2009-02, Brno Univ. of Technology, 2009.

[3] P. Abdulla, L. Holík, L. Kaati, and T. Vojnar. A Uniform (Bi-)simulation-based Framework for Reducing Tree Automata. *ENTCS*, 2009(251):27–48, 2009.

[4] A. Farzan, Y.-F. Chen, E. Clarke, Y.-K. Tsay, and B.-Y. Wang. Extending Automated Compositional Verification to the Full Class of Omega-regular Languages. In *Proc. of TACAS'08*, LNCS 4963, Springer, 2008.

[5] C. Fritz and T. Wilke. State Space Reductions for Alternating Büchi Automata: Quotienting by Simulation Equivalences. In *Proc. of FSTTCS'02*, LNCS 2556, Springer, 2002.

[6] C. Fritz and T. Wilke. Simulation Relations for Alternating Büchi Automata. *Theoretical Computer Science*, 338(1-3):275–314, 2005.

[7] P. Gastin and D. Oddoux. Fast LTL to Büchi Automata Translations. In *Proc. of CAV'01*, LNCS 2102, Springer, 2001.

[8] S. Gurumurthy, O. Kupferman, F. Somenzi, and M. Vardi. On Complementing Nondeterministic Büchi Automata. In *Proc. of CHARME'03*, LNCS 2860, Springer, 2003.

[9] O. Kupferman and M. Vardi. Weak Alternating Automata Are Not That Weak. *ACM Transactions on Computational Logic*, 2(3):408–429, 2001.

[10] F. Somenzi and R. Bloem. Efficient Büchi Automata from LTL Formulae. In *Proc. of CAV'00*, LNCS 1855, Springer, 2000.

[11] Y.K. Tsay, Y.F. Chen, M.H. Tsai, K.N. Wu, W.C. Chan. GOAL: A Graphical Tool for Manipulating Büchi Automata & Temp. Formulae. *TACAS'07*, LNCS 4424, Springer, 2007.

[12] M. Vardi. Automata-theoretic Model Checking Revisited. In *Proc. of VMCAI'07*, LNCS 4349, Springer, 2007.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Algorithms for Message Ferrying on Mobile ad hoc Networks

Mostafa Ammar[1], Deeparnab Chakrabarty[2],Atish Das Sarma[1], Subrahmanyam Kalyanasundaram[1], Richard J. Lipton[1]

[1] Georgia Institute of Technology
{ammar, atish, subruk, rjl}@cc.gatech.edu

[2] University of Waterloo
deepc@math.uwaterloo.ca

ABSTRACT. Message Ferrying is a mobility assisted technique for working around the disconnectedness and sparsity of Mobile ad hoc networks. One of the important questions which arise in this context is to determine the routing of the ferry, so as to minimize the buffers used to store data at the nodes in the network. We introduce a simple model to capture the ferry routing problem. We characterize *stable* solutions of the system and provide efficient approximation algorithms for the MIN-MAX BUFFER PROBLEM for the case when the nodes are on hierarchically separated metric spaces.

1 Introduction

Message Ferrying is a new approach developed to assist communication in Mobile ad-hoc networks [6, 15, 16, 17, 18]. Mobile ad-hoc networks are typically deployed with limited infrastructure. Moreover, due to various conditions like limited radio range, physical obstacles or inclement weather, some nodes in the network might not be able to communicate with others. This could result in a disconnected network. In such situations, a typical network protocol might not yield good results. Message Ferrying is an approach which works around such problems. The message ferrying technique makes use of mobile nodes, called "ferries", which are able to collect and transport data from one node to another. Message ferries move around the deployed area according to known routes and communicate with other nodes they meet. By using ferries as relays, nodes can communicate asynchronously with other nodes that are disconnected.

The Message Ferrying scheme raises many theoretical questions that are currently open. For example, Zhao et.al [17] have developed *ad hoc* codes that decide how the ferries should move. While these codes appear to perform well in simulations there are no bounds on the performance of their heuristic methods. The data at the nodes has to be locally stored in buffers till it can be passed on to the ferries. In this paper, we look at the buffer optimization problem for the nodes in the network. We devise routing schemes for the ferries so that the maximum buffer utilization at any node is minimized.

We do the following in this paper.

- Formalize models for the Message Ferry routing problems.
- State exact conditions for the *Stability Problem* of ensuring finite buffers.

© Ammar, Chakrabarty, Das Sarma, Kalyanasundaram, Lipton; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 13–24
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2303

- Devise approximation algorithms for buffer optimization.

We feel that the main contribution of this work is in the creation of the precise models for the Message Ferrying problem. We feel that similarly motivated problems, stemming from direct practical applications could be important in the future. The algorithms and techniques in this paper are fairly preliminary; these are just the first steps towards understanding the Message Ferrying problem.

1.1 Model and Problem Statement

We model the problem in the following manner. We look at all the connected components of the network. Since the network can communicate within a connected component using traditional protocols, we can model each connected component as a node. The connected components are modeled as nodes numbered $[n]$ in a metric space. The metric space induces a distance $d(\cdot,\cdot)$ on the nodes. The ferries are assumed to be devices with infinite storage capacity traveling across the space at unit speed. We assume that node i generates data at a rate r_i which is to be passed on to the ferries (the data can then in turn be transferred from the ferry to other nodes). The rate of data transfer from the nodes to the ferries is given by r_F. Unless otherwise stated, we shall assume that there is only one ferry. We wish to provide message ferrying schemes which are *optimal*.

A Message Ferrying scheme is an (infinite) sequence σ of tuples (i,t), where i denotes the node visited and t the time spent exchanging data with i in this visit. We need to look at the case where the ferry can read data at an infinite speed, i.e. $r_F = \infty$, separately. In this case, it is clearly undesirable to spend any time at a node, as all the data is read instantaneously. Then all we need to specify is a sequence of nodes. Although the sequence could be infinite, we prove that periodically repeating sequences of finite periods suffice for our purposes.

Any viable Message Ferrying scheme would need to be optimized over a large number of disparate parameters like delay minimization and packet loss. Currently, ad hoc and complex measures are used for performance evaluation [7, 18]. We propose two concrete measures and present results for the same. Our first measure is the notion of *stability*. A Message Ferrying scheme is called stable, if the maximum amount of data stored at a node at any point of time is bounded. This is clearly desirable as a node can have only a fixed finite buffer.

DEFINITION 1. STABILITY PROBLEM
Given the rates r_i and r_F, and the distance metric d, is there a Message Ferrying scheme such that the required buffers at all the nodes are bounded?

Our second problem is to optimize the maximum buffer size required over all nodes. That is, given that the rates satisfy the stability conditions, we need to find the scheme minimizing the maximum buffer of any node. Later, we shall see (theorem 20) that the stability criterion for the problem with multiple ferries reduces to that with a single ferry. So it makes sense to look at the optimization problem when only one ferry is involved.

DEFINITION 2. MIN-MAX BUFFER PROBLEM (MMBP)
Given the data rates r_i and ferry rate r_F, and the distance metric d, find the order of ferry visits so as to minimize the maximum buffer required at any node.

As we shall see in Section 3, the above problem turns out to be as hard as solving TSP on the same metric. The MMBP problem is thus a variant of the TSP problem where the objective is not to optimize the length of the tour, rather minimize a sort of weighted delay. Variants of TSP [1, 13], have been studied extensively in the past, although none imply anything about the problem we study.

In section 2, we characterize the necessary and sufficient conditions for stability of a solution. In section 3, we first prove the NP hardness of MMBP problem. We subsequently obtain approximation algorithms for MMBP under restricted metric spaces. In particular, we give a constant factor approximation algorithm for *hierarchically well separated trees* (HSTs) of constant height, and, a $\frac{4}{3}$-factor approximation ratio for the uniform metric case. We extend this to an $O(n)$-factor algorithm for HSTs of height $O(\log n)$. Notice that even though n may be large, this approximation ratio holds for arbitrarily large rates r_i of data generation at nodes as well. We look at some simple extensions to the MMBP problem in section 4. In section 5, we conclude with some remarks and open problems suggesting specific future directions of work.

2 Characterization of stable instances

In this section, we give necessary and sufficient conditions for the existence of stable solutions. We consider the case when there is only one ferry, the node data rates are r_i, ferry rate is r_F and the data is only sent from nodes to the ferry. We then use this to obtain results for the general case.

For the case mentioned above, note that $\sum_{i=1}^{n} r_i < r_F$ is a necessary condition for a stable solution if any of the distances are non-zero. Otherwise the total rate of generation of data in the nodes exceeds the rate at which it can be read by the ferry. In the next theorem, we show that this necessary condition is also sufficient.

THEOREM 3. *Given rates r_i for the nodes, ferry rate r_F, any distances d, a stable solution exists if $\sum r_i < r_F$.*

PROOF. Consider the sequence that visits nodes in order 1 to n, spending time $t_1, t_2, \ldots, t_n$ at them respectively and repeats. Suppose it takes time T to travel from 1 to n in that order.

Now notice that the following is enough for stability: for every node j, the amount of data consumed by the ferry in one visit must be at least the amount that is generated between two visits of the ferry to the node j. That is, for every j we have

$$r_F \cdot t_j \geq r_j \cdot (T + \sum_{i=1}^{n} t_i)$$

Adding these equations over all j we get

$$r_F \cdot \sum_{j=1}^{n} t_j \geq (\sum_{j=1}^{n} r_j) \cdot (T + \sum_{i=1}^{n} t_i)$$

When $r_F = \sum_{j=1}^{n} r_j + \varepsilon$, for some $\varepsilon > 0$, it is easy to check that $t_i = \frac{T}{\varepsilon} r_i$ satisfies the inequalities, implying a stable solution. ∎

3 Min-max Buffer Problem (MMBP)

In this section, we look at the general min-max buffer problem. Throughout this section, we assume that the instance is stable. Also, hereafter, we would be assuming that there is only one ferry node. A solution is called *periodic* if the nodes are visited in a periodic pattern (note that this pattern could have some nodes occurring more than once). In the following proposition, we show that we could look for periodic optimal solutions, since they are as good as optimal solutions, in case of rational rates.

PROPOSITION 4. *For any instance of the* MMBP *with rational rates and distances, there exists a periodic Message Ferrying scheme which is optimal.*

PROOF. Suppose there is an optimal aperiodic solution with maximum buffer B. By hypothesis, the rates and distances are rational. This implies that the optimal solution is rational, and when the ferry reaches a node, the buffer state is rational. Scaling the states to be integral, and recalculating B, each buffer can be one of $0, 1, 2, \ldots, B$ at any given point of time. If there are n nodes, there can be at most $(B+1)^n$ possible buffer states. There are n nodes, we can consider a combined notion of states $S = (\bar{B}, i)$, where $\bar{B}$ is a vector denoting the buffer state across all nodes and i denotes the node visited. So the optimal aperiodic solution returns to at least one of these states S more than once. Let us say that the repeated state is $S^* = (\bar{B}^*, i^*)$. Consider a new periodic solution where this subsequence (between two repetitions of S^*) is repeated indefinitely. Since the same visits are conducted between the two visits to S^*, upon returning to i^*, the buffers have again come back to $\bar{B}^*$. Since the buffers never overflowed in the original aperiodic sequence, they do not overflow in this repeated sequence. This is because we go through the states which were all part of the original aperiodic sequence. Thus we have a periodic sequence which is optimal. ∎

Henceforth, our solutions will be a sequence that is repeated periodically. The following proposition shows the relation of MMBP to the TSP if all the data production rates are identical. Note that this is not true in general. When the rates are different, the solution given by the TSP can be arbitrarily bad for the message ferrying problem.

PROPOSITION 5. *For any underlying metric $d(\cdot,\cdot)$, if the rates of all nodes are equal, i.e., $r_i = 1$, for all i, and the rate of the ferry $r_F = \infty$, then finding the optimal solution to the* MMBP *is the sequence generated by the optimal Traveling Salesman Problem (TSP) tour, and hence NP-hard.*

PROOF. Recall that since $r_F = \infty$, we need to only specify the routing order, there is no need to mention wait times at each node. It is enough to show that the optimal sequence for the MMBP must be generated by a tour. Since all rates are the same, the maximum buffer for the sequence generated by a tour is proportional to the cost of the tour. This implies that the optimal ferry route for the MMBP is the optimal TSP tour.

Assume that the optimal sequence is not a tour. Let σ be the sequence of the optimal solution. Let us relabel the nodes according to the order that we see them in the optimal

solution. By the choice of the labeling, the last node to be visited is n. Since σ is not the optimal TSP tour, n will be seen for the first time after the ferry has traveled a distance greater than the cost of the optimal TSP tour. Since every node has the same rate, the buffer of n is the largest of all the nodes till we visit n for the first time. Consider a solution τ, where the ferry visits all the nodes repeatedly as in the optimal TSP solution. The maximum buffer of n in τ is less than the buffer of n in σ. Moreover, the maximum buffer of every node in τ is the same. Thus τ has a strictly lesser maximum buffer than the route σ. ∎

The above proposition states that solving MMBP is at least as hard as solving TSP on the same underlying metric. Papadimitriou and Yannakakis in [14] prove that the TSP is NP hard even when the distances of the graph are restricted to 1 and 2. This implies that the MMBP is NP hard, even for the case when the distances are restricted to 1 and 2.

In the next two sections, we investigate approximation algorithms when the rate of the ferry is infinite.

3.1 Uniform Metric Case

Here we look at the uniform metric case, where the distance between all nodes are the same. That is, $d(i, j) = 1$ for $i \neq j$. The nodes have rates r_i and we have one ferry, with rate $r_F = \infty$. Once again, recall that since the ferry rate is infinite, we just need to mention the next node to be visited and there is no need to specify wait times. For this case we prove the following theorem.

THEOREM 6. *There is a $\frac{4}{3}$-factor approximation algorithm for* MIN-MAX BUFFER PROBLEM *in the case when the metric is uniform, and the ferry rate $r_F = \infty$.*

The algorithm outline is as follows. Given a guess of the max-buffer B, the algorithm checks *approximate* feasibility of B. That is, the algorithm rejects B only if it is infeasible, otherwise it returns a solution with a max-buffer guarantee of $\frac{4}{3}B$. The $\frac{4}{3}$-approximation follows from a binary search on the possible values of B.

Let σ be any (infinite) feasible sequence of the node visits with max-buffer B. Each node $i \in [n]$ must be visited once in every B/r_i steps. If $d(i)$ denotes the maximum distance between two consecutive appearances of i in σ, we must have $d(i) \leq \lfloor B/r_i \rfloor$. Hence the feasibility solution for the uniform metric case reduces to the following combinatorial problem, called the *pinwheel scheduling problem*. Let us set $m_i = \lfloor B/r_i \rfloor$.

> PINWHEEL SCHEDULING PROBLEM:
> Given integers $m_1 \leq \cdots \leq m_n$, is there a (infinite) sequence σ of $[n]$, such that, for each $1 \leq i \leq n$, the maximum distance between any two consecutive appearances of i is at most m_i? If it does, we call $(m_1, m_2, \cdots, m_n)$ feasible.

This scheduling problem is of independent interest and has been studied previously [3, 4, 9, 10, 11, 12]. Here are some observations about this problem.

PROPOSITION 7. *An instance of the pinwheel scheduling problem $(m_1, m_2, \cdots, m_n)$ has a feasible solution, only if $\sum_{i=1}^{n} \frac{1}{m_i} \leq 1$.*

PROOF. Consider a snapshot of any feasible σ of length $Z = m_1 m_2 \cdots m_n$. For each i, It must contain at least Z/m_i occurrences of i. Since there are only Z possible slots, we have $Z \geq \sum_{i=1}^{n} Z/m_i$ proving the lemma. ∎

Remarks: Notice that this condition is necessary but not sufficient; consider $(m_1, m_2, m_3) = (2, 3, N)$. In this case, $\sum \frac{1}{m_i}$ is approximately $\frac{5}{6}$ for large N, but there is no sequence that can satisfy this for any finite N.

Let OPT be the optimal maximum buffer value, amongst all feasible routes of the ferry. Notice that the optimal routing solves the sequence feasibility problem for the rates $(OPT/r_1, OPT/r_2, \cdots, OPT/r_n)$. So this sequence is feasible, and so proposition 7 implies the following lemma.

LEMMA 8. *If the nodes have rates $r_1, r_2, \ldots, r_n$, with uniform metric, and $r_F = \infty$, we have*

$$OPT \geq \sum_{i=1}^{n} r_i$$

where OPT is the optimal maximum buffer value.

We now have a direct reduction from the Pinwheel Scheduling problem to the ferry routing problem.

LEMMA 9. *Let $\alpha \geq 1$. If we have an algorithm for the pinwheel scheduling problem for m_i such that $\sum_{i=1}^{n} \frac{1}{m_i} \leq \frac{1}{\alpha}$, then we have an α-approximation algorithm for the MMBP problem, with uniform metric, and $r_F = \infty$.*

PROOF. Given a target buffer B, let $m_i = \lfloor \alpha B / r_i \rfloor$. Note that m_i is the maximum allowed time gap between any two consecutive visits to the node i, if we want to bound the buffer by αB. If $\sum_{i=1}^{n} \frac{1}{m_i} > \frac{1}{\alpha}$, then $\sum_{i=1}^{n} \lfloor B/r_i \rfloor^{-1} > 1$. This by Lemma 7 implies that B is infeasible, and the algorithm rejects it. If not, then the algorithm for pinwheel scheduling returns a feasible sequence for $(m_1, m_2, \cdots, m_n)$. For this sequence, the maximum buffer of any node is at most αB. Thus this is an α-approximation.

The only remaining decision is the choice of B. By lemma 8, we have $OPT \geq \sum_{i=1}^{n} r_i$. Also, we can see that $OPT \leq B_{\max}$ where $B_{\max} = \alpha \sum_{i=1}^{n} r_i + r_{\max}$. This is because if we set $B = B_{\max}$, then the corresponding value $\sum_{i=1}^{n} \frac{1}{m_i} \leq \frac{1}{\alpha}$. So in order to complete the approximation algorithm, we need to do a binary search for B between $\sum_{i=1}^{n} r_i$ to $\alpha \sum_{i=1}^{n} r_i + r_{\max}$. ∎

We can use the above lemma 9, with an approximation algorithm for pinwheel scheduling. Fishburn and Lagarias in [9] gave an algorithm for pinwheel scheduling as long as the following condition is met.

THEOREM 10.[Fishburn, Lagarias] *There exists an algorithm for the pinwheel scheduling problem when $\sum_{i=1}^{n} \frac{1}{m_i} \leq 0.75$.*

Theorem 10 along with lemma 9 gives us the following approximation algorithm.

THEOREM 11. *There is a $\frac{4}{3}$-factor approximation algorithm for the* MIN-MAX BUFFER PROBLEM *in the case when the metric is uniform, and the ferry rate $r_F = \infty$.*

The above theorem straightaway implies a $\frac{4}{3}\frac{D_{max}}{D_{min}}$ factor for general metrics where D_{max} (D_{min}) is the maximum (minimum) distance between two points. In particular, in the case where the distances are 1 and 2, this implies a $\frac{8}{3}$ factor approximation. Note that by Proposition 5 and the paper [14], this instance is already NP-hard.

Theorem 10, together with lemma 9 also implies the following lemma, which is used in the section 3.2.

LEMMA 12. *Given nodes of rate $r_1, \cdots, r_n$ and a distance 1 between each node, there exists a ferry routing with maximum buffer at most $\frac{4}{3}\sum r_i + r_{max}$.*

A Simpler Algorithm for Pinwheel Scheduling

Fishburn and Lagarias' algorithm for the pinwheel scheduling problem is quite involved. The algorithm involves case based analysis for several small sets of problem instances, classifies the small sets and extends it to bigger sets based on the classification. We therefore now give a simpler algorithm for pinwheel scheduling, with a slightly worse bound. Our algorithm works when $\sum_{i=1}^{n} \frac{1}{m_i} \leq 1/2$.

LEMMA 13. *If $\sum_{i=1}^{n} \frac{1}{m_i} \leq 1/2$, then $(m_1, m_2, \cdots, m_n)$ is feasible for pinwheel scheduling.*

PROOF. We prove by induction on n. The base case of $n = 1, m_1 = 2$ is trivial. For $n \geq 2$, we have $\sum_{i=1}^{n} \frac{1}{m_i} \leq 1/2$. Rearranging and dividing we get $\sum_{i=2}^{n} \frac{1}{\bar{m}_2} \leq 1/2$, where

$$\bar{m}_i = \left\lceil 2m_i(\frac{1}{2} - \frac{1}{m_1}) \right\rceil$$

By induction, we get $(\bar{m}_2, \cdots, \bar{m}_n)$ is feasible. Let σ' be the feasible pinwheel scheduling sequence. Obtain σ by putting 1 in σ' every m_1 positions. This increases the distance between two i's to $\bar{m}_i + \left\lceil \frac{\bar{m}_i}{m_1 - 1} \right\rceil$. We would have a feasible sequence for $(m_1, m_2, \cdots, m_n)$ if $\bar{m}_i + \left\lceil \frac{\bar{m}_i}{m_1 - 1} \right\rceil \leq m_i$. The following claim shows that this is indeed the case. ∎

CLAIM 14. *For any integers $1 \leq m_1 \leq m_i$ let $\bar{m}_i = \left\lceil m_i - \frac{2m_i}{m_1} \right\rceil$, we have that*

$$\bar{m}_i + \left\lceil \frac{\bar{m}_i}{m_1 - 1} \right\rceil \leq m_i$$

PROOF. Let $x = \frac{2m_i}{m_1}$ and let $k = \lfloor x \rfloor$. Note $\bar{m}_i = m_i - k$. Thus to prove the claim, it suffices to show $\left\lceil \frac{m_i - k}{m_1 - 1} \right\rceil \leq k$. Since $m_i - k = m_1 x/2 - k$, we have

$$\left\lceil \frac{m_i - k}{m_1 - 1} \right\rceil = \left\lceil \frac{x}{2} + \frac{1}{m_1 - 1}(\frac{x}{2} - k) \right\rceil \leq k$$

because $\frac{x}{2} < \lfloor x \rfloor = k$, for all $x > 1$. ∎

This sufficiency condition is constructive as well. Given $(m_1, m_2, \cdots, m_n)$ such that $\sum_{i=1}^{n} 1/m_i \leq 1/2$, recursively run on $(\bar{m}_2, \bar{m}_3, \cdots, \bar{m}_n)$ and put 1 every m_1 spots in the sequence returned.

In the section 3.2, we give a constant factor algorithm for the metrics induced by hierarchically separated trees of constant depth.

3.2 Metrics induced by HSTs of constant depth

In this section, we generalize the results of the previous section to a greater class of metrics. We have seen that we can get a constant factor approximation algorithm for the uniform metric. In this section, we shall look at metrics induced by hierarchically well separated trees (HST).

DEFINITION 15. *A* Hierarchically well Separated Tree *(HST) is a rooted tree such that any pair of leaves that have the least common ancestor at height i, are separated by a distance of D^{i-1}, where D is a parameter.*

HSTs induce a metric on the nodes of the tree. These metrics have been widely studied [2, 5, 8] in the area of metric embeddings. HSTs are interesting because it is possible to get low-distortion embeddings of general metrics into those induced by HSTs.

In this section, we show a constant factor approximation for metrics induced by HSTs of constant height. Note that the uniform case is an HST of height 1. For the sake of clarity, we first look at the case of height 2. We call these metrics $\{1, D\}$-metrics. (Note that any metric with distances only 1 and D, with $D > 2$, can be thought of as an $\{1, D\}$-metric).

In this case, we can partition the point sets into clusters $P_1, P_2, \cdots, P_t$ with each pair in any cluster being at distance 1, and any two points in different clusters at distance D.

We fix some notation. Let $R_i := \max_{j \in P_i} r_j$ and $S_i := \sum_{j \in P_i} r_j$ be the maximum rate and sum of rates for nodes in P_i. Our algorithm would maintain $B_1, \cdots, B_t$ as the max buffers needed for the various clusters. Note that the max-buffer $B = \max_i B_i$.

We now state lower bounds on OPT for this instance.

LEMMA 16.
1. $OPT \geq \sum_{i=1}^{t} S_i$
2. $OPT \geq \sum_{i=1}^{t} DR_i$

PROOF. Note that if we shrink distances between the nodes and or delete nodes, we could only decrease the optimum buffer value. If the distance D were shrunk to 1, then by Lemma 8 we get $OPT \geq \sum_i r_i = \sum_i S_i$. If we delete all points other than the ones with maximum rate, again by Lemma 8 (recall that we are on an instance where distances are scaled by a factor D), we get $OPT/D \geq \sum_i R_i$. ∎

Suppose σ_i be the sequence corresponding to Lemma 12 in Section 3.1 for the nodes in P_i. This guarantees a max-buffer of $\frac{4}{3}S_i + R_i$. We use this fact to develop the following algorithm for the $\{1, D\}$ case. We run the uniform metric algorithm at two separate levels for this case.

Algorithm $\{1, D\}$
1. Visit each cluster P_i once in every window of k_i clusters visited. (k_i will be determined later)
2. When at P_i, run D steps of σ_i starting from the point where it was when it last left P_i.

What this algorithm essentially does is to spend *some* time in each cluster of nodes (here a cluster is a set of nodes with pairwise distances 1). Across the clusters, the algorithm simulates the uniform metric algorithm, where the rate of data generation of a cluster is simply the sum of rates of data generation of the nodes in the cluster. Further, whenever the algorithm spends time inside a given cluster, the algorithm again simulates the uniform metric algorithm within it. Notice, however, that the algorithm may not necessarily be able to perform an entire loop over all nodes within a cluster in one visit. Therefore, it resumes the optimal algorithm within the cluster once it returns to the cluster the next time.

THEOREM 17. *The above algorithm achieves a constant factor approximation for the* MIN-MAX BUFFER PROBLEM *on* $\{1, D\}$ *metrics.*

PROOF. We argue cluster by cluster. We have two cases.
Case 1: Every node in P_i is visited after at least D steps in σ_i. Since for each D steps of σ_i, the algorithm spends $2k_iD$ time outside P_i (k_iD for traveling across clusters and D in each of the k_i clusters), the time between two consecutive occurrences of the point is increased by a factor of at most $2k_iD/D = 2k_i$. Thus we have $B_i \leq 2k_i(\frac{4}{3}S_i + R_i) \leq \frac{14}{3}k_iS_i$ from the lemma 12.
Case 2: There is a node in P_i which is visited with a gap strictly less than D in σ_i. This implies that it is visited every time we visit the cluster, and thus its buffer is at most $R_i(2k_iD)$.

The two cases give $B_i \leq \max(\frac{14}{3}k_iS_i, 2R_ik_iD) \leq (\frac{14}{3}S_i + 2R_iD)k_i$. By choosing $k_i = \left\lceil \frac{\frac{4}{3}\sum_{i=1}^{r}(2R_iD+\frac{14}{3}S_i)}{2R_iD+\frac{14}{3}S_i} \right\rceil$, we get $B_i \leq \frac{4}{3}\sum_{i=1}^{r}(2R_iD + \frac{14}{3}S_i) \leq 9 \cdot OPT$ from Lemma 16.

We complete the proof by noting that a visiting sequence for the clusters for these k_i's can be achieved since $\sum_{i=1}^{t} 1/k_i \leq 0.75$ and we are done by Lemma 8. ∎

THEOREM 18. *There is a constant factor approximation to the MMBP for HSTs of constant height.*

PROOF. Assume that we have a C-factor approximation algorithm to an HST of height k; let this algorithm be A_C. Now consider an HST of height $(k+1)$, say it has t subtrees of height k. We claim that the analogous extension to $Algorithm\{1, D\}$ works here:
1. Visit the points in subtree i, P_i, in every window of k_i subtrees visited among the t subtrees.
2. When at P_i, run D^k steps of algorithm A_C from the point where it was when it last left P_i.

Running through a similar analysis as in $Algorithm\{1, D\}$, we see that with an increase of 1 in the height of the HST, the approximation ratio increases by a factor at most 7. Thus we get an approximation factor of $\frac{4}{3} \cdot 7^k$ where the height of the HST is k. ∎

Remarks: By the above algorithm, we achieve an $O(n)$-factor approximation ratio on an HST of height $O(\log n)$.

Remarks: Standard metric embedding results of Bartal [2] and Fakcheroenphol et al. [8] give a probabilistic embedding of HSTs of height $\log n$ into arbitrary metrics with an expected distortion of $O(\log n)$. This has been used in obtaining approximation algorithms for various problems like the buy-at-bulk network design, metric labeling, etc. which solve the problem on HST instances and extend it to general metrics via results of [8].

Unfortunately these results do not help us guarantee any approximation on the MMBP problem for general metrics as we deal with *maximum* buffers and the expectation of the maximum buffer can be much larger than the maximum of the expected buffer sizes.

4 Extensions

In this section, we show simple extensions of the stability conditions for data exchange, and for the case when there are multiple ferry nodes collecting data.

4.1 Data Exchange Problem

Suppose a node i generates data at a rate a_i and the ferry generates data, to be passed on to the node i, at a rate b_i. The following lemma follows easily from Theorem 3.

LEMMA 19. *If the ferry can only receive or send data at a time, a stable Message Ferrying scheme exists if and only if* $\sum a_i + b_i < r_F$. *If the ferry can receive and send simultaneously, a stable Message Ferrying scheme exists if and only if* $\sum \max(a_i, b_i) < r_F$.

Note that $\max(\sum a_i, \sum b_i) < r_F$ is not a sufficient condition for the simultaneous case of the above lemma. A simple example is two nodes, with $a_1 = b_2 = 0.8, a_2 = b_1 = 0.1, r_F = 1$.

Consider the stability problem in a situation where the ferry (or ferries) have bounded buffers. Suppose each of the ferry has a limited buffer size. Notice that in such a case, we cannot get a similar theorem like theorem 3. When one limits the ferry buffer size, the stability is not just a function of the rates of the nodes. The stability would depend on the topology of the nodes as well. To see this, consider the following problem: there is one ferry and two nodes, and the ferry has to transport data from one node to the other. The rates r_1, r_2 correspond to the transfer rates at the nodes. Let $r_1 + r_2$ be infinitesimally close but still less than r_F. By theorem 3, we would still be stable if the ferry had no bounds on its buffer. Theorem 3 achieves this by making the ferry stay very long at either node, and then moving only occasionally. But a bound of the ferry buffer size would force the ferry to move earlier, and thus risk losing data or being not stable. Thus a bound on the ferry's buffer size would mean that the stability depends on the topology of the problem.

4.2 Multiple Ferries

Suppose there are m ferries each with the same ferry transfer rate, r_F. Also assume that at any node only one ferry can operate at a time. We have the following necessary and sufficient conditions for this case.

THEOREM 20. *Consider the case when there are m ferries each with the same ferry transfer rate, r_F. Also, at any node only one ferry can operate at a time. The necessary and sufficient condition for stability is $\sum r_i < mr_F$ and $r_i \leq r_F$ for all i.*

PROOF. One of the necessary conditions, $\sum r_i < mr_F$ is immediate. If $r_i > r_F$ for any node, notice that at most one ferry can serve that node at any time. So we would require unbounded buffer at that node. So $r_i \leq r_F$ is a necessary condition.

To see why the conditions are sufficient, let $s_i = \frac{r_i}{m}$, then the first condition is equivalent to $\sum s_i < r_F$. One ferry could solve this instance with rates s_i. Consider a stable cyclic solution for this instance with one ferry, with rate r_F. Let this solution take time T for each cycle period (inclusive of waiting times at each node). Start the m ferries in the given solution at points $0, \frac{T}{m}, 2\frac{T}{m}, \ldots, (n-1)\frac{T}{m}$. Pretend that each of these ferries is solving an instance with rates s_i. This is a stable solution, provided that the ferries never run into each other at any node. But notice that $\max s_i = \frac{1}{m} \max r_i \leq \frac{r_F}{m}$. So the ferry spends time at most T/m at each node for the instance with rates s_i. Since the ferries are equally spaced in time, no two ferries would have to serve the same node at a given time. ∎

5 Conclusions

In this paper, we formalize the Message Ferrying model for Mobile ad hoc networks. We characterize stability conditions for problem instances on a node distribution and efficient approximation algorithms for a restricted class of metric node distributions. An interesting question is to extend our results to the more realistic and interesting case of finite ferry rate. Another direction is to generalize the algorithm for a larger class of metrics on which the nodes are distributed. Also, while the ferry problem seems intriguingly similar to the TSP, there seems to be no formal connection. Is there a way to translate the TSP approximation algorithms (on generic metric spaces) to the ferry problem? Also, in this work, we assumed the rate at which a node is producing data to be constant; however this is could be an unreasonable assumption, depending on the application. Is there a natural way to model and solve the more general case?

Acknowledgement. We thank Shiva Kintali for helpful initial discussions.

References

[1] Nikhil Bansal, Avrim Blum, Shuchi Chawla and Adam Meyerson, "Approximation algorithms for deadline-TSP and vehicle routing with time-windows", *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, 166–174, 2004.

[2] Yair Bartal, "Probabilistic approximation of metric spaces and its algorithmic applications", *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, 184–193, 1996.

[3] Mee Yee Chan and Francis Y. L. Chin, "General Schedulers for the Pinwheel Problem Based on Double-Integer Reduction", *IEEE Transactions on Computers*, Volume 41, Issue 6, 755–768, June 1992.

[4] Mee Yee Chan and Francis Y. L. Chin, "Schedulers for larger class of pinwheel instances", *Algorithmica*, Volume 9, Issue 5, 425–462, May 1993.

[5] Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha and Serge A. Plotkin, "Approximating a Finite Metric by a Small Number of Tree Metrics", *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, 379–388, 1998.

[6] Yang Chen, Jeonghwa Yang, Wenrui Zhao, Mostafa Ammar and Ellen Zegura, "Multicasting in Sparse MANETs Using Message Ferrying", *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC) 2006*, Volume 2, 691–696, April 2006.

[7] Thomas D. Dyer and Rajendra V. Boppana, "A Comparison of TCP Performance over Three Routing Protocols for Mobile Ad Hoc Networks", *Proceedings of the 2nd ACM International Symposium on Mobile ad hoc Networking and Computing*, 56–66, 2001.

[8] Jittat Fakcheroenphol, Kunal Talwar and Satish Rao, "A tight bound on approximating arbitrary metrics by tree metrics", *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, 448–455, 2003.

[9] Peter C. Fishburn, P. C. and Jeffrey C. Lagarias, "Pinwheel scheduling: achievable densities", *Algorithmica*, Volume 34, Issue 1, 14–38, 2002.

[10] Robert Holte, Al Mok, Louis Rosier, Igor Tulchinsky and Donald Varvel, "The pinwheel: A real-time scheduling problem", *Proceedings of the 22nd Hawaii International Conference of System Sciences*, 693–702, January 1989.

[11] Robert Holte, Louis Rosier, Igor Tulchinsky and Donald Varvel, "Pinwheel scheduling with two distinct numbers", *Mathematical Foundations of Computer Science 1989* (*Porpolhk abka-Kozubnik 1989*), *Lecture Notes in Computer Science*, Volume 379, 281–290, 1989, and *Theoretical Computer Science* Volume 100, Issue 1, 105–135, 1992.

[12] Shun-Shii Lin and Kwei-Jay Lin, "A pinwheel scheduler for three distinct numbers with a tight schedulability bound", *Algorithmica*, Volume 19, Issue 4, 411–426, 1997.

[13] Viswanath Nagarajan and R. Ravi, "Minimum vehicle routing with a common deadline", *Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) 2006*, 212–223, 2006.

[14] Christos H. Papadimitriou and Mihalis Yannakakis, "The traveling salesman problem with distances one and two", *Mathematics of Operations Research*, Volume 18, Issue 1, 1–11, February 1993.

[15] Barath Petit, Mostafa Ammar and Richard Fujimoto, "Protocols for Roadside-to-Roadside Data Relaying over Vehicular Networks", *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC) 2006*, Volume 1, 294–299, April 2006.

[16] Jeonghwa Yang, Yang Chen, Mostafa Ammar and Chung-Ki Lee, "Ferry Replacement Protocols in Sparse MANET Message Ferrying Systems", *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC) 2005*, Vol. 4, 2038–2044, March 2005.

[17] Wenrui Zhao, Mostafa Ammar and Ellen Zegura, "A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks", *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, 187–198, May 2004.

[18] Wenrui Zhao, Mostafa Ammar and Ellen Zegura, "Controlling the Mobility of Multiple Data Transport Ferries in a Delay-Tolerant Network", *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2005)*, Volume 2, 1407–1418, 2005.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Arithmetic Circuits and the Hadamard Product of Polynomials

V. Arvind, Pushkar S. Joglekar, Srikanth Srinivasan

Institute of Mathematical Sciences
C.I.T Campus,Chennai 600 113, India
{arvind,pushkar,srikanth}@imsc.res.in

ABSTRACT. Motivated by the Hadamard product of matrices we define the Hadamard product of multivariate polynomials and study its arithmetic circuit and branching program complexity. We also give applications and connections to polynomial identity testing. Our main results are the following.

- We show that noncommutative polynomial identity testing for algebraic branching programs over rationals is complete for the logspace counting class $C_=L$, and over fields of characteristic p the problem is in $Mod_pL/poly$.
- We show an exponential lower bound for expressing the Raz-Yehudayoff polynomial as the Hadamard product of two monotone multilinear polynomials. In contrast the Permanent can be expressed as the Hadamard product of two monotone multilinear formulas of quadratic size.

1 Introduction

In this paper we define the *Hadamard product* of two polynomials f and g in $\mathbb{F}\langle X\rangle$ and study its expressive power and applications to the complexity of arithmetic circuits and algebraic branching programs. We also apply it to give a fairly tight characterization of polynomial identity testing for algebraic branching programs over the field of rationals.

Suppose $X = \{x_1, x_2, \cdots, x_n\}$ is a set of n noncommuting variables. The free monoid X^* consists of all words over these variables. For a field $\mathbb{F}$ let $\mathbb{F}\langle x_1, x_2, \cdots, x_n\rangle$ denote the free noncommutative polynomial ring over $\mathbb{F}$ generated by the variables in X. Thus, the polynomials in this ring are $\mathbb{F}$-linear combinations of words over X. For a given polynomial $f \in \mathbb{F}\langle X\rangle$, let $\mathrm{mon}(f) = \{m \in X^* \mid m \text{ is a nonzero monomial in } f\}$. If $X = \{x_1, x_2, \cdots, x_n\}$ is a set of *n commuting variables* then $\mathbb{F}[X]$ denotes the commutative polynomial ring with coefficients from $\mathbb{F}$.

Motivated by the well-known Hadamard product of matrices (see e.g. [6]) we define the Hadamard product of polynomials.

© V. Arvind, P. S. Joglekar, S. Srinivasan; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 25–36
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2304

DEFINITION 1. *Let $f, g \in \mathbb{F}\langle X\rangle$ where $X = \{x_1, x_2, \cdots, x_n\}$. The* Hadamard product *of f and g, denoted $f \circ g$, is the polynomial $f \circ g = \sum_m a_m b_m m$, where $f = \sum_m a_m m$ and $g = \sum_m b_m m$, where the sums index over monomials m.*

Complexity theory preliminaries We recall some definitions of logspace counting classes from [3]. Let L denote the class of languages accepted by deterministic logspace machines.

GapL is the class of functions $f : \Sigma^* \to \mathbb{Z}$, for which there is a logspace bounded NDTM M such that for each input $x \in \Sigma^*$, we have $f(x) = acc_M(x) - rej_M(x)$, where $acc_M(x)$ and $rej_M(x)$ are the number of accepting and rejecting paths of M on input x, respectively.

A language L is in $C_=L$ if there exists a function $f \in$ GapL such that $x \in L$ if and only if $f(x) = 0$. For a prime p, a language L is in the complexity class Mod_pL if there exists a function $f \in$ GapL such that $x \in L$ if and only if $f(x) = 0 (\text{mod } p)$.

It is shown in [3] that checking if an integer matrix is singular is complete for $C_=L$ with respect to logspace many-one reductions. The same problem is known to be complete for Mod_pL over a field of characteristic p. It is useful to recall that both $C_=L$ and Mod_pL are contained in TC^1 (which, in turn, is contained in NC^2).

An *Algebraic Branching Program* (ABP) [13, 14] over a field $\mathbb{F}$ and variables $x_1, x_2, \cdots, x_n$ is a *layered* directed acyclic graph with one *source* vertex of indegree zero and one *sink* vertex of outdegree zero. Let the layers be numbered $0, 1, \cdots, d$. The source and sink are the unique layer 0 and layer d vertices, respectively. Edges only go from layer i to $i+1$ for each i. Each edge in the ABP is labeled with a linear form over $\mathbb{F}$ in the input variables. Each source to sink path in the ABP computes the product of the linear forms labelling the edges on the path, and the sum of these polynomials over all source to sink paths is the polynomial computed by the ABP. The size of the ABP is the number of vertices.

Main results. We show that the *noncommutative* branching program complexity of the Hadamard product $f \circ g$ is upper bounded by the product of the branching program sizes for f and g.This upper bound is natural because we know from Nisan's seminal work [13] that the algebraic branching program (ABP) complexity $B(f)$ is well characterized by the ranks of its "communication" matrices $M_k(f)$, and the rank of Hadamard product $A \circ B$ of two matrices A and B is upper bounded by the product of their ranks. Our proof is constructive: we give a deterministic logspace algorithm for computing an ABP for $f \circ g$.

We then apply this result to polynomial identity testing. It is shown by Raz and Shpilka [14] that polynomial identity testing of noncommutative ABPs can be done in deterministic polynomial time. A simple divide and conquer algorithm can be easily designed to show that the problem is in deterministic NC^3. What then is the precise complexity of polynomial identity testing for noncommutative ABPs? For noncommutative ABPs over *rationals* we give a tight characterization by showing that the problem is $C_=L$-complete. We prove this result using the result on Hadamard product of ABPs explained above.

For noncommutative ABPs over a finite field of characteristic p, we show that identity test-

ing is in the nonuniform class $\text{Mod}_p\text{L/poly}$ (more precisely, in randomized Mod_pL). Furthermore, the problem turns out to be hard (w.r.t. logspace many-one reductions) for both NL and Mod_pL. Hence, it is not likely to be easy to improve this upper bound unconditionally to Mod_pL (it would imply that NL is contained in Mod_pL). However, under a hardness assumption we can apply standard arguments [4, 12] to derandomize this algorithm and put the problem in Mod_pL.

In Section 4 we consider the Hadamard product for commutative polynomials. We show an exponential lower bound for expressing the Raz-Yehudayoff polynomial [15] as the Hadamard product of two monotone multilinear polynomials. In contrast the Permanent can be expressed as the Hadamard product of two monotone multilinear formulas of quadratic size.

2 The Hadamard Product

Let $f, g \in \mathbb{F}\langle X\rangle$ where $X = \{x_1, x_2, \cdots, x_n\}$. Clearly, $\text{mon}(f \circ g) = \text{mon}(f) \cap \text{mon}(g)$. Thus, the Hadamard product can be seen as an algebraic version of the intersection of formal languages. Our definition of the Hadamard product of polynomials is actually motivated by the well-known Hadamard product $A \circ B$ of two $m \times n$ matrices A and B. We recall the following well-known bound for the rank of the Hadamard product.

PROPOSITION 2. *Let A and B be $m \times n$ matrices over a field $\mathbb{F}$. Then*

$$\text{rank}(A \circ B) \leq \text{rank}(A)\text{rank}(B)$$

It is known from Nisan's work [13] that the ABP complexity $B(f)$ of a polynomial $f \in \mathbb{F}\langle X\rangle$ is closely connected with the ranks of the communication matrices $M_k(f)$, where $M_k(f)$ has its rows indexed by degree k monomials and columns by degree $d-k$ monomials and the $(m, m')^{th}$ entry of $M_k(f)$ is the coefficient of mm' in f. Nisan showed that $B(f) = \sum_k \text{rank}(M_k(f))$. Indeed, Nisan's result and the above proposition easily imply the following bound on the ABP complexity of $f \circ g$.

LEMMA 3. *For $f, g \in \mathbb{F}\langle X\rangle$ we have $B(f \circ g) \leq B(f)B(g)$.*

Proof. By Nisan's result $B(f \circ g) = \sum_k \text{rank}(M_k(f \circ g))$. The above proposition implies

$$\sum_k \text{rank}(M_k(f \circ g)) \leq \sum_k \text{rank}(M_k(f))\text{rank}(M_k(g)) \leq (\sum_k \text{rank}(M_k(f))(\sum_k \text{rank}(M_k(g))),$$

and the claim follows. ∎

We now show an algorithmic version of this upper bound.

THEOREM 4. *Let P and Q be two given ABP's computing polynomials f and g in $\mathbb{F}\langle x_1, x_2, \ldots, x_n\rangle$, respectively. Then there is a deterministic polynomial-time algorithm that will output an ABP R for the polynomial $f \circ g$ such that the size of R is a constant multiple of the product of the sizes of P and Q. (Indeed, R can be computed in deterministic logspace.)*

Proof. Let f_i and g_i denote the i^{th} homogeneous parts of f and g respectively. Then $f = \sum_{i=0}^{d} f_i$ and $g = \sum_{i=0}^{d} g_i$. Since the Hadamard product is distributive over addition and $f_i \circ g_j = 0$ for $i \neq j$ we have $f \circ g = \sum_{i=0}^{d} f_i \circ g_i$. Thus, we can assume that both P and Q are homogeneous ABP's of degree d. Otherwise, we can easily construct an ABP to compute $f_i \circ g_i$ separately for each i and put them together. Note that we can easily compute ABPs for f_i and g_i in logspace given as input the ABPs for f and g.

By allowing parallel edges between nodes of P and Q we can assume that the labels associated with each edge in an ABP is either 0 or αx_i for some variable x_i and scalar $\alpha \in \mathbb{F}$. Let s_1 and s_2 bound the number of nodes in each layer of P and Q respectively. Denote the j^{th} node in layer i by $\langle i, j\rangle$ for ABPs P and Q. Now we describe the construction of the ABP R for computing the polynomial $f \circ g$. Each layer i, $1 \leq i \leq d$ of R will have $s_1 \cdot s_2$ nodes, with node labeled $\langle i, a, b\rangle$ corresponding to the node $\langle i, a\rangle$ of P and the node $\langle i, b\rangle$ of Q. We can assume there is an edge from every node in layer i to every node in layer $i+1$ for both ABPs. For, if there is no such edge we can always include it with label 0.

In the new ABP R we put an edge from $\langle i, a, b\rangle$ to $\langle i+1, c, e\rangle$ with label $\alpha\beta x_t$ if and only if there is an edge from node $\langle i, a\rangle$ to $\langle i+1, c\rangle$ with label αx_t in P and an edge from $\langle i, b\rangle$ to $\langle i+1, e\rangle$ with label βx_t in ABP Q. Let $\langle 0, a, b\rangle$ and $\langle d, c, e\rangle$ denote the source and the sink nodes of ABP R, where $\langle 0, a\rangle, \langle 0, b\rangle$ are the source nodes of P and Q, and $\langle d, c\rangle, \langle d, e\rangle$ are the sink nodes of P and Q respectively. It is easy to see that ABP R can be computed in deterministic logspace. Let $h_{\langle i,a,b\rangle}$ denote the polynomial computed at node $\langle i, a, b\rangle$ of ABP R. Similarly, let $f_{\langle i,a\rangle}$ and $g_{\langle i,b\rangle}$ denote the polynomials computed at node $\langle i, a\rangle$ of P and node $\langle i, b\rangle$ of Q. We can easily check that $h_{\langle i,a,b\rangle} = f_{\langle i,a\rangle} \circ g_{\langle i,b\rangle}$ by an induction argument on the number of layers in the ABPs. It follows from this inductive argument that the ABP R computes the polynomial $f \circ g$ at its sink node. The bound on the size of R also follows easily. ∎

Applying the above theorem we can give a *tight* complexity theoretic upper bound for identity testing of noncommutative ABPs over rationals.

THEOREM 5. *The problem of polynomial identity testing for noncommutative algebraic branching programs over $\mathbb{Q}$ is in NC^2. More precisely, it complete for the logspace counting class $\mathrm{C}_=\mathrm{L}$ under logspace reductions.*

Proof. Let P be the given ABP computing $f \in \mathbb{Q}\langle X\rangle$. We apply the construction of Theorem 4 to compute a polynomial sized ABP R for the Hadamard product $f \circ f$ (i.e. of f with itself). Notice that $f \circ f$ is nonzero iff f is nonzero. Now, we crucially use the fact that $f \circ f$ is a polynomial whose nonzero coefficients are all *positive*. Hence, $f \circ f$ is nonzero iff

it evaluates to nonzero on the all 1's input. The problem thus boils down to checking if R evaluates to nonzero on the all 1's input.

By Theorem 4, the ABP R for polynomial $f \circ f$ is computable in deterministic logspace, given as input an ABP for f. Furthermore, evaluating the ABP R on the all 1's input can be easily converted to iterated integer matrix multiplication (one matrix for each layer of the ABP), and checking if R evaluates to nonzero can be done by checking if a specific entry of the product matrix is nonzero. It is well known that checking if a specific entry of an iterated integer matrix product is zero is in the logspace counting class $\mathrm{C_{=}L}$ (e.g. see [3, 1]). However, $\mathrm{C_{=}L}$ is contained in NC^2, in fact in TC^1.

We now argue the hardness of this problem for $\mathrm{C_{=}L}$. The problem of checking if an integer matrix A is singular is well known to be complete for $\mathrm{C_{=}L}$ under deterministic logspace reductions. The standard GapL algorithm for computing $\det(A)$ [16] can be converted to an ABP P_A which will compute $\det(A)$.* Hence the ABP P_A computes the identically zero polynomial iff A is singular. Putting it all together, it follows that identity testing of noncommutative ABPs over rationals is complete for the class $\mathrm{C_{=}L}$. ■

An iterative matrix product problem Suppose B is a noncommutative ABP computing a homogeneous polynomial in $\mathbb{F}\langle X\rangle$ of degree d, where each edge of the ABP is labeled by a homogeneous linear form in variables from X.

Let n_ℓ denote the number of nodes of B in layer ℓ, $0 \leq \ell \leq d$. For each x_i and layer ℓ, we associate an $n_\ell \times n_{\ell+1}$ matrix $A_{i,\ell}$ where the $(k,j)^{th}$ entry of matrix $A_{i,\ell}$ is the coefficient of x_i in the linear form associated with the (v_k, u_j) edge in the ABP B. Here v_k is the k^{th} node in layer ℓ and u_j the j^{th} node in the layer $\ell+1$. The following claim is easy to see and relates these matrices to the ABP B.

CLAIM 6. *The coefficient of any degree d monomial $x_{i_1}x_{i_2}\cdots x_{i_d}$ in the polynomial computed by the ABP B is the matrix product $A_{i_1,0}A_{i_2,1}\cdots A_{i_d,d-1}$ (which is a scalar since $A_{i_1,0}$ is a row and $A_{i_d,d-1}$ is a column).*

Let i and j be any two nodes in the ABP B. We denote by $B(i,j)$ the algebraic branching program obtained from the ABP B by designating node i in B as the source node and node j as the sink node. Clearly, $B(i,j)$ computes a homogeneous polynomial of degree $b-a$ if i appears in layer a and j in layer b.

For layers a, b, $0 \leq a < b \leq d$ let $t = b - a$ and $P(a,b) = \{A_{s_1,a}A_{s_2,a+1}\ldots A_{s_t,b-1} | 1 \leq s_j \leq n$, for $1 \leq j \leq t\}$. $P(a,b)$ consists of $n_a \times n_b$ matrices. Thus the dimension of the linear space spanned by $P(a,b)$ is bounded by $n_a n_b$. It follows from Claim 6 that the linear span of $P(a,b)$ is the zero space iff the polynomial computed by ABP $B(i,j)$ is identically zero for every $1 \leq i \leq n_a$ and $1 \leq j \leq n_b$.

*Notice that the polynomial computed by the ABP P_A is a constant since P_A has only constants and no variables.

Thus, it suffices to compute a basis for the space spanned by matrices in $P(0,d)$ to check whether the polynomial computed by B is identically zero. We can easily give a deterministic NC^3 algorithm for this problem over any field $\mathbb{F}$: First recursively compute bases M_1 and M_2 for the space spanned by matrices in $P(0,d/2)$ and $P(d/2+1,d)$ respectively. From bases M_1 and M_2 we can compute in deterministic NC^2 a basis M for space spanned by matrices in $P(0,d)$ as follows. We compute the set S of pairwise products of matrices in M_1 and M_2 and then we can compute a maximal linearly independent subset of S in NC^2 (see e.g. [1]). This gives an easy NC^3 algorithm to compute a basis for the linear span of $P(0,d)$. This proves the following.

PROPOSITION 7. *The problem of polynomial identity testing for noncommutative algebraic branching programs over any field (in particular, finite fields $\mathbb{F}$) is in deterministic* NC^3.

Can we give a tight complexity characterization for identity testing of noncommutative ABPs over finite fields? We show that the problem is in nonuniform $\mathrm{Mod}_p\mathrm{L}$ and is hard for $\mathrm{Mod}_p\mathrm{L}$ under logspace reductions. Furthermore, the problem is hard for NL. Hence, it appears difficult to improve the upper bound to uniform $\mathrm{Mod}_p\mathrm{L}$ (as NL is not known to be contained in uniform $\mathrm{Mod}_p\mathrm{L}$).

THEOREM 8. *The problem of polynomial identity testing for noncommutative algebraic branching programs over a finite field $\mathbb{F}$ of characteristic p is in* $\mathrm{Mod}_p\mathrm{L}/\mathrm{poly}$.

Proof. Consider a new ABP B' in which we replace the variables x_i, $1 \le i \le n$ appearing in the linear form associated with an edge from some node in layer l to a node in layer $l+1$ of ABP B by new variable $x_{i,l}$, for layers $l = 0, 1, \ldots, d-1$. Let $g \in \mathbb{F}[X]$ denotes the polynomial computed by ABP B' in *commuting* variables $x_{i,l}, 1 \le i \le n, 1 \le l < d$. It is easy to see that the commutative polynomial $g \in \mathbb{F}[X]$ is identically zero iff the noncommutative polynomial $f \in \mathbb{F}\langle X\rangle$ computed by ABP B is identically zero. Now, we can apply the standard Schwartz-Zippel lemma to check if g is identically zero by substituting random values for the variables $x_{i,l}$ from $\mathbb{F}$ (or a suitable finite extension of $\mathbb{F}$). After substitution of field elements, we are left with an iterated matrix product over a field of characteristic p which can be done in $\mathrm{Mod}_p\mathrm{L}$. This gives us a randomized $\mathrm{Mod}_p\mathrm{L}$ algorithm. By standard amplification it follows that the problem is in $\mathrm{Mod}_p\mathrm{L}/\mathrm{poly}$. ∎

Next we show that identity testing noncommutative ABPs over any field is hard for NL by a reduction from directed graph reachability. Let (G,s,t) be a reachability instance. Without loss of generality, we assume that G is a layered directed acyclic graph. The graph G defines an ABP with source s and sink t as follows: label each edge e in G with a *distinct* variable x_e and for each absent edge put the label 0. The polynomial computed by the ABP is nonzero if and only if there is a directed s-t path in G.

THEOREM 9. *The problem of polynomial identity testing for noncommutative algebraic branching programs over any field is hard for* NL.

3 Hadamard product of noncommutative circuits

Analogous to Theorem 4 we show that $f \circ g$ has small circuits if f has a small circuit and g has a small ABP. For the proof refer to the full version of the paper [2].

THEOREM 10. *Let $f, h \in \mathbb{F}\langle x_1, x_2, \cdots, x_n\rangle$ be given by a degree d circuit C and a degree d ABP P respectively, where $d = O(n^{O(1)})$. Then we can compute in polynomial time a circuit C' that computes $f \circ h$ where the size of C' is polynomially bounded in the sizes of C and P.*

On the other hand, suppose f and g individually have small circuit complexity. Does $f \circ g$ have small circuit complexity? Can we compute such a circuit for $f \circ g$ from circuits for f and g? We first consider these questions for monotone circuits. It is useful to understand the connection between monotone noncommutative circuits and context-free grammars. We recall the following definition.

DEFINITION 11. *We call a context-free grammar $G = (V, T, P, S)$ an* acyclic CFG *if for any nonterminal $A \in V$ there does not exist any derivation of the form $A \Rightarrow^* uAw$, and for each production $A \Rightarrow \beta$ we have $|\beta| \leq 2$.*

The size *size*(G) of an acyclic CFG $G = (V, T, P, S)$ is defined as $|V| + |T| + \text{size}(P)$, where V, T, and P are the sets of variables, terminals, and production rules. We note the following easy proposition that relates acyclic CFGs to monotone noncommutative circuits over X.

PROPOSITION 12. *Let C be a monotone circuit of size s computing a polynomial $f \in \mathbb{Q}\langle X\rangle$. Then there is an acyclic CFG G for mon(f) with* size$(G) = O(s)$. *Conversely, if G is an acyclic CFG of size s computing some finite set $L \subset X^*$ of monomials over X, there exists a monotone circuit of size $O(s)$ that computes a polynomial $\sum_{m \in L} a_m m \in \mathbb{Q}\langle X\rangle$, where the positive integer a_m is the number of derivation trees for m in the grammar G.*

THEOREM 13. *There are monotone circuits C and C' computing polynomials f and g in $\mathbb{Q}\langle X\rangle$ respectively, such that the polynomial $f \circ g$ requires monotone circuits of size exponential in $|X|$,* size(C), *and* size(C').

Proof. Let $X = \{x_1, \cdots, x_n\}$. Define the finite language $L_1 = \{zww^r \mid z, w \in X^*, |z| = |w| = n\}$ and the corresponding polynomial $f = \sum_{m_\alpha \in L_1} m_\alpha$. Similarly let $L_2 = \{ww^rz \mid z, w \in X^*, |z| = |w| = n\}$, and the corresponding polynomial $g = \sum_{m_\alpha \in L_2} m_\alpha$. It is easy to see that there are poly(n) size *unambiguous* acyclic CFGs for L_1 and L_2. Hence, by Proposition 12 there are monotone circuits C_1 and C_2 of size poly(n) such that C_1 computes polynomial f and C_2 computes polynomial g. We first show that the finite language $L_1 \cap L_2$ cannot be generated by any acyclic CFG of size $2^{o(n \lg n)}$. Assume to the contrary that there is an acyclic CFG $G = (V, T, P, S)$ for $L_1 \cap L_2$ of size $2^{o(n \lg n)}$. Notice that $L_1 \cap L_2 = \{t \mid t = ww^rw, w \in X^*, |w| = n\}$.

Consider any derivation tree T' for a word $ww^rw = w_1w_2 \dots w_nw_nw_{n-1} \dots w_2w_1w_1 \dots w_n$. Starting from the root of the binary tree T', we traverse down the tree always picking the child with larger yield. Clearly, there must be a nonterminal $A \in V$ in this path of the derivation tree such that $A \Rightarrow^* u$, $u \in X^*$ and $n \leq |u| < 2n$. Crucially, note that any word that A generates must have same length since every word generated by the grammar G is in $L_1 \cap L_2$ and hence of length $3n$. Let $ww^rw = s_1us_2$ where $|s_1| = k$. As $|u| < 2n$, the string s_1s_2 completely determines the string ww^rw. Hence, the nonterminal A can derive at most one string u. Furthermore, this string u can occur in at most $2n$ positions in a string of length $3n$. Notice that for each position in which u can occur it completely determines a string of the form ww^rw. Therefore, A can participate in the derivation of at most $2n$ strings from $L_1 \cap L_2$. Since there are n^n distinct words in $L_1 \cap L_2$, it follows that there must be at least $\frac{n^n}{2n}$ *distinct* nonterminals in V. This contradicts the size assumption of G.

Since $L_1 \cap L_2$ cannot be generated by any acyclic CFG of size $2^{o(n \log n)}$, it follows from Lemma 12 that the polynomial $f \circ g$ can not be computed by any monotone circuit of $2^{o(n \log n)}$ size. ∎

Theorem 13 shows that the Hadamard product of monotone circuits is more expressive than monotone circuits. It raises the question whether the permanent polynomial can be expressed as the Hadamard product of polynomial-size (or even subexponential size) monotone circuits. We note here that the permanent *can* be easily expressed as the Hadamard product of $O(n^3)$ many monotone circuits (in fact, monotone ABPs).

THEOREM 14. *Suppose there is a deterministic subexponential-time algorithm that takes two circuits as input, computing polynomials f and g in $\mathbb{Q}\langle x_1, \cdots, x_n \rangle$, and outputs a circuit for $f \circ g$. Then either* NEXP *is not in* P/poly *or the Permanent does not have polynomial size noncommutative circuits.*

Proof. Let C_1 be a circuit computing some polynomial $h \in \mathbb{Q}\langle x_1, \dots, x_n \rangle$. By assumption, we can compute a circuit C_2 for $h \circ h$ in subexponential time. Therefore, h is identically zero iff $h \circ h$ is identically zero iff C_2 evaluates to 0 on the all 1's input. We can easily check if C_2 evaluates to 0 on all 1's input by substitution and evaluation. This gives a deterministic subexponential time algorithm for testing if h is identically zero. By the noncommutative analogue of [11], shown in [5], it follows that either NEXP $\not\subset$ P/poly or the Permanent does not have polynomial size noncommutative circuits. ∎

Next, We show that the identity testing problem: given $f, g \in \mathbb{F}\langle X \rangle$ by circuits test if $f \circ g$ is identically zero is coNP hard via a reduction from *bounded* Post Correspondence Problem. For the proof refer to the full version of the paper [2].

THEOREM 15. *Given two* monotone polynomial-degree *circuits C and C' computing polynomial $f, g \in \mathbb{Q}\langle X \rangle$ it is* coNP*-complete to check if $f \circ g$ is identically zero.*

4 Hadamard product of monotone multilinear circuits

In this section we study the Hadamard product of *commutative* polynomials (defined as in the noncommutative case). First we introduce some notation useful for this section. Given a polynomial $f \in \mathbb{F}[X]$, and a monomial m over the variables X, we define $f(m)$ to be the coefficient of the monomial m in the polynomial f.[†] Recall the Definition 1 of the Hadamard product of two polynomials in $\mathbb{F}\langle X\rangle$. We define the Hadamard product in the commutative case analogously. Thus, for polynomials $f, g \in \mathbb{F}[X]$ we have $F(m) = f(m)g(m)$ for any monomial m, where $F = f \circ g$.

In this section our interest is the expressive power of the Hadamard product. Can we express a hard explicit polynomial (like the Permanent) as the Hadamard product $f \circ g$ where f and g have small arithmetic circuits? It turns out that we easily can.

PROPOSITION 16. *There are multilinear polynomials $f, g \in \mathbb{F}[x_{11}, x_{12}, \cdots, x_{nn}]$ such that both f and g have arithmetic formulas of size $O(n^2)$ and $f \circ g$ is the Permanent polynomial. Furthermore, for $\mathbb{F} = \mathbb{Q}$ these formulas for f and g are* monotone.

Proof. Define the polynomials f and g on the variables $\{x_{ij} \mid 1 \leq i, j \leq n\}$ as follows $f = \prod_{i=1}^{n}(\sum_{j=1}^{n} x_{ij})$ and $g = \prod_{j=1}^{n}(\sum_{i=1}^{n} x_{ij})$. Clearly, their Hadamard product is $Perm(x_{11}, \cdots, x_{nn})$. The formulas for f and g over rationals are monotone. ∎

Nevertheless, we will define an explicit monotone multilinear polynomial that cannot be written as the Hadamard product of multilinear polynomials computed by subexponential sized monotone arithmetic circuits. Our construction adapts a result of Raz and Yehudayoff [15] describing an explicit monotone polynomial that has no monotone arithmetic circuits of size $2^{\epsilon n}$, for some constant $\epsilon > 0$. Our proof closely follows the arguments in [15]. Due to lack of space, we provide only proof sketches for several technical statements.

DEFINITION 17. *For $\epsilon > 0$, a multilinear polynomial $f \in \mathbb{C}[x_1, \ldots, x_n]$ is an* ϵ-product polynomial *if there are disjoint sets $A, B \subseteq X = \{x_1, \ldots, x_n\}$ such that $|A| \geq \epsilon n$ and $|B| \geq \epsilon n$ and $f = gh$ where $g \in \mathbb{C}[A]$ and $h \in \mathbb{C}[B]$.*

In the sequel, we often identify a multilinear polynomial f in $\mathbb{C}[X]$ with its coefficients vector (indexed by monomials in the natural lexicographic order). The complex inner product of vectors $w, w' \in \mathbb{C}^k$ is $\langle w, w'\rangle = \sum_i w_i \overline{w'_i}$. Let $\mathcal{M}(X)$ denote the set of multilinear monomials over the variables in X.

DEFINITION 18. *The* correlation *of multilinear polynomials f and g in $\mathbb{C}[X]$ is defined as* $\mathrm{Corr}(f, g) = |\sum_{m \in \mathcal{M}(X)} f(m)\overline{g(m)}|$. *Notice that* $\mathrm{Corr}(f, f)$ *is the ℓ_2-norm $\|f\|$ of f.*

[†] There should be no confusion with evaluating the multivariate polynomial f at a point $(a_1, \cdots, a_n)$ as we denote that by $f(a_1, a_2, \cdots, a_n)$.

The explicit polynomial from [15]

The explicit polynomial F we define is essentially the same as the one in [15] (the difference is in the constants). Let $s \in \mathbb{N}$ be a constant, to be chosen later and $t = 40s$. Let $n = tp = 40sp$, for a prime p, and $X = \{x_1, \ldots, x_n\}$. Partition X into t many sets of variables $X(1), \ldots, X(t)$ with p variables each, where $X(i) = \{x_{(i-1)p+j} \mid j \in [p]\}$.

In poly(n) time we can construct the field $\mathbb{F} = \mathbb{F}_{2^p}$ which is in bijective correspondence with $\{0,1\}^p$. We can assume that $0 \in \mathbb{F}$ is associated with the all 0s vector 0^p. Fix a nontrivial additive character ψ of $\mathbb{F}$. Since $\mathrm{char}(\mathbb{F}) = 2$ we have $\psi(x) = \pm 1$ for all $x \in \mathbb{F}$. Each monomial $m \in \mathcal{M}(X)$ defines a subset A_m of X and is thus represented by its characteristic vector $w \in \{0,1\}^n$. Split w into t blocks $w_1, \ldots, w_t$ of size p each (w_i is the characteristic vector of $A_m \cap X(i)$), and consider the p field elements $y_1(m), y_2(m), \ldots, y_t(m) \in \mathbb{F}$ associated with these strings. The bijection between $\mathbb{F}$ and $\{0,1\}^p$ implies for any $m \in \mathcal{M}(X)$ that $y_i(m) = 0$ iff no variable $x \in X(i)$ appears in m.

Let us now define the polynomial F. Given a monomial $m \in \mathcal{M}(X)$, we define $F(m)$ to be $\psi(\prod_{i=1}^{t} y_i(m))$. We define a polynomial $f \in \mathbb{F}[X]$ to be *explicit* if the coefficient $f(m)$ of any monomial m can be computed in time polynomial in n. Note that the polynomial F is explicit.

We now state our main correlation result using which we will obtain the lower bound against the Hadamard product of monotone multilinear polynomials in $\mathbb{C}[x_1, \ldots, x_n]$. A proof sketch is given in the full version of the paper [2].

LEMMA 19. *Let $F \in \mathbb{C}[x_1, \ldots, x_n]$ be the explicit multilinear polynomial defined above and $f_1, f_2 \in \mathbb{C}[x_1, \ldots, x_n]$ be any $1/3$-product polynomials. Then*

1. $\sum_{m \in \mathcal{M}(\{x_1, \ldots, x_n\})} F(m) \geq 0$.
2. $\mathrm{Corr}\,(F, f_1 \circ f_2) \leq 2^{-\alpha n} \|F\| \|f_1 \circ f_2\|$, *for a constant $\alpha > 0$ that is independent of f_1 and f_2.*

Using the above lemma bounding the correlation between F and the Hadamard product of $1/3$-product polynomials, we will prove the main lower bound. We first recall a crucial lemma of Raz and Yehudayoff [15].

LEMMA 20. *For $n \geq 3$, let $F \in \mathbb{C}[x_1, \ldots, x_n]$ be a monotone multilinear polynomial computed by a monotone circuit of size s (i.e. the circuit has at most s edges). Then, there are $s+1$ monotone $1/3$-product polynomials $f_1, f_2, \ldots, f_{s+1}$ such that $F = \sum_{i=1}^{s+1} f_i$.*

THEOREM 21. *For large enough $n \in \mathbb{N}$, there is an explicit monotone multilinear polynomial $F' \in \mathbb{Q}[x_1, \ldots, x_n]$ that cannot be written as a Hadamard product of two monotone multilinear polynomials $f_1, f_2 \in \mathbb{R}[x_1, \ldots, x_n]$ such that each f_i is computed by monotone circuits of size less than $2^{\alpha n}$, where $\alpha > 0$ is an absolute constant.*

Proof. By the density of primes it suffices to consider n of the form tp, for prime p, where t is the constant in the definition of F. Let X denote the set of variables $\{x_1, \ldots, x_n\}$, and let F be the explicit polynomial mentioned in Lemma 19 above. For any monomial $m \in \mathcal{M}(X)$, let $F'(m) = (F(m)+1)/2$. Clearly, the coefficients of F' all lie in $\{0,1\}$. Consider the correlation between F and F', $\text{Corr}\,(F,F') = \left|\sum_{m:F(m)=1} 1\right| \geq 2^{n-1}$, where the inequality follows from the point 1 of Lemma 19.

Let us assume that F' can be written as $f_1 \circ f_2$, where f_1 and f_2 are multilinear polynomials computed by monotone arithmetic circuits of size at most s. We assume $n \geq 3$, so that Lemma 20 is applicable. By Lemma 20, there exist monotone 1/3-product polynomials $f_{1,1}, \ldots, f_{1,s+1}, f_{2,1}, \ldots, f_{2,s+1}$ such that $f_i = \sum_{j=1}^{s+1} f_{i,j}$, for each $i \in \{1,2\}$. Thus, we have,

$$F' = \left(\sum_{j=1}^{s+1} f_{1,j}\right) \circ \left(\sum_{k=1}^{s+1} f_{2,k}\right) = \sum_{1\leq j,k\leq s+1} f_{1,j} \circ f_{2,k}$$

Taking correlation with F on both sides, we see that,

$$2^{n-1} \leq \sum_{1\leq j,k\leq s+1} \text{Corr}\left(F, f_{1,j} \circ f_{2,k}\right) \leq \sum_{1\leq j,k\leq s+1} 2^{-\beta n}\|F\|\|f_{1,j} \circ f_{2,k}\|,$$

by applying triangle inequality and then part 2 of Lemma 19, where $\beta > 0$ is some constant.

Since, $f_{1,j} \circ f_{2,k}$'s are monotone polynomials adding up to F', it follows that for any monomial $m \in \mathcal{M}(X)$ its coefficient in $f_{1,j} \circ f_{2,k}$ is at most 1. Hence, $\|f_{1,j} \circ f_{2,k}\| \leq \|F\|$ and we have

$$2^{n-1} \leq \sum_{1\leq j,k\leq s+1} 2^{-\beta n}\|F\|^2 = (s+1)^2 2^{n-\beta n}$$

Consequently, we have $s \geq 2^{\beta n/4}$, for large enough n. ∎

References

[1] E. Allender, R. Beals, and M. Ogihara, The complexity of matrix rank and feasible systems of linear equations, *Computational Complexity* , 8(2):99-126, 1999.

[2] V. Arvind, P. S. Joglekar, S. Srinivasan Arithmetic Circuits and the Hadamard Product of Polynomials *CoRR* abs/0907.4006: (2009)

[3] E. Allender, M. Ogihara, Relationships among PL, #L and the determinant. *RAIRO - Theoretical Informatics and Applications*, 30:1–21, 1996.

[4] E. Allender, K. Reinhardt, S. Zhou, Isolation, matching and counting uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181, 1999.

[5] V. Arvind, P. Mukhopadhyay, S. Srinivasan New results on Noncommutative Polynomial Identity Testing*In Proc. of Annual IEEE Conference on Computational Complexity*,268-279,2008.

[6] R. Bhatia, Matrix Analysis, Springer-Verlag Publishing Company, 1997.
[7] J. Bourgain: "On the Construction of Affine Extractors", Geometric & Functional Analysis GAFA, Vol. 17, No. 1. (April 2007), pp. 33-57.
[8] J. Bourgain, A. Glibichuk, S. Konyagin: "Estimate for the number of sums and products and for exponential sums in fields of prime order", J. London Math. Soc. 73 (2006), pp. 380-398.
[9] M. R. Garey, D. S. Johnson Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman.*p. 228. ISBN 0-7167-1045-5*, 1979.
[10] J. E. Hopcroft, R. Motawani, J. D. Ullman, Introduction to Automata Theory Languages and Computation,*Second Edition*, Pearson Education Publishing Company.
[11] V. Kabanets, R. Impagliazzo, Derandomization of polynomial identity test means proving circuit lower bounds, *In Proc. of 35th ACM Sym. on Theory of Computing*,355-364,2003.
[12] A. Klivans, D. van Melkebeek, Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.
[13] N. Nisan, Lower bounds for noncommutative computation *In Proc. of 23rd ACM Sym. on Theory of Computing*, 410-418, 1991.
[14] R. Raz, A. Shpilka, Deterministic polynomial identity testing in non commutative models *Computational Complexity*,14(1):1-19, 2005.
[15] Ran Raz, Amir Yehudayoff, "Multilinear Formulas, Maximal-Partition Discrepancy and Mixed-Sources Extractors." FOCS 2008: 273-282.
[16] S. Toda, Counting Problems Computationally Equivalant to the Determinant, manuscript.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Kernels for Feedback Arc Set In Tournaments

Stéphane Bessy[1], Fedor V. Fomin[2], Serge Gaspers[1], Christophe Paul[1], Anthony Perez[1], Saket Saurabh[2], Stéphan Thomassé[1]

[1] LIRMM – Université de Montpellier 2, CNRS, Montpellier, France.
{bessy|gaspers|paul|perez|thomasse}@lirmm.fr

[2] Department of Informatics, University of Bergen, Bergen, Norway.
{fedor.fomin|saket.saurabh}@ii.uib.no

ABSTRACT. A tournament $T = (V, A)$ is a directed graph in which there is exactly one arc between every pair of distinct vertices. Given a digraph on n vertices and an integer parameter k, the FEEDBACK ARC SET problem asks whether the given digraph has a set of k arcs whose removal results in an acyclic digraph. The FEEDBACK ARC SET problem restricted to tournaments is known as the k-FEEDBACK ARC SET IN TOURNAMENTS (k-FAST) problem. In this paper we obtain a linear vertex kernel for k-FAST. That is, we give a polynomial time algorithm which given an input instance T to k-FAST obtains an equivalent instance T' on $O(k)$ vertices. In fact, given any fixed $\epsilon > 0$, the kernelized instance has at most $(2+\epsilon)k$ vertices. Our result improves the previous known bound of $O(k^2)$ on the kernel size for k-FAST. Our kernelization algorithm solves the problem on a subclass of tournaments in polynomial time and uses a known polynomial time approximation scheme for k-FAST.

1 Introduction

Given a directed graph $G = (V, A)$ on n vertices and an integer parameter k, the FEEDBACK ARC SET problem asks whether the given digraph has a set of k arcs whose removal results in an acyclic directed graph. In this paper, we consider this problem in a special class of directed graphs, *tournaments*. A tournament $T = (V, A)$ is a directed graph in which there is exactly one directed arc between every pair of vertices. More formally the problem we consider is defined as follows.

> k-FEEDBACK ARC SET IN TOURNAMENTS (k-FAST): Given a tournament $T = (V, A)$ and a positive integer k, does there exist a subset $F \subseteq A$ of at most k arcs whose removal makes T acyclic.

In the weighted version of k-FAST, we are also given integer weights (each weight is at least one) on the arcs and the objective is to find a feedback arc set of weight at most k. This problem is called k-WEIGHTED FEEDBACK ARC SET IN TOURNAMENTS (k-WFAST).

Feedback arc sets in tournaments are well studied from the combinatorial [17, 18, 24, 25, 28, 32], statistical [26] and algorithmic [1, 2, 12, 21, 30, 31] points of view. The problems k-FAST and k-WFAST have several applications. In *rank aggregation* we are given several

© Bessy, Fomin, Gaspers, Paul, Perez, Saurabh, Thomassé; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 37–47
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2305

rankings of a set of objects, and we wish to produce a single ranking that on average is as consistent as possible with the given ones, according to some chosen measure of consistency. This problem has been studied in the context of voting [7, 11], machine learning [10], and search engine ranking [15, 16]. A natural consistency measure for rank aggregation is the number of pairs that occur in a different order in the two rankings. This leads to *Kemeny rank aggregation* [19, 20], a special case of k-WFAST.

The k-FAST problem is known to be NP-complete by recent results of Alon [2] and Charbit *et al.* [9] while k-WFAST is known to be NP-complete by Bartholdi III *et al.* [4]. From an approximation perspective, k-WFAST is APX-hard [27] but admits a polynomial time approximation scheme when the edge weights are bounded by a constant [21]. The problem is also well studied in parameterized complexity. In this area, a problem with input size n and a parameter k is said to be fixed parameter tractable (FPT) if there exists an algorithm to solve this problem in time $f(k) \cdot n^{O(1)}$, where f is an arbitrary function of k. Raman and Saurabh [23] showed that k-FAST and k-WFAST are FPT by obtaining an algorithm running in time $O(2.415^k \cdot k^{4.752} + n^{O(1)})$. Recently, Alon *et al.* [3] have improved this result by giving an algorithm for k-WFAST running in time $O(2^{O(\sqrt{k}\log^2 k)} + n^{O(1)})$. This algorithm runs in sub-exponential time, a trait uncommon to parameterized algorithms. In this paper we investigate k-FAST from the view point of kernelization, currently one of the most active subfields of parameterized algorithms.

A parameterized problem is said to admit a *polynomial kernel* if there is a polynomial (in n) time algorithm, called a *kernelization* algorithm, that reduces the input instance to an instance whose size is bounded by a polynomial $p(k)$ in k, while preserving the answer. This reduced instance is called a $p(k)$ *kernel* for the problem. When $p(k)$ is a linear function of k then the corresponding kernel is a linear kernel. Kernelization has been at the forefront of research in parameterized complexity in the last couple of years, leading to various new polynomial kernels as well as tools to show that several problems do not have a polynomial kernel under some complexity-theoretic assumptions [5, 6, 8, 14, 29]. In this paper we continue the current theme of research on kernelization and obtain a *linear vertex* kernel for k-FAST. That is, we give a polynomial time algorithm which given an input instance T to k-FAST obtains an equivalent instance T' on $O(k)$ vertices. More precisely, given any fixed $\epsilon > 0$, we find a kernel with a most $(2+\epsilon)k$ vertices in polynomial time. The reason we call it a linear *vertex* kernel is that, even though the number of vertices in the reduced instance is at most $O(k)$, the number of arcs is still $O(k^2)$. Our result improves the previous known bound of $O(k^2)$ on the vertex kernel size for k-FAST [3, 13]. For our kernelization algorithm we find a subclass of tournaments where one can find a minimum sized feedback arc set in polynomial time (see Lemma 12) and use the known polynomial time approximation scheme for k-FAST by Kenyon-Mathieu and Schudy [21]. The polynomial time algorithm for a subclass of tournaments could be of independent interest.

The paper is organized as follows. In Section 2, we give some definition and preliminary results regarding feedback arc sets. In Section 3 we give a linear vertex kernel for k-FAST. Finally we conclude with some remarks in Section 4.

2 Preliminaries

Let $T = (V, A)$ be a tournament on n vertices. We use $T_\sigma = (V_\sigma, A)$ to denote a tournament whose vertices are ordered under a fixed ordering $\sigma = v_1, \ldots, v_n$ (we also use D_σ for an ordered directed graph). We say that an arc v_iv_j of T_σ is a *backward arc* if $i > j$, otherwise we call it a *forward arc*. Moreover, given any partition $\mathcal{P} := \{V_1, \ldots, V_l\}$ of V_σ, where every V_i is an interval according to the ordering of T_σ, we use A_B to denote all arcs between the intervals (having their endpoints in different intervals), and A_I for all arcs within the intervals. If T_σ contains no backward arc, then we say that it is *transitive*.

For a vertex $v \in V$ we denote its *in-neighborhood* by $N^-(v) := \{u \in V \mid uv \in A\}$ and its *out-neighborhood* by $N^+(v) := \{u \in V \mid vu \in A\}$. A set of vertices $M \subseteq V$ is a *module* if and only if $N^+(u) \setminus M = N^+(v) \setminus M$ for every $u, v \in M$. For a subset of arcs $A' \subseteq A$, we define $T[A']$ to be the digraph (V', A') where V' is the union of endpoints of the arcs in A'. Given an ordered digraph D_σ and an arc $e = v_iv_j$, $S(e) = \{v_i, \ldots, v_j\}$ denotes the *span* of e. The number of vertices in $S(e)$ is called the *length* of e and is denoted by $l(e)$. Thus, for every arc $e = v_iv_j$, $l(e) = |i - j| + 1$. Finally, for every vertex v in the span of e, we say that e is *above* v.

In this paper, we will use the well-known fact that every acyclic tournament admits a transitive ordering. In particular, we will consider *maximal transitive modules*. We also need the following result for our kernelization algorithm.

LEMMA 1. *([23]) Let $D = (V, A)$ be a directed graph and F be a minimal feedback arc set of D. Let D' be the graph obtained from D by reversing the arcs of F in D, then D' is acyclic.*

In this paper whenever we say *circuit*, we mean a directed cycle. Next we introduce a definition which is useful for a lemma we prove later.

DEFINITION 2. *Let $D_\sigma = (V_\sigma, A)$ be an ordered directed graph and let $f = vu$ be a backward arc of D_σ. We call* certificate *of f, and denote it by $c(f)$, any directed path from u to v using only forward arcs in the span of f in D_σ.*

Observe that such a directed path together with the backward arc f forms a directed cycle in D_σ whose only backward arc is f.

DEFINITION 3. *Let $D_\sigma = (V_\sigma, A)$ be an ordered directed graph, and let $F \subseteq A$ be a set of backward arcs of D_σ. We say that we can* certify *F whenever it is possible to find a set $\mathcal{F} = \{c(f) : f \in F\}$ of arc-disjoint certificates for the arcs in F.*

Let $D_\sigma = (V_\sigma, A)$ be an ordered directed graph, and let $F \subseteq A$ be a subset of backward arcs of D_σ. We say that we can certify the set F using *only arcs from* $A' \subseteq A$ if F can be certified by a collection $\mathcal{F}$ such that the union of the arcs of the certificates in $\mathcal{F}$ is contained in A'. In the following, $fas(D)$ denotes the *size* of a minimum feedback arc set, that is, the cardinality of a minimum sized set F of arcs whose removal makes D acyclic.

LEMMA 4. *Let D_σ be an ordered directed graph, and let $\mathcal{P} = \{V_1, \ldots, V_l\}$ be a partition of D_σ into intervals. Assume that the set F of all backward arcs of $D_\sigma[A_B]$ can be certified using only arcs from A_B. Then $fas(D_\sigma) = fas(D_\sigma[A_I]) + fas(D_\sigma[A_B])$. Moreover, there exists a minimum sized feedback arc set of D_σ containing F.*

PROOF. For any bipartition of the arc set A into A_1 and A_2, $fas(D_\sigma) \geq fas(D_\sigma[A_1]) + fas(D_\sigma[A_2])$. Hence, in particular for a partition of the arc set A into A_I and A_B we have

that $fas(D_\sigma) \geq fas(D_\sigma[A_I]) + fas(D_\sigma[A_B])$. Next, we show that $fas(D_\sigma) \leq fas(D_\sigma[A_I]) + fas(D_\sigma[A_B])$. This follows from the fact that once we reverse all the arcs in F, each remaining circuit lies in $D_\sigma[V_i]$ for some $i \in \{1, \ldots, l\}$. In other words once we reverse all the arcs in F, every circuit is completely contained in $D_\sigma[A_I]$. This concludes the proof of the first part of the lemma. In fact, what we have shown is that there exists a minimum sized feedback arc set of D_σ containing F. This concludes the proof of the lemma. ∎

3 Kernels for k-FAST

In this section we first give a subquadratic vertex kernel of size $O(k\sqrt{k})$ for k-FAST and then improve on it to get our final vertex kernel of size $O(k)$. We start by giving a few reduction rules that will be needed to bound the size of the kernels.

Rule 3.1 *If a vertex v is not contained in any triangle, delete v from T.*

Rule 3.2 *If there exists an arc uv that belongs to more than k distinct triangles, then reverse uv and decrease k by* 1.

We say that a reduction rule is *sound*, if whenever the rule is applied to an instance (T, k) to obtain an instance (T', k'), T has a feedback arc set of size at most k if and only if T' has a feedback arc set of size at most k'.

LEMMA 5. *([3, 13]) Rules 3.1 and 3.2 are sound and can be applied in polynomial time.*

The Rules 3.1 and 3.2 together led to a quadratic kernel for k-WFAST [3]. Earlier, these rules were used by Dom *et al.* [13] to obtain a quadratic kernel for k-FAST. We now add a new reduction rule that will allow us to obtain the claimed bound on the kernel sizes for k-FAST. Given an ordered tournament $T_\sigma = (V_\sigma, A)$, we say that $\mathcal{P} = \{V_1, \ldots, V_l\}$ is a *safe partition* of V_σ into intervals whenever it is possible to certify the backward arcs of $T_\sigma[A_B]$ using only arcs from A_B.

Rule 3.3 *Let T_σ be an ordered tournament, $\mathcal{P} = \{V_1, \ldots, V_l\}$ be a safe partition of V_σ into intervals and F be the set of backward arcs of $T_\sigma[A_B]$. Then reverse all the arcs of F and decrease k by $|F|$.*

LEMMA 6. *Rule 3.3 is sound.*

PROOF. Let $\mathcal{P}$ be a safe partition of T_σ. Observe that it is possible to certify all the backward arcs, that is F, using only arcs in A_B. Hence using Lemma 4 we have that $fas(T_\sigma) = fas(T_\sigma[A_I]) + fas(T_\sigma[A_B])$. Furthermore, by Lemma 4 we also know that there exists a minimum sized feedback arc set of D_σ containing F. Thus, T_σ has a feedback arc set of size at most k if and only if the tournament T'_σ obtained from T_σ by reversing all the arcs of F has a feedback arc set of size at most $k - |F|$. ∎

3.1 A subquadratic kernel for k-FAST

In this section, we show how to obtain an $O(k\sqrt{k})$ sized vertex kernel for k-FAST. To do so, we introduce the following reduction rule.

Rule 3.4 *Let V_m be a* maximal transitive module *of size p, and I and O be the set of in-neighbors and out-neighbors of the vertices of V_m in T, respectively. Let Z be the set of arcs uv such that $u \in O$ and $v \in I$. If $q = |Z| < p$ then reverse all the arcs in Z and decrease k by q.*

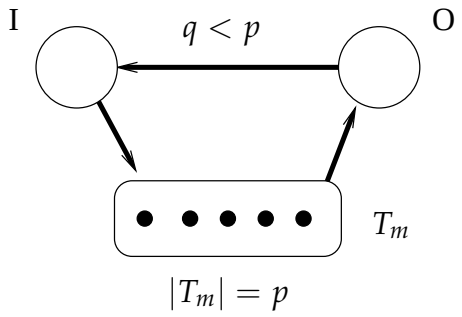


Figure 1: A transitive module on which Rule 3.4 applies.

LEMMA 7. *Rule 3.4 is sound and can be applied in linear time.*

PROOF. We first prove that the partition $\mathcal{P} = \{I, V_m, O\}$ forms a safe partition of the input tournament. Let $V'_m = \{w_1, \ldots, w_q\} \subseteq V_m$ be an arbitrary subset of size q of V_m and let $Z = \{u_i v_i \mid 1 \leq i \leq q\}$. Consider the collection $\mathcal{F} = \{v_i w_i u_i \mid u_i v_i \in Z, w_i \in V'_m\}$ and notice that it certifies all the arcs in Z. In fact we have managed to certify all the backwards arcs of the partition using only arcs from A_B and hence $\mathcal{P}$ forms a safe partition. Thus, by Rule 3.3, it is safe to reverse all the arcs from O to I. The time complexity follows from the fact that computing a modular decomposition tree can be done in $O(n+m)$ time on directed graphs [22]. ∎

We show that any YES-instance to which none of the Rules 3.1, 3.2 and 3.4 could be applied has at most $O(k\sqrt{k})$ vertices.

THEOREM 8. *Let $(T = (V, A), k)$ be a* YES*-instance to k-*FAST *which has been reduced according to Rules 3.1, 3.2 and 3.4. Then T has at most $O(k\sqrt{k})$ vertices.*

PROOF. Let S be a feedback arc set of size at most k of T and let T' be the tournament obtained from T by reversing all the arcs in S. Let σ be the transitive ordering of T' and $T_\sigma = (V_\sigma, A)$ be the ordered tournament corresponding to the ordering σ. We say that a vertex is *affected* if it is incident to some arc in S. Thus, the number of affected vertices is at most $2|S| \leq 2k$. The reduction Rule 3.1 ensures that the first and last vertex of T_σ are affected. To see this note that if the first vertex in V_σ is not affected then it is a source vertex (vertex with in-degree 0) and hence it is not part of any triangle and thus Rule 3.1 would have applied. We can similarly argue for the last vertex. Next we argue that there is no backward arc e of length greater than $2k+2$ in T_σ. Assume to the contrary that $e = uv$ is a backward arc with $S(e) = \{v, x_1, x_2, \ldots, x_{2k+1}, \ldots, u\}$ and hence $l(e) > 2k+2$. Consider the collection $\mathcal{T} = \{v x_i u \mid 1 \leq i \leq 2k\}$ and observe that at most k of these triples can contain an arc from $S \setminus \{e\}$ and hence there exist at least $k+1$ triplets in $\mathcal{T}$ which corresponds to distinct triangles all containing e. But then e would have been reversed by an application

of Rule 3.2. Hence, we have shown that there is no backward arc e of length greater than $2k+2$ in T_σ. Thus $\sum_{e\in S} l(e) \leq 2k^2+2k$.

We also know that between two consecutive affected vertices there is exactly one maximal transitive module. Let us denote by t_i the number of vertices in these modules, where $i \in \{1,\ldots,2k-1\}$. The objective here is to bound the number of vertices in V_σ or V using $\sum_{i=1}^{2k-1} t_i$. To do so, observe that since T is reduced under the Rule 3.4, there are at least t_i backward arcs above every module with t_i vertices, each of length at least t_i. This implies that $\sum_{i=1}^{2k-1} t_i^2 \leq \sum_{e\in S} l(e) \leq 2k^2+2k$. Now, using the Cauchy-Schwarz inequality we can show the following.

$$\sum_{i=1}^{2k-1} t_i = \sum_{i=1}^{2k-1} t_i \cdot 1 \leq \sqrt{\sum_{i=1}^{2k-1} t_i^2 \cdot \sum_{i=1}^{2k-1} 1} \leq \sqrt{(2k^2+2k)\cdot(2k-1)} = \sqrt{4k^3+2k^2-k}.$$

Thus every reduced YES-instance has at most $\sqrt{4k^3+2k^2-k}+2k = O(k\sqrt{k})$ vertices. ∎

3.2 A linear kernel for k-FAST

We begin this subsection by showing some general properties about tournaments which will be useful in obtaining a linear kernel for k-FAST.

Backward Weighted Tournaments

Let T_σ be an ordered tournament with weights on its backward arcs. We call such a tournament a *backward weighted tournament* and denote it by T_ω, and use $\omega(e)$ to denote the weight of a backward arc e. For every interval $I := [v_i,\ldots,v_j]$ we use $\omega(I)$ to denote the total weight of all backward arcs having both their endpoints in I, that is, $\omega(I) = \sum_{e=uv} w(e)$ where $u,v \in I$ and e is a backward arc.

DEFINITION 9.(Contraction) *Let $T_\omega = (V_\sigma, A)$ be an ordered tournament with weights on its backward arcs and $I = [v_i,\ldots,v_j]$ be an interval. The contracted tournament is defined as $T_{\omega'} = (V_{\sigma'} = V_\sigma \setminus \{I\} \cup \{c_I\}, A')$. The arc set A' is defined as follows.*

- *It contains all the arcs $A_1 = \{uv \mid uv \in A, u \notin I, v \notin I\}$*
- *Add $A_2 = \{uc_I \mid uv \in A, u \notin I, v \in I\}$ and $A_3 = \{c_I v \mid uv \in A, u \in I, v \notin I\}$.*
- *Finally, we remove every forward arc involved in a 2-cycle after the addition of arcs in the previous step.*

The order σ' for $T_{\omega'}$ is provided by $\sigma' = v_1,\ldots,v_{i-1},c_I,v_{j+1},\ldots,v_n$. We define the weight of a backward arc $e = xy$ of A' as follows.

$$w'(xy) = \begin{cases} w(xy) & \text{if } xy \in A_1 \\ \sum_{\{xz\in A \mid z\in I\}} w(xz) & \text{if } xy \in A_2 \\ \sum_{\{zy\in A \mid z\in I\}} w(zy) & \text{if } xy \in A_3 \end{cases}$$

We refer to Figure 2 for an illustration.

Next we generalize the notions of certificate and certification (Definitions 2 and 3) to backward weighted tournaments.

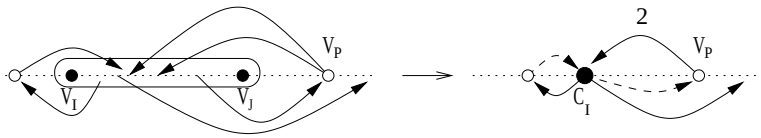


Figure 2: Illustration of the contraction step for the interval $I := [v_i, \ldots, v_j]$.

DEFINITION 10. *Let $T_\omega = (V_\sigma, A)$ be a backward weighted tournament, and let $f = vu \in A$ be a backward arc of T_ω. We call* ω-certificate *of f, and denote it by $\mathcal{C}(f)$, a collection of $\omega(f)$ arc-disjoint directed paths going from u to v and using only forward arcs in the span of f in T_ω.*

DEFINITION 11. *Let $T_\omega = (V_\sigma, A)$ be a backward weighted tournament, and let $F \subseteq A$ be a subset of backward arcs of T_ω. We say that we can* ω-certify *F whenever it is possible to find a set $\mathcal{F} = \{\mathcal{C}(f) : f \in F\}$ of arc-disjoint ω-certificates for the arcs in F.*

LEMMA 12. *Let $T_\omega = (V_\sigma, A)$ be a backward weighted tournament such that for every interval $I := [v_i, \ldots, v_j]$ the following holds:*

$$2 \cdot \omega(I) \leq |I| - 1 \tag{1}$$

Then it is possible to ω-certify the backward arcs of T_ω.

PROOF. Let $V_\sigma = v_1, \ldots, v_n$. The proof is by induction on n, the number of vertices. Note that by applying (1) to the interval $I = [v_1, \ldots, v_n]$, we have that there exists a vertex v_i in T_ω that is not incident to any backward arc. Let $T'_\omega = (V'_\sigma, A')$ denote the tournament $T_\omega \setminus \{v_i\}$. We say that an interval I is *critical* whenever $|I| \geq 2$ and $2 \cdot \omega(I) = |I| - 1$. We now consider several cases, based on different types of critical intervals.

(i) Suppose that there are no critical intervals. Thus, in T'_ω, every interval satisfies (1), and hence by induction on n the result holds.

(ii) Suppose now that the only critical interval is $I = [v_1, \ldots, v_n]$, and let $e = vu$ be a backward arc above v_i with the maximum length. Note that since v_i does not belong to any backward arc, we can use it to form a directed path $c(e) = uv_iv$, which is a certificate for e. We now consider T'_ω where the weight of e has been decreased by 1. In this process if $\omega(e)$ becomes 0 then we reverse the arc e. We now show that every interval of T'_ω respects (1). If an interval $I' \in T'_\omega$ does not contain v_i in the corresponding interval in T_ω, then by our assumption we have that $2 \cdot \omega(I') \leq |I'| - 1$. Now we assume that the interval corresponding to I' in T_ω contains v_i but either $u \notin I' \cup \{v_i\}$ or $v \notin I' \cup \{v_i\}$. Then we have $2 \cdot \omega(I') = 2 \cdot \omega(I) < |I| - 1 = |I'|$ and hence we get that $2 \cdot \omega(I') \leq |I'| - 1$. Finally, we assume that the interval corresponding to I' in T_ω contains v_i and $u, v \in I' \cup \{v_i\}$. In this case, $2 \cdot \omega(I') = 2 \cdot (\omega(I) - 1) \leq |I| - 1 - 2 < |I'| - 1$. Thus, by the induction hypothesis, we obtain a family of arc-disjoint ω-certificates $\mathcal{F}'$ which ω-certify the backward arcs of T'_ω. Observe that the maximality of $l(e)$ ensures that if e is reversed then it will not be used in any ω-certificate of $\mathcal{F}'$, thus implying that $\mathcal{F}' \cup c(e)$ is a family ω-certifying the backward arcs of T_ω.

(iii) Finally, suppose that there exists a critical interval $I \subsetneq V_\sigma$. Roughly speaking, we will show that I and $V_\sigma \setminus I$ can be certified separately. To do so, we first show the following.
Claim. Let $I \subset V_\sigma$ be a critical interval. Then the tournament $T_{\omega'} = (V_{\sigma'}, A')$ obtained from T_ω by contracting I satisfies the conditions of the lemma.
PROOF. Let H' be any interval of $T_{\omega'}$. As before if H' does not contain c_I then the result holds by hypothesis. Otherwise, let H be the interval corresponding to H' in T_ω. We will show that $2\omega(H') \leq |H'| - 1$. By hypothesis, we know that $2\omega(H) \leq |H| - 1$ and that $2\omega(I) = |I| - 1$. Thus we have the following.

$$2\omega(H') = 2 \cdot (\omega(H) - \omega(I)) \leq |H| - 1 - |I| + 1 = (|H| + 1 - |I|) - 1 = |H'| - 1$$

Thus, we have shown that the tournament $T_{\omega'}$ satisfies the conditions of the lemma. ∎

We now consider a minimal critical interval I. By induction, and using the claim, we know that we can obtain a family of arc-disjoint ω-certificates $\mathcal{F}'$ which ω-certifies the backward arcs of $T_{\omega'}$ without using any arc within I. Now, by minimality of I, we can use (ii) to obtain a family of arc-disjoint ω-certificates $\mathcal{F}''$ which ω-certifies the backward arcs of I using only arcs within I. Thus, $\mathcal{F}' \cup \mathcal{F}''$ is a family ω-certifying all backward arcs of T_ω.

This concludes the proof of the lemma. ∎

In the following, any interval that does not respect condition (1) is said to be a *dense interval*.

LEMMA 13. *Let $T_\omega = (V_\sigma, A)$ be a backward weighted tournament with $|V_\sigma| \geq 2p + 1$ and $\omega(V_\sigma) \leq p$. Then there exists a safe partition of V_σ with at least one backward arc between the intervals and it can be computed in polynomial time.*

PROOF. The proof is by induction on $n = |V_\sigma|$. Observe that the statement is true for $n = 3$, which is our base case.

For the inductive step, we assume first that there is no dense interval in T_ω. In this case Lemma 12 ensures that the partition of V_σ into singletons of vertices is a safe partition. So from now on we assume that there exists at least one dense interval.

Let I be a dense interval. By definition of I, we have that $\omega(I) \geq \frac{1}{2} \cdot |I|$. We now contract I and obtain the backward weighted tournament $T_{\omega'} = (V_{\sigma'}, A')$. In the contracted tournament $T_{\omega'}$, we have:

$$\begin{cases} |V_{\sigma'}| & \geq 2p + 1 - (|I| - 1) = 2p - |I| + 2; \\ \omega'(V_{\sigma'}) & \leq p - \frac{1}{2} \cdot |I|. \end{cases}$$

Thus, if we set $r := p - \frac{1}{2} \cdot |I|$, we get that $|V_{\sigma'}| \geq 2r + 1$ and $\omega'(V_{\sigma'}) \leq r$. Since $|V_{\sigma'}| < |V_\sigma|$, by the induction hypothesis we can find a safe partition $\mathcal{P}$ of $T_{\omega'}$, and thus obtain a family $\mathcal{F}_{\omega'}$ that ω-certifies the backward arcs of $T_{\omega'}[A_B]$ using only arcs in A_B.

We claim that $\mathcal{P}'$ obtained from $\mathcal{P}$ by substituting c_I by its corresponding interval I is a safe partition in T_ω. To see this, first observe that if c_I has not been used to ω-certify the backward arcs in $T_{\omega'}[A_B]$, that is, c_I is not an end point of any arc in the ω-certificates, then we are done. So from now on we assume that c_I has been part of a ω-certificate for some

backward arc. Let e be a backward arc in $T_{\omega'}[A_B]$, and let $c_{\omega'}(e) \in \mathcal{F}_{\omega'}$ be a ω-certificate of e. First we assume that c_I is not the first vertex of the certificate $c_{\omega'}(e)$ (with respect to ordering σ'), and let c_1 and c_2 be the left (in-) and right (out-) neighbors of c_I in $c_{\omega'}(e)$. By definition of the contraction step together with the fact that there is a forward arc between c_1 and c_I and between c_I and c_2 in $T_{\omega'}$, we have that there were no backward arcs between any vertex in the interval corresponding to c_I and c_1 and c_2 in the original tournament T_ω. So we can always find a vertex in I to replace c_I in $c_{\omega'}(e)$, thus obtaining a certificate $c(e)$ for e in $T_\omega[A_B]$ (observe that e remains a backward arc even in T_ω). Now we assume that c_I is either a first or last vertex in the certificate $c_{\omega'}(e)$. Let e' be an arc corresponding to e in $T_{\omega'}$ with one of its endpoints being $e_I \in I$. To certify e' in $T_\omega[A_B]$, we need to show that we can construct a certificate $c(e')$ using only arcs of $T_\omega[A_B]$. We have two cases to deal with.

(i) If c_I is the first vertex of $c_{\omega'}(e)$ then let c_1 be its right neighbor in $c_{\omega'}(e)$. Using the same argument as before, there are only forward arcs between any vertex in I and c_1. In particular, there is a forward arc $e_I c_1$ in T_ω, meaning that we can construct a ω-certificate for e' in T_ω by setting $c(e') := (c_{\omega'}(e) \setminus \{c_I\}) \cup \{e_I\}$.

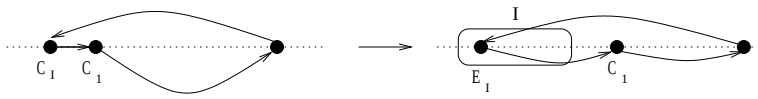


Figure 3: On the left, the ω-certificate $c_{\omega'}(e) \in \mathcal{F}_{\omega'}$. On the right, the corresponding ω-certificate obtained in T_ω by replacing c_I by the interval I.

(ii) If c_I is the last vertex of $c_{\omega'}(e)$ then let c_q be its left neighbor in $c_{\omega'}(e)$. Once again, we have that there are only forward arcs between c_q and vertices in I, and thus between c_q and e_I. So using this we can construct a ω-certificate for e' in T_ω.

Notice that the fact that all ω-certificates are pairwise arc-disjoint in $T_{\omega'}[A_B]$ implies that the corresponding ω-certificates are arc-disjoint in $T_\omega[A_B]$, and so $\mathcal{P}'$ is indeed a safe partition of V_σ. ∎

We are now ready to give the linear size kernel for k-FAST. To do so, we make use of the fact that there exists a polynomial time approximation scheme for this problem [21].

THEOREM 14. *For every fixed $\epsilon > 0$, there exists a vertex kernel for k-FAST with at most $(2+\epsilon)k$ vertices that can be computed in polynomial time.*

PROOF. Let $(T = (V, A), k)$ be an instance of k-FAST. For a fixed $\epsilon > 0$, we start by computing a feedback arc set S of size at most $(1 + \frac{\epsilon}{2})k$. To find such a set S, we use the known polynomial time approximation scheme for k-FAST [21]. Then, we order T with the transitive ordering of the tournament obtained by reversing every arc of S in T. Let T_σ denote the resulting ordered tournament. By the upper bound on the size of S, we know that T_σ has at most $(1 + \frac{\epsilon}{2})k$ backward arcs. Thus, if T_σ has more than $(2+\epsilon)k$ vertices then Lemma 13 ensures that we can find a safe partition with at least one backward arc between the intervals in polynomial time. Hence we can reduce the tournament by applying Rule 3.3. We then apply Rule 3.1, and repeat the previous steps until we do not find a safe partition or $k = 0$. In the former case, we know by Lemma 13 that T can have at most $(2+\epsilon)k$ vertices, thus implying the result. In all other cases we return NO. ∎

4 Conclusion

In this paper we obtained linear vertex kernel for k-FAST, in fact, a vertex kernel of size $(2+\epsilon)k$ for any fixed $\epsilon > 0$. The new bound on the kernel size improves the previous known bound of $O(k^2)$ on the vertex kernel size for k-FAST given in [3, 13]. It would be interesting to see if one can obtain kernels for other problems using either polynomial time approximation schemes or a constant factor approximation algorithm for the corresponding problem. An interesting problem which remains unanswered is, whether there exists a linear or even a $o(k^2)$ vertex kernel for the k-FEEDBACK VERTEX SET IN TOURNAMENTS (k-FVST) problem. In the k-FVST problem we are given a tournament T and a positive integer k and the aim is to find a set of at most k vertices whose deletion makes the input tournament acyclic. The smallest known kernel for k-FVST has size $O(k^2)$.

References

[1] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *ACM Symposium on Theory of Computing (STOC)*, pages 684–693, 2005.

[2] N. Alon. Ranking tournaments. *SIAM J. Discrete Math.*, 20(1):137–142, 2006.

[3] N. Alon, D. Lokshtanov, and S. Saurabh. Fast FAST. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2009. (to appear).

[4] J. Bartholdi III, C. A. Tovey, and M. A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[5] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels (extended abstract). In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5125 of *LNCS*, pages 563–574, 2008.

[6] H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (meta) kernelization. *CoRR*, abs/0904.0727, 2009.

[7] J. Borda. Mémoire sur les élections au scrutin. *Histoire de l'Académie Royale des Sciences*, 1781.

[8] N. Bousquet, J. Daligault, S. Thomassé, and A. Yeo. A polynomial kernel for multicut in trees. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 183–194, 2009.

[9] P. Charbit, S. Thomassé, and A. Yeo. The minimum feedback arc set problem is NP-hard for tournaments. *Combin. Probab. Comput.*, 16(1):1–4, 2007.

[10] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. In *Advances in neural information processing systems (NIPS)*, pages 451–457, 1997.

[11] M. Condorcet. Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix, 1785.

[12] D. Coppersmith, L. Fleischer, and A. Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 776–782, 2006.

[13] M. Dom, J. Guo, F. Hüffner, R. Niedermeier, and A. Truß. Fixed-parameter tractability results for feedback set problems in tournaments. In *Conference on Algorithms and Complexity (CIAC)*, volume 3998 of *LNCS*, pages 320–331, 2006.

[14] M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through colors and IDs. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2009. (to appear).

[15] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation revisited. Technical report.

[16] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *World Wide Web Conference (WWW)*, 2001.

[17] P. Erdös and J. W. Moon. On sets on consistent arcs in tournaments. *Canad. Math. Bull.*, 8:269–271, 1965.

[18] H. A. Jung. On subgraphs without cycles in tournaments. *Combinatorial Theory and its Applications II*, pages 675–677, 1970.

[19] J. Kemeny. Mathematics without numbers. *Daedalus*, 88:571–591, 1959.

[20] J. Kemeny and J. Snell. *Mathematical models in the social sciences*. Blaisdell, 1962.

[21] C. Kenyon-Mathieu and W. Schudy. How to rank with few errors. In *ACM Symposium on Theory of Computing (STOC)*, pages 95–103, 2007.

[22] R. M. McConnell and F. de Montgolfier. Linear-time modular decomposition of directed graphs. *Discrete Appl. Math.*, 145(2):198–209, 2005.

[23] V. Raman and S. Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theor. Comput. Sci*, 351(3):446–458, 2006.

[24] K. D. Reid and E. T. Parker. Disproof of a conjecture of Erdös and Moser on tournaments. *J. Combin. Theory*, 9:225–238, 1970.

[25] S. Seshu and M. B. Reed. *Linear Graphs and Electrical Networks*. Addison-Wesley, 1961.

[26] P. Slater. Inconsistencies in a schedule of paired comparisons. *Biometrika*, 48:303–312, 1961.

[27] E. Speckenmeyer. On feedback problems in digraphs. In *Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 411 of *Lecture Notes in Computer Science*, pages 218–231. Springer, 1989.

[28] J. Spencer. Optimal ranking of tournaments. *Networks*, 1:135–138, 1971.

[29] S. Thomassé. A quadratic kernel for feedback vertex set. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 115–119, 2009.

[30] A. van Zuylen. Deterministic approximation algorithms for ranking and clusterings. Technical Report 1431, Cornell ORIE, 2005.

[31] A. van Zuylen, R. Hegde, K. Jain, and D. P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 405–414, 2007.

[32] D. H. Younger. Minimum feedback arc sets for a directed graph. *IEEE Trans. Circuit Theory*, 10:238–245, 1963.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

On the Memory Consumption of Probabilistic Pushdown Automata *

Tomáš Brázdil[1], **Javier Esparza**[2], **Stefan Kiefer**[2]

[1] Faculty of Informatics, Masaryk University, Brno, Czech Republic
xbrazdil@fi.muni.cz

[2] Institut für Informatik, Technische Universität München, Germany
{esparza,kiefer}@in.tum.de

ABSTRACT. We investigate the problem of evaluating memory consumption for systems modelled by probabilistic pushdown automata (pPDA). The space needed by a run of a pPDA is the maximal height reached by the stack during the run. The problem is motivated by the investigation of depth-first computations that play an important role for space-efficient schedulings of multithreaded programs.

We study the computation of both the distribution of the memory consumption and its expectation. For the distribution, we show that a naive method incurs an exponential blow-up, and that it can be avoided using linear equation systems. We also suggest a possibly even faster approximation method. Given $\varepsilon > 0$, these methods allow to compute bounds on the memory consumption that are exceeded with a probability of at most ε.

For the expected memory consumption, we show that whether it is infinite can be decided in polynomial time for stateless pPDA (pBPA) and in polynomial space for pPDA. We also provide an iterative method for approximating the expectation. We show how to compute error bounds of our approximation method and analyze its convergence speed. We prove that our method converges linearly, i.e., the number of accurate bits of the approximation is a linear function of the number of iterations.

1 Introduction

The verification of probabilistic programs with possibly recursive procedures has been intensely studied in the last years. The Markov chains or Markov Decision Processes underlying these systems may have infinitely many states. Despite this fact, which prevents the direct application of the rich theory of finite Markov Chains, many positive results have been obtained. Model-checking algorithms have been proposed for both linear and branching temporal logics [8, 11, 18], the long-run behavior of the systems has been analyzed [6, 9], and algorithms deciding properties of games have been described (see e.g. [10]).

In all these papers programs are modelled as probabilistic pushdown automata (pPDA) or as recursive Markov chains; the two models are very close, and nearly all results obtained for one of them can be easily translated to the other [7]. In this paper we consider pPDA. Loosely speaking, a pPDA is a pushdown automaton whose transitions carry probabilities. The *configurations* of a pPDA are pairs containing the current control state and the current stack content. A *run* is a sequence of configurations, each one obtained from its predecessor

*The first author was supported by the research center Institute for Theoretical Computer Science (ITI), project No. 1M0545.

© T. Brázdil, J. Esparza, S. Kiefer; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 49–60
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2306

by applying a transition, which may modify the control state and modify the top of the stack. If a run reaches a configuration with empty stack, it stays in this configuration forever. We say "it terminates".

The memory consumption of a pPDA is modelled by the random variable M that assigns to a run the *maximal* stack height of the configurations visited along it (which may be infinite). We study the distribution and the expected value of M. The execution time and memory consumption of pPDA were studied in [9], but the results about the latter were much weaker. More precisely, all it was shown in [9] was that $\mathcal{P}(M = \infty)$ can be compared with 0 or 1 in polynomial space and with other rationals in exponential time.

A probabilistic recursive program whose variables have finite range can be modelled by a pPDA, and in this case M models the amount of memory needed for the recursion stack. But M is also an important parameter for the problem of scheduling multithreaded computations (see [15, 3] among other papers). When a multithreaded program is executed by one processor, a scheduler decides which thread to execute next, and the current states of all other active threads are stored. When threads are lightweight, this makes the memory requirements of the program heavily depend on the thread scheduler [15]. Under the usual assumption that a thread can terminate only after all threads spawned by it terminate, the space-optimal scheduler is the one that, when A spawns B, interrupts the execution of A and continues with B; this is called the *depth-first scheduler* in [15, 3]. The depth-first scheduler can be modelled by a pushdown automaton. To give an example, consider a multithreaded system with two types of threads, X and Y. Imagine that through statistical sampling we know that a thread of type X spawns either a thread of type Y or no new threads, both with probability $1/2$; a thread of type Y spawns another thread of type Y with probability $1/3$, or no new thread with probability $2/3$. The depth-first execution of this multithreaded program is modelled by a pPDA with one control state, stack symbols X, Y, and rules $X \stackrel{1/2}{\to} YX$, $X \stackrel{1/2}{\to} \varepsilon$, $Y \stackrel{1/3}{\to} YY$, $Y \stackrel{2/3}{\to} \varepsilon$. Notice that the rule $X \stackrel{1/2}{\to} YX$ indeed models the depth-first idea: the new thread Y is executed before the thread X.

In this simple model, pPDA for multithreaded systems have one single control state. Stack symbols represent currently active threads and pushdown rules model whether a thread dies or spawns a new thread. On the other hand, pPDA with more than one control state can model global variables with finite range (the possible values of the global store are encoded into the control states of the pPDA) [4]. For these reasons we study arbitrary pPDA in this paper, but also specialize our results (and in particular the complexity of algorithms) to so-called pBPA, which are pPDA with a single control state. As we shall see, while some algorithms are polynomial for pBPA, this is unlikely to be the case for pPDA.

Our contribution. We specifically address the problem of computing $\mathcal{P}(M \geq n)$, or at least an upper bound, for a given n. This allows us to determine the size that the stack (or the store for threads awaiting execution) must have in order to guarantee that the probability of a memory overflow does not exceed a given bound. In Section 3 we obtain a system of recurrence equations for $\mathcal{P}(M \geq n)$, and show that for a pPDA with set Q of control states and set Γ of stack symbols, $\mathcal{P}(M \geq n)$ can be computed in time $\mathcal{O}\left(n \cdot (|Q|^2 \cdot |\Gamma|)^3\right)$ (time $\mathcal{O}\left(n \cdot |\Gamma|^3\right)$ for pBPAs) in the Blum-Shub-Smale model, the computation model in which an arithmetic operation takes one time unit, independently of the size of the operands. How-

ever, this result does not provide any information on the asymptotic behavior of $\mathcal{P}(M \geq n)$ when n grows, and moreover the algorithm is computationally inefficient for large values of n. We address these problems for pPDA in which the expected value of M is finite. We show in Section 3.2 that in this case $\mathcal{P}(M \geq n) \in \Theta(\rho^n)$, where $\rho < 1$ is the spectral radius of a certain matrix. This determines the exact asymptotic behavior up to a constant, and also leads to an algorithm for computing a bound on $\mathcal{P}(M \geq n)$ with logarithmic runtime in n.

Then we turn to computing the expectation of M. In Section 3.3 we provide an algorithm that approximates the expectation, give an error bound and analyze its convergence speed. Finally, in Section 4 we study the problem of deciding whether the expected value of M is finite. We show that the problem is polynomial for pBPAs. For arbitrary pPDA we show that the problem is in PSPACE and at least as hard as the SQRT-SUM and PosSLP problems. Notice that already the problem of deciding if the termination probability of a pPDA exceeds a given bound has this same complexity.

The full version of this paper [5] includes the proofs and more discussion.

Related work. Much work has been done also on the analysis of other well-structured infinite-state Markov chains, such as quasi-birth-death processes and Jackson queueing networks [16] and probabilistic lossy channel systems [17]. However, none of these classes contain pPDA or even pBPA. There is also work on general infinite-state (continuous-time) Markov chains which analyzes the behavior of the chain up to a finite depth [12]. This method is very general, but it is inefficient for pPDA, because it has not been designed to exploit the pushdown structure. Our analysis techniques are strongly based on linear algebra and matrix theory, in particular Perron-Frobenius theory [2]. The closest work to ours is [11] which also uses Perron-Frobenius theory for assessing the termination probability of recursive Markov chains.

2 Preliminaries

In the rest of this paper, $\mathbb{N}$ and $\mathbb{R}$ denote the set of positive integers and real numbers, respectively. The set of all finite words over a given alphabet Σ is denoted by Σ^*, and the set of all infinite words over Σ is denoted by Σ^ω. We write ε for the empty word. Given two sets $K \subseteq \Sigma^*$ and $L \subseteq \Sigma^* \cup \Sigma^\omega$, we use $K \cdot L$ (or just KL) to denote the concatenation of K and L, i.e., $KL = \{ww' \mid w \in K, w' \in L\}$. The length of a given $w \in \Sigma^* \cup \Sigma^\omega$ is denoted by $|w|$, where the length of an infinite word is ∞. Given a word (finite or infinite) over Σ, the individual letters of w are denoted by $w(0), w(1), \ldots$

Vectors and Matrices. Given a set S, we regard the elements of $\mathbb{R}^S$ as *vectors*. We use bold letters, like $\mathbf{u}$, for vectors. The vector whose components are all 0 (resp. all 1) is denoted by $\mathbf{0}$ (resp. $\mathbf{1}$). We write $\mathbf{u} = \mathbf{v}$ (resp. $\mathbf{u} \leq \mathbf{v}$) if $\mathbf{u}(s) = \mathbf{v}(s)$ (resp. $\mathbf{u}(s) \leq \mathbf{v}(s)$) holds for all $s \in S$. If $S' \subseteq S$, we write $\mathbf{u}_{|S'}$ for the vector $\mathbf{v} \in \mathbb{R}^{S'}$ with $\mathbf{v}(s) = \mathbf{u}(s)$ for all $s \in S'$.

Given two vector spaces $\mathbb{R}^S, \mathbb{R}^T$ we identify a linear function $A : \mathbb{R}^S \to \mathbb{R}^T$ with its corresponding matrix $A \in \mathbb{R}^{T \times S}$. We use capital letters for matrices and I for the identity matrix. We call a matrix nonnegative if all its entries are nonnegative. For nonnegative square matrices $A \in \mathbb{R}^{S \times S}$ we define the matrix sum $A^* = \sum_{i=0}^{\infty} A^i = I + A + AA + \cdots$. It is a well-known fact (see e.g. [13]) that A^* converges (or "exists") iff $\rho(A) < 1$, where

$\rho(A)$ denotes the spectral radius of A, i.e., the largest absolute value of the eigenvalues of A. Perron-Frobenius theory for nonnegative matrices (see e.g. [2]) states that $\rho(A)$ is an eigenvalue of A. If A^* exists, then $A^* = (I - A)^{-1}$.

Markov Chains. Our models of interest induce (infinite-state) Markov chains.

DEFINITION 1. *A* Markov chain *is a triple $M = (S, \to, Prob)$ where S is a finite or countably infinite set of* states*, $\to \subseteq S \times S$ is a* transition relation*, and $Prob$ is a function which to each transition $s \to t$ of M assigns its probability $Prob(s \to t) > 0$ so that for every $s \in S$ we have $\sum_{s \to t} Prob(s \to t) = 1$ (as usual, we write $s \xrightarrow{x} t$ instead of $Prob(s \to t) = x$).*

A *path* in M is a finite or infinite word $w \in S^+ \cup S^\omega$ such that $w(i-1) \to w(i)$ for every $1 \leq i < |w|$. A *run* in M is an infinite path in M. We denote by $Run[M]$ the set of all runs in M. The set of all runs that start with a given finite path w is denoted by $Run[M](w)$. When M is understood, we write Run (or $Run(w)$) instead of $Run[M]$ (or $Run[M](w)$, resp.).

To every $s \in S$ we associate the probability space $(Run(s), \mathcal{F}, \mathcal{P})$ where $\mathcal{F}$ is the σ-field generated by all *basic cylinders* $Run(w)$ where w is a finite path starting with s, and $\mathcal{P} : \mathcal{F} \to [0,1]$ is the unique probability measure such that $\mathcal{P}(Run(w)) = \Pi_{i=1}^{|w|-1} x_i$ where $w(i-1) \xrightarrow{x_i} w(i)$ for every $1 \leq i < |w|$. If $|w| = 1$, we put $\mathcal{P}(Run(w)) = 1$. Only certain subsets of $Run(s)$ are $\mathcal{P}$-measurable, but in this paper we only deal with "safe" subsets that are guaranteed to be in $\mathcal{F}$. Given $s \in S$ and $A \subseteq S$, we say *A is reachable from s* if $\mathcal{P}(\{w \in Run(s) \mid \exists i \geq 0 : w(i) \in A\}) > 0$.

Probabilistic Pushdown Automata (pPDA).

DEFINITION 2. *A* probabilistic pushdown automaton (pPDA) *is a tuple $\Delta = (Q, \Gamma, \delta, Prob)$ where Q is a finite set of* control states*, Γ is a finite* stack alphabet*, $\delta \subseteq Q \times \Gamma \times Q \times \Gamma^{\leq 2}$ (where $\Gamma^{\leq 2} = \{\alpha \in \Gamma^*, |\alpha| \leq 2\}$) is a* transition relation*, and $Prob$ is a function which to each transition $pX \to q\alpha$ assigns a rational probability $Prob(pX \to q\alpha) > 0$ so that for all $p \in Q$ and $X \in \Gamma$ we have that $\sum_{pX \to q\alpha} Prob(pX \to q\alpha) = 1$ (as usual, we write $pX \xrightarrow{x} q\alpha$ instead of $Prob(pX \to q\alpha) = x$).*

Elements of $Q \times \Gamma^*$ are called *configurations* of Δ. A pPDA with just one control state is called pBPA (pBPAs correspond to 1-exit recursive Markov chains defined in [11]). In what follows, configurations of pBPAs are usually written without the control state (i.e., we write only α instead of $p\alpha$).

EXAMPLE 3 *As a running example we choose the pBPA $\Delta = (\{p\}, \{X, Y, Z, W\}, \delta, Prob)$ with δ and $Prob$ given as follows.*

$$\begin{array}{llll} X \xrightarrow{1/4} \varepsilon & X \xrightarrow{1/4} Y & Y \xrightarrow{2/3} \varepsilon & Z \xrightarrow{1} Z \\ X \xrightarrow{1/4} XX & X \xrightarrow{1/4} Z & Y \xrightarrow{1/3} YY & W \xrightarrow{1} YW \end{array}$$

We can interpret this example as a model of a multithreaded system with four kinds of threads. Notice that threads of type Z and W do not terminate (our results also deal with this possibility). We are interested in the minimal number of threads n such that the probability that the execution of X requires to store more than n threads is at most 10^{-5}.

We define the size $|\Delta|$ of a pPDA Δ as follows: $|\Delta| = |Q| + |\Gamma| + |\delta| + |Prob|$ where $|Prob|$ equals the sum of sizes of binary representations of values of $Prob$. To Δ we associate the Markov chain M_Δ with $Q \times \Gamma^*$ as set of states and transitions defined as follows:

- $p\varepsilon \xrightarrow{1} p\varepsilon$ for each $p \in Q$;
- $pX\beta \xrightarrow{x} q\alpha\beta$ is a transition of M_Δ iff $pX \xrightarrow{x} q\alpha$ is a transition of Δ.

Given $p, q \in Q$ and $X \in \Gamma$, we often write pXq to denote (p, X, q). Given pXq we define

$$Run(pXq) = \{w \in Run(pX) \mid \exists i \geq 0 : w(i) = q\varepsilon\} \quad \text{and} \quad [pXq] = \mathcal{P}(Run(pXq)) .$$

Maximal Stack Height. Given $p\alpha \in Q \times \Gamma^*$, we denote by $height(p\alpha) = |\alpha|$ the stack height of $p\alpha$. Given $pX \in Q \times \Gamma$, the maximal stack height of a run is defined by setting

$$M_{pX}(w) = \sup\{height(w(i)) \mid i \geq 0\} \quad \text{for all runs } w \in Run(pX).$$

It is easy to show that for all $n \in \mathbb{N} \cup \{\infty\}$ the set $M_{pX}^{-1}(n) = \{w \in Run(pX) \mid M_{pX}(w) = n\}$ is measurable. Hence the expectation EM_{pX} of M_{pX} exists and we have

$$EM_{pX} = \sum_{n \in \mathbb{N} \cup \{\infty\}} n \cdot \mathcal{P}(M_{pX}^{-1}(n)) .$$

For what follows, we fix a pPDA $\Delta = (Q, \Gamma, \delta, Prob)$ with initial configuration $p_0X_0 \in Q \times \Gamma$. We are interested in the random variable $M_{p_0X_0}$ modelling the memory consumption of Δ. We wish to compute or approximate the distribution of $M_{p_0X_0}$ and its expectation.

3 Computing the Memory Consumption

The problem of computing the distribution of the maximal stack height is the problem of computing the probability of reaching a given height. So, for every $n \geq 1$ we define a vector $\mathbf{P}[n] \in \mathbb{R}^{Q \times \Gamma}$ with

$$\mathbf{P}[n](pX) = \mathcal{P}(\{w \in Run(pX) \mid M_{pX}(w) \geq n\}) \quad \text{for every } pX \in Q \times \Gamma ,$$

i.e., $\mathbf{P}[n](pX)$ is the probability that the maximal stack height is $\geq n$ in a run of $Run(pX)$.

There is a "naive" method to compute $\mathbf{P}[n](p_0X_0)$. (Recall that M_Δ is the Markov chain associated with Δ.) First, compute the Markov chain M_Δ^{n+1} obtained from M_Δ by restricting it to the states with a height of at most $n+1$. Note that M_Δ^{n+1} has finitely many states. Then compute $\mathbf{P}[n](p_0X_0)$ by computing the probability of reaching a state of height $n+1$ starting from p_0X_0. This can be done as usual by solving a linear equation system. The problem with this approach is that the number of states in M_Δ^{n+1} is $\Theta(|Q| \cdot |\Gamma|^n)$, i.e., exponential in n, and the linear equation system has equally many equations.

A better algorithm is obtained by observing that the Markov chain induced by a pPDA has a certain regular structure. We exploit this to get rid of the state explosion in the "naive" method. (This has also been observed in the analysis of other structured infinite-state systems, see e.g. [16].) In the following we describe the improved method, which is based on linear recurrences. We are mainly interested in the probabilities $\mathbf{P}[n]$ to reach height n, but as

an auxiliary quantity we use the probability of not exceeding height n in terminating runs. Formally, for every $n \geq 0$ we define a vector $\mathbf{T}[n] \in \mathbb{R}^{Q \times \Gamma \times Q}$ such that

$$\mathbf{T}[n](pXq) = \mathcal{P}(\{w \in Run(pXq) \mid M_{pX}(w) \leq n\}) \quad \text{for every } pXq \in Q \times \Gamma \times Q\,,$$

i.e., $\mathbf{T}[n](pXq)$ is the probability of all runs of $Run(pX)$ that terminate at q and do not exceed the height n. To every $pXq \in Q \times \Gamma \times Q$ we associate a variable $\mathbf{T}\langle n\rangle(pXq)$. Consider the following equation system: If $\mathbf{T}[n](pXq) = 0$, then we put $\mathbf{T}\langle n\rangle(pXq) = 0$. Otherwise, we put

$$\mathbf{T}\langle n\rangle(pXq) = \sum_{pX \xrightarrow{y} q\varepsilon} y + \sum_{pX \xrightarrow{y} rY} y\mathbf{T}\langle n\rangle(rYq) + \sum_{pX \xrightarrow{y} rYZ} \sum_{s \in Q} y\mathbf{T}[n-1](rYs)\mathbf{T}\langle n\rangle(sZq)\,.$$

PROPOSITION 4. *For $n \geq 0$, the vector $\mathbf{T}[n]$ is the* unique *solution of that equation system.*

The values $\mathbf{T}[n]$ can be used to set up an equation system for $\mathbf{P}[n]$. To every $pX \in Q \times \Gamma$ we associate a variable $\mathbf{P}\langle n\rangle(pX)$. Consider the following equation system: We put $\mathbf{P}\langle 1\rangle(pX) = 1$. If $\mathbf{P}[n](pX) = 0$, then we put $\mathbf{P}\langle n\rangle(pX) = 0$. Otherwise, we put

$$\mathbf{P}\langle n\rangle(pX) = \sum_{pX \xrightarrow{y} qY} y\mathbf{P}\langle n\rangle(qY) + \sum_{pX \xrightarrow{y} qYZ} y\mathbf{P}[n-1](qY) + \sum_{pX \xrightarrow{y} qYZ} \sum_{r \in Q} y\mathbf{T}[n-2](qYr)\mathbf{P}\langle n\rangle(rZ)\,.$$

PROPOSITION 5. *For $n \geq 1$, the vector $\mathbf{P}[n]$ is the* unique *solution of that equation system.*

EXAMPLE 6 *In our example we have for $n \geq 1$*

$$\begin{aligned}
\mathbf{T}[n](X) &= 1/4 + 1/4\ \mathbf{T}[n](Y) + 1/4\ \mathbf{T}[n](Z) + 1/4\ \mathbf{T}[n-1](X)\mathbf{T}[n](X)\\
\mathbf{T}[n](Y) &= 2/3 + 1/3\ \mathbf{T}[n-1](Y)\mathbf{T}[n](Y)\\
\mathbf{T}[n](Z) &= 0\\
\mathbf{T}[n](W) &= 0
\end{aligned}$$

and for $n \geq 2$

$$\begin{aligned}
\mathbf{P}[n](X) &= 1/4\ \mathbf{P}[n](Y) + 1/4\ \mathbf{P}[n](Z) + 1/4\ \mathbf{P}[n-1](X) + 1/4\ \mathbf{T}[n-2](X)\mathbf{P}[n](X)\\
\mathbf{P}[n](Y) &= 1/3\ \mathbf{P}[n-1](Y) + 1/3\ \mathbf{T}[n-2](Y)\mathbf{P}[n](Y)\\
\mathbf{P}[n](Z) &= 0\\
\mathbf{P}[n](W) &= \mathbf{P}[n-1](Y) + \mathbf{T}[n-2](Y)\mathbf{P}[n](W)\,.
\end{aligned}$$

Solving those systems successively for increasing n shows that $n = 17$ is the smallest number n such that $\mathbf{P}[n](X) \leq 10^{-5}$. In the interpretation as a multithreaded system this means that the probability that 17 or more threads need to be stored is at most 10^{-5}.

Using the above equation systems, we can compute $\mathbf{T}[n]$ and $\mathbf{P}[n]$ iteratively for increasing n by plugging in the values obtained in earlier iterations. The cost of each iteration is dominated by solving the equation system for $\mathbf{T}[n]$, which can be done, using Gaussian elimination, in time $\mathcal{O}\left((|Q|^2 \cdot |\Gamma|)^3\right)$ in the Blum-Shub-Smale model. So the total time to compute $\mathbf{P}[n]$ is linear in n.

PROPOSITION 7. *The value* $\mathbf{P}[n]$ *can be computed by setting up and solving the equation systems of Propositions 4 and 5 in time* $\mathcal{O}\left(n \cdot (|Q|^2 \cdot |\Gamma|)^3\right)$ *in the Blum-Shub-Smale model.*

The values $\mathbf{P}[n]$ that can be computed by Proposition 7 also allow to approximate the expectation $EM_{p_0X_0}$: Since $EY = \sum_{n=1}^{\infty} \mathcal{P}(Y \geq n)$ holds for any random variable Y with values in $\mathbb{N}$, we have $EM_{p_0X_0} = \sum_{n=1}^{\infty} \mathbf{P}[n](p_0X_0)$, so one can approximate $EM_{p_0X_0}$ by computing $\sum_{n=1}^{k} \mathbf{P}[n](p_0X_0)$ for some finite k.

Proposition 7 is simple and effective, but not fully satisfying for several reasons. First, it does not indicate how fast $\mathbf{P}[n](p_0X_0)$ decreases (if at all) for increasing n. Second, although computing $\mathbf{P}[n]$ using Proposition 7 is more efficient than using the "naive" method, it may still be too costly for large n, especially if Q or Γ are large. Instead, one may prefer an upper bound on $\mathbf{P}[n]$ if it is fast to compute. Finally, we wish for an approximation method for $EM_{p_0X_0}$ that comes with an error bound.

In the following we achieve these goals for pPDAs in which the expected memory consumption is finite. So we assume the following on the pPDA Δ for the rest of the section.
ASSUMPTION: The expectation $EM_{p_0X_0}$ is finite.
Notice that from the practical point of view this is a mild assumption: systems with infinite expected memory consumption also have infinite expected running time, and are unlikely to be considered suitable in reasonable scenarios. In Section 4 we show that whether $EM_{p_0X_0}$ is finite can be decided in polynomial time for pBPA, but also that this problem is unlikely to be decidable in polynomial time for general pPDA.

3.1 The Matrix A

This subsection leads to a matrix A which is crucial for our analysis. It is useful to get rid of certain irregularities in the equation systems of Propositions 4 and 5. The following lemma shows that the variables in the equation systems do not change from 0 to positive (or from positive to 0) if n is sufficiently large. (Recall that, by definition, $\mathbf{T}[n] \leq \mathbf{T}[n+1]$ and $\mathbf{P}[n] \geq \mathbf{P}[n+1]$ for all $n \geq 1$.)

LEMMA 8.
1. $\mathbf{T}[|Q|^2|\Gamma|+1](pXq) > 0 \iff$ *for all* $n \geq |Q|^2|\Gamma|+1 : \mathbf{T}[n](pXq) > 0 \iff [pXq] > 0$;
2. $\mathbf{P}[|Q||\Gamma|+1](pX) > 0 \iff$ *for all* $n \geq 1 : \mathbf{P}[n](pX) > 0$.

Another irregularity can be removed by restricting $\mathbf{T}[n]$ and $\mathbf{P}[n]$ to their "interesting" components; in particular, we filter out entries of $\mathbf{P}[n]$ that cannot create large stacks. Let $\mathcal{T} \subseteq Q \times \Gamma \times Q$ denote the set of all pXq such that $pX\Gamma^*$ is reachable from p_0X_0, and $[pXq] > 0$. Let $\mathcal{H} \subseteq Q \times \Gamma$ denote the set of all pX such that $pX\Gamma^*$ is reachable from p_0X_0, and $\mathbf{P}[n](pX) > 0$ for all $n \geq 1$.

LEMMA 9. *The sets* $\mathcal{T}$ *and* $\mathcal{H}$ *are computable in polynomial time.*

EXAMPLE 10 *For our running example, we fix X as the initial configuration. Then* $W\Gamma^*$ *is not reachable and* $\mathbf{P}[n](Z) = 0$ *for* $n \geq 2$, *hence* $\mathcal{H} = \{X, Y\}$. *Furthermore,* $\mathcal{T} = \{X, Y\}$.
We define $\mathbf{t}[n] \in \mathbb{R}^{\mathcal{T}}$ by $\mathbf{t}[n] := \mathbf{T}[n]_{|\mathcal{T}}$, i.e., $\mathbf{t}[n] \in \mathbb{R}^{\mathcal{T}}$ is the restriction of $\mathbf{T}[n]$ to $\mathcal{T}$. Similarly, we define $\mathbf{p}[n] := \mathbf{P}[n]_{|\mathcal{H}}$. Now we bring the equation systems for $\mathbf{t}[n]$ and $\mathbf{p}[n]$ from Propositions 4 and 5 in a compact matrix form.

For $\mathbf{t}[n]$, we define a vector $\mathbf{c} \in \mathbb{R}^{\mathcal{T}}$, a linear function $\tilde{L}$ on $\mathbb{R}^{\mathcal{T}}$, and a bilinear function $\tilde{Q} : \mathbb{R}^{\mathcal{T}} \times \mathbb{R}^{\mathcal{T}} \to \mathbb{R}^{\mathcal{T}}$ as follows:

$$(\mathbf{c})(pXq) = \sum_{pX \xrightarrow{y} q\varepsilon} y \qquad (\tilde{L}\mathbf{v})(pXq) = \sum_{pX \xrightarrow{y} rY,\ rYq \in \mathcal{T}} y\mathbf{v}(rYq)$$
$$(\tilde{Q}(\mathbf{u},\mathbf{v}))(pXq) = \sum_{pX \xrightarrow{y} rYZ} \ \sum_{s \in Q,\ rYs \in \mathcal{T},\ sZq \in \mathcal{T}} y\mathbf{u}(rYs)\mathbf{v}(sZq)$$

By $\tilde{Q}(\mathbf{u},\cdot)$ we denote a linear function satisfying $\tilde{Q}(\mathbf{u},\cdot)(\mathbf{v}) = \tilde{Q}(\mathbf{u},\mathbf{v})$.

For $\mathbf{p}[n]$, we define linear functions L and L' on $\mathbb{R}^{\mathcal{H}}$, and a bilinear function $Q : \mathbb{R}^{\mathcal{T}} \times \mathbb{R}^{\mathcal{H}} \to \mathbb{R}^{\mathcal{H}}$ as follows:

$$(L\mathbf{v})(pX) = \sum_{pX \xrightarrow{y} qY,\ qY \in \mathcal{H}} y\mathbf{v}(qY) \qquad (L'\mathbf{v})(pX) = \sum_{pX \xrightarrow{y} qYZ,\ qY \in \mathcal{H}} y\mathbf{v}(qY)$$
$$(Q(\mathbf{u},\mathbf{v}))(pX) = \sum_{pX \xrightarrow{y} qYZ} \ \sum_{r \in Q,\ qYr \in \mathcal{T},\ rZ \in \mathcal{H}} y\mathbf{u}(qYr)\mathbf{v}(rZ)$$

By $Q(\mathbf{u},\cdot)$ we denote a linear function satisfying $Q(\mathbf{u},\cdot)(\mathbf{v}) = Q(\mathbf{u},\mathbf{v})$.
Using Propositions 4 and 5 we obtain for $n \geq |Q|^2|\Gamma| + 3$ (recall Lemma 8):

PROPOSITION 11. *The following equations hold for all $n \geq |Q|^2|\Gamma| + 3$:*

$$\mathbf{t}[n] = \mathbf{c} + \tilde{L}\mathbf{t}[n] + \tilde{Q}(\mathbf{t}[n-1], \mathbf{t}[n]) \quad \text{and} \quad \mathbf{p}[n] = L\mathbf{p}[n] + L'\mathbf{p}[n-1] + Q(\mathbf{t}[n-2], \mathbf{p}[n])$$

EXAMPLE 12 *In our example we have for $n \geq 1$*

$$\mathbf{t}[n] = \overbrace{\begin{pmatrix} 1/4\ \mathbf{t}[n-1](X) & 1/4 \\ 0 & 1/3\ \mathbf{t}[n-1](Y) \end{pmatrix}}^{\tilde{L}+\tilde{Q}(\mathbf{t}[n-1],\cdot)} \mathbf{t}[n] + \overbrace{\begin{pmatrix} 1/4 \\ 2/3 \end{pmatrix}}^{\mathbf{c}}$$

and for $n \geq 2$

$$\mathbf{p}[n] = \overbrace{\begin{pmatrix} 1/4\ \mathbf{t}[n-2](X) & 1/4 \\ 0 & 1/3\ \mathbf{t}[n-2](Y) \end{pmatrix}}^{L+Q(\mathbf{t}[n-2],\cdot)} \mathbf{p}[n] + \overbrace{\begin{pmatrix} 1/4 & 0 \\ 0 & 1/3 \end{pmatrix}}^{L'} \mathbf{p}[n-1]\,.$$

Unlike $\mathbf{P}[n]$, the vector $\mathbf{p}[n]$ can be expressed in the form $A_n\mathbf{p}[n-1]$ for a suitable matrix A_n:

PROPOSITION 13. *Let $A_n := (L + Q(\mathbf{t}[n-2],\cdot))^* L'$. Then for every $n \geq |Q|^2|\Gamma| + 3$ the matrix A_n exists and $\mathbf{p}[n] = A_n\mathbf{p}[n-1]$.*

The key of our further analysis is to replace the matrix A_n by $A = \lim_{n\to\infty} A_n$. Since $A_n = (L + Q(\mathbf{t}[n-2],\cdot))^* L'$, we have

$$A := (L + Q(\mathbf{t},\cdot))^* L'$$

where we define $\mathbf{t} = \lim_{n\to\infty} \mathbf{t}[n]$. (Observe that $\mathbf{t}(pXq) = [pXq]$.) It is not immediate from Proposition 13 that A exists, but it can be proved:

PROPOSITION 14. *The matrix A exists and its spectral radius ρ satisfies $\rho < 1$.*

Proposition 14 is the technical core of this paper. Its proof is quite involved and relies on Perron-Frobenius theory [2]. We give a proof sketch and a full proof in [5].

EXAMPLE 15 *The termination probabilities $\mathbf{t}$ can be computed as the least solution of a nonlinear equation system [8, 11]. Applied to our example we obtain $\mathbf{t}(X) = 2 - \sqrt{2} \approx 0.586$ and $\mathbf{t}(Y) = 1$. Basic computations yield the following matrix A whose spectral radius is $\rho = 1/2$.*

$$A = \begin{pmatrix} 1/(2+\sqrt{2}) & 1/(4+2\sqrt{2}) \\ 0 & 1/2 \end{pmatrix}$$

3.2 Approximating the Distribution and a Tail Bound

We can assume $p_0 X_0 \in \mathcal{H}$ in the following, because otherwise, by Lemma 8, we would have $\mathbf{P}[n](p_0 X_0) = 0$ for $n \geq |Q|^2|\Gamma| + 3$, removing any need for further analysis.
The following theorem suggests an efficient approximation algorithm.

THEOREM 16. *Let $n_\perp := |Q|^2|\Gamma| + 3$ and $\widehat{\mathbf{p}}[n] := \mathbf{p}[n]$ for $n < n_\perp$ and $\widehat{\mathbf{p}}[n_\perp + n] := A^n \mathbf{p}[n_\perp]$ for $n \geq 0$. Then $\mathbf{p}[n] \leq \widehat{\mathbf{p}}[n]$ holds for all $n \geq 1$. Moreover, there exists d with $0 < d \leq 1$ and*

$$d \cdot \widehat{\mathbf{p}}[n](p_0 X_0) \leq \mathbf{p}[n](p_0 X_0) \leq \widehat{\mathbf{p}}[n](p_0 X_0) \,.$$

The proposition shows that $\mathbf{p}[n](p_0 X_0)$ and the approximation $\widehat{\mathbf{p}}[n](p_0 X_0)$ differ at most by a constant factor. Given A, the matrix powers A^n can be computed by repeated squaring, which allows to compute this upper bound in time $\mathcal{O}\left((|Q| \cdot |\Gamma|)^3 \cdot \log n\right)$ in the Blum-Shub-Smale model. To compute $A = (L + Q(\mathbf{t}, \cdot))^* L'$ itself, we can compute the matrix star via the matrix inverse, as stated in the preliminaries. Computing the vector $\mathbf{t}$ of termination probabilities requires a more detailed discussion. The vector is the least solution of a nonlinear equation system, and its components may be irrational and even non-expressible by radicals [8, 11]. However, there are several ways to compute at least upper bounds on $\mathbf{t}$ (which suffices to obtain upper bounds on $\mathbf{p}[n]$, as A depends monotonically on $\mathbf{t}$), or lower-bound approximations sufficiently accurate for all practical purposes, see [5] for a discussion. Theorem 16 provides a tail bound for $\mathbf{p}[n](p_0 X_0)$:

COROLLARY 17. *We have $\mathbf{p}[n](p_0 X_0) \in \Theta(\rho^n)$.*

EXAMPLE 18 *Since in our example Proposition 13 holds already for $n \geq 2$, we have $\widehat{\mathbf{p}}[n] = A^{n-1}\mathbf{1}$ for $n \geq 1$. With the matrix A from Example 15 and using $\mathbf{p}[n] \leq \widehat{\mathbf{p}}[n]$ we obtain:*

$$\mathbf{p}[2] \leq 0.5 \cdot \mathbf{1}, \quad \mathbf{p}[5] \leq 0.07 \cdot \mathbf{1}, \quad \mathbf{p}[17] \leq 10^{-4} \cdot \mathbf{1}, \quad \mathbf{p}[65] \leq 10^{-19} \cdot \mathbf{1}, \quad \ldots$$

Binary search can be used to determine that $n = 18$ is the least number n for which $\mathbf{p}[n] \leq \widehat{\mathbf{p}}[n] \leq 10^{-5} \cdot \mathbf{1}$ holds, so the comparison with Example 6 shows that the overapproximation is quite tight here. As $\rho = 1/2$, Corollary 17 yields $\mathbf{p}[n](p_0 X_0) \in \Theta(1/2^n)$.

3.3 Approximating the Expectation

We define an approximation method for the expectation $EM_{p_0X_0}$, and bound its error. As mentioned below Proposition 7, we have $EM_{p_0X_0} = \sum_{n=1}^{\infty} \mathbf{p}[n](p_0X_0)$, which can be (under-) approximated by the partial sums $\sum_{n=1}^{k} \mathbf{p}[n](p_0X_0)$. The values $\mathbf{p}[n](p_0X_0)$ can be computed using Proposition 11.

The following theorem gives error bounds on this approximation method and shows that it converges linearly, i.e., the number of accurate bits (as defined in [14]) is a linear function of the number of iterations. (Recall for the following statement that for a vector $\mathbf{v} \in \mathbb{R}^{\mathcal{H}}$ its 1-norm $\|\mathbf{v}\|_1$ is defined as $\sum_{h\in\mathcal{H}} |\mathbf{v}(h)|$, and that for a matrix B its 1-norm $\|B\|_1$ is the maximal 1-norm of its columns.)

THEOREM 19. *Let* $UM_{p_0X_0}(k) := \sum_{n=1}^{k} \mathbf{p}[n](p_0X_0)$. *For all* $k \geq |Q|^2|\Gamma| + 3$

$$EM_{p_0X_0} - UM_{p_0X_0}(k) \leq \|A^*\|_1 \, \|\mathbf{p}[k]\|_1 \leq ab^k$$

where $a > 0$ *and* $0 < b < 1$ *are computable rational numbers. Hence, the sequence* $(UM_{p_0X_0}(k))_k$ *converges linearly to* $EM_{p_0X_0}$.

The computation procedure of the constants a and b from Theorem 19 is somewhat involved, but the first inequality of Theorem 19 gives concrete error bounds as well:

EXAMPLE 20 *Using Proposition 11 we compute* $\sum_{n=1}^{12} \mathbf{p}[n](X) = 1.5731\ldots$ *and furthermore* $\|\mathbf{p}[12]\|_1 \approx 0.00042$. *We have* $\|A^*\|_1 = 1 + \sqrt{2} \approx 2.4$. *Theorem 19 yields*

$$1.57 < EM_X \leq 1.5731\ldots + \|A^*\|_1 \cdot \|\mathbf{p}[12]\|_1 < 1.58\,.$$

4 Finiteness of the Expected Memory Consumption

In this section we study the complexity of the finite-expectation problem that asks whether the expectation of the memory consumption is finite.

4.1 Expected Memory Consumption of pPDA

For pPDA we can show the following theorem.

THEOREM 21. *The problem whether* $EM_{p_0X_0}$ *is finite is decidable in polynomial space.*

The proof is based on the following proposition which strengthens Proposition 14 from the previous section which stated that, under the assumption that $EM_{p_0X_0}$ is finite, the spectral radius ρ of A satisfies $\rho < 1$.

PROPOSITION 22. *Suppose* $\mathcal{P}(M_{p_0X_0} < \infty) = 1$. *Then the matrix* A *exists. Moreover, its spectral radius* ρ *satisfies* $\rho < 1$ *if and only if* $EM_{p_0X_0}$ *is finite.*

The condition $\mathcal{P}(M_{p_0X_0} < \infty) = 1$ can be checked in polynomial space [9]. If it does not hold, then clearly $EM_{p_0X_0} = \infty$. Otherwise one checks $\rho \geq 1$. Roughly speaking, this can be done in polynomial space because the matrix A is given in terms of the termination probabilities $\mathbf{t}$ which can be expressed in the existential theory of the reals.

We can also show that this upper complexity bound from Theorem 21 cannot be significantly lowered without a major breakthrough on long-standing and fundamental problems on numerical computations, namely the SQRT-SUM and the PosSLP problems [1, 11, 5]:

THEOREM 23. *The* PosSLP *problem is P-time many-one reducible to the decision problem whether the expected maximal height of a pPDA is finite.*

It follows that SQRT-SUM is (Turing) reducible to the finite-stack problem, because SQRT-SUM is (Turing) reducible to PosSLP [1, 11].

4.2 Expected Memory Consumption of pBPA

Now we show that for pBPA the finite-expectation problem can be decided in polynomial time. Let us fix a pBPA $\Delta = (\{p\}, \Gamma, \delta, Prob)$, and fix an initial configuration $X_0 \in \Gamma$. Let Γ_0 denote the set of all symbols $Y \in \Gamma$ such that $Y\Gamma^*$ is reachable from X_0. Let *Term* be the set of all symbols $X \in \Gamma_0$ such that $\mathbf{t}(X) = 1$, i.e., a run from a *Term*-symbol terminates almost surely. We define $NTerm = \Gamma_0 \setminus Term$. The following proposition follows from [11].

PROPOSITION 24. *The sets Term and NTerm can be computed in polynomial time.*

ALGORITHM DECIDING WHETHER EM_{X_0} IS FINITE:

1. Compute the sets *Term* and *NTerm* (using Proposition 24).
2. Decide in polynomial time [5] whether all $Y \in NTerm$ satisfy $\mathcal{P}(M_Y < \infty) = 1$. If no, then stop and return 'no'.
3. Decide in polynomial time [5] whether all $Y \in Term$ satisfy $EM_Y < \infty$. If no, then return 'no'. Otherwise return 'yes'.

THEOREM 25. *The above algorithm is polynomial. It returns 'yes' iff EM_{X_0} is finite.*

5 Conclusions

We have investigated the memory consumption of probabilistic pushdown automata (pPDA). Technically speaking, we have studied the random variable M returning the maximal stack height of a pPDA. In [9] a PSPACE algorithm was provided for deciding whether the runs with $M = \infty$ have nonzero probability, but the distribution of M and its expectation have not been studied. For computing the distribution of M, we have shown that the exponential blow-up of the naive method can be avoided using a system of linear equations. We have also provided an approximation method that gives upper bounds. This can be used, e.g., for providing space that suffices with a probability of, say, 99%.

Computing the expectation EM was mentioned in [9] as "harder problem" and left open. Using novel proof techniques, we have provided a rather complete solution. We have shown that whether the expected maximal stack height of a pBPA is finite can be decided in polynomial time, while for general pPDA the problem is in PSPACE. By means of a reduction to the PosSLP and SQRT-SUM problems we have furthermore shown that this complexity cannot be significantly lowered without major breakthroughs. Finally, we have defined an iterative method for approximating the expected maximal stack height, and

have shown that it converges linearly. The complexity of the decision problem $EM_{p_0X_0} < k$ for a finite bound k is an open question.

References

[1] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. In *IEEE Conference on Computational Complexity*, pages 331–339. IEEE Computer Society, 2006.

[2] A. Berman and R.J. Plemmons. *Nonnegative matrices in the mathematical sciences*. Academic Press, 1979.

[3] R.D. Blumofe and C.E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.

[4] A. Bouajjani and J. Esparza. Rewriting models of boolean programs. In *Proceedings of RTA 2006*, Seattle, USA, 2006.

[5] T. Brázdil, J. Esparza, and S. Kiefer. On the memory consumption of probabilistic pushdown automata. Technical Report FIMU-RS-2009-07, Masaryk University, 2009. Available at `http://www.fi.muni.cz/reports/files/2009/FIMU-RS-2009-07.pdf`.

[6] T. Brázdil, J. Esparza, and A. Kučera. Analysis and prediction of the long-run behavior of probabilistic sequential programs with recursion. In *FOCS'05*, pages 521–530, 2005.

[7] J. Esparza and K. Etessami. Verifying probabilistic procedural programs. In *FSTTCS 2004*, pages 16–31, 2004.

[8] J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. In *LICS 2004*, pages 12–21. IEEE, 2004.

[9] J. Esparza, A. Kučera, and R. Mayr. Quantitative analysis of probabilistic pushdown automata: Expectations and variances. In *LICS'05*, pages 117–126. IEEE, 2005.

[10] K. Etessami and M. Yannakakis. Recursive concurrent stochastic games. *Logical Methods in Computer Science*, 4(4), 2008.

[11] K. Etessami and M. Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1–66, 2009.

[12] E.M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. INFAMY: An infinite-state Markov model checker. In *CAV*, LNCS 5643, pages 641–647, 2009.

[13] R.A. Horn and C.A. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

[14] S. Kiefer, M. Luttenberger, and J. Esparza. On the convergence of Newton's method for monotone systems of polynomial equations. In *STOC 2007*, pages 217–226, 2007.

[15] G.J. Narlikar and G.E. Belloch. Space-efficient scheduling of nested parallelism. *ACM TOPLAS*, 21(1):138–173, 1999.

[16] A. Remke, B. Haverkort, and L. Cloth. CSL model checking algorithms for QBDs. *Theoretical Computer Science*, 382(1):24–41, 2007.

[17] P. Schnoebelen. The verification of probabilistic lossy channel systems. In *Validation of Stochastic Systems*, LNCS 2925, pages 445–465. Springer, 2004.

[18] M. Yannakakis and K. Etessami. Checking LTL properties of recursive Markov chains. In *QEST 2005*, pages 155–165, 2005.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Continuous-Time Stochastic Games with Time-Bounded Reachability

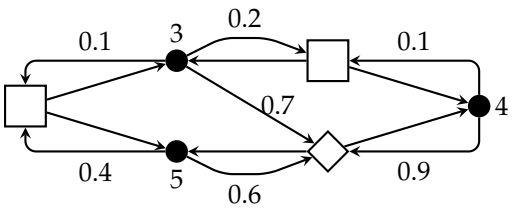
T. Brázdil, V. Forejt, J. Krčál, J. Křetínský, A. Kučera
Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic
{brazdil,forejt,krcal,xkretins,kucera}@fi.muni.cz

ABSTRACT. We study continuous-time stochastic games with time-bounded reachability objectives. We show that each vertex in such a game has a *value* (i.e., an equilibrium probability), and we classify the conditions under which optimal strategies exist. Finally, we show how to compute optimal strategies in finite uniform games, and how to compute ε-optimal strategies in finitely-branching games with bounded rates (for finite games, we provide detailed complexity estimations).

1 Introduction

Markov models are widely used in many diverse areas such as economics, biology, or physics. More recently, they have also been used for performance and dependability analysis of computer systems. Since faithful modeling of computer systems often requires both *randomized* and *non-deterministic* choice, a lot of attention has been devoted to Markov models where these two phenomena co-exist, such as *Markov decision processes* and *stochastic games*. The latter model of stochastic games is particularly apt for analyzing the interaction between a system and its environment, which are formalized as two *players* with antagonistic objectives (we refer to, e.g., [10, 5, 11] for more comprehensive expositions of results related to games in formal analysis and verification of computer systems). So far, most of the existing results concern *discrete-time* Markov decision processes and stochastic games, and the accompanying theory is relatively well-developed (see, e.g., [9, 4]).

In this paper, we study *continuous-time stochastic games (CTGs)* and hence also *continuous-time Markov decision processes (CTMDPs)* with time-bounded reachability objectives. Roughly speaking, a CTG is a finite or countably infinite graph with three types of vertices—controllable vertices (boxes), adversarial vertices (diamonds), and actions (circles). The outgoing edges of controllable and adversarial vertices lead to the actions that are *enabled* at a given vertex. The outgoing edges of actions lead to controllable or adversarial vertices, and every edge is assigned a positive probability so that the total sum of these probabilities is equal to 1. Further, each action is assigned a positive real *rate*. A simple finite CTG is shown below.



© Brázdil, Forejt, Krčál, Křetínský, Kučera; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 61–72
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2307

A game is played by two players, $\Box$ and $\Diamond$, who are responsible for selecting the actions (i.e., resolving the non-deterministic choice) in the controllable and adversarial vertices, respectively. The selection is timeless, but performing a selected action takes time which is exponentially distributed (the parameter is the rate of a given action). When a given action is finished, the next vertex is chosen randomly according to the fixed probability distribution over the outgoing edges of the action. A *time-bounded reachability objective* is specified by a set T of target vertices and a time bound $t > 0$. The goal of player $\Box$ is to maximize the probability of reaching a target vertex before time t, while player $\Diamond$ aims at minimizing this probability.

Note that events such as component failures, user requests, message receipts, exceptions, etc., are essentially history-independent, which means that the time between two successive occurrences of such events is exponentially distributed. CTGs provide a natural and convenient formal model for systems exhibiting these features, and time-bounded reachability objectives allow to formalize basic liveness and safety properties of these systems.

Previous work. Although the practical relevance of CTGs with time-bounded reachability objectives to verification problems is obvious, to the best of our knowledge there are no previous results concerning even very basic properties of such games. A more restricted model of uniform CTMDPs is studied in [2, 7]. Intuitively, a uniform CTMDP is a CTG where all non-deterministic vertices are controlled just by one player, and all actions are assigned the same rate. In [2], it is shown that the maximal and minimal probability of reaching a target vertex before time t is efficiently computable up to an arbitrarily small given error, and that the associated strategy is also effectively computable. An open question explicitly raised in [2] is whether this result can be extended to all (not necessarily uniform) CTMDP. In [2], it is also shown that time-dependent strategies are more powerful than time-abstract ones, and this issue is addressed in greater detail in [7] where the mutual relationship between various classes of time-dependent strategies in CTMDPs is studied. Furthermore, in [1] reward-bounded objectives in CTMDPs are studied.

Our contribution is twofold. Firstly, we examine the *fundamental properties* of CTGs, where we aim at obtaining as general (and tight) results as possible. Secondly, we consider the associated *algorithmic issues*. Concrete results are discussed in the following paragraphs.

Fundamental properties of CTGs. We start by showing that each vertex $\hat{v}$ in a CTG with time-bounded reachability objectives has a *value*, i.e., an *equilibrium probability* of reaching a target vertex before time t. The value is equal to $\sup_\sigma \inf_\pi \mathcal{P}^{\sigma,\pi}_{\hat{v}}(\mathit{Reach}^{\leq t}(T))$ and $\inf_\pi \sup_\sigma \mathcal{P}^{\sigma,\pi}_{\hat{v}}(\mathit{Reach}^{\leq t}(T))$, where σ and π range over all time-abstract strategies of player $\Box$ and player $\Diamond$, and $\mathcal{P}^{\sigma,\pi}_{\hat{v}}(\mathit{Reach}^{\leq t}(T))$ is the probability of reaching T before time t in a play obtained by applying the strategies σ and π. This result holds for *arbitrary* CTGs which may have countably many vertices and actions. This immediately raises the question whether each player has an *optimal* strategy which achieves the outcome equal to or better than the value against every strategy of the opponent. We show that the answer is negative in general, but an optimal strategy for player $\Diamond$ is guaranteed to exist in *finitely-branching* CTGs, and an optimal strategy for player $\Box$ is guaranteed to exist in *finitely-branching* CTGs with *bounded rates* (see Definition 2). These results are tight, which is documented by appropriate counterexamples. Moreover, we show that in the subclasses of CTGs just mentioned, the players have also optimal CD strategies (a strategy is CD if it is deterministic and "count-

ing", i.e., it only depends on the number of actions in the history of a play, where actions with the same rate are identified). Note that CD strategies still use infinite memory and in general they do not admit a finite description. A special attention is devoted to finite uniform CTGs, where we show a somewhat surprising result—both players have *finite memory optimal strategies* (these finite memory strategies are deterministic and their decision is based on "bounded counting" of actions; hence, we call them "BCD").

Algorithms. We show that for finite CTGs, ε-optimal strategies for both players are computable in $|V|^2 \cdot |A| \cdot bp \cdot (|\mathcal{R}|+1)^{\mathcal{O}((\max \mathcal{R}) \cdot t + \ln \frac{1}{\varepsilon})}$ time, where $|V|$ and $|A|$ is the number of vertices and actions, resp., bp is the maximum bit-length of transition probabilities and rates (we assume that rates and the probabilities in distributions assigned to the actions are represented as fractions of integers encoded in binary), $|\mathcal{R}|$ is the number of rates, $\max \mathcal{R}$ is the maximal rate, and t is the time bound. This solves the open problem of [2] (in fact, our result is more general as it applies to finite CTGs, not just to finite CTMDPs). Actually, the algorithm works also for *infinite-state* CTGs as long as they are finitely-branching, have bounded rates, and satisfy some natural "effectivity assumptions" (see Corollary 14). For example, this is applicable to the class of infinite-state CTGs definable by pushdown automata (where the rate of a given configuration depends just on the current control state), and also to other automata-theoretic models. Finally, we show how to compute the optimal BCD strategies for both players in finite uniform CTGs.

Due to space constraints, proofs are omitted here. Full proofs can be found in [3]. In the following we just try to indicate basic ideas behind the proofs. This is not always possible, because some arguments are tricky and occasionally we also rely on relatively advanced calculations. Nevertheless, the results themselves should be easy to understand.

2 Definitions

In this paper, the sets of all positive integers, non-negative integers, rational numbers, real numbers, non-negative real numbers, and positive real numbers are denoted by $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{R}^{\geq 0}$, and $\mathbb{R}^{>0}$, respectively. Let A be a finite or countably infinite set. A *probability distribution* on A is a function $f : A \to \mathbb{R}^{\geq 0}$ such that $\sum_{a \in A} f(a) = 1$. The *support* of f is the set of all $a \in A$ where $f(a) > 0$. A distribution f is *rational* if $f(a) \in \mathbb{Q}$ for every $a \in A$, *positive* if $f(a) > 0$ for every $a \in A$, and *Dirac* if $f(a) = 1$ for some $a \in A$. The set of all distributions on A is denoted by $\mathcal{D}(A)$. A *σ-field* over a set Ω is a set $\mathcal{F} \subseteq 2^\Omega$ that includes Ω and is closed under complement and countable union. A *measurable space* is a pair $(\Omega, \mathcal{F})$ where Ω is a set called *sample space* and $\mathcal{F}$ is a σ-field over Ω whose elements are called *measurable sets*. A *probability measure* over a measurable space $(\Omega, \mathcal{F})$ is a function $\mathcal{P} : \mathcal{F} \to \mathbb{R}^{\geq 0}$ such that, for each countable collection $\{X_i\}_{i \in I}$ of pairwise disjoint elements of $\mathcal{F}$, $\mathcal{P}(\bigcup_{i \in I} X_i) = \sum_{i \in I} \mathcal{P}(X_i)$, and moreover $\mathcal{P}(\Omega) = 1$. A *probability space* is a triple $(\Omega, \mathcal{F}, \mathcal{P})$, where $(\Omega, \mathcal{F})$ is a measurable space and $\mathcal{P}$ is a probability measure over $(\Omega, \mathcal{F})$. Given two measurable sets $X, Y \in \mathcal{F}$ such that $\mathcal{P}(Y) > 0$, the *conditional probability* of X under the condition Y is defined as $\mathcal{P}(X \mid Y) = \mathcal{P}(X \cap Y)/\mathcal{P}(Y)$. We say that a property $A \subseteq \Omega$ holds *for almost all* elements of a measurable set Y if $\mathcal{P}(Y) > 0$, $A \cap Y \in \mathcal{F}$, and $\mathcal{P}((A \cap Y) \mid Y) = 1$.

In our next definition we introduce continuous-time Markov chains (CTMCs). The

literature offers several equivalent definitions of CTMCs (see, e.g., [8]). For purposes of this paper, we adopt the variant where transitions have discrete probabilities and the rates are assigned to states.

DEFINITION 1. *A* continuous-time Markov chain (CTMC) *is a tuple* $\mathcal{M} = (M, \to, Prob, \mathbf{R}, Init)$*, where* M *is a finite or countably infinite set of* states*,* $\to \subseteq M \times M$ *is a* transition relation *such that every* $s \in M$ *has at least one outgoing transition,* $Prob$ *is a function which to each* $s \in M$ *assigns a positive probability distribution over the set of its outgoing transitions,* $\mathbf{R}$ *is a function which to each* $s \in M$ *assigns a positive real* rate*, and* $Init$ *is the* initial probability distribution *on* M*.*

We write $s \xrightarrow{x} s'$ to indicate that $s \to s'$ and $Prob(s)(s \to s') = x$. A *time-abstract path* is a finite or infinite sequence $u = u_0, u_1, \ldots$ of states such that $u_{i-1} \to u_i$ for every $1 \leq i < length(u)$, where $length(u)$ is the length of u (the length of an infinite sequence is ∞). A *timed path* (or just *path*) is a pair $w = (u, t)$, where u is a time-abstract path and $t = t_1, t_2, \ldots$ is a sequence of positive reals such that $length(t) = length(u)$. We put $length(w) = length(u)$, and for every $0 \leq i < length(w)$, we usually write $w(i)$ and $w[i]$ instead of u_i and t_i, respectively.

Infinite paths are also called *runs*. The set of all runs in $\mathcal{M}$ is denoted $Run_{\mathcal{M}}$, or just Run when $\mathcal{M}$ is clear from the context. A *template* is a pair (u, I), where $u = u_0, u_1, \ldots$ is a finite time-abstract path and $I = I_0, I_1, \ldots$ a finite sequence of non-empty intervals in $\mathbb{R}^{\geq 0}$ such that $length(u) = length(I)$. Every template (u, I) determines a *basic cylinder* $Run(u, I)$ consisting of all runs w such that $w(i) = u_i$ for all $0 \leq i < length(u)$, and $w[j] \in I_j$ for all $0 \leq i < length(u) - 1$. To $\mathcal{M}$ we associate the probability space $(Run, \mathcal{F}, \mathcal{P})$ where $\mathcal{F}$ is the σ-field generated by all basic cylinders $Run(u, I)$ and $\mathcal{P} : \mathcal{F} \to \mathbb{R}^{\geq 0}$ is the unique probability measure on $\mathcal{F}$ such that

$$\mathcal{P}(Run(u, I)) = Init(u_0) \cdot \prod_{i=0}^{length(u)-2} Prob(u_i)(u_i \to u_{i+1}) \cdot \left(e^{-\mathbf{R}(u_i) \cdot \sup(I_i)} - e^{-\mathbf{R}(u_i) \cdot \inf(I_i)} \right)$$

Note that if $length(u) = 1$, the "big product" above is empty and hence equal to 1.

Now we formally define continuous-time games, which generalize continuous-time Markov chains by allowing not only probabilistic but also *non-deterministic* choice. Continuous-time games also generalize the model of continuous-time Markov decision processes studied in [2, 7] by splitting the non-deterministic vertices into two disjoint subsets of *controllable* and *adversarial* vertices, which are controlled by two "players" with antagonistic objectives. Thus, one can model the interaction between a system and its environment.

DEFINITION 2. *A* continuous-time game (CTG) *is a tuple* $G = (V, A, \mathbf{E}, (V_\Box, V_\Diamond), \mathbf{P}, \mathbf{R})$ *where* V *is a finite or countably infinite set of* vertices*,* A *is a finite or countably infinite set of* actions*,* $\mathbf{E}$ *is a function which to every* $v \in V$ *assigns a non-empty set of actions* enabled *in* v*,* $(V_\Box, V_\Diamond)$ *is a partition of* V*,* $\mathbf{P}$ *is a function which assigns to every* $a \in A$ *a probability distribution on* V*, and* $\mathbf{R}$ *is a function which assigns a positive real* rate *to every* $a \in A$*.*

We require that $V \cap A = \emptyset$ *and use* N *to denote the set* $V \cup A$*. We say that* G *is* finitely-branching *if for each* $v \in V$ *the set* $\mathbf{E}(v)$ *is finite (note that* $\mathbf{P}(a)$ *for a given* $a \in A$ *can still have an infinite support.) We say that* G *has* bounded rates *if* $\sup_{a \in A} \mathbf{R}(a) < \infty$*, and that* G *is* uniform *if* $\mathbf{R}$ *is a constant function. Finally, we say that* G *is* finite *if both* V *and* A *are finite.*

If $V_\Box$ or $V_\Diamond$ is empty (i.e., there is just one type of vertices), then G is a *continuous-time Markov decision process (CTMDP).* Technically, our definition of CTMDP is slightly different from the one used in [2, 7], but the difference is only cosmetic. The two models are equivalent in a well-defined sense (a detailed explanation is included in [3]). Also note that $\mathbf{P}$ and $\mathbf{R}$ associate the probability distributions and rates directly to actions, not to pairs of $V \times A$. This is perhaps somewhat non-standard, but leads to simpler notation (since each vertex can have its "private" set of enabled actions, this is no restriction).

A *play* of G is initiated in some vertex. The non-deterministic choice is resolved by two players, $\Box$ and $\Diamond$, who select the actions in the vertices of $V_\Box$ and $V_\Diamond$, respectively. The selection itself is timeless, but some time is spent by performing the selected action (the time is exponentially distributed with the rate $\mathbf{R}(a)$), and then a transition to the next vertex is chosen randomly according to the distribution $\mathbf{P}(a)$. The players can also select the actions *randomly*, i.e., they select not just a single action but a *probability distribution* on the enabled actions. Moreover, the players are allowed to play differently when the same vertex is revisited. We assume that both players can see the history of a play, but cannot measure the elapsed time.

Let $\odot \in \{\Box, \Diamond\}$. A *strategy* for player $\odot$ is a function which to each $wv \in N^* V_\odot$ assigns a probability distribution on $\mathbf{E}(v)$. The sets of all strategies for player $\Box$ and player $\Diamond$ are denoted by Σ and Π, respectively. Each pair of strategies $(\sigma, \pi) \in \Sigma \times \Pi$ together with an initial vertex $\hat{v} \in V$ determine a unique *play* of the game G, which is a CTMC $G(\hat{v}, \sigma, \pi)$ where N^*A is the set of states, the rate of a given $wa \in N^*A$ is $\mathbf{R}(a)$ (the rate function of $G(\hat{v}, \sigma, \pi)$ is also denoted by $\mathbf{R}$), and transitions exist only between states of the form wa and $wava'$, where $wa \xrightarrow{x} wava'$ iff one of the following conditions is satisfied:

- $v \in V_\Box$, $a' \in \mathbf{E}(v)$, and $x = \mathbf{P}(a)(v) \cdot \sigma(wv)(a') > 0$
- $v \in V_\Diamond$, $a' \in \mathbf{E}(v)$, and $x = \mathbf{P}(a)(v) \cdot \pi(wv)(a') > 0$

The initial distribution is determined as follows:

- $\mathit{Init}(\hat{v}a) = \sigma(\hat{v})(a)$ if $\hat{v} \in V_\Box$ and $a \in \mathbf{E}(\hat{v})$;
- $\mathit{Init}(\hat{v}a) = \pi(\hat{v})(a)$ if $\hat{v} \in V_\Diamond$ and $a \in \mathbf{E}(\hat{v})$;
- in the other cases, *Init* returns zero.

Note that the set of states of $G(\hat{v}, \sigma, \pi)$ is infinite. Also note that all states reachable from a state $\hat{v}a$, where $\mathit{Init}(\hat{v}a) > 0$, are alternating sequences of vertices and actions. We say that a state w of $G(\hat{v}, \sigma, \pi)$ *hits* a vertex $v \in V$ if v is the last vertex which appears in w (for example, $v_1 a_1 v_2 a_2$ hits v_2). Further, we say that w hits $T \subseteq V$ if w hits some vertex of T. From now on, the paths (both finite and infinite) in $G(\hat{v}, \sigma, \pi)$ are denoted by Greek letters $\alpha, \beta, \ldots$. Note that for every $\alpha \in \mathit{Run}_{G(\hat{v},\sigma,\pi)}$ and every $i \in \mathbb{N}_0$ we have that $\alpha(i) = wa$ where $wa \in N^*A$.

We denote by $\mathcal{R}(G)$ the set of all rates used in G (i.e., $\mathcal{R}(G) = \{\mathbf{R}(a) \mid a \in A\}$), and by $\mathcal{H}(G)$ the set of all vectors of the form $\mathbf{i} : \mathcal{R}(G) \to \mathbb{N}_0$ satisfying $\sum_{r \in \mathcal{R}(G)} \mathbf{i}(r) < \infty$. When G is clear from the context, we write just $\mathcal{R}$ and $\mathcal{H}$ instead of $\mathcal{R}(G)$ and $\mathcal{H}(G)$, respectively. For every $\mathbf{i} \in \mathcal{H}$, we put $|\mathbf{i}| = \sum_{r \in \mathcal{R}} \mathbf{i}(r)$. For every $r \in \mathcal{R}$, we denote by $\mathbf{1}_r$ the vector of $\mathcal{H}$ such that $\mathbf{1}_r(r) = 1$ and $\mathbf{1}_r(r') = 0$ if $r' \neq r$. Further, for every $wx \in N^*N$ we define the vector $\mathbf{i}_{wx} \in \mathcal{H}$ such that $\mathbf{i}_{wx}(r)$ returns the cardinality of the set $\{j \in \mathbb{N}_0 \mid 0 \leq j < \mathit{length}(w), w(j) \in A, \mathbf{R}(w(j)) = r\}$ (Note that the last element x of wx is disregarded.) Given $\mathbf{i} \in \mathcal{H}$ and $wx \in N^*N$, we say that wx *matches* $\mathbf{i}$ if $\mathbf{i} = \mathbf{i}_{wx}$.

We say that a strategy τ is *counting (C)* if $\tau(wv) = \tau(w'v)$ for all $w, w' \in N^*$ such that $\mathbf{i}_{wv} = \mathbf{i}_{w'v}$. In other words, a strategy τ is counting if the only information about the history of a play w which influences the decision of τ is the vector $\mathbf{i}_{wv}$. Hence, every counting strategy τ can be considered as a function from $\mathcal{H} \times V$ to $\mathcal{D}(A)$, where $\tau(\mathbf{i}, v)$ corresponds to the value of $\tau(wv)$ for every wv matching $\mathbf{i}$. A counting strategy τ is *bounded counting (BC)* if there is $k \in \mathbb{N}$ such that $\tau(wv) = \tau(w'v)$ whenever $|w|, |w'| \geq k$. A strategy τ is *deterministic (D)* if $\tau(wv)$ is a Dirac distribution for all wv. Strategies that are not necessarily counting are called *history-dependent (H)*, and strategies that are not necessarily deterministic are called *randomized (R)*. Thus, we obtain the following six types of strategies: BCD, BCR, CD, CR, HD, and HR. The most general (unrestricted) type is HR, and the importance of the other types of strategies becomes clear in subsequent sections.

In this paper, we are interested in continuous-time games with *time-bounded reachability objectives*, which are specified by a set $T \subseteq V$ of *target* vertices and a *time bound* $t \in \mathbb{R}^{>0}$. The goal of player $\Box$ is to maximize the probability of reaching a target vertex before the time bound t, while player $\Diamond$ aims at minimizing this probability. Let $\hat{v}$ be the initial vertex. Then each pair of strategies $(\sigma, \pi) \in \Sigma \times \Pi$ determines a unique *outcome* $\mathcal{P}_{\hat{v}}^{\sigma,\pi}(Reach^{\leq t}(T))$, which is the probability of all $\alpha \in Run_{G(\hat{v},\sigma,\pi)}$ that visit T before time t (i.e., there is $i \in \mathbb{N}_0$ such that $\alpha(i)$ hits T and $\sum_{i=0}^{i-1} \alpha[i] \leq t$). A fundamental question (answered in Section 3) is whether continuous-time games with time-bounded reachability objectives have a *value*, i.e., a unique equilibrium outcome. We say that $\hat{v} \in V$ *has a value* if

$$\sup_{\sigma \in \Sigma} \inf_{\pi \in \Pi} \mathcal{P}_{\hat{v}}^{\sigma,\pi}(Reach^{\leq t}(T)) = \inf_{\pi \in \Pi} \sup_{\sigma \in \Sigma} \mathcal{P}_{\hat{v}}^{\sigma,\pi}(Reach^{\leq t}(T))$$

If $\hat{v}$ has a value, then $val(\hat{v})$ denotes the *value of* $\hat{v}$ defined by the above equality. Further, if $\hat{v}$ has a value, it makes sense to define *ε-optimal* and *optimal* strategies in $\hat{v}$. Let $\varepsilon \geq 0$. We say that a strategy $\sigma \in \Sigma$ is an *ε-optimal maximizing strategy in $\hat{v}$* (or just *ε-optimal in $\hat{v}$*) if

$$\inf_{\pi \in \Pi} \mathcal{P}_{\hat{v}}^{\sigma,\pi}(Reach^{\leq t}(T)) \;\geq\; val(\hat{v}) - \varepsilon\,,$$

and that a strategy $\pi \in \Pi$ is an *ε-optimal minimizing strategy in $\hat{v}$* (or just *ε-optimal in $\hat{v}$*) if

$$\sup_{\sigma \in \Sigma} \mathcal{P}_{\hat{v}}^{\sigma,\pi}(Reach^{\leq t}(T)) \;\leq\; val(v) + \varepsilon$$

A strategy is *optimal* in $\hat{v}$ if it is 0-optimal in $\hat{v}$, and just *optimal* if it is optimal in every $\hat{v}$.

3 The Existence of Values and Optimal Strategies

In this section we first prove that every vertex in a CTG with time-bounded reachability objectives has a value. This result holds without any additional restrictions (i.e., for CTGs with possibly countable state-space and infinite branching degree). From this we immediately obtain the existence of ε-optimal strategies for both players for every $\varepsilon > 0$. Then, we study the existence of optimal strategies. We show that even though optimal minimizing strategies may not exist in infinitely-branching CTGs, they always exist in finitely-branching ones. As for optimal maximizing strategies, we show that they do not necessarily exist

even in finitely-branching CTGs, but they are guaranteed to exist if a game is both finitely-branching and has bounded rates (see Definition 2).

For the rest of this section, we fix a CTG $G = (V, A, \mathbf{E}, (V_\Box, V_\Diamond), \mathbf{P}, \mathbf{R})$, a set $T \subseteq V$ of target vertices, and a time bound $t > 0$. Given $\mathbf{i} \in \mathcal{H}$ where $|\mathbf{i}| > 0$, we denote by $F_\mathbf{i}$ the probability distribution function of the random variable $\sum_{r\in\mathcal{R}} \sum_{i=1}^{\mathbf{i}(r)} X_i^{(r)}$ where all $X_i^{(r)}$ are mutually independent and each $X_i^{(r)}$ is an exponentially distributed random variable with the rate r (for reader's convenience, basic properties of exponentially distributed random variables are recalled in [3]). We also define $F_\mathbf{0}$ as a constant function returning 1 for every argument (here $\mathbf{0} \in \mathcal{H}$ is the empty history, i.e., $|\mathbf{0}| = 0$). In the special case when $\mathcal{R}$ is a singleton, we use F_ℓ and f_ℓ to denote $F_\mathbf{i}$ and $f_\mathbf{i}$ such that $\mathbf{i}(r) = \ell$, where r is the only element of $\mathcal{R}$. Further, given $\sim \in \{<, \leq, =\}$ and $k \in \mathbb{N}$, we denote by $\mathcal{P}_\hat{v}^{\sigma,\pi}(Reach_{\sim k}^{\leq t}(T))$ the probability of all $\alpha \in Run_{G(\hat{v},\sigma,\pi)}$ that visit T for the first time in the number of steps satisfying $\sim k$ and before time t (i.e., there is $i \in \mathbb{N}_0$ such that $i = \min\{j \mid \alpha(j) \text{ hits } T\} \sim k$ and $\sum_{i=0}^{i-1} \alpha[i] \leq t$).

The following theorem says that every vertex in a CTG with bounded reachability objectives has a value. Let us note that the powerful result of Martin [6] about weak determinacy of Blackwell games cannot be applied in this setting, at least not immediately. As we shall see, the ideas presented in the proof of Theorem 3 are useful also for designing an algorithm which for a given $\varepsilon > 0$ computes ε-optimal strategies for both players.

THEOREM 3. *Every vertex $v \in V$ has a value.*

Roughly speaking, Theorem 3 is proved in the following way. Given $\sigma \in \Sigma$, $\pi \in \Pi$, $\mathbf{j} \in \mathcal{H}$, and $u \in V$, we denote by $P^{\sigma,\pi}(u, \mathbf{j})$ the probability of all runs $\alpha \in Run_{G(u,\sigma,\pi)}$ such that for some $n \in \mathbb{N}_0$ the state $\alpha(n)$ hits T and matches $\mathbf{j}$, and for all $0 \leq j < n$ we have that $\alpha(j)$ does not hit T. Then we introduce two functions $\mathcal{A}, \mathcal{B} : \mathcal{H} \times V \to [0,1]$ where

$$\mathcal{A}(\mathbf{i}, v) = \sup_{\sigma\in\Sigma} \inf_{\pi\in\Pi} \sum_{\mathbf{j}\in\mathcal{H}} F_{\mathbf{i}+\mathbf{j}}(t) \cdot P^{\sigma,\pi}(v, \mathbf{j}) \qquad \mathcal{B}(\mathbf{i}, v) = \inf_{\pi\in\Pi} \sup_{\sigma\in\Sigma} \sum_{\mathbf{j}\in\mathcal{H}} F_{\mathbf{i}+\mathbf{j}}(t) \cdot P^{\sigma,\pi}(v, \mathbf{j})$$

Intuitively, $\mathcal{A}(\mathbf{i}, v)$ and $\mathcal{B}(\mathbf{i}, v)$ give the "best" probability achievable by player $\Box$ and player $\Diamond$ in a vertex v, assuming that the history of a play matches $\mathbf{i}$. Hence, it suffices to prove that $\mathcal{A} = \mathcal{B}$, because then also $\mathcal{A}(\mathbf{0}, v) = \mathcal{B}(\mathbf{0}, v) = val(v)$, where $\mathbf{0}$ returns zero for every argument. The equality $\mathcal{A} = \mathcal{B}$ is obtained by demonstrating that both $\mathcal{A}$ and $\mathcal{B}$ are equal to the (unique) least fixed point of a monotonic function $\mathcal{V} : (\mathcal{H} \times V \to [0,1]) \to (\mathcal{H} \times V \to [0,1])$ defined as follows: for every $H : \mathcal{H} \times V \to [0,1]$, $\mathbf{i} \in \mathcal{H}$, and $v \in V$ we have that

$$\mathcal{V}(H)(\mathbf{i}, v) = \begin{cases} F_\mathbf{i}(t) & v \in T \\ \sup_{a\in\mathbf{E}(v)} \sum_{u\in V} \mathbf{P}(a)(u) \cdot H(\mathbf{i} + \mathbf{1}_{\mathbf{R}(a)}, u) & v \in V_\Box \setminus T \\ \inf_{a\in\mathbf{E}(v)} \sum_{u\in V} \mathbf{P}(a)(u) \cdot H(\mathbf{i} + \mathbf{1}_{\mathbf{R}(a)}, u) & v \in V_\Diamond \setminus T \end{cases}$$

The details are technical and can be found in [3].

Observe that due to Theorem 3, both players have ε-optimal strategies in every vertex v (for every $\varepsilon > 0$). This follows directly from the definition of $val(v)$ given in Section 2. Now we examine the existence of *optimal* strategies. We start by observing that optimal minimizing and optimal maximizing strategies do not necessarily exist, even if we restrict

ourselves to games with finitely many rates (i.e., $\mathcal{R}(G)$ is finite) and finitely many distinct transition probabilities.

OBSERVATION 4. *Optimal minimizing and optimal maximizing strategies in continuous-time games with time-bounded reachability objectives do not necessarily exist, even if we restrict ourselves to games with finitely many rates (i.e., $\mathcal{R}(G)$ is finite) and finitely many distinct transition probabilities.*

However, if G is finitely-branching, then the existence of an optimal minimizing CD strategy can be established by adapting the construction used in the proof of Theorem 3. Observe that we do not require that G has bounded rates.

THEOREM 5. *If G is finitely-branching, then there is an optimal minimizing CD strategy.*

The issue with optimal maximizing strategies is slightly more complicated. First, we observe that optimal maximizing strategies do not necessarily exist even in finitely-branching games.

OBSERVATION 6. *Optimal maximizing strategies in continuous-time games with time-bounded reachability objectives may not exist, even if we restrict ourselves to finitely-branching games.*

Now we show that if G is finitely-branching *and* has bounded rates, then there is an optimal maximizing CD strategy. To achieve that, we introduce the notion of *k-step optimal* strategies, which optimize the outcome in finite plays of length k. Observe that, due to Theorem 3, for all $k \in \mathbb{N}$ and $v \in V$ we have that $\sup_{\sigma\in\Sigma} \inf_{\pi\in\Pi} \mathcal{P}_v^{\sigma,\pi}(Reach_{\leq k}^{\leq t}(T)) = \inf_{\pi\in\Pi} \sup_{\sigma\in\Sigma} \mathcal{P}_v^{\sigma,\pi}(Reach_{\leq k}^{\leq t}(T))$. We use $val^k(v)$ to denote the *k-step value* defined by this equality, and we say that strategies $\sigma^k \in \Sigma$ and $\pi^k \in \Pi$ are *k-step optimal* if for all $v \in V$, $\pi \in \Pi$, and $\sigma \in \Sigma$ we have $\inf_{\pi\in\Pi} \mathcal{P}_v^{\sigma^k,\pi}(Reach_{\leq k}^{\leq t}(T)) = \sup_{\sigma\in\Sigma} \mathcal{P}_v^{\sigma,\pi^k}(Reach_{\leq k}^{\leq t}(T)) = val^k(v)$. The existence and basic properties of k-step optimal strategies are stated in our next lemma.

LEMMA 7. *If G is finitely-branching and has bounded rates, then we have the following:*

1. *For all $\varepsilon > 0$, $k \geq (\sup \mathcal{R})te^2 - \ln \varepsilon$, $\sigma \in \Sigma$, $\pi \in \Pi$, and $v \in V$ we have that*

$$\mathcal{P}_v^{\sigma,\pi}(Reach^{\leq t}(T)) - \varepsilon \;\leq\; \mathcal{P}_v^{\sigma,\pi}(Reach_{\leq k}^{\leq t}(T)) \;\leq\; \mathcal{P}_v^{\sigma,\pi}(Reach^{\leq t}(T))$$

2. *For every $k \in \mathbb{N}$, there are k-step optimal BCD strategies $\sigma^k \in \Sigma$ and $\pi^k \in \Pi$. Further, for all $\varepsilon > 0$ and $k \geq (\sup \mathcal{R})te^2 - \ln \varepsilon$ we have that every k-step optimal strategy is also an ε-optimal strategy.*

If G is finitely-branching and has bounded rates, one may be tempted to construct an optimal maximizing strategy σ by selecting those actions that are selected by infinitely many k-step optimal BCD strategies for all $k \in \mathbb{N}$ (these strategies are guaranteed to exist by Lemma 7 (2)). However, this is not so straightforward, because the distributions assigned to actions in finitely-branching games can still have an infinite support. Intuitively, this issue is overcome by considering larger and larger finite subsets of the support so that the total probability of all of the infinitely many omitted elements approaches zero. Hence, a proof of the following theorem is somewhat technical.

THEOREM 8. *If G is finitely-branching and has bounded rates, then there is an optimal maximizing CD strategy.*

3.1 Optimal Strategies in Finite Uniform CTGs

In this subsection, we restrict ourselves to finite uniform CTGs and prove that both players have *optimal BCD strategies* in such games. Roughly speaking, the result is obtained by showing that optimal CD strategies (which are guaranteed to exist by Theorem 5 and Theorem 8) can be safely redefined into *greedy* strategies after performing a finite (and effectively computable) number of steps. Greedy strategies try to maximize/minimize the probability of reaching T in as few steps as possible, and hence they can ignore the history of a play. Hence, the original optimal CD strategies become stationary after a finite number of steps, which means that they are in fact BCD. We also show that this result is tight in the sense that optimal BCD strategies do not necessarily exist in uniform CTGs with infinitely many states. In Section 4, we use these results to design an algorithm which *computes* the optimal BCD strategies in finite uniform games.

In this subsection, we assume that the previously fixed CTG G is finite and that $\mathbf{R}(a) = r > 0$ for all $a \in A$. We start by introducing greedy strategies.

DEFINITION 9. *A strategy $\sigma \in \Sigma$ is* greedily maximizing *if for all $v \in V$ and $\sigma' \in \Sigma$ one of the following two conditions is satisfied:*

- *For all $i \in \mathbb{N}_0$ we have* $\inf_{\pi\in\Pi} \mathcal{P}_v^{\sigma,\pi}(Reach_{\leq i}^{<\infty}(T)) = \inf_{\pi\in\Pi} \mathcal{P}_v^{\sigma',\pi}(Reach_{\leq i}^{<\infty}(T))$.
- *There is $i \in \mathbb{N}_0$ such that* $\inf_{\pi\in\Pi} \mathcal{P}_v^{\sigma,\pi}(Reach_{\leq i}^{<\infty}(T)) > \inf_{\pi\in\Pi} \mathcal{P}_v^{\sigma',\pi}(Reach_{\leq i}^{<\infty}(T))$ *and for all $j < i$ we have* $\inf_{\pi\in\Pi} \mathcal{P}_v^{\sigma,\pi}(Reach_{\leq j}^{<\infty}(T)) = \inf_{\pi\in\Pi} \mathcal{P}_v^{\sigma',\pi}(Reach_{\leq j}^{<\infty}(T))$.

Similarly, $\pi \in \Pi$ is greedily minimizing *if for all $v \in V$ and $\pi' \in \Pi$ one of the following conditions holds:*

- *For all $i \in \mathbb{N}_0$ we have* $\sup_{\sigma\in\Sigma} \mathcal{P}_v^{\sigma,\pi}(Reach_{\leq i}^{<\infty}(T)) = \sup_{\sigma\in\Sigma} \mathcal{P}_v^{\sigma,\pi'}(Reach_{\leq i}^{<\infty}(T))$.
- *There is $i \in \mathbb{N}_0$ such that* $\sup_{\sigma\in\Sigma} \mathcal{P}_v^{\sigma,\pi}(Reach_{\leq i}^{<\infty}(T)) < \sup_{\sigma\in\Sigma} \mathcal{P}_v^{\sigma,\pi'}(Reach_{\leq i}^{<\infty}(T))$ *and for all $j < i$ we have* $\sup_{\sigma\in\Sigma} \mathcal{P}_v^{\sigma,\pi}(Reach_{\leq j}^{<\infty}(T)) = \sup_{\sigma\in\Sigma} \mathcal{P}_v^{\sigma,\pi'}(Reach_{\leq j}^{<\infty}(T))$.

A strategy τ is stationary *if τ is deterministic and $\tau(wv)$ depends just on v for every vertex v.*

Note that time plays no role in greedily maximizing/minimizing strategies. Our next lemma reveals that greedy *stationary* strategies exist and can be effectively computed in polynomial time in finite CTGs.

LEMMA 10. *There is a greedily maximizing stationary strategy σ_g, and a greedily minimizing stationary strategy π_g. Moreover, the strategies σ_g and π_g are computable in polynomial time.*

Now we can state the main theorem of this subsection.

THEOREM 11. *Let σ_g be a greedily maximizing stationary strategy, and π_g a greedily minimizing stationary strategy. Let σ be an optimal maximizing CD strategy, and π an optimal minimizing CD strategy. Then for all sufficiently large $k \in \mathbb{N}$ we have that BCD strategies $\sigma' \in \Sigma$ and $\pi' \in \Pi$ defined by*

$$\sigma'(i,v) = \begin{cases} \sigma(i,v) & \text{if } i < k; \\ \sigma_g(v) & \text{otherwise.} \end{cases} \qquad \pi'(i,v) = \begin{cases} \pi(i,v) & \text{if } i < k; \\ \pi_g(v) & \text{otherwise.} \end{cases}$$

are optimal. Moreover, if all transition probabilities in G are rational, then σ' and π' are optimal for all $k \geq rt(1 + m^{|A|^2\cdot|V|^2})$, where m is the maximal denominator of transition probabilities.

A natural question is whether Theorem 11 can be extended to infinite-state uniform CTGs. The question is answered in our next observation.

OBSERVATION 12. *Optimal BCD strategies do not necessarily exist in uniform infinite-state CTGs, even if they are finitely-branching and use only finitely many distinct transition probabilities.*

4 Algorithms

Now we present algorithms which compute ε-optimal BCD strategies in finitely-branching CTGs with bounded rates and optimal BCD strategies in finite uniform CTGs. In this section, we assume that all rates and distributions used in the considered CTGs are *rational*.

4.1 Computing ε-optimal BCD strategies

For the rest of this subsection, let us fix a CTG $G = (V, A, \mathbf{E}, (V_\Box, V_\Diamond), \mathbf{P}, \mathbf{R})$, a set $T \subseteq V$ of target vertices, a time bound $t > 0$, and some $\varepsilon > 0$. For simplicity, let us first assume that G is finite; as we shall see, our algorithm does not really depend on this assumption, as long as the game is finitely-branching, has bounded rates, and its structure can be effectively generated (see Corollary 14). Let $k = (\max \mathcal{R})te^2 - \ln(\frac{\varepsilon}{2})$. Then, due to Lemma 7, all k-step optimal strategies are $\frac{\varepsilon}{2}$-optimal.

For every $\mathbf{i} \in \mathcal{H}$, where $|\mathbf{i}| \leq k$, and for every $v \in V$, our algorithm computes an action $C(\mathbf{i}, v) \in \mathbf{E}(v)$ which represents the choice of the constructed ε-optimal BCD strategies $\sigma_\varepsilon \in \Sigma$ and $\pi_\varepsilon \in \Pi$. That is, for every $\mathbf{i} \in \mathcal{H}$, where $|\mathbf{i}| \leq k$, and for every $v \in V_\Box$, we put $\sigma_\varepsilon(\mathbf{i}, v)(C(\mathbf{i}, v)) = 1$, and for the other arguments we define σ_ε arbitrarily so that σ_ε remains a BCD strategy. The strategy π_ε is induced by the function C in the same way.

To compute $C(\mathbf{i}, v)$, our algorithm uses a family of probabilities $R(\mathbf{i}, u)$ of reaching T from u before time t in at most $k - |\mathbf{i}|$ steps using the strategies σ_ε and π_ε and assuming that the history matches $\mathbf{i}$. Actually, our algorithm computes the probabilities $R(\mathbf{i}, u)$ only up to a sufficiently small error so that the actions chosen by C are "sufficiently optimal" (i.e., the strategies σ_ε and π_ε are ε-optimal, but they are not necessarily k-step optimal for the k chosen above). Our algorithm works in two phases:

1. For $\mathbf{i} \in \mathcal{H}$, where $|\mathbf{i}| \leq k$, compute a number $\ell_\mathbf{i}(t) > 0$ such that $\frac{|F_\mathbf{i}(t) - \ell_\mathbf{i}(t)|}{F_\mathbf{i}(t)} \leq \frac{\varepsilon^{2|\mathbf{i}|+1}}{2^{2|\mathbf{i}|+1}}$. For every $a \in A$ and $u \in V$, compute a floating point representation $\mathbf{p}(a)(u)$ of $\mathbf{P}(a)(u)$ satisfying $\frac{|\mathbf{P}(a)(u) - \mathbf{p}(a)(u)|}{\mathbf{P}(a)(u)} \leq \frac{\varepsilon^{2k+1}}{2^{2k+1}}$.
2. Compute (in a bottom up fashion) the functions R and C as follows: Given $\mathbf{i} \in \mathcal{H}$, where $|\mathbf{i}| \leq k$, and $v \in V$, we have that

$$R(\mathbf{i}, v) = \begin{cases} \ell_\mathbf{i}(t) & \text{if } v \in T \\ 0 & \text{if } v \notin T \text{ and } |\mathbf{i}| = k \\ max_{a \in \mathbf{E}(v)} \sum_{u \in V} \mathbf{p}(a)(u) \cdot R(\mathbf{i} + \mathbf{1}_{\mathbf{R}(a)}, u) & \text{if } v \in V_\Box \setminus T \text{ and } |\mathbf{i}| < k \\ min_{a \in \mathbf{E}(v)} \sum_{u \in V} \mathbf{p}(a)(u) \cdot R(\mathbf{i} + \mathbf{1}_{\mathbf{R}(a)}, u) & \text{if } v \in V_\Diamond \setminus T \text{ and } |\mathbf{i}| < k \end{cases}$$

   For all $|\mathbf{i}| < k$ and $v \notin T$, we put $C(\mathbf{i}, v) = a$ where a is an action that realizes the maximum (or minimum).

In [3] we show that the strategies σ_ε and π_ε are indeed ε-optimal. Complexity analysis of the algorithm reveals the following (*bp* denotes the maximum bit-length of $\mathbf{P}(a)(v)$ and rates, assuming that we represent $\mathbf{P}(a)(v)$ and rates as fractions of integers encoded in binary).

THEOREM 13. *Assume that G is finite. Then for every $\varepsilon > 0$ there are ε-optimal BCD strategies $\sigma_\varepsilon \in \Sigma$ and $\pi_\varepsilon \in \Pi$ computable in time $|V|^2 \cdot |A| \cdot bp \cdot (|\mathcal{R}|+1)^{\mathcal{O}((\max \mathcal{R}) \cdot t + \ln \frac{1}{\varepsilon})}$.*

Note that our algorithm needs to analyze only a finite part of G. Hence, it also works for infinite games which satisfy the conditions formulated in the next corollary.

COROLLARY 14. *Let G be a finitely-branching game with bounded rates and let $v \in V$. Assume that the vertices and actions of G reachable from v in a given finite number of steps are effectively computable, and that an upper bound on rates is also effectively computable. Then for every $\varepsilon > 0$ there are effectively computable BCD strategies $\sigma_\varepsilon \in \Sigma$ and $\pi_\varepsilon \in \Pi$ that are ε-optimal in v.*

4.2 Computing optimal BCD strategies in uniform finite games

For the rest of this subsection, we fix a finite uniform CTG $G = (V, A, \mathbf{E}, (V_\Box, V_\Diamond), \mathbf{P}, \mathbf{R})$ where $\mathbf{R}(a) = r > 0$ for all $a \in A$. Let $k = rt(1 + m^{|A|^2 \cdot |V|^2})$ (see Theorem 11).

The algorithm works similarly as the one of Section 4.1, but there are also some differences. Since we have just one rate, the vector $\mathbf{i}$ becomes just a number i. Similarly as in Section 4.1, our algorithm computes an action $C(i,v) \in \mathbf{E}(v)$ representing the choice of the constructed optimal BCD strategies $\sigma_{max} \in \Sigma$ and $\pi_{min} \in \Pi$. By Lemma 11, every optimal strategy can, from the k-th step on, start to behave as a fixed greedy stationary strategy, and we can compute such a greedy stationary strategy in polynomial time. Hence, the optimal BCD strategies σ_{max} and π_{min} are defined as follows:

$$\sigma_{max}(i,v) = \begin{cases} C(i,v) & \text{if } i < k; \\ \sigma_g(v) & \text{otherwise.} \end{cases} \qquad \pi_{min}(i,v) = \begin{cases} C(i,v) & \text{if } i < k; \\ \pi_g(v) & \text{otherwise.} \end{cases}$$

To compute the function C, our algorithm uses a table of symbolic representations of the (precise) probabilities $R(i,v)$ (here $i \leq k$ and $v \in V$) of reaching T from v before time t in at most $k - i$ steps using the strategies σ_{max} and π_{min} and assuming that the history matches i.

The function C and the family of all $R(i,v)$ are computed (in a bottom up fashion) as follows: For all $0 \leq i \leq k$ and $v \in V$ we have that

$$R(i,v) = \begin{cases} F_i(t) & \text{if } v \in T \\ \sum_{j=0}^{\infty} F_{i+j}(t) \cdot \mathcal{P}_v^{\sigma_g, \pi_g}(Reach_{=j}^{<\infty}(T)) & \text{if } v \notin T \text{ and } i = k \\ max_{a \in \mathbf{E}(v)} \sum_{u \in V} \mathbf{P}(a)(u) \cdot R(i+1,u) & \text{if } v \in V_\Box \setminus T \text{ and } i < k \\ min_{a \in \mathbf{E}(v)} \sum_{u \in V} \mathbf{P}(a)(u) \cdot R(i+1,u) & \text{if } v \in V_\Diamond \setminus T \text{ and } i < k \end{cases}$$

For all $i < k$ and $v \in V$, we put $C(i,v) = a$ where a is an action maximizing or minimizing $\sum_{u \in V} \mathbf{P}(a)(u) \cdot R(i+1,u)$, depending on whether $v \in V_\Box$ or $v \in V_\Diamond$, respectively. The effectivity of computing such an action (this issue is not trivial) is discussed in the proof of the following theorem.

THEOREM 15. *The BCD strategies σ_{max} and π_{min} are optimal and effectively computable.*

5 Conclusions, Future Work

We have shown that vertices in CTGs with time bounded reachability objectives have a value, and we classified the subclasses of CTGs where a given player has an optimal strategy. We also proved that in finite uniform CTGs, both players have optimal BCD strategies. Finally, we designed algorithms which compute ε-optimal BCD strategies in finitely-branching CTGs with bounded rates, and optimal BCD strategies in finite uniform CTGs.

There are several interesting directions for future research. First, we can consider more general classes of strategies that depend on the elapsed time (in our setting, strategies are time-abstract). In [2], it is demonstrated that time-dependent strategies can achieve better results than the time-abstract ones. Further, [7] shows the power of time-dependent strategies differs when the player knows only the time consumed by the last action, or the complete timed history of a play. It is not immediately clear whether Theorem 3 still holds for time-dependent strategies, and whether it makes sense to think about optimal strategies in this setting. Second, a generalization to semi-Markov processes and games, where arbitrary (not only exponential) distributions are considered, would be desirable. Another interesting open problem is the existence of optimal BCD strategies in (not necessarily uniform) games.

References

[1] C. Baier, B.R. Haverkort, H. Hermanns, and J.-P. Katoen. Reachability in continuous-time Markov reward decision processes. In E. Graedel, J. Flum, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, volume 2 of Texts in Logics and Games, pages 53–72. Amsterdam University Press, 2008.

[2] C. Baier, H. Hermanns, J.-P. Katoen, and B.R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *TCS*, 345:2–26, 2005.

[3] T. Brázdil, V. Forejt, J. Krčál, J. Křetínský, and A. Kučera. Continuous-time stochastic games with time-bounded reachability. Technical report FIMU-RS-2009-09, Faculty of Informatics, Masaryk University, Brno, 2009.

[4] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1996.

[5] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games*. Number 2500 in LNCS. Springer, 2002.

[6] D.A. Martin. The determinacy of Blackwell games. *JSL*, 63(4):1565–1581, 1998.

[7] M. Neuhäußer, M. Stoelinga, and J.-P. Katoen. Delayed nondeterminism in continuous-time Markov decision processes. In *Proceedings of FoSSaCS 2009*, volume 5504 of *LNCS*, pages 364–379. Springer, 2009.

[8] J.R. Norris. *Markov Chains*. Cambridge University Press, 1998.

[9] M.L. Puterman. *Markov Decision Processes*. Wiley, 1994.

[10] W. Thomas. Infinite games and verification. In *Proceedings of CAV 2003*, volume 2725 of *LNCS*, pages 58–64. Springer, 2003.

[11] I. Walukiewicz. A landscape with games in the background. In *Proceedings of LICS 2004*, pages 356–366. IEEE, 2004.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Deterministic Automata and Extensions of Weak MSO

Mikołaj Bojańczyk, Szymon Toruńczyk*
University of Warsaw
{bojan,szymtor}@mimuw.edu.pl

ABSTRACT. We introduce a new class of automata on infinite words, called min-automata. We prove that min-automata have the same expressive power as weak monadic second-order logic (weak MSO) extended with a new quantifier, the recurrence quantifier. These results are dual to a framework presented in [2], where max-automata were proved equivalent to weak MSO extended with an unbounding quantifier. We also present a general framework, which tries to explain which types of automata on infinite words correspond to extensions of weak MSO. As another example for the usefulness framework, apart from min- and max-automata, we define an extension of weak MSO with a quantifier that talks about ultimately periodic sets.

Introduction

In [2], a new class of languages of infinite words was defined. This class had two equivalent descriptions: in terms of a deterministic counter automaton (called a max-automaton), and in terms of an extension of weak monadic second-order logic (weak MSO). The argument raised in [2] was that there are robust extensions of ω-regular languages, extensions that have descriptions in terms of both automata and logic. This paper further investigates that argument. These are the contributions:

1. We define a type of automaton dual to max-automata, called a min-automaton, and prove that it is equivalent to a certain extension of weak MSO.
2. We show that min- and max-automata fit in a general picture, where deterministic automata with prefix-closed acceptance conditions define extensions of weak MSO.
3. As another example of the general picture, we present an extension of weak MSO, together with a corresponding automaton, that talks about ultimately periodic sets.

Below we describe these contributions in more detail.

Min-automata. A max-automaton, as defined in [2], works as follows. It is a deterministic automaton, but it also has a finite set C of counters, which store natural numbers. Each transition is decorated by a sequence of counter operations, which are from the set

$$Op = \{c := c + 1\,, \quad c := \max(d, e) \qquad : \qquad c, d, e \in C\}.$$

(The toolkit of operations in [2] was slightly different, but the simpler one above is equivalent.) There are two key properties of the model. First, the automaton is deterministic, which is important for the connection with weak MSO. Second, the choice of the next state is not influenced by the counter values, but only the current state and input letter; one can

*Work partially funded by the Polish government grant no. N206 008 32/0810

© Bojańczyk,Toruńczyk; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 73–84
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2308

somehow think of the counter operations being applied after the run is chosen. The only place where the counters are read is the acceptance condition, which is a boolean combination of conditions

$$\limsup_{i\to\infty} val(c, a_1 a_2 \dots a_i) = \infty,$$

where $val(c, u)$ is the value of counter c after reading a finite prefix u of the input word.

The main contribution of [2] is that max-automata are equivalent to weak MSO extended with a quantifier, called the unbounding quantifier. The unbounding quantifier binds a set variable X in a formula $\varphi(X)$ and is true if there are sets X of arbitrarily large finite size that satisfy $\varphi(X)$.

If an automaton with the max operation has a matching logic, then what about min? What if we use lim inf instead of lim sup in the acceptance condition? In this paper we analyze such an automaton model, called a min-automaton, where min is used instead of max, and the acceptance condition uses lim inf instead of lim sup. We show that min-automata also have a corresponding logic. Note that there are other combinations, which we do not study here, such as automata that use max and lim inf.

What is the logic that corresponds to min-automata? As was the case for max-automata, this is an extension of weak MSO, where a new quantifier is added. The quantifier for min-automata, which we introduce in this paper and call the recurrence quantifier, says: "there is some $n \in \mathbb{N}$ such that infinitely many sets of size n satisfy $\varphi(X)$". One of our main results, Theorem 8, is that min-automata have the same expressive power as weak MSO extended with the recurrence quantifier.

General Framework. Although we think that min-automata are interesting in their own right, we also think that they are part of a bigger picture for deterministic automata on infinite words. The bigger picture is that any "reasonable" acceptance condition seems to give a robust class of languages extending weak MSO. We present some preliminary results that attempt to formalise these ideas.

One consequence of our results is a normal form theorem: any formula of weak MSO extended with both the unbounding quantifier (the quantifier related to max-automata) and the recurrence quantifier (the quantifier related to min-automata) is effectively equivalent to a boolean combination of formulas, each of which has at most one occurrence of the new quantifiers (bounding or recurrence). In other words, mutual nesting of the new quantifiers does not contribute to the expressive power. This normal form can be used to decide satisfiability for weak MSO extended with both quantifiers, since the algorithm only needs to test emptiness for boolean combinations of (actually, conjunctions of) max- and min-automata.

Ultimately Periodic Quantifier. As an example of the bigger picture, we consider an extension of weak MSO with the ultimately periodic quantifier. This quantifier binds a first-order variable in a formula $\varphi(x)$ and says that the set of word positions that satisfy $\varphi(x)$ is ultimately periodic. We present an equivalent automaton model, where the acceptance condition says that certain states appear in an ultimately periodic way, and certain other states

do not. Using this model, and some combinatorics, we prove that satisfiability is decidable for weak MSO with the ultimately periodic quantifier.

Background and related work. The idea of considering extensions of ω-regular languages is not new, dating back to the sixties. One line of work has been to add new predicates, such as a predicate $square(x)$, which holds for positions that are square numbers. This line was started by [7], and continued in [5, 11, 10].

More closely related to this paper is the work on the unbounding quantifier. This quantifier was introduced in [3]. The satisfiability problem for full MSO (as opposed to weak MSO, the subject of this paper) extended with the unbounding quantifier was tackled [4]. By introducing an automaton model, called a BS-automaton, [4] provided some fragments of full MSO with the unbounding quantifier that have decidable satisfiability over infinite words. A BS-automaton is a counter automaton with acceptance conditions as in max- and min-automata, but, crucially, it is nondeterministic. Nondeterminism is important for full MSO, where existential quantification over infinite sets is allowed. Nondeterminism also increases the flexibility of the model (for instance, the max and min operations become redundant). There is no free lunch, however: nondeterministic BS-automata are not closed under complement, and it is not clear what is the correct automaton model for full MSO with the unbounding quantifier. It is still an open problem if full MSO extended with the unbounding quantifier has decidable satisfiability over infinite words.

BS-automata have also been considered in [1], under the name of R-automata. BS-automata are also closely related to distance desert automata, which were used by Kirsten to decide the star height problem [8]. A tree variant of distance desert automata was introduced in [6], to decide star height for tree languages.

Acknowledgments. We would like to thank Eryk Kopczyński, Sławomir Lasota, Aymeric Vincent and Thomas Wilke for many stimulating discussions.

1 Min-automata

In this section we introduce min-automata. The idea is that a min-automaton has a finite set of counters that store natural numbers, and each transition is labeled by a finite sequence of counter operations, taken from the set

$$Op_C = \{c := c + 1\,, \quad c := \min(d, e) \qquad : \qquad c, d, e \in C\}.$$

Formally, a deterministic min-automaton consists of:

A	The alphabet of the automaton
Q	A finite set of states of the automaton
C	A finite set of counters of the automaton
δ	The state transition function, $\delta\colon Q \times A \to Q$
γ	The counter update function, $\gamma\colon Q \times A \to (Op_C)^*$
q_0	The initial state, $q_0 \in Q$
v_0	The vector of initial counter values, $v_0 \in \mathbb{N}^C$
F	The acceptance condition, described below.

Given a finite word $w \in A^*$, the automaton produces a unique run $\rho \in Q^*$. By applying the counter update function γ to this run, we get a sequence $\pi \in (Op_C)^*$ of counter operations. By applying this sequence of operations to the initial counter valuation v_0, we get a counter valuation written $val(c, w)$.

The acceptance condition F is the only place where the counters are read. It talks about the asymptotic[†] values of the counters when reading an input word $a_1 a_2 \cdots \in A^\omega$. It is a boolean combination of conditions

$$\liminf_{i \to \infty} val(c, a_1 \cdots a_i) = \infty. \tag{1}$$

In the automaton, the above condition is represented in the formula F by an atom c for short.

In particular, the class of languages accepted by min-automata is closed under complementation, since replacing the acceptance condition F by $\neg F$ gives an automaton recognizing the complement language, thanks to determinism. Closure under alternative and conjunction follows from the usual cartesian product construction.

If the counters would influence the states, such as by having a zero-test counter operation, we would lose all the robust decidability of the model. It is crucial that as far as choosing the states is concerned, a min-automaton behaves just like a finite deterministic automaton.

EXAMPLE 1. With each infinite sequence of natural numbers $n_1, n_2, n_3 \ldots$, we may associate an infinite word

$$a^{n_1} b\, a^{n_2} b\, a^{n_3} b \ldots$$

Let L be the set of words associated with sequences where $\liminf n_i < \infty$. Then L is recognized by a deterministic min-automaton with one state, three counters c, d, z and the following instructions.

- when reading a, do $c := c + 1$,
- when reading b, do $d := min(c, c)$; $c := z$.

The initial valuation is $(0, 0, 0)$. Counter c stores the size of the current a block, while counter d stores the size of the last complete a block. Counter z always stores 0, and is used to reset

[†]Since the acceptance condition is insensitive to finite perturbations, the initial counter valuation does not influence the accepted language. The initial counter valuation will play a role for automata in matrix form.

counter c when a block of a's is finished. The acceptance condition is $F = \neg c \wedge \neg d$: both counters c and d should have $\liminf < \infty$ (counter z is not mentioned in the acceptance condition).

The above example shows how counter operations $c := 0$ and $d := c$ can be implemented in the model.

The following lower bound on the complexity of emptiness is via a reduction from the universality problem for nondeterministic automata. This is also a partial answer to a question posed in [2], which asked about the complexity of emptiness for max-automata (the same proof works for max-automata).

THEOREM 2. *Emptiness is* PSPACE*-hard for min-automata.*

Determinism. Does determinism restrict the expressive power of min-automata? It does for max-automata: in [2], it was shown that nondeterministic max-automata can, while deterministic max-automata cannot, recognize the language

$$L = \{a^{n_1} b\, a^{n_2} b\, a^{n_3}\, b \ldots : \liminf n_i < \infty\}.$$

The reason why a nondeterministic max-automaton can recognize L is that a sequence has $\liminf < \infty$ if and only if it has a subsequence of $\limsup < \infty$, and the subsequence can be nondeterministically guessed. The reason why deterministic max-automata cannot recognize this language is that L is on level Σ_3 of the Borel hierarchy, while deterministic max-automata can only recognize languages that are boolean combinations of Σ_2 languages.

For min-automata, one can prove that nondeterministic min-automata can, while deterministic min-automata cannot, recognize the language

$$K = \{a^{n_1} b\, a^{n_2} b\, a^{n_3}\, b \ldots : \limsup n_i = \infty\}.$$

The reason why a nondeterministic min-automaton can recognize K is the same as in the counterexample for max-automata. However, how does one prove that a deterministic min-automaton cannot recognize K? The topological argument no longer works, since K is on level Π_2 of the Borel hierarchy, while deterministic min-automata can recognize even Σ_3 languages, such as the language L. One idea would be to change the topology, to one where min-automata would be simpler than max-automata, but we could not find such a topology. Our solution uses pumping arguments.

Relationship with BS-automata. In this section we talk about translating min- and max-automata into BS-automata, as defined in [4]. BS-automata are like min- or max-automata, with three differences: (i) they are nondeterministic; (ii) they do not have the min and max counter operations, only increment and reset; and (iii) the acceptance condition can speak of both $\liminf$ and $\limsup$. In [2] it was shown how to convert a max-automaton to a nondeterministic BS-automaton. The same technique works for min-automata, so we get:

THEOREM 3. *Every max-automaton is effectively equivalent to a nondeterministic BS-automaton. The same holds for min-automata.*

COROLLARY 4. *Emptiness is decidable for boolean combinations of min- and max-automata.*

PROOF. Since max- and min-automata are closed under boolean operations, the problem is equivalent to testing emptiness for positive boolean combinations. Since BS-automata are closed under positive boolean combinations, every boolean combination of max- and min-automata is effectively equivalent to a BS-automaton. Emptiness of BS-automata is decidable by [4]. ∎

The complexity of the above procedure is quite high, especially due to the high cost of translating a max-automaton into a BS-automaton (the current algorithm is nonelementary). It would be nice to get an upper bound that is closer to the PSPACE lower bound from Theorem 2.

BS-automata do not have the min operation, and yet they are still able to capture min-automata. The translation from min-automata to BS-automata introduces nondeterminism. One might ask: is the min counter operation necessary in a deterministic min-automaton? (After removing the min-operation, we add a substitution operation $c := d$ and a reset operation $c := 0$, and we still keep the acceptance condition that talks about lim inf.) Notice how the automaton in Example 1 does not really use the min operation, only the substitution. In preliminary work, we have proved that min-automata without min are less expressive.

Below we describe the separating example. The alphabet is a, b, c, d. Let

$$w = a^{n_1} b a^{n_2} b \cdots a^{n_k} b$$

be a word in $(a^*b)^+$. For $\sigma \in \{c, d\}$ we define $\overline{w\sigma}$ to be $\min(n_1, \ldots, n_k)$ if $\sigma = c$ and ∞ otherwise. The separating language is

$$\{w_1\sigma_1 w_2\sigma_2 \ldots \in ((a^*b)^+(c+d))^\omega : \liminf \overline{w_i\sigma_i} = \infty\}$$

It is easy to define a min-automaton that recognizes the above language. The proof that an automaton without min cannot recognize this language requires a pumping argument, and will be given in a full version of this paper.

A matrix representation. In this section we represent the automata by matrices.

We extend slightly the definition of min-automata and allow an additional value $\top$, called the *undefined* value. As far as the min operation is concerned, the values are ordered $0 < 1 < \ldots < \top$. We extend addition to the new counter values by setting:

$$\top + x = x + \top = \top \qquad \text{for all } x.$$

We write $\mathcal{T}$ for the extended set $\{0, 1, 2, \ldots, \top\}$ of counter values. Together with the two operations above $\mathcal{T}$ forms a semiring, where the addition operation is min and the multiplication operation is $+$. This semiring is called *the tropical semiring*, or $(min, +)$ semiring, see e.g. [9].

The new counter values can be eliminated, by storing in the states the information about which counters are $\top$. The undefined counter value $\top$ will become important in the matrix representation, where it will be used to eliminate states from the automaton.

Let $\mathbb{M}_C\mathcal{T}$ denote the semiring of $C \times C$ matrices with entries from $\mathcal{T}$. Suppose that $M \in \mathbb{M}_C\mathcal{T}$. We can treat M as a counter operation, which changes a counter valuation, treated as a vector $v \in \mathcal{T}^C$, to $v \cdot M \in \mathcal{T}^C$. This type of operation can be implemented by a min-automaton, possibly after introducing auxiliary counters.

EXAMPLE 5. Let us return to the automaton from Example 1. When reading a letter a, the automaton would perform the operations $c := c + 1$. In matrix form, this is written as

$$\begin{pmatrix} c & d & z \end{pmatrix} \quad := \quad \begin{pmatrix} c & d & z \end{pmatrix} \cdot \begin{pmatrix} 1 & \top & \top \\ \top & 0 & \top \\ \top & \top & 0 \end{pmatrix}.$$

When reading b, the automaton would do $d := min(c, c);\ c := z$. In matrix form, this is

$$\begin{pmatrix} c & d & z \end{pmatrix} \quad := \quad \begin{pmatrix} c & d & z \end{pmatrix} \cdot \begin{pmatrix} \top & 0 & \top \\ \top & \top & \top \\ 0 & \top & 0 \end{pmatrix}.$$

Every sequence of counter operations can be represented in a matrix form as in the above example. In a *min-automaton in matrix* form, the counter operations are implemented by matrices, and the choice of the matrix only depends on the last letter seen (so there is no state). Such an automaton is given by an initial vector and a matrix for each letter of the input alphabet, so it is a tuple

$$\langle A, C, \gamma : A \to \mathbb{M}_C\mathcal{T}, v_0 \in \mathcal{T}^C, F \rangle.$$

After reading a word $a_1 \cdots a_n$, the counter valuation is

$$v_0 \cdot \gamma(a_1) \cdot \gamma(a_2) \cdots \gamma(a_n).$$

PROPOSITION 6. *For every min-automaton one can construct an equivalent min-automaton in matrix form. If the input automaton has n states and m counters, the output automaton has $(m + 1) \times n$ counters.*

PROOF. [sketch] By storing the state information in the counters which use the value $\top$. Each counter has one copy corresponding to each of the automaton states, and all but one of the copies are undefined at any moment. ∎

What is the point of the matrix representation? One advantage is that it underlies the close connection with existing work on distance automata and formal power series, where matrices over the tropical semiring play an important role. We would like to further investigate this connection, especially how the PSPACE upper bound on the limitedness problem for distance automata can be used for testing emptiness of min automata.

Another advantage is that we can eliminate states from the automaton. This is more an advantage of the $\top$ counter value. Having a stateless automaton enormously simplifies combinatorics, for instance in the proof that deterministic min-automata cannot recognize the language K defined earlier, and hence nondeterministic min-automata cannot be determinized.

2 Weak MSO with the recurrence quantifier

In [2], max-automata were proved to have the same expressive power as weak MSO extended with a new quantifier, called the unbounding quantifier (denoted U). For min-automata, the situation is the same, only a different quantifier is needed. Before introducing the new quantifier, we recall the definition of weak MSO. In weak MSO over infinite words we may:

- quantify over finite sets of positions (the $\exists_{\text{fin}} X$ quantifier) and single positions (the $\exists x$ quantifier),
- verify that a position belongs to a set of positions ($x \in X$),
- verify that one position comes before another ($x \leq y$),
- check the label standing on a position ($a(x)$ for each label $a \in A$),
- use boolean operations ($\wedge, \vee, \neg$).

Weak MSO corresponds to deterministic Muller automata over infinite words, which, thanks to the theorem of McNaughton, define all ω-regular languages. The goal of this section is to show this correspondence for min-automata, by adding a new quantifier, called the recurrence quantifier.

The recurrence quantifier The recurrence quantifier, written R, binds a set variable X in a formula $\varphi(X)$ and is true if there are infinitely many sets X of equal size that satisfy $\varphi(X)$. More precisely, $\mathsf{R}X.\varphi(X)$ is satisfied in a word w if there exists a number $N \in \mathbb{N}$ and infinitely many sets X of size N such that $\varphi(X)$ is satisfied in w.

EXAMPLE 7. Let φ be a formula with a free set-variable X which says that X is connected and has at least two b's. Formally,

$$\varphi(X) = \wedge \begin{cases} \forall x \forall y \forall z & x \in X \quad \wedge \quad z \in X \quad \wedge \quad x \leq y \leq z \quad \Rightarrow \quad y \in X \\ \exists x \exists y & x < y \quad \wedge \quad b(x) \quad \wedge \quad b(y) \quad \wedge \quad x \in X \quad \wedge \quad y \in X \end{cases}$$

A word $a^{n_1} b\, a^{n_2} b \ldots$ satisfies $\mathsf{R}X.\varphi(X)$ if and only if $\liminf n_i < \infty$. Therefore, the set of words with infinitely many b's that satisfy $\mathsf{R}X.\varphi(X)$ is the language L from Example 1.

THEOREM 8. *Weak MSO logic with the recurrence quantifier recognizes the same class of languages as min-automata.*

This theorem is a special case of Theorem 11, stated in the next section.

3 General framework

In the previous section, we defined min-automata and stated that they are equivalent to weak MSO with the recurrence quantifier. This is analogous to the situation for max-automata, where the appropriate quantifier is the unbounding quantifier. Converting an automaton into a formula is straightforward, while converting a formula into an automaton can be done thanks to some general properties shared by min- and max-automata. We would like to bring out these similarities, by introducing a more abstract framework.

The automaton side The control structure of deterministic min-automata, max-automata, Büchi automata, etc. is always the same, it is only the mode of acceptance that changes. We give an abstract definition below, by modeling an acceptance condition as a language $F \subseteq B^\omega$. The definition uses the notion of a letter to letter transducer, by which we understand a finite deterministic automaton with input alphabet A, whose transitions are labelled by letters of an output alphabet B. This transducer maps every word in A^* to a word in B^* of same length. We will use a transducer on infinite words, where it will give a function $A^\omega \to B^\omega$. Note that the transducers have no acceptance condition.

DEFINITION 9. *An automaton with acceptance condition $F \subseteq B^\omega$ (or simply F-automaton) $\mathcal{A}$ is a deterministic letter-to-letter transducer with input alphabet A and output alphabet B. We say that $\mathcal{A}$ accepts an input word $w \in A^\omega$ if the output word belongs to F. Languages accepted by F-automata are called F-regular.*

One example of this definition is a Büchi automaton. In this case, the acceptance condition is any language of the form $(B^*C)^\omega \subseteq B^\omega$, for $C \subseteq B$. In a similar way we can encode Muller or parity automata.

For min- or max-automata, the same can be done. In this case, the alphabet of the acceptance condition consists of words over the set of counter operations, and the acceptance condition contains those infinite sequences of counter operations where the appropriate limits are ∞.

We are mainly interested in *prefix-independent* acceptance conditions, namely languages $F \subseteq B^\omega$ that satisfy $F = B^*F$. All the examples mentioned above are prefix-independent. (In the case of min- or max-automata, to get prefix-independence we should not use the matrix form of automata, but the original definition, where the counters have values in $\mathbb{N}$.)

The logic side Let us call a *locus* any family $\mathcal{X}$ of finite sets of positions. Let a given input word be fixed. A formula $\varphi(X)$ with a free set-type variable X defines its locus $\mathcal{X}_\varphi$ as the family of finite sets of positions X which satisfy φ. A *locus property* Q is any set of loci. If Q is a locus property, then we write $\mathsf{Q}X.\varphi(X)$ if $\mathcal{X}_\varphi \in \mathsf{Q}$. The quantifiers $\exists_{\mathsf{fin}}$, U, R, P (defined in the next section) all arise in this fashion. For instance, for a locus $\mathcal{X}$, $\mathcal{X} \in \exists_{\mathsf{fin}}$ if it is nonempty, while $\mathcal{X} \in \mathsf{U}$ if it contains arbitrarily large sets.

For two loci $\mathcal{X}$ and $\mathcal{Y}$, we write $\mathcal{X} \simeq \mathcal{Y}$ if $\mathcal{X}$ and $\mathcal{Y}$ differ by a finite number of sets. We call Q *finitely invariant* if Q is invariant under $\simeq$, i.e. if $\mathcal{X} \in \mathsf{Q}$ and $\mathcal{X} \simeq \mathcal{Y}$, then $\mathcal{Y} \in \mathsf{Q}$. Examples of finitely invariant locus properties are $\mathsf{U}, \mathsf{R}, \mathsf{P}$. On the other hand, $\exists_{\mathsf{fin}}$ is not finitely invariant.

A Q-*formula* is a formula $\mathsf{Q}X.\varphi(X)$ where $\varphi(X)$ is a formula of WMSO with only one free variable, namely X. An open Q-*formula* is a Q-formula where φ is open in the following sense: if a word w together with a set X satisfies $\varphi(X)$, then there is some finite prefix of w such that changing the word w on positions outside the prefix does not affect the truth value of $\varphi(X)$.

Quantifier elimination Here we present our main result, which shows how quantifiers can be denested in the scope of a formula of WMSO. Since the theorem talks about automata and languages, a quantifier is viewed as an operation on languages, which takes a language

over an alphabet $A \times \{0,1\}$ and returns a language over an alphabet A. In the following, for a word w over alphabet A and a set of positions X, we write $w \otimes X$ for the word over alphabet $A \times \{0,1\}$ that has the labels of w on the first coordinate and the characteristic function of X on the second coordinate.

THEOREM 10. *Let F be a prefix-independent acceptance condition and let Q be a locus property. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language*

$$\mathsf{Q}L = \{w \in A^\omega : \mathsf{Q}X.[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages, ω-regular languages, and Q-formulas. Moreover, if Q is finitely invariant, then the Q-formulas are open.

Here is an important corollary of the above result.

THEOREM 11. *Weak MSO extended by both the recurrence quantifier R and the unbounding quantifier U defines the same languages as boolean combinations of max-automata and min-automata. If the formula does not use R, then min-automata are not used in the combination, likewise for U and max-automata.*

The above theorem also gives a normal form for weak MSO with the quantifiers R and U. Take a formula φ of the logic, compile it into a boolean combination of automata as in the above corollary, and then compile each of those automata back into a formula. What we end up with is a boolean combination of formulas of the form $\mathsf{R}X.\varphi(X)$ or $\mathsf{U}X.\varphi(X)$, where $\varphi(X)$ is a formula of weak MSO without R or U. In other words, nesting the quantifiers R and U does not contribute anything to the expressive power of weak MSO.

4 Ultimately Periodic Quantifier

In this section we present another extension of weak MSO, and use the general framework to show that its emptiness problem is decidable.

The *ultimately periodic quantifier*, written P, is used to say that a set of positions is ultimately periodic. Specifically, if φ is a formula, and x is a first-order variable free in φ, then $\mathsf{P}x.\varphi(x)$ is true in a word if the set of positions x that satisfy φ is ultimately periodic (the variable x gets bound by the quantifier).

We now use the framework from the previous section to present an automaton model that captures weak MSO extended with the ultimately periodic quantifier. For $L \subseteq A^\omega$ and a word $a_1a_2 \ldots \in A^\omega$, we write

$$\text{suffix}_L(a_1a_2\ldots) = \{i \in \mathbb{N} : a_ia_{i+1}\ldots \in L\}$$

We define PS_L to be the set of words $w \in A^\omega$ where $\text{suffix}_L(w)$ is ultimately periodic. Any language of the form PS_L is called an *ultimately periodic acceptance condition*.

COROLLARY 12. *Weak MSO extended with the ultimately periodic quantifier has the same expressive power as boolean combinations of deterministic automata with Büchi and ultimately periodic acceptance conditions.*

PROOF. The nontrivial translation, from logic to automata, follows from Theorem 10. ∎

THEOREM 13. *Satisfiability is decidable for weak MSO extended with the ultimately periodic quantifier.*

PROOF. By Corollary 12, it suffices to decide emptiness for a boolean combination of deterministic automata with Büchi and ultimately periodic acceptance conditions. (The translations between formulas and automata are effective.) Since the acceptance conditions concerned are closed under homomorphic images, we may assume that the same transducer $f : A^\omega \to B^\omega$ is used by all automata. We may also assume that the boolean combination is in DNF form, and as far as emptiness is concerned, has only one disjunct (which is a conjunction of, possibly negated, acceptance conditions). Finally, by collapsing the Büchi languages into a single ω-regular language, we may assume one conjunct is ω-regular, and all others involve ultimately periodic acceptance conditions.

Summing up: we want to decide if the transducer f can output a word in an intersection $K \cap K_1 \cap \cdots \cap K_n$, where K is ω-regular and each K_i is either a language PS_{L_i} or its complement, for some ω-regular language L_i. It is not difficult to see that the following language over alphabet $\{0,1\}^n$ is ω-regular:

$$M = \{X_1 \otimes \cdots \otimes X_n : \text{ exists } w \in f(A^\omega) \cap K \text{ such that } X_i = \mathrm{suffix}_{L_i}(w) \text{ for all } i = 1, \ldots, n\}$$

(here $\otimes$ combines characteristic functions of sets into a word over the product alphabet).

The emptiness problem boils down to testing if the set M above contains a word, whose projection onto coordinates i corresponding to languages PS_{L_i} is an ultimately periodic word, and whose projection onto coordinates i corresponding to complements of languages PS_{L_i} is not ultimately periodic. This way we have reduced our satisfiability problem to the following combinatorial result, which can be solved using standard automata techniques. ∎

THEOREM 14. *The following problem is decidable*

- *Input: An ω-regular language $L \subseteq B^\omega$, letter-to-letter homomorphisms $\pi_i : B^\omega \to B_i^\omega$ for $i = 1, \ldots, n$, and a set $F \subseteq \{1, \ldots, n\}$.*
- *Question: Is there some $w \in L$ such that $F = \{i : \pi_i(w)$ is ultimately periodic$\}$.*

We could go even further, and consider an extension of weak MSO where all the new quantifiers mentioned in this work are allowed: the bounding quantifier, the recurrence quantifier, and the ultimately periodic quantifier. As previously, the automaton model would simply be boolean combinations of the three automata models: min-automata, max-automata, and automata with ultimately periodic acceptance condition. The emptiness problem would require solving a variant of Theorem 14 where the language L is not ω-regular, but recognized by a nondeterministic BS-automaton (since these are strong enough to capture both max- and min-automata).

5 Conclusions

In this paper we presented several new classes of languages of infinite words. These classes are robust: they have good closure properties, they admit logical and automaton characterizations, they have decidable emptiness. We hope that the examples from this paper,

together with the max-automata from [2], offer convincing proof that there are interesting generalizations of the concept of ω-regular language. The general theme is to look at deterministic automata with conditions that talk about asymptotic behavior, conditions more subtle than the usual "state q appears infinitely often".

One direction of future research is investigating the exact relationship between min-automata and the existing theory of distance automata and formal power series. Preliminary results show that such connections result in a PSPACE-upper bound for deciding emptiness of boolean combinations of min- and max-automata.

Finally, we intend to investigate a similar theory for tree languages.

References

[1] P. A. Abdulla, P. Krcál, and W. Yi. R-automata. In *CONCUR*, pages 67–81, 2008.

[2] M. Bojańczyk. Weak MSO with the unbounding quantifier. submitted.

[3] M. Bojańczyk. A bounding quantifier. In *Computer Science Logic*, volume 3210 of *Lecture Notes in Computer Science*, pages 41–55, 2004.

[4] M. Bojańczyk and T. Colcombet. Omega-regular expressions with bounds. In *Logic in Computer Science*, pages 285–296, 2006.

[5] O. Carton and W. Thomas. The monadic theory of morphic infinite words and generalizations. In *Mathematical Foundations of Computer Science*, volume 1893 of *Lecture Notes in Computer Science*, pages 275–284, 2000.

[6] T. Colcombet and C. Löding. The nesting-depth of disjunctive mu-calculus for tree languages and the limitedness problem. In *Computer Science Logic*, volume 5213 of *Lecture Notes in Computer Science*, 2008.

[7] C. C. Elgot and M. O. Rabin. Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *Journal of Symbolic Logic*, 31:169–181, 1966.

[8] D. Kirsten. Distance desert automata and the star height problem. *Theoretical Informatics and Applications*, 39(3):455–511, 2005.

[9] J.-É. Pin. Tropical semirings. In *Idempotency*, pages 50–69. Cambridge University Press, 1998.

[10] A. Rabinovich. On decidability of monadic logic of order over the naturals extended by monadic predicates. *Inf. Comput.*, 205(6):870–889, 2007.

[11] A. Rabinovich and W. Thomas. Decidable theories of the ordering of natural numbers with unary predicates. In *CSL*, pages 562–574, 2006.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

On Timed Alternating Simulation for Concurrent Timed Games

Laura Bozzelli[1], **Axel Legay**[1], **Sophie Pinchinat**[1]
IRISA, Campus de Beaulieu, 35042 Rennes Cedex, FRANCE.

ABSTRACT. We address the problem of alternating simulation refinement for concurrent timed games (TG). We show that checking timed alternating simulation between TG is EXPTIME-complete, and provide a logical characterization of this preorder in terms of a meaningful fragment of a new logic, TAMTL*. TAMTL* is an action-based timed extension of standard alternating-time temporal logic ATL*, which allows to quantify on strategies where the designated player is not responsible for blocking time. While for full TAMTL*, model-checking TG is undecidable, we show that for its fragment TAMTL, corresponding to the timed version of ATL, the problem is instead in EXPTIME.

1 Introduction

Refinement preorders constitute the standard mathematical approach to formalize the relation between abstract and concrete versions of the same system. Intuitively, an implementation I refines an abstraction A when each behavior of I is allowed by A. Refinement usually comes together with a logical setting to formally express the requirements preserved by the preorder. The goal is to ensure that the properties proved about the abstract description continue to hold in the refined version (i.e., the implementation). This scenario may arise either because the design is being carried out in an incremental fashion, or because the system is too complex and an abstraction needs to be used to verify its properties.

In the design and analysis of reactive and distributed component-based systems, refinement usually refers to a single component, whose behavior depends on assumptions on its environment (the other components). In this context, traditional refinement preorders, like simulation, are inappropriate because they do not distinguish between the behaviors of the component and those of its environment; so that, refinement also restrict the environment behaviors. Recently, [5, 10, 8] have addressed this problem and succeeded in an elegant solution for finite-state systems based on the game paradigm: the system is modeled by a multi-player finite-state concurrent game, where at each step, the next state is determined by considering the "intersection" between the choices (behavioral options) made simultaneously and independently by all the players (the components). Thus, one can keep all assumptions about a component separated from those of its environment. In this framework, simulation refinement becomes *alternating simulation* [5], a preorder which exploits the game setting and is defined according to a designated player (component):* an implementation I refines an abstraction A of the same component whenever any possible behavioral option of I is allowed by A, and controvariantly, any possible behavioral option of the environment of A is allowed by the environment of I. In this way, the refinement restricts the component behaviors without restricting the permissible environment behaviors.

*or, more in general, w.r.t. any subset of players (coalitions)

© Bozzelli, Legay, Pinchinat; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 85–96
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2309

While classical simulation preserves universal fragments of standard branching temporal logics designed for closed systems such as CTL* [11], alternating simulation for a given player preserves expressive fragments of *alternating-time temporal logics* designed for open systems such as ATL* [5, 4]. The latter is a convenient formalism for component-based systems modeled by finite-state concurrent games, where properties need to be guaranteed by a player irrespective of the behavior of the other players.

Our contribution. We address the problem of refinement for real-time component-based systems, agreeing on the crucial role of timed information in practical applications, *e.g.* in embedded–system applications. We extend the notion of alternating simulation refinement for finite-state concurrent games to the setting of (perfect-information) timed concurrent games (TG) with the element of surprise introduced in [9]. In this setting, at each step, players choose simultaneously and independently moves consisting of delayed actions: the move with the smallest delay is carried out and determines the next state (if the smallest delay is proposed by several players, then the move of one of them is chosen nondeterministically). Moreover, we propose the new logic TAMTL* as a language for specifying properties of timed component-based systems modeled by TG. TAMTL* is a real-time action-based extension of ATL*, in which the temporal operators correspond to those of the timed linear-time temporal logic MTL [13]. Differently from the known real-time extension of ATL*, namely TATL* [12], which is based on a dense-time continuous semantic (the system is observed at any point in time), we adopt a dense-time pointwise semantics (the system is observed through events) [16]. Furthermore and more importantly, we generalize the class of atomic formulas of MTL by introducing the notion of (*timed*) *multi-action constraint*. Intuitively, such constraints express requirements on the "observable" part of single steps along TG runs, i.e., the delay-action chosen by each player and the player which is selected in the current step. In this way we can directly express important properties such as the existence of reasonable strategies, that are strategies where the designated player is not responsible for blocking time progress. In TATL*, this is not directly possible: to express the above requirement we have to artificially extend the infinite labeled transition system (LTS) of the given TG in order to obtain another LTS that cannot be associated to any TG specification. Our main results are the following:

1. We show that checking timed alternating simulation between TG for a given player is EXPTIME-complete. The upper bound is proved by a non-trivial generalization of the region-abstraction approach used for checking timed simulation/bisimulation [7, 17]. The matching lower bound is shown by an easy and linear-time reduction from the problem of checking timed simulation, which is known to be EXPTIME-hard [14].
2. We provide a logical characterization of timed alternating simulation for a given player σ in terms of a meaningful fragment, σ-TAMTL$^*_P$, of TAMTL*, where strategy quantifiers are parameterized by σ and negation applies only to multi-action constraints. We show that a TG $\mathcal{A}$ is timed σ-simulated by a TG $\mathcal{B}$ precisely when each σ-TAMTL$^*_P$ formula that holds in $\mathcal{A}$ also holds in $\mathcal{B}$. To the best of our knowledge, this is the first paper that provides a full logical characterization for a timed refinement preorder.
3. While for unrestricted TAMTL*, model checking TG is undecidable (since TAMTL* subsumes MTL over infinite words [15]), we show that for its fragment TAMTL, where each temporal operator is immediately preceded by a strategy quantifier, the problem

is instead in EXPTIME. To do so, for each player σ, we associate to the given TG a region-abstraction finite-state turn-based game G_σ, and recursively reduce the problem to solving the games G_σ w.r.t. regular objectives. Compared to the TATL model checking algorithm in [12], our approach is direct and provides more insight on TG.

Here, we restrict our attention to the two-player case, but all results can be extended to the multi-player setting, where players play in coalitions. Details of this extension are deferred to the full version of this paper.

Related work. Refinement of real-time closed systems has been addressed in many papers (e.g. [3, 1, 17]), where systems are modeled by standard *timed automata* (TA) [3]. Timed language containment for TA is undecidable [3], while *timed simulation* [1, 17] between TA, which preserves the universal fragment of timed CTL (TCTL) [2], is EXPTIME-complete [17, 14]. For the open system setting, we are only aware of the recent work of Bulychev et al. [6], who propose timed simulation preorders for two-player timed games where partial observability is also taken into account. However, the games exploited there are asymmetric, which prevents a natural extension to the multi-player setting. Moreover, there are some significant restrictions on the model. For example, a player is enforced to play a discrete action if the invariant at the current location expires. Furthermore, their notion of preorder differs from ours in at least one crucial point: in their case, there is no interaction between the choices of opponent players in the underlying simulation game.

2 Preliminaries

2.1 Concurrent Timed Games

Let $\mathbb{R}_{\geq 0}$ be the set of non-negative reals and $\mathbb{Q}_{\geq 0}$ be the set of non-negative rational numbers. Fix a finite set of *clock* variables X. The set $C(X)$ of *clock constraints* (*over* X) is the set of boolean combinations of formulas of the form $x \sim c$, where $x \in X$, c is a natural number, and $\sim \in \{\leq, <\}$. A (*clock*) *valuation* (*over* X) is a function $v : X \to \mathbb{R}_{\geq 0}$ that maps every clock to a non-negative real number. Whether a valuation v *satisfies* a clock constraint $g \in C(X)$, denoted $v \models g$, is defined in a natural way. For $t \in \mathbb{R}_{\geq 0}$, the valuation $v + t$ is defined as $(v + t)(x) = v(x) + t$ for all $x \in X$. For $Y \subseteq X$, the valuation $v[Y := 0]$ is defined as $(v[Y := 0])(x) = 0$ if $x \in Y$ and $(v[Y := 0])(x) = v(x)$ otherwise.

DEFINITION 1.[3] *A* timed transition table (TT) *is a tuple* $\mathcal{T} = \langle Act, X, Q, \Delta, Inv \rangle$, *where* Act *is a finite set of actions,* Q *is a finite set of locations,* $\Delta \subseteq Q \times (Act \cup \{\bot\}) \times C(X) \times 2^X \times Q$ *is a finite* transition relation, *where* $\bot \notin Act$ *is the* null action, *and* $Inv : Q \to C(X)$ *maps each location to an invariant. We require that for each* $q \in Q$, *there is exactly one transition* $(q, \bot, g, Y, q')$ *from* q *associated with the null action; moreover,* $q' = q$, $g = true$, *and* $Y = \emptyset$.

A *state* of $\mathcal{T}$ is a pair (q, v) such that $q \in Q$, v is a valuation, and the invariant at location q is satisfied by v, i.e. $v \models Inv(q)$. The TT $\mathcal{T}$ induces an infinite-state labeled transition system (LTS) $[\![\mathcal{T}]\!] = \langle S, \to \rangle$ over the set of labels $\mathbb{R}_{\geq 0} \times (Act \cup \{\bot\}) \times \Delta$, where S is the set of $\mathcal{T}$-states, and the set of labeled edges $\to \subseteq S \times [\mathbb{R}_{\geq 0} \times (Act \cup \{\bot\}) \times \Delta] \times S$ is defined as: $(q, v) \xrightarrow{t,a,\delta} (q', v')$ *iff* $\delta = (q, a, g, Y, q')$, $v + t \models g$, $v' = (v + t)[Y := 0]$, and $v + t' \models Inv(q)$ for each $0 < t' \leq t$. Note that if $(q, v) \xrightarrow{t,\bot,\delta} (q', v')$, then $q' = q$ and $v' = v + t$.

DEFINITION 2.[9] *A* (two-player concurrent) timed game (TG) *is a tuple* $\mathcal{A} = \langle \mathcal{T}, s^0, Act_0, Act_1 \rangle$, *where* $\mathcal{T} = \langle Act, X, Q, \Delta, Inv \rangle$ *is a* TT, s^0 *is a designated initial state of* $\mathcal{T}$ *whose clock values are in* $\mathbb{Q}_{\geq 0}$, *and* $\{Act_0, Act_1\}$ *is a partition of* Act *with* $Act_0, Act_1 \neq \emptyset$.

A state of $\mathcal{A}$ is a state of $[\![\mathcal{T}]\!]$. For each $\sigma \in \{0,1\}$, let $Act_\sigma^\perp = Act_\sigma \cup \{\perp\}$. Intuitively, $Act_\sigma^\perp$ represents the set of actions for player σ. The set of moves $Mov_{\mathcal{A}}(\sigma)$ of player σ is given by $\mathbb{R}_{\geq 0} \times Act_\sigma^\perp \times \Delta$. For a state s, the set of *moves available to player* σ *in* s, written $Mov_{\mathcal{A}}(\sigma, s)$, is the set of moves $(t, a, \delta) \in Mov_{\mathcal{A}}(\sigma)$ such that $s \xrightarrow{t,a,\delta} s'$ for some state s', which is uniquely determined and is denoted by $Next_{\mathcal{A}}(s, \langle t, a, \delta \rangle)$. Observe that $Mov_{\mathcal{A}}(\sigma, s)$ is not empty since $(0, \perp, (q, \perp, true, \emptyset, q)) \in Mov_{\mathcal{A}}(\sigma, s)$, where q is the location of s.

The timed game is intuitively played as follows. In each state s, each player σ chooses a move $(t, a, \delta) \in Mov_{\mathcal{A}}(\sigma, s)$ indicating that the player wants to play the transition δ associated with the action a after a delay of t time units. The null action $\perp$ signifies the player's intention to remain idle for the specified time delay. The move with the shorter proposed time delay determines the next state of the game; if both player propose the same delay, then one of the chosen moves occurs non-deterministically. An outcome of the game corresponds to an infinite path of $[\![\mathcal{T}]\!]$ augmented with additional information. Before formalizing these notions, we recall that in the standard definition of TG (see e.g. [9]) a move of a player just consists of a timed delay followed by an action. This because the underlying TT is assumed to be time-deterministic, i.e. for each $(t, a) \in \mathbb{R}_{\geq 0} \times Act$ and state s, there is at most one transition δ such that $s \xrightarrow{t,a,\delta} s'$. Here, we have removed this restriction. Thus, to uniquely determine the next state, a player has to specify also the transition to be taken.

For moves $(t_0, a_0, \delta_0) \in Mov_{\mathcal{A}}(0, s)$ and $(t_1, a_1, \delta_1) \in Mov_{\mathcal{A}}(1, s)$, the *joint destination move*, written $JDM(\langle t_0, a_0, \delta_0 \rangle, \langle t_1, a_1, \delta_1 \rangle)$, is $\{\langle t_0, a_0, \delta_0 \rangle, \langle t_1, a_1, \delta_1 \rangle\}$ if $t_0 = t_1$, and the singleton $\{\langle t_k, a_k, \delta_k \rangle\}$ for the unique $k \in \{0,1\}$ such that $t_k < t_{1-k}$ otherwise.
A run of $\mathcal{A}$ is a finite or infinite sequence $\pi = s_0, \langle m_1^0, m_1^1, \sigma_1 \rangle, s_1, \langle m_2^0, m_2^1, \sigma_2 \rangle, s_2, \ldots$ such that for any k, $s_k \in S$, $m_{k+1}^0 \in Mov_{\mathcal{A}}(0, s_k)$, $m_{k+1}^1 \in Mov_{\mathcal{A}}(1, s_k)$, $\sigma_{k+1} \in \{0,1\}$, $m_{k+1}^{\sigma_{k+1}} \in JDM(m_{k+1}^0, m_{k+1}^1)$, and $s_{k+1} = Next_{\mathcal{A}}(s_k, m_{k+1}^{\sigma_{k+1}})$. For each k, we denote by π^k the suffix-run of π starting from state s_k, and by $\pi[0,k]$ the prefix-run of π leading to state s_k. The *duration* $DUR(\pi)$ of π is the sum of timestamps of the selected moves $m_{k+1}^{\sigma_{k+1}}$ along π. An infinite run π is *divergent* if $DUR(\pi) = +\infty$. Let $FRuns$ be the set of finite runs of $\mathcal{A}$. For $\pi \in FRuns$, we denote by $last(\pi)$ the last state of π. A *strategy* f_σ for player $\sigma \in \{0,1\}$ is a mapping $f_\sigma : FRuns \to Mov_{\mathcal{A}}(\sigma)$ assigning to each finite run π a move to be proposed by player σ at $last(\pi)$ such that $f_\sigma(\pi) \in Mov_A(\sigma, last(\pi))$. For each state s, the set of *outcomes of strategy* f_σ *from* s, $Outcomes_{\mathcal{A}}(\sigma, s, f_\sigma)$, is the set of all *infinite* runs $s_0, \langle m_1^0, m_1^1, \sigma_1 \rangle, s_1, \langle m_2^0, m_2^1, \sigma_2 \rangle, s_2 \ldots$ such that $s_0 = s$, and for each $k \geq 0$, $f_\sigma(s_0, \langle m_1^0, m_1^1, \sigma_1 \rangle, s_1, \ldots s_k) = m_{k+1}^\sigma$. Let $\pi = s_0, \langle m_1^0, m_1^1, \sigma_1 \rangle, s_1, \langle m_2^0, m_2^1, \sigma_2 \rangle, s_2 \ldots$ with $m_k^j = (t_k^j, a_k^j, \delta_k^j)$ (for each $j = 0, 1$ and $k \geq 1$). The *trace of* π, written $trace(\pi)$, is $\langle (t_1^0, a_1^0), (t_1^1, a_1^1), \sigma_1 \rangle, \langle (t_2^0, a_2^0), (t_2^1, a_2^1), \sigma_2 \rangle, \ldots$

We are also interested in strategies f_σ of player $\sigma \in \{0,1\}$ such that player σ is not responsible for blocking time progress [9]. Let $Blameless_\sigma$ be the set of infinite runs $\pi = s_0, \langle m_1^0, m_1^1, \sigma_1 \rangle, s_1, \ldots$ such that player σ is responsible only for finitely many steps, i.e. such that there is $k \geq 1$ so that for all $j \geq k$, $\sigma_j = 1 - \sigma$. Note that $Blameless_\sigma$ does not distinguish between runs which have the same trace. A strategy f_σ for player σ is *reasonable* in a state s iff for all runs π in $Outcomes_{\mathcal{A}}(\sigma, s, f_\sigma)$, either π is divergent or $\pi \in Blameless_\sigma$.

2.2 The logic TAMTL*

In this subsection, we introduce a real-time action-based extension of the alternating-time temporal logic ATL* [4], called TAMTL*, based on a dense-time pointwise semantics.
Fix two nonempty and disjoint sets of actions Act_0 and Act_1. A (*timed*) *multi-action* over (Act_0, Act_1) is a triple $\theta = \langle (t_0, a_0), (t_1, a_1), \sigma \rangle$, where $(t_i, a_i) \in \mathbb{R}_{\geq 0} \times Act_i^{\perp}$ for $i = 0, 1$, $\sigma \in \{0, 1\}$, and $t_\sigma \leq t_0, t_1$. Note that the traces of runs in TG on (Act_0, Act_1) are sequence of multi-actions. A (*timed*) *multi-action constraint* χ is a triple $\chi = \langle (a'_0, \sim_0 c_0), (a'_1, \sim_1 c_1), \sigma' \rangle$, where $\sigma' \in \{0, 1\}$ and for $i = 0, 1$, $a'_i \in Act_i^{\perp}$, $\sim_i \in \{=, <, \leq, >, \geq\}$, and $c_i \in \mathbb{Q}_{\geq 0}$. The above multi-action θ *satisfies* χ, written $\theta \models \chi$, iff $\sigma = \sigma'$ and for $i = 0, 1$, $a_i = a'_i$ and $t_i \sim_i c_i$.
The sets of *state formulas* φ and *path formulas* ψ of TAMTL* over (Act_0, Act_1) are defined as:

$$\varphi := true \mid \neg \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle\langle \sigma \rangle\rangle \psi \mid \langle\langle \sigma \rangle\rangle_{re} \psi$$
$$\psi := \chi \mid \varphi \mid \neg \psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \psi \, \mathcal{U}_I \, \psi \mid \psi \widetilde{\mathcal{U}}_I \psi$$

where $\sigma \in \{0, 1\}$, $\langle\langle \sigma \rangle\rangle$ and $\langle\langle \sigma \rangle\rangle_{re}$ are *strategy quantifiers*, where, intuitively, $\langle\langle \sigma \rangle\rangle_{re}$ is restricted to σ-reasonable strategies, χ is a multi-action constraint, $\mathcal{U}_I$ is the *constrained strict until operator*, where I is an interval with bounds in $\mathbb{Q}_{\geq 0} \cup \{+\infty\}$, and $\widetilde{\mathcal{U}}_I$ is the dual of $\mathcal{U}_I$.
The set of state formulas φ forms the language TAMTL*. TAMTL* is interpreted over states of TG. Let $\mathcal{A}$ be a TG over (Act_0, Act_1), s be a state of $\mathcal{A}$, and π be an infinite run of $\mathcal{A}$. For a state formula φ and a path formula ψ, the satisfaction relations $(\mathcal{A}, s) \models \varphi$ and $(\mathcal{A}, \pi) \models \psi$ are defined by induction as follows (we omit the rules for boolean connectives):

$(\mathcal{A}, s) \models \langle\langle \sigma \rangle\rangle \psi$	iff	there is a strategy f of player σ such that for all $\pi \in Outcomes_{\mathcal{A}}(\sigma, s, f)$, $(\mathcal{A}, \pi) \models \psi$
$(\mathcal{A}, s) \models \langle\langle \sigma \rangle\rangle_{re} \psi$	iff	there is a *reasonable* strategy f of player σ such that for all $\pi \in Outcomes_{\mathcal{A}}(\sigma, s, f)$, $(\mathcal{A}, \pi) \models \psi$ if π is divergent
$(\mathcal{A}, \pi) \models \chi$	iff	$trace(\pi) = \theta_0, \theta_1, \ldots$ and $\theta_0 \models \chi$
$(\mathcal{A}, \pi) \models \varphi$	iff	$\pi = s_0, \ldots$ and $(\mathcal{A}, s_0) \models \varphi$
$(\mathcal{A}, \pi) \models \psi_1 \, \mathcal{U}_I \, \psi_2$	iff	there is $i > 0$ such that $DUR(\pi[0, i]) \in I$, $(\mathcal{A}, \pi^i) \models \psi_2$, and $(\mathcal{A}, \pi^k) \models \psi_1$ for all $0 < k < i$
$(\mathcal{A}, \pi) \models \psi_1 \widetilde{\mathcal{U}}_I \psi_2$	iff	$(\mathcal{A}, \pi) \models \neg((\neg\psi_1) \, \mathcal{U}_I (\neg\psi_2))$

We write $\mathcal{A} \models \varphi$ to mean that $(\mathcal{A}, s^0) \models \varphi$ for the initial state s^0 of $\mathcal{A}$. We use some standard shortcuts: $\Diamond_I \psi := true \, \mathcal{U}_I \, \psi$ (*eventually*), $\Box_I \psi := \neg \Diamond_I \neg \psi$ (*always*), and $\bigcirc_I \psi := (\neg true) \, \mathcal{U}_I \, \psi$ (*next*). We omit the subscript I when $I = \mathbb{R}_{\geq 0}$. We denote by TAMTL the fragment of TAMTL* consisting of the formulas in which every temporal operator is immediately preceded by a strategy quantifier. Moreover, for $\sigma \in \{0, 1\}$, let σ-TAMTL$^*_P$ be the fragment of TAMTL* (not closed under negation) in which all strategy quantifiers are parameterized by σ, and negation is applied only to multi-action constraints. Intuitively, σ-TAMTL$^*_P$ formulas describe behaviors that player σ can enforce no matter what player $1 - \sigma$ does. Note that since TG are not determined [9], the dual σ-$\widetilde{\text{TAMTL}}^*_P$ of σ-TAMTL$^*_P$ does not correspond to $(1 - \sigma)$-TAMTL$^*_P$. By the equivalence below, it follows that in fact $\langle\langle \sigma \rangle\rangle_{re}$ is a *derivate operator* in TAMTL* and also in σ-TAMTL$^*_P$. However, this does not hold for the logic TAMTL.

$$\langle\langle \sigma \rangle\rangle_{re} \psi \equiv \langle\langle \sigma \rangle\rangle \big(((\Box \Diamond_{[1,\infty[} true) \rightarrow \psi) \wedge (\neg(\Box \Diamond_{[1,\infty[} true) \rightarrow \psi_{blameless_\sigma}) \big)$$
$$\psi_{blameless_\sigma} := \Diamond \Box \Big(\bigvee_{a_0 \in Act_0^{\perp}} \bigvee_{a_1 \in Act_1^{\perp}} \langle (a_0, \geq 0), (a_1, \geq 0), 1 - \sigma \rangle \Big)$$

EXAMPLE 3. *Let* $Act_0 = \{a\}$ *and* $Act_1 = \{b\}$. *The* 0-*TAMTL*$^*_P$ *formula* $\langle\langle 0\rangle\rangle_{re} \Box(\langle\langle(a, \geq 0), (b, \geq 0), 0\rangle \rightarrow \Diamond_{[1,1]}\langle\langle(a, \geq 0), (b, \geq 0), 1\rangle\rangle)$ *requires that player* 0 *has a reasonable strategy ensuring that along every its divergent outcome, every* a*-event (i.e., the action* a *is selected in the current step) is followed one time unit later by a* b*-event.*

3 Timed Alternating Simulation

In this section, we introduce the notion of timed alternating simulation between TG which generalizes alternating simulation between finite-state concurrent games [5].

Fix two *comparable* TG $\mathcal{A} = \langle \mathcal{T}_\mathcal{A}, s_0^\mathcal{A}, Act_0^\mathcal{A}, Act_1^\mathcal{A}\rangle$ and $\mathcal{B} = \langle \mathcal{T}_\mathcal{B}, s_0^\mathcal{B}, Act_0^\mathcal{B}, Act_1^\mathcal{B}\rangle$, i.e. such that $Act_0^\mathcal{A} = Act_0^\mathcal{B}$ and $Act_1^\mathcal{A} = Act_1^\mathcal{B}$. Let $S_\mathcal{A}$ (resp., $S_\mathcal{B}$) be the set of states of $\mathcal{A}$ (resp., $\mathcal{B}$).

DEFINITION 4. *For a player* $\sigma \in \{0,1\}$, *a relation* $H \subseteq S_\mathcal{A} \times S_\mathcal{B}$ *is a* timed alternating simulation for player σ from $\mathcal{A}$ to $\mathcal{B}$ *iff for all* $(s_\mathcal{A}, s_\mathcal{B}) \in H$, *the following holds:*

- *for every move* $m_\sigma^\mathcal{A} = (t, a, \delta_\mathcal{A}) \in Mov_\mathcal{A}(\sigma, s_\mathcal{A})$, *there is a matching move* $m_\sigma^\mathcal{B} = (t, a, \delta_\mathcal{B}) \in Mov_\mathcal{B}(\sigma, s_\mathcal{B})$ *such that for every move* $m_{1-\sigma}^\mathcal{B} = (t', b, \delta'_\mathcal{B}) \in Mov_\mathcal{B}(1-\sigma, s_\mathcal{B})$, *there is a matching move* $m_{1-\sigma}^\mathcal{A} = (t', b, \delta'_\mathcal{A}) \in Mov_\mathcal{A}(1-\sigma, s_\mathcal{A})$ *so that for all* $i = 0, 1$,

$$m_i^\mathcal{A} \in JDM(m_0^\mathcal{A}, m_1^\mathcal{A}) \quad \textit{implies} \quad (Next_\mathcal{A}(s_\mathcal{A}, m_i^\mathcal{A}), Next_\mathcal{B}(s_\mathcal{B}, m_i^\mathcal{B})) \in H$$

Note that $m_i^\mathcal{B} \in JDM(m_0^\mathcal{B}, m_1^\mathcal{B})$. If there is a timed alternating simulation H for player σ from $\mathcal{A}$ to $\mathcal{B}$ such that $(s_0^\mathcal{A}, s_0^\mathcal{B}) \in H$, we say that $\mathcal{B}$ *timed* σ*-simulates* $\mathcal{A}$, and we write $\mathcal{A} \preceq_\sigma \mathcal{B}$. Note that $\preceq_\sigma$ is a preorder on TG. We can give a game-theoretic interpretation of timed alternating simulation for a player $\sigma \in \{0,1\}$. Consider the following two-player *turn-based* game whose set of *main* positions is $S_\mathcal{A} \times S_\mathcal{B}$. The initial position is $(s_0^\mathcal{A}, s_0^\mathcal{B})$. Each round consists of five steps as follows. Assume that the current main position is $(s_\mathcal{A}, s_\mathcal{B})$. Then:

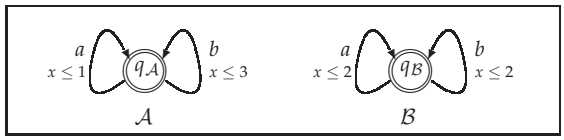
1. The antagonist chooses a move $m_\sigma^\mathcal{A} = (t, a, \delta_\mathcal{A}) \in Mov_\mathcal{A}(\sigma, s_\mathcal{A})$ of player σ in $\mathcal{A}$ available at state $s_\mathcal{A}$, and moves to position $p_1 = (s_\mathcal{A}, s_\mathcal{B}, m_\sigma^\mathcal{A})$.
2. The protagonist, from p_1, chooses a matching move $m_\sigma^\mathcal{B} = (t, a, \delta_\mathcal{B}) \in Mov_\mathcal{B}(\sigma, s_\mathcal{B})$ of player σ in $\mathcal{B}$ available at state $s_\mathcal{B}$, and moves to position $p_2 = (s_\mathcal{A}, s_\mathcal{B}, m_\sigma^\mathcal{A}, m_\sigma^\mathcal{B})$.
3. The antagonist, from p_2, chooses a move $m_{1-\sigma}^\mathcal{B} = (t', b, \delta'_\mathcal{B}) \in Mov_\mathcal{B}(1-\sigma, s_\mathcal{B})$ of player $1-\sigma$ in $\mathcal{B}$ available at state $s_\mathcal{B}$, and moves to position $p_3 = (s_\mathcal{A}, s_\mathcal{B}, m_\sigma^\mathcal{A}, m_\sigma^\mathcal{B}, m_{1-\sigma}^\mathcal{B})$.
4. The protagonist, from p_3, chooses a matching move $m_{1-\sigma}^\mathcal{A} = (t', b, \delta'_\mathcal{A}) \in Mov_\mathcal{A}(\sigma, s_\mathcal{A})$ of player $1-\sigma$ in $\mathcal{A}$ available at state $s_\mathcal{A}$, and moves to $p_4 = (s_\mathcal{A}, s_\mathcal{B}, m_\sigma^\mathcal{A}, m_\sigma^\mathcal{B}, m_{1-\sigma}^\mathcal{B}, m_{1-\sigma}^\mathcal{A})$.
5. The antagonist, from position p_4, chooses $i = 0, 1$ such that $m_i^\mathcal{A} \in JDM(m_0^\mathcal{A}, m_1^\mathcal{A})$, and moves to the main position $(Next_\mathcal{A}(s_\mathcal{A}, m_i^\mathcal{A}), Next_\mathcal{B}(s_\mathcal{B}, m_i^\mathcal{B}))$.

If the game proceeds ad infinitum, then the protagonist wins. Otherwise, the game reaches a position from which the protagonist cannot choose in steps 2 or 4 above a matching move, and the antagonist wins. It easily follows that $\mathcal{B}$ timed σ-simulates $\mathcal{A}$ iff the protagonist has a winning strategy. Note that for each $\sigma \in \{0,1\}$, we have a different turn-based game.

Intuitively, $\mathcal{B}$ timed σ-simulates $\mathcal{A}$ iff player σ is more powerful in game $\mathcal{B}$ than in game $\mathcal{A}$, i.e. each behavior that player σ can induce in $\mathcal{A}$, it can also induce in $\mathcal{B}$. The following lemma formalizes this intuition. Let $H \subseteq S_\mathcal{A} \times S_\mathcal{B}$. For a run π of $\mathcal{A}$ and a run π' of $\mathcal{B}$ having the same length, we write $H(\pi, \pi')$ to mean that for each prefix-run $\pi[0,k]$ of π, $(last(\pi[0,k]), last(\pi'[0,k])) \in H$.

LEMMA 5. *Let H be a timed alternating simulation for player $\sigma \in \{0,1\}$ from $\mathcal{A}$ to $\mathcal{B}$. Then, for all $(s_\mathcal{A}, s_\mathcal{B}) \in H$ and strategy $f_\mathcal{A}$ of player σ in $\mathcal{A}$, there exists a strategy $f_\mathcal{B}$ of player σ in $\mathcal{B}$ such that for every run $\pi_\mathcal{B} \in Outcomes_\mathcal{B}(\sigma, s_\mathcal{B}, f_\mathcal{B})$, there exists a run $\pi_\mathcal{A} \in Outcomes_\mathcal{A}(\sigma, s_\mathcal{A}, f_\mathcal{A})$ so that $H(\pi_\mathcal{A}, \pi_\mathcal{B})$ and $trace(\pi_\mathcal{A}) = trace(\pi_\mathcal{B})$.*

EXAMPLE 6. *The figure depicts two simple TG $\mathcal{A}$ and $\mathcal{B}$ with $Act_0 = \{a\}$ and $Act_1 = \{b\}$. Let $s^0_\mathcal{A} = (q_\mathcal{A}, v)$ and $s^0_\mathcal{B} = (q_\mathcal{B}, v)$ be the initial states of $\mathcal{A}$ and B, where v is any valuation with $v(x) \leq 1$. It easily follows that $\mathcal{B}$ timed 0-simulates $\mathcal{A}$ and $\mathcal{A}$ timed 1-simulates B, but the vice versa of each of two conditions does not hold. Moreover, note that there exists no (standard) timed simulation from $\mathcal{A}$ to $\mathcal{B}$ and vice versa (w.r.t. the given initial states).*



3.1 Checking Timed Alternating Simulation

In this subsection, we show that for given comparable TG $\mathcal{A}$ and $\mathcal{B}$, and player $\sigma \in \{0,1\}$, checking whether $\mathcal{A} \preceq_\sigma \mathcal{B}$ is decidable via a suitable *region* abstraction, and the check can be done in exponential time. Fix two comparable TG $\mathcal{A} = \langle \mathcal{T}_\mathcal{A}, s^\mathcal{A}_0, Act_0, Act_1 \rangle$ and $\mathcal{B} = \langle \mathcal{T}_\mathcal{B}, s^\mathcal{B}_0, Act_0, Act_1 \rangle$. Let $S_\mathcal{A}$ (resp., $S_\mathcal{B}$) be the set of states of $\mathcal{A}$ (resp., $\mathcal{B}$), and let $X_\mathcal{A}$ (resp., $X_\mathcal{B}$) be the set of clocks of $\mathcal{A}$ (resp., $\mathcal{B}$). W.l.o.g. we can assume that $X_\mathcal{A} \cap X_\mathcal{B} = \emptyset$.

Region equivalence [3]: we denote by K_{max} the largest constant occurring in the clock constraints of $\mathcal{A}$ and $\mathcal{B}$. Given a clock valuation $v_\mathcal{A}$ over $X_\mathcal{A}$ and a clock valuation $v_\mathcal{B}$ over $X_\mathcal{B}$, the clock valuation $v_\mathcal{A} \| v_\mathcal{B}$ over $X_\mathcal{A} \cup X_\mathcal{B}$ is defined in the obvious way (recall that $X_\mathcal{A} \cap X_\mathcal{B} = \emptyset$). For $t \in \mathbb{R}_{\geq 0}$, $\lfloor t \rfloor$ denotes the integral part of t and $fract(t)$ denotes its fractional part. The *region equivalence relation over $S_\mathcal{A} \times S_\mathcal{B}$*, written $\approx_{\mathcal{A}\|\mathcal{B}}$, is defined as follows: $((q_\mathcal{A}, v_\mathcal{A}), (q_\mathcal{B}, v_\mathcal{B})) \approx_{\mathcal{A}\|\mathcal{B}} ((q'_\mathcal{A}, v'_\mathcal{A}), (q'_\mathcal{B}, v'_\mathcal{B}))$ iff $q_\mathcal{A} = q'_\mathcal{A}$, $q_\mathcal{B} = q'_\mathcal{B}$, and for each $x \in X_\mathcal{A} \cup X_\mathcal{B}$, either both $(v_\mathcal{A} \| v_\mathcal{B})(x), (v'_\mathcal{A} \| v'_\mathcal{B})(x) > K_{max}$, or the following holds:

- $\lfloor (v_\mathcal{A} \| v_\mathcal{B})(x) \rfloor = \lfloor (v'_\mathcal{A} \| v'_\mathcal{B})(x) \rfloor$ and $fract((v_\mathcal{A} \| v_\mathcal{B})(x)) = 0$ iff $fract((v'_\mathcal{A} \| v'_\mathcal{B})(x)) = 0$;
- for each $y \in X_\mathcal{A} \cup X_\mathcal{B}$ s.t. $(v_\mathcal{A} \| v_\mathcal{B})(y) \leq K_{max}$, $fract((v_\mathcal{A} \| v_\mathcal{B})(x)) \leq fract((v_\mathcal{A} \| v_\mathcal{B})(y))$ iff $fract((v'_\mathcal{A} \| v'_\mathcal{B})(x)) \leq fract((v'_\mathcal{A} \| v'_\mathcal{B})(y))$ (ordering of the fractional parts).

Let $Reg_{\mathcal{A}\|\mathcal{B}}$ be the set of equivalence classes of $\approx_{\mathcal{A}\|\mathcal{B}}$, called *regions*. By [3], $Reg_{\mathcal{A}\|\mathcal{B}}$ is finite and its size is singly exponential in the sizes of $\mathcal{A}$ and $\mathcal{B}$.

Finite Sampling of $\mathbb{R}_{\geq 0}$: let $(s_\mathcal{A}, s_\mathcal{B}) \in S_\mathcal{A} \times S_\mathcal{B}$ and $x_1, \ldots, x_n$ be the clocks in $X_\mathcal{A} \cup X_\mathcal{B}$ whose values $t_1, \ldots, t_n$ in $(s_\mathcal{A}, s_\mathcal{B})$ are not greater than K_{max}. Assume w.l.o.g. that $\tau_1 \leq \tau_2 \leq \ldots \leq \tau_n$, where $\tau_i = fract(t_i)$ for $1 \leq i \leq n$. Let $\tau_0 = 0$, $\tau_{n+1} = 1$, and $min(s_\mathcal{A}, s_\mathcal{B}) = min\{\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor\}$. We consider the following *finite* set of real numbers:

$$Times(s_\mathcal{A}, s_B) = \{h - \tfrac{1}{2}(\tau_i + \tau_{i+1}) \mid i = 0, \ldots, n \text{ and } h = 1, \ldots, K_{max} - min(s_\mathcal{A}, s_\mathcal{B})\} \cup$$
$$\{h - \tau_i \mid i = 1, \ldots, n \text{ and } h = 1, \ldots, K_{max} - min(s_\mathcal{A}, s_\mathcal{B})\} \cup \{0, \ldots, K_{max} + 1 - min(s_\mathcal{A}, s_\mathcal{B})\}\}$$

Thus, $Times(s_\mathcal{A}, s_B)$ consists of the integers in $\{0, \ldots, K_{max} + 1 - min(s_\mathcal{A}, s_\mathcal{B})\}$ plus the distances between the points p and the integers $1, \ldots, K_{max} - min(s_\mathcal{A}, s_\mathcal{B})$, where p is either a fractional part τ_j or the mid-point of some interval $[\tau_i, \tau_{i+1}]$ with $0 \leq i \leq n$. Intuitively, the *distance* d between a mid-point $\frac{1}{2}(\tau_i + \tau_{i+1})$ and an integer $h = 1, \ldots, K_{max} - min(s_\mathcal{A}, s_\mathcal{B})$

is used as a representative for all timestamps t such that $h - \tau_{i+1} < t < h - \tau_i$ (formally, $(v_{\mathcal{A}} \| v_{\mathcal{B}}) + t \approx_{\mathcal{A}\|\mathcal{B}} (v_{\mathcal{A}} \| v_{\mathcal{B}}) + d$, where $v_{\mathcal{A}}$ and $v_{\mathcal{B}}$ are the clock valuations of $s_{\mathcal{A}}$ and $s_{\mathcal{B}}$).

Checking if $\mathcal{A} \preceq_\sigma \mathcal{B}$ for $\sigma \in \{0,1\}$: let H_σ^{max} be the maximal timed alternating simulation for player σ from $\mathcal{A}$ to $\mathcal{B}$. We show that H_σ^{max} is a computable union of regions.

DEFINITION 7. *[Goodness] Let $\Gamma \subseteq Reg_{\mathcal{A}\|\mathcal{B}}$ be a set of regions and let $R \in \Gamma$. We say that R* is good in Γ w.r.t. player $\sigma \in \{0,1\}$ iff *there is $(s_{\mathcal{A}}, s_{\mathcal{B}}) \in R$ such that:*

1. *for every move $m_\sigma^{\mathcal{A}} = (t, a, \delta_{\mathcal{A}}) \in Mov_{\mathcal{A}}(\sigma, s_{\mathcal{A}})$ with $t \in Times(s_{\mathcal{A}}, s_{\mathcal{B}})$, there is a matching move $m_\sigma^{\mathcal{B}} = (t, a, \delta_{\mathcal{B}}) \in Mov_{\mathcal{B}}(\sigma, s_{\mathcal{B}})$ such that for every move $m_{1-\sigma}^{\mathcal{B}} = (t', b, \delta'_{\mathcal{B}}) \in Mov_{\mathcal{B}}(1-\sigma, s_{\mathcal{B}})$ with $t' \in Times(s_{\mathcal{A}}, s_{\mathcal{B}})$, there is a matching move $m_{1-\sigma}^{\mathcal{A}} = (t', b, \delta'_{\mathcal{A}}) \in Mov_{\mathcal{A}}(1-\sigma, s_{\mathcal{A}})$ so that for all $i = 0, 1$ with $m_i^{\mathcal{A}} \in JDM(m_0^{\mathcal{A}}, m_1^{\mathcal{A}})$,*
$$(Next_{\mathcal{A}}(s_{\mathcal{A}}, m_i^{\mathcal{A}}), Next_{\mathcal{B}}(s_{\mathcal{B}}, m_i^{\mathcal{B}})) \in R_i \text{ for some } R_i \in \Gamma$$

Fix $\sigma \in \{0,1\}$ and let $\Omega_\sigma : 2^{Reg_{\mathcal{A}\|\mathcal{B}}} \to 2^{Reg_{\mathcal{A}\|\mathcal{B}}}$ be the *monotone* operator defined as follows: $\Omega_\sigma(\Gamma) = \{R \in \Gamma \mid R \text{ is good in } \Gamma \text{ w.r.t. player } \sigma\}$. We show that Ω_σ is computable and $H_\sigma^{max} = \bigcup_{R \in \Gamma_{max}} R$, where Γ_{max} is the greatest fixpoint of Ω_σ. For this, we need the following crucial technical lemma, which extends the result given in [17] for timed simulation.

LEMMA 8. *Let $\Gamma \subseteq Reg_{\mathcal{A}\|\mathcal{B}}$ be a set of regions and $R \in \Gamma$ such that R is good in Γ w.r.t. player σ. Then, each $(s_{\mathcal{A}}, s_{\mathcal{B}}) \in R$ satisfies the condition obtained from Condition 1 in Definition 7 by removing the constraint that the timestamps have to be chosen in $Times(s_{\mathcal{A}}, s_{\mathcal{B}})$.*

Let $H \subseteq S_{\mathcal{A}} \times S_{\mathcal{B}}$ be a timed alternating simulation for player σ from $\mathcal{A}$ to $\mathcal{B}$. We denote by Γ_H the set $\Gamma_H = \{R \in Reg_{\mathcal{A}\|\mathcal{B}} \mid R \cap H \neq \emptyset\}$. By Definition 7, the following holds.

LEMMA 9. *If $H \subseteq S_{\mathcal{A}} \times S_{\mathcal{B}}$ is a timed alternating simulation for player σ from $\mathcal{A}$ to $\mathcal{B}$, then Γ_H is a fixpoint of Ω_σ.*

By Lemmata 8 and 9, we obtain the following results.

COROLLARY 10. *If $H \subseteq S_{\mathcal{A}} \times S_{\mathcal{B}}$ is a timed alternating simulation for player σ from $\mathcal{A}$ to $\mathcal{B}$, then $\bigcup_{R \in \Gamma_H} R$ is a timed alternating simulation for player σ from $\mathcal{A}$ to $\mathcal{B}$.*

COROLLARY 11. *Let $\Gamma \subseteq Reg_{\mathcal{A}\|\mathcal{B}}$ be a set of regions. Then, $\Omega_\sigma(\Gamma) = \Gamma$ iff $\bigcup_{R \in \Gamma} R$ is a timed alternating simulation for player σ from $\mathcal{A}$ to $\mathcal{B}$.*

By Corollary 10, H_σ^{max} is a union of regions in $Reg_{\mathcal{A}\|\mathcal{B}}$, and by Corollary 11, $H_\sigma^{max} = \bigcup_{R \in \Gamma_{max}} R$, where Γ_{max} is the greatest fixpoint of Ω_σ. Note that Γ_{max} can be obtained by iterative applications of Ω_σ starting with $\Gamma_0 = Reg_{\mathcal{A}\|\mathcal{B}}$. There can be at most $|Reg_{\mathcal{A}\|\mathcal{B}}|$ many iterations. Moreover, by Lemma 8, Condition 1 in Definition 7 is independent on what representative is chosen for the given equivalence class. Since $|Times(s_{\mathcal{A}}, s_{\mathcal{B}})|$ for $(s_{\mathcal{A}}, s_{\mathcal{B}}) \in S_{\mathcal{A}} \times S_{\mathcal{B}}$ and $|Reg_{\mathcal{A}\|\mathcal{B}}|$ are singly exponential in the sizes of $\mathcal{A}$ and $\mathcal{B}$, if follows that $\Omega_\sigma(\Gamma)$ for given $\Gamma \subseteq Reg_{\mathcal{A}\|\mathcal{B}}$ can be computed in single exponential time in the sizes of $\mathcal{A}$ and $\mathcal{B}$. Since $\mathcal{A} \preceq_\sigma \mathcal{B}$ iff $(s_0^{\mathcal{A}}, s_0^{\mathcal{B}}) \in H_\sigma^{max}$, checking whether $\mathcal{A} \preceq_\sigma \mathcal{B}$ is in EXPTIME. We can show that the problem is also EXPTIME-hard by a straightforward and linear reduction from checking timed simulation between TT, which is EXPTIME-hard [14]. Thus, we obtain the following.

THEOREM 12. *Given two comparable TG $\mathcal{A}$ and $\mathcal{B}$ and player $\sigma \in \{0,1\}$, the problem of checking whether $\mathcal{A} \preceq_\sigma \mathcal{B}$ is* EXPTIME*-complete.*

3.2 Logical characterization of timed alternating simulation

In this subsection, we give a logical characterization of timed alternating simulation for a given player $\sigma \in \{0,1\}$ in terms of the fragment σ-TAMTL$^*_P$ of TAMTL*.

THEOREM 13. *Let $\mathcal{A}$ and $\mathcal{B}$ be two TG over (Act_0, Act_1) and $\sigma \in \{0,1\}$. Then, $\mathcal{A} \preceq_\sigma \mathcal{B}$ if and only if for every σ-TAMTL$^*_P$ formula φ, $\mathcal{A} \models \varphi$ implies $\mathcal{B} \models \varphi$. Hence, $\mathcal{A} \preceq_\sigma \mathcal{B}$ if and only if for every σ-$\widetilde{\text{TAMTL}}^*_P$ formula $\widetilde{\varphi}$, $\mathcal{B} \models \widetilde{\varphi}$ implies $\mathcal{A} \models \widetilde{\varphi}$.*

Sketched proof. For the direct implication ($\Rightarrow$), it suffices to show that if H is a timed alternating simulation for player σ from $\mathcal{A}$ to $\mathcal{B}$, then the following holds:

1. for all σ-TAMTL$^*_P$ (state) formulas φ and $(s_\mathcal{A}, s_\mathcal{B}) \in H$, $(\mathcal{A}, s_\mathcal{A}) \models \varphi$ implies $(\mathcal{B}, s_\mathcal{B}) \models \varphi$.
2. for all path formulas ψ of σ-TAMTL$^*_P$ and for all infinite runs $\pi_\mathcal{A}$ of $\mathcal{A}$ and $\pi_\mathcal{B}$ of $\mathcal{B}$ s.t. $H(\pi_\mathcal{A}, \pi_\mathcal{B})$ and $trace(\pi_\mathcal{A}) = trace(\pi_\mathcal{B})$, $(\mathcal{A}, \pi_\mathcal{A}) \models \psi$ implies $(\mathcal{B}, \pi_\mathcal{B}) \models \psi$.

The proof is by induction on the structure of formulas. The non-trivial case is that of state formulas of the form $\langle\langle\sigma\rangle\rangle\psi$ (recall that $\langle\langle\sigma\rangle\rangle_{re}$ is a derivate operator in σ-TAMTL$^*_P$). Assume that $(s_\mathcal{A}, s_\mathcal{B}) \in H$ and $(\mathcal{A}, s_\mathcal{A}) \models \langle\langle\sigma\rangle\rangle\psi$. Thus, there is a strategy $f_\mathcal{A}$ of player σ in $\mathcal{A}$ such that for each outcome $\pi_\mathcal{A}$ of $f_\mathcal{A}$ from $s_\mathcal{A}$, $(\mathcal{A}, \pi_\mathcal{A}) \models \psi$. Since $(s_\mathcal{A}, s_\mathcal{B}) \in H$, by Lemma 5, there is a strategy $f_\mathcal{B}$ of player σ in $\mathcal{B}$ such that for each outcome $\pi_\mathcal{B}$ of $f_\mathcal{B}$ from $s_\mathcal{B}$, there is an outcome $\pi_\mathcal{A}$ of $f_\mathcal{A}$ from $s_\mathcal{A}$ so that $H(\pi_\mathcal{A}, \pi_\mathcal{B})$ and $trace(\pi_\mathcal{A}) = trace(\pi_\mathcal{B})$. By ind. hyp., Property 2 holds for the path formula ψ. Hence, evidently, the result follows.

For the converse implication ($\Leftarrow$) of the theorem, assume that $\mathcal{A} \not\preceq_\sigma \mathcal{B}$. Let $\sigma = 0$ (the other case is symmetric). We need to prove that for some 0-TAMTL$^*_P$ formula φ, $\mathcal{A} \models \varphi$ and $\mathcal{B} \not\models \varphi$. Consider the turn-based 0-simulation game G_0 between the antagonist and the protagonist on page 90. By the results of Subsection 3.1 we can assume that the timestamps chosen by the antagonist are in the finite set $Times(s_\mathcal{A}, s_\mathcal{B})$, where $(s_\mathcal{A}, s_\mathcal{B})$ is the main current position of the game. It follows that G_0 is finitely-branching. Since $\mathcal{A} \not\preceq_0 \mathcal{B}$, the antagonist has a winning strategy f starting from $(s_0^\mathcal{A}, s_0^\mathcal{B})$, where $s_0^\mathcal{A}$ (resp., $s_0^\mathcal{B}$) is the initial state of $\mathcal{A}$ (resp., $\mathcal{B}$) whose clock-values are *rational*. Hence, the strategy-tree T_f of f from $(s_0^\mathcal{A}, s_0^\mathcal{B})$ is *finite*, and (by def. of *Times*) the timestamps of the moves along the edges of T_f are *rational*. We claim that for each node x_p of T_f labeled by a main position $p = (s_\mathcal{A}, s_\mathcal{B}) \in S_\mathcal{A} \times S_\mathcal{B}$, there is a 0-TAMTL$^*_P$ formula φ_p such that $(\mathcal{A}, s_\mathcal{A}) \models \varphi_p$ and $(\mathcal{B}, s_\mathcal{B}) \not\models \varphi_p$. Hence, the result follows. The proof is by induction on the height of the subtree of T_f rooted at node x_p. By construction, x_p has exactly one child, say x'_p, and the edge from x_p to x'_p corresponds to a move $m^0_\mathcal{A} = (t, a, \delta_\mathcal{A})$ for player 0 in $Mov_\mathcal{A}(0, s_\mathcal{A})$ with $t \in Times(s_\mathcal{A}, s_\mathcal{B}) \subseteq \mathbb{Q}_{\geq 0}$ chosen by the antagonist in Step 1 on page 90. Moreover, the edges from x'_p to its children $y_1, \ldots, y_n$, if any, correspond to *all and only* the matching moves $m^0_\mathcal{B} = (t, a, \delta_\mathcal{B}) \in Mov_\mathcal{B}(0, s_\mathcal{B})$ of $m^0_\mathcal{A}$ for player 0 in $\mathcal{B}$ from $s_\mathcal{B}$. If $n = 0$ (base case), there is no such a matching move. In this case, the 0-TAMTL$^*_P$ formula φ_p satisfying the claim is $\langle\langle 0\rangle\rangle(\bigvee_{b \in Act_1^\perp} \bigvee_{\kappa \in \{0,1\}} \langle (a, = t), (b, \geq 0), \kappa\rangle)$.

Now, assume that $n \geq 1$. By construction, for each $1 \leq i \leq n$, y_i has a unique child y'_i and the edge from y_i to y'_i is associated with some move $(t', b, \delta'_\mathcal{B}) \in Mov_\mathcal{B}(1, s_\mathcal{B})$ (depending on i) with $t' \in Times(s_\mathcal{A}, s_\mathcal{B}) \subseteq \mathbb{Q}_{\geq 0}$ chosen by the antagonist in Step 3 on page 90. Moreover, the edges from y'_i to its children $z_{i,1}, \ldots, z_{i,m_i}$ represent *all and only* the possible matching moves $(t', b, \delta'_\mathcal{A}) \in Mov_\mathcal{A}(1, s_\mathcal{A})$ (for player 1 in $\mathcal{A}$ from $s_\mathcal{A}$) of the move $(t', b, \delta'_\mathcal{B}) \in Mov_\mathcal{B}(1, s_\mathcal{B})$. Assume that for each $1 \leq i \leq n$, $m_i \geq 1$, i.e. y'_i is not a leaf (the

other case being simpler). By construction, for each $1 \leq l \leq m_i$, $z_{i,l}$ has a unique child $z'_{i,l}$, which is labeled by a main position in $S_{\mathcal{A}} \times S_{\mathcal{B}}$, and the edge from $z_{i,l}$ to $z'_{i,l}$ corresponds to a choice $\kappa = 0, 1$ of the antagonist in Step 5 on page 90. We distinguish two cases:

- $\exists 1 \leq i \leq n.\ \forall 1 \leq l \leq m_i$: the edge from $z_{i,l}$ to $z'_{i,l}$ is associated with the choice $\kappa = 1$;
- $\forall 1 \leq i \leq n.\ \exists 1 \leq l \leq m_i$: the edge from $z_{i,l}$ to $z'_{i,l}$ is associated with the choice $\kappa = 0$.

Here, we focus on the first case. Let $m^1_{\mathcal{B}} = (t', b, \delta'_{\mathcal{B}}) \in Mov_{\mathcal{B}}(1, s_{\mathcal{B}})$ be the move associated with the edge from y_i to y'_i, where $t' \in \mathbb{Q}_{\geq 0}$, and $w_{\mathcal{B}} = Next_{\mathcal{B}}(s_{\mathcal{B}}, m^1_{\mathcal{B}})$. By construction, $t' \leq t$ and the nodes $z'_{i,1}, \ldots, z'_{i,m_i}$ are labeled by positions $(w^1_{\mathcal{A}}, w_{\mathcal{B}}), \ldots, (w^{m_i}_{\mathcal{A}}, w_{\mathcal{B}})$, respectively, where $w^1_{\mathcal{A}}, \ldots, w^{m_i}_{\mathcal{A}}$ are the states of $\mathcal{A}$ obtained from $s_{\mathcal{A}}$ applying all and only the matching moves $(t', b, \delta'_{\mathcal{A}}) \in Mov_{\mathcal{A}}(1, s_{\mathcal{A}})$ of $m^1_{\mathcal{B}}$. By ind. hyp. for each $1 \leq l \leq m_i$, there is a 0-TAMTL$^*_P$ formula ϕ_l s.t. $(\mathcal{A}, w^l_{\mathcal{A}}) \models \phi_l$ and $(\mathcal{B}, w_{\mathcal{B}}) \not\models \phi_l$. Let φ_p be the 0-TAMTL$^*_P$ formula given by

$$\langle\langle 0 \rangle\rangle \{ \big(\bigvee_{c \in Act^{\perp}_1} \bigvee_{\kappa \in \{0,1\}} \langle (a, = t), (c, \geq 0), \kappa \rangle \big) \wedge \big(\langle (a, = t), (b, = t'), 1 \rangle \rightarrow \bigcirc(\phi_1 \vee \ldots \vee \phi_{m_i}) \big) \}$$

Evidently, $(\mathcal{A}, s_{\mathcal{A}}) \models \varphi_p$. Moreover, $(\mathcal{B}, s_{\mathcal{B}}) \not\models \varphi_p$, since for every strategy of player 0 in $\mathcal{B}$ which initially selects from $s_{\mathcal{B}}$ a move of the form (t, a, δ), there is an outcome from $s_{\mathcal{B}}$ of the form $\pi = s_{\mathcal{B}}, \langle (t, a, \delta), (t', b, \delta_{\mathcal{B}}), 1 \rangle, w_{\mathcal{B}}, \ldots$, where by hypothesis $w_{\mathcal{B}} \not\models (\phi_1 \vee \ldots \vee \phi_{m_i})$. ∎

From the proof of Theorem 13, it follows that timed alternating simulation for player σ can also be logically characterized by the small fragment of σ-TAMTL$^*_P$ which only uses the boolean connectives, the next temporal modality $\bigcirc$, and the strategy quantifier $\langle\langle \sigma \rangle\rangle$.

4 Model checking TG against TAMTL

Fix a TG $\mathcal{A}_{in}$ over (Act_0, Act_1) and a TAMTL formula φ. By [3], w.l.o.g. we can assume that the constants occurring in φ are natural numbers. Moreover, we can assume that $\mathcal{A}_{in}$ uses a clock x_{div}, which is reset whenever the constraint $x_{div} \geq 1$ holds. Let $\mathcal{A}$ be the TG obtained from $\mathcal{A}_{in}$ by simply adding a new clock, say x_φ (note that x_φ is never used by $\mathcal{A}$). Let K_{max} be the largest constant occurring in $\mathcal{A}$ and φ. We denote by $Reg_{\mathcal{A}_{in}}$ (resp., $Reg_{\mathcal{A}}$) the finite set of equivalences classes of the *region equivalence* on the set $S_{\mathcal{A}_{in}}$ (resp., $S_{\mathcal{A}}$) of states of $\mathcal{A}_{in}$ (resp., $\mathcal{A}$) w.r.t. the constant K_{max} [3], which is defined similarly to the set $Reg_{\mathcal{A}\|\mathcal{B}}$ in Subsection 3.1. We show that checking whether $\mathcal{A}_{in} \models \varphi$ (model-checking problem) can be reduced to solving finite-state games w.r.t. regular objectives. For this, we associate to $\mathcal{A}$ two finite-state games which abstract away from precise time information.

Let $R \in Reg_{\mathcal{A}}$. A region $R' \in Reg_{\mathcal{A}}$ is an abstract time-successor of R, written $R \leq R'$, if there is $(q, v) \in R$ such that for some $t \in \mathbb{R}_{\geq 0}$, $(q, v + t) \in R'$ and $v + t' \models Inv(q)$ for each $0 < t' < t$. By [3], the previous condition is independent on what representative is chosen in R. Moreover, if $R \leq R'$ and $R \leq R''$, then either $R' \leq R''$ or $R'' \leq R'$. The set of *abstract moves* available to player σ in R, written $Mov^{abs}_{\mathcal{A}}(\sigma, R)$, is the set of triples $(R', a, \delta) \in Reg_{\mathcal{A}} \times Act^{\perp}_{\sigma} \times \Delta$, such that $R \leq R'$, $\delta = (q, a, g, Y, q')$, q is the location associated with R', and g holds in R'. Given $m = (R', a, \delta) \in Mov^{abs}_{\mathcal{A}}(\sigma, R)$ with $\delta = (q, a, g, Y, q')$, we denote by $Next^{abs}_{\mathcal{A}}(R, m)$ the unique region R'' such that there is $(q, v) \in R'$ so that $(q, v[Y := 0]) \in R''$. By [3], the previous condition is independent on what representative is chosen in R'.

Let $\sigma \in \{0, 1\}$. The finite-state *turn-based* two-player game $\mathcal{A}^{abs}_{\sigma} = \langle P_\sigma = P^{\sigma}_{\sigma} \cup P^{1-\sigma}_{\sigma}, E_\sigma \rangle$ is defined as follows: $P^{\sigma}_{\sigma} = Reg_{\mathcal{A}} \times \{0, 1\}$ is the set of states (or positions) for player σ,

$P_\sigma^{1-\sigma} = \{\langle R, m, l\rangle \mid R \in Reg_{\mathcal{A}}, m \in Mov_{\mathcal{A}}^{abs}(\sigma, R), \text{and } l \in \{0,1\}\}$ is the set of positions for player $1-\sigma$, and $E_\sigma \subseteq (P_\sigma^\sigma \times P_\sigma^{1-\sigma}) \cup (P_\sigma^{1-\sigma} \times P_\sigma^\sigma)$ consists of following edges:

- $(R, l) \twoheadrightarrow (R, m, l)$ for all $R \in Reg_{\mathcal{A}}$, $m \in Mov_{\mathcal{A}}^{abs}(\sigma, R)$, and $l \in \{0,1\}$;
- $(R, \langle R_1, a, \delta_1\rangle, l) \twoheadrightarrow (R', l')$ iff $\exists \langle R_2, b, \delta_2\rangle \in Mov_{\mathcal{A}}^{abs}(1-\sigma, R)$ s.t. *either* $l' = \sigma$, $R_1 \leq R_2$, and $R' = Next_{\mathcal{A}}^{abs}(R, \langle R_1, a, \delta_1\rangle)$, *or* $l' = 1-\sigma$, $R_2 \leq R_1$, and $R' = Next_{\mathcal{A}}^{abs}(R, \langle R_2, b, \delta_2\rangle)$.

A strategy for player σ in $\mathcal{A}_\sigma^{abs}$ is a function $f : P_\sigma^* \cdot P_\sigma^\sigma \to P_\sigma^{1-\sigma}$ such that for each $\pi = \pi' \cdot p \in P_\sigma^* \cdot P_\sigma^\sigma$, $p \twoheadrightarrow f(\pi)$ is an edge of $\mathcal{A}_\sigma^{abs}$. For each $p \in P_\sigma$, the set $Outcomes_{\mathcal{A}_\sigma^{abs}}(\sigma, p, f)$ of *infinite outcomes of f from p* is defined in the usual way. For a finite set of propositions $Prop$, a labeling function $L : P_\sigma \to 2^{Prop}$, a standard LTL formula ξ over $Prop$, and position $p \in P_\sigma$, we say that the strategy f is *winning* in p w.r.t. L and the objective ξ if for each outcome $p_0, p_1, \ldots \in Outcomes_{\mathcal{A}_\sigma^{abs}}(\sigma, p, f)$, $L(p_0), L(p_1), \ldots$ satisfies ξ. The following two lemmata show the connection between the strategies of player σ in $\mathcal{A}$ and the strategies of player σ in $\mathcal{A}_\sigma^{abs}$.

LEMMA 14. *Let $\sigma \in \{0,1\}$, f be a strategy of player σ in $\mathcal{A}$, $R_0 \in Reg_{\mathcal{A}}$, and $s_0 \in R_0$. Then, there is a strategy f_{abs} of player σ in $\mathcal{A}_\sigma^{abs}$ such that for each path $\pi_{abs} = (R_0, 0), p_0, (R_1, \underline{\sigma_1}), p_1, (R_2, \underline{\sigma_2}) \ldots \in Outcomes_{\mathcal{A}_\sigma^{abs}}(\sigma, (R_0, 0), f_{abs})$, there exists a run $\pi \in Outcomes_{\mathcal{A}}(\sigma, s_0, f)$ of the form $\pi = s_0, \langle m_1^0, m_1^1, \underline{\sigma_1}\rangle, s_1, \langle m_2^0, m_2^1, \underline{\sigma_2}\rangle, \ldots$ so that for each $h \geq 1$, $s_h \in R_h$.*

LEMMA 15. *Let $\sigma \in \{0,1\}$ and f_{abs} be a strategy of player σ in $\mathcal{A}_\sigma^{abs}$, and $R_0 \in Reg_{\mathcal{A}}$. Then, there is a strategy f of player σ in $\mathcal{A}$ s.t. for each $\pi = s_0, \langle m_1^0, m_1^1, \underline{\sigma_1}\rangle, s_1, \langle m_2^0, m_2^1, \underline{\sigma_2}\rangle, \ldots \in Outcomes_{\mathcal{A}}(\sigma, s_0, f)$ with $s_0 \in R_0$, there is $\pi_{abs} \in Outcomes_{\mathcal{A}_\sigma^{abs}}(\sigma, (R_0, 0), f_{abs})$ of the form $\pi_{abs} = (R_0, 0), p_0, (R_1, \underline{\sigma_1}), p_1, (R_2, \underline{\sigma_2}) \ldots$ so that for each $h \geq 1$, $s_h \in R_h$.*

THEOREM 16. *The set of states s_{in} of $\mathcal{A}_{in}$ such that $(\mathcal{A}_{in}, s_{in}) \models \varphi$ is a union of regions in $Reg_{\mathcal{A}_{in}}$, and its (region) representation can be computed in exponential time. Hence, model checking TG against TAMTL is in* EXPTIME.

PROOF. We prove by induction on the structure of the formulas that the result holds for each state subformula ϕ of φ. Here, we illustrate the case in which $\phi = \langle\langle\sigma\rangle\rangle_{re}(\phi_1 \mathcal{U}_I \phi_2)$ for some $\sigma \in \{0,1\}$. For $s \in S_{\mathcal{A}}$, we denote by $Proj(s)$ the associated state in $S_{\mathcal{A}_{in}}$. Let $S_{\mathcal{A}}[x_\varphi := 0]$ be the set of states in $S_{\mathcal{A}}$ such that the value of clock x_φ is 0. Note that for each $s \in S_{\mathcal{A}}$, $(\mathcal{A}, s) \models \phi$ iff $(\mathcal{A}_{in}, Proj(s)) \models \phi$. By ind. hyp. it follows that for each $i = 1, 2$, the set of states $s \in S_{\mathcal{A}}$ such that $(\mathcal{A}, s) \models \phi_i$ is a union of regions in $Reg_{\mathcal{A}}$ whose representation can be computed in exponential time. Evidently, it suffices to show that the last condition continues to hold for the set of states s in $S_{\mathcal{A}}[x_\varphi := 0]$ such that $(\mathcal{A}, s) \models \langle\langle\sigma\rangle\rangle_{re}(\phi_1 \mathcal{U}_I \phi_2)$. Note that by the previous observations, for each $s_0 \in S_{\mathcal{A}}[x_\varphi := 0]$, $(\mathcal{A}, s_0) \models \langle\langle\sigma\rangle\rangle_{re}(\phi_1 \mathcal{U}_I \phi_2)$ iff there is a strategy f of player σ in $\mathcal{A}$ such that for each $\pi = s_0, \langle m_1^0, m_1^1, \sigma_1\rangle, s_1, \langle m_2^0, m_2^1, \sigma_2\rangle, \ldots \in Outcomes_{\mathcal{A}}(\sigma, s_0, f)$, the associate sequence $Reg(s_0), \sigma_1, Reg(s_1), \sigma_2, \ldots$, where $Reg(s_j)$ is the region of s_j, satisfies the following: either (1) for infinitely many $j \geq 0$, $Reg(s_j)$ satisfies the constraint $x_{div} \geq 1$, and there is $k > 0$ such that $Reg(s_k)$ satisfies ϕ_2 and the constraint $x_\varphi \in I$, and $Reg(s_h)$ satisfies ϕ_1 for each $0 < h < k$, or (2) there is $k \geq 0$ such that for each $j \geq k$, $\sigma_j \neq \sigma$ and $Reg(s_j)$ satisfies $x_{div} < 1$. Let $L : P_\sigma \to \{p_{\phi_2}, p_{\phi_1}, (x_{div} \geq 1), (x_\varphi \in I), 0, 1\}$ be the labeling of $\mathcal{A}_\sigma^{abs}$ defined in the obvious way. Then, by Lemmata 14 and 15, for all regions $R_0 \in Reg_{\mathcal{A}}$ satisfying $x_\varphi = 0$ and $s_0 \in R_0$, it holds that $(\mathcal{A}, s_0) \models \langle\langle\sigma\rangle\rangle_{re}(\phi_1 \mathcal{U}_I \phi_2)$ iff there is a winning strategy f_{abs} of player σ in $\mathcal{A}_\sigma^{abs}$ in position $(R_0, 0)$ w.r.t. the labeling L and the LTL objective: $[\Box \Diamond (x_{div} \geq 1) \wedge (p_{\phi_1} \mathcal{U} (p_{\phi_2} \wedge (x_\varphi \in I)))] \vee [\Diamond \Box (\neg (x_{div} \geq 1) \wedge (1 - \sigma))]$

Since LTL finite-state games for a fixed LTL formula can be solved in polynomial time [18] and since the size of $\mathcal{A}_\sigma^{abs}$ is exponential in the size of $\mathcal{A}_{in}$, the result follows. ∎

References

[1] L. Aceto and A. Jeffrey. A Complete Axiomatization of Timed Bisimulation for a Class of Timed Regular Behaviours. *Theoretical Computer Science*, 152(2):251–268, 1995.

[2] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[3] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[5] R. Alur, T.A. Henzinger, O. Kupferman, and M .Y. Vardi. Alternating Refinement Relations. In *Proc. 9th CONCUR*, *LNCS* 1466, pp. 163–178, 1998.

[6] P. Bulychev, T. Chatain, A. David, and K.G. Larsen. Checking simulation relation between timed game automata. In *Proc. 7th FORMATS*, *LNCS*, 2009. To appear

[7] K. Čerāns. Decidability of Bisimulation Equivalences for Parallel Timer Processes. In *Proc. 4th CAV*, *LNCS* 663, pp. 302–315, 1992.

[8] L. de Alfaro, L. Dias da Silva, M. Faella, A. Legay, P. Roy, and M. Sorea. Sociable Interfaces. In *Proc. 5th FROCOS*, *LNCS* 3717, pp. 81–105, 2005.

[9] L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, and M. Stoelinga. The Element of Surprise in Timed Games. In *Proc. 14th CONCUR*, *LNCS* 2761, pp. 142–156, 2003.

[10] L. de Alfaro and T. A. Henzinger. Interface automata. In *Proc. 9th FSE*, *ACM* , pp. 109–120, 2001.

[11] E.A. Emerson and J.Y. Halpern. Sometimes and Not Never Revisited: On Branching Versus Linear Time. *Journal of the ACM*, 33(1):151–178, 1986.

[12] T.A. Henzinger and V.S. Prabhu. Timed Alternating-Time Temporal Logic. In *Proc. 4th FORMATS*, *LNCS* 4202, pp. 1–17, 2006.

[13] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[14] F. Laroussinie and Ph. Schnoebelen. The State Explosion Problem from Trace to Bisimulation Equivalence. In *Proc. 3rd FOSSACS*, *LNCS* 1784, pp. 192–207, 2000.

[15] J. Ouaknine and J. Worrell. On Metric Temporal Logic and Faulty Turing Machines. In *Proc. 9th FOSSACS*, *LNCS* 3921, pp. 217–230, 2006.

[16] J. Ouaknine and J. Worrell. On the Decidability of Metric Temporal Logic. In *Proc. 20th LICS*, pp. 188-197, IEEE Computer Society Press, 2005.

[17] S. Tasiran, R. Alur, R .P. Kurshan, and R.K. Brayton. Verifying Abstractions of Timed Systems. In *Proc. 7th CONCUR*, *LNCS* 1119, pp. 546–562,1996.

[18] W. Thomas. Automata on Infinite Objects. *Handbook of Theoretical Computer Science: Formal Models and Sematics*, B:133–192, 1990.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Covering of ordinals

Laurent Braud
Université Paris-Est Marne-la-Vallée
laurent.braud@univ-mlv.fr

ABSTRACT. The paper focuses on the structure of fundamental sequences of ordinals smaller than ε_0. A first result is the construction of a monadic second-order formula identifying a given structure, whereas such a formula cannot exist for ordinals themselves. The structures are precisely classified in the pushdown hierarchy. Ordinals are also located in the hierarchy, and a direct presentation is given.

A recurrent question in computational model theory is the problem of model checking, i.e. the way to decide whether a given formula holds in a structure or not. When studying infinite structures, first-order logic only brings local properties whereas second-order logic is most of the time undecidable, so monadic second-order logic or one of its variants is often a balanced option. In the field of countable ordinals, results of Büchi [3] and Shelah [15] both brought decidability of the monadic theory via different ways. This positive outcome is tainted with the following property : the monadic theory of a countable ordinal only depends on a small portion of it, called the ω-tail [3, Th. 4.9]. In other words, many ordinals greater than ω^ω share the same monadic theories and cannot be distinguished.

Another class of structures enjoying a decidable monadic second-order theory is the pushdown hierarchy [6], which takes its source in the Muller and Schupp characterization of transition graphs of pushdown automata [11]. In the same way, each level of the hierarchy has two characterizations : an internal by higher-order pushdown automata [4], and an external presentation by graph transformations [5]. This paper will use the latter by the means of monadic interpretation and treegraph operations.

The original motivation of this paper was the localization of ordinals smaller than ε_0 in the hierarchy. Because of the above property, ordinals themselves are not easy to manipulate with monadic interpretations. There is therefore a need of structures as expressive as ordinals (in terms of interpretations) but having additional properties, such as the existence of a monadic formula precisely identifying the structure.

A well-known object answers to this request. Each countable limit ordinal may be defined as the limit of a so-called fundamental sequence. For ordinals smaller than ε_0, it is easy to have a unique definition for this sequence using the Cantor normal form. We note $\alpha \prec \beta$ when α is in the fundamental sequence of β or $\alpha + 1 = \beta$. When restricted to ordinals smaller than λ, we call the resulting structure the covering graph of λ. In Section 2, we present precisely this structure and give some of its properties. In particular, the out-degree of its vertices is studied intensively. This eventually yields a specific formula for each covering graph.

Section 3 locates the covering graph of any ordinal α smaller than ε_0 in the level n of the hierarchy, where n is the largest size of the ω-tower smaller than α. The result also applies to ordinals themselves. This was already shown for ordinals up to ω^{ω^ω} in [1]. In Section 4, the result in strengthened by proving that covering graphs are not in the lower levels; the

© Laurent Braud; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 97–108
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2310

question is still open for ordinals. Eventually, we produce a direct presentation for towers of ω through prefix-recognizable relations of order n, but involving a more technical proof.

Similar attempts of characterization of ordinals has been made in the field of automaticity [8, 10], but in the other way around : word- and tree-automatic ordinals are shown to be respectively less than ω^ω and ω^{ω^ω}.

1 Definitions

In this paper, ordinals are often considered from a graph theory point of view. The set of vertices of α is the set of ordinals smaller than α, and the set of arcs is the relation $<$.

1.1 Graphs

Graphs are finite or infinite sets of labeled arcs. A Σ-graph is a set $G \subseteq V \times \Sigma \times V$, where V (or V_G if unclear) is the *support*, i.e. a finite or countably infinite set of *vertices*, and Σ a finite set of *labels*. An element (p, a, q) of G is called an *arc* and noted $p \xrightarrow{a} q$. Each label $a \in \Sigma$ is associated to a relation $R_a = \{(p, q) \mid p \xrightarrow{a} q\}$ on V. A finite sequence of arcs $p \xrightarrow{a_1} \dots \xrightarrow{a_n} q$ is a *path* and noted $p \xrightarrow{a_1 \dots a_n} q$. This is extended to languages with $p \xrightarrow{L} q$ iff $\exists u \in L$ such that $p \xrightarrow{u} q$. Isomorphism between graphs is noted $\simeq$.

The *monadic second-order* (MSO) logic is defined as usual; see for instance [9]. We take a set of (lowercase) first-order variables and a set of (uppercase) second-order variables. For a given set of labels Σ, atomic formulas are $x \in X$, $x = y$ and $x \xrightarrow{a} y$ for all $a \in \Sigma$ and x, y, X variables. Formulas are then closed by the propositional connectives $\neg, \wedge$ and the quantifier $\exists$. Graphs are seen as relational structures over the signature consisting of the relations $\{R_a\}_{a \in \Sigma}$. The set of closed monadic formulas satisfied by a graph G is noted $\mathrm{MTh}(G)$.

Given a binary relation R, the *in-degree* (respectively *out-degree*) of x is the cardinality of $\{y \mid yRx\}$ (resp. $\{y \mid xRy\}$). The output degree in a graph G of $x \in V$ is the cardinal of $\{y \mid \exists a, (x, a, y) \in G\}$. The output degree of a graph is the maximal output degree of its vertices if it exists.

1.2 Ordinals

For a general introduction to ordinal theory, see [14, 13]. An order is a well-order when each non-empty subset has a smallest element. Ordinals are well-ordered by the relation $\in$, and satisfy $\forall x (x \in \alpha \Rightarrow x \subset \alpha)$. Since any well-ordered set is isomorphic to a unique ordinal, we will often consider an ordinal up to isomorphism. In terms of graphs, the set of labels of an ordinal is a singleton often noted $\Sigma = \{<\}$ and the graph respects the following monadic properties :

$$\begin{array}{ll}
\text{(strict order)} & \left\{ \begin{array}{c} \forall p, q (\neg (p \xrightarrow{<} q \wedge q \xrightarrow{<} p)) \\ \forall p, q, r ((p \xrightarrow{<} q \wedge (q \xrightarrow{<} r) \Rightarrow p \xrightarrow{<} r) \end{array} \right. \\
\text{(total order)} & \forall p, q (p \xrightarrow{<} q \vee q \xrightarrow{<} p \vee p = q) \\
\text{(well order)} & \forall X \neq \emptyset \; \exists x (x \in X \wedge \forall y (y \in X \Rightarrow (x \xrightarrow{<} y \vee x = y)))
\end{array}$$

The ordinal arithmetics define operations on ordinals such as addition, multiplication, exponentiation. The bound of ordinals investigated here is ε_0, the smallest ordinal such that $\varepsilon_0 = \omega^{\varepsilon_0}$; therefore **the declaration "$< \varepsilon_0$" is implicit through the rest of the paper**. To simplify the writing of towers of ω, the notation $\Uparrow$ is used to note the iteration of exponentiation ie. $a \Uparrow b = a^{a^{\cdots^a}} \Big\} b$ times. In particular, $a \Uparrow 0 = 1$ is the (right) exponentiation identity.

Classic operations are not commutative in ordinal theory : for instance $\omega + \omega^2 = \omega^2 < \omega^2 + \omega$. This leads to many writings for a single ordinal. Fortunately, all ordinals smaller than ε_0 may uniquely be written in the Cantor normal form (CNF)

$$\alpha = \omega^{\alpha_0} + \cdots + \omega^{\alpha_k}$$

where $\alpha_k \leq \cdots \leq \alpha_0 < \alpha$. An alternative we will call *reduced* Cantor normal form (RCNF) is $\alpha = \omega^{\alpha_0}.c_0 + \cdots + \omega^{\alpha_k}.c_k$ where $\alpha_k < \cdots < \alpha_0 < \alpha$ and $c_1, \ldots, c_k$ are non-zero integers. To express ordinals smaller than ε_0 from natural numbers and ω, the only operations needed are thus addition and exponentiation.

2 Covering graphs

In this section, we define the covering graph of an ordinal as the graph of successor and fundamental sequence relations. Then, we prove some of its important properties. One of them is the finite degree property, which is worked out to bring a specific monadic formula for each covering graph, thus allowing to differentiate them.

2.1 Fundamental sequence

The cofinality [14] of any countable ordinal is ω. To each limit ordinal α we may associate a ω-sequence whose bound is α. For $\alpha \leq \varepsilon_0$, $\alpha = \beta + \omega^\gamma$ with $\beta < \alpha$, $\gamma < \alpha$ and ω^γ is the last term in the CNF of α, we define the fundamental sequence $(\alpha[n])_{n<\omega}$ as follows :

$$\alpha[n] = \begin{cases} \beta + \omega^{\gamma'}.(n+1) & \text{if } \gamma = \gamma' + 1 \\ \beta + \omega^{\gamma[n]} & \text{otherwise.} \end{cases}$$

We define $\alpha' \prec \alpha$ whenever there is k such that $\alpha' = \alpha[k]$, or if $\alpha' + 1 = \alpha$.

For instance, the fundamental sequence of ω is the sequence of integers starting from 1. The sequence of ω^ω is therefore $(\omega, \omega^2, \omega^3, \ldots)$. The fundamental sequence merged with the successor relation yields for instance

$$0 \prec 1 \prec \omega \prec \omega + 1 \prec \omega.2 \prec \omega^2 \prec \omega^\omega.$$

Taking the transitive closure of this relation gives back the original order, so there no information loss.

Lemma 1 *The transitive closure of $\prec$ is $<$.*

Moreover, the relation is *crossing-free* as described below, which is a helpful technical tool.

Lemma 2 *If $\alpha_1 < \lambda_1 < \alpha_2$, $\alpha_1 \prec \alpha_2$ and $\lambda_1 \prec \lambda_2$, then $\lambda_2 \leq \alpha_2$.*

This is the forbidden case :

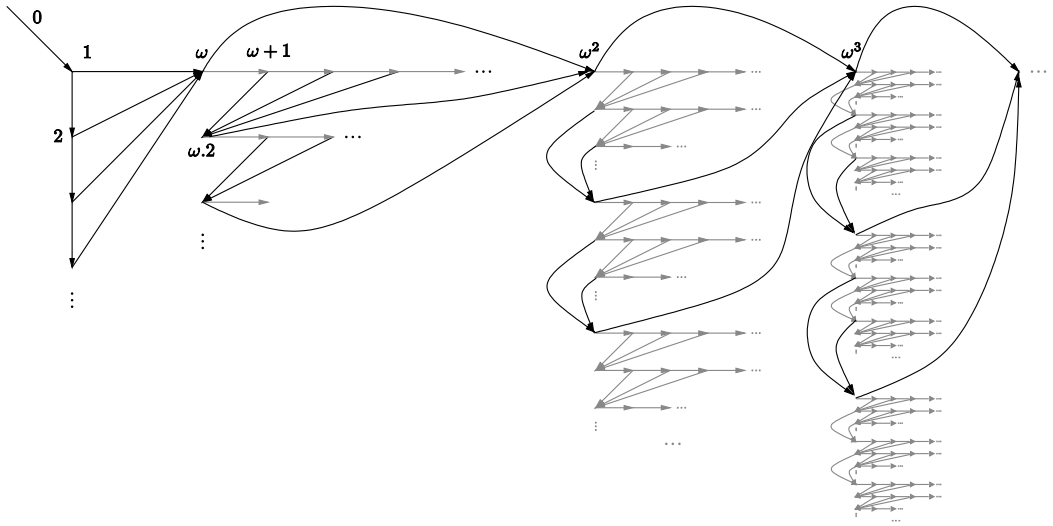


Figure 1: covering graph of ω^ω.

2.2 Covering graphs

Let $\mathcal{G}_\alpha = \{\lambda_1 \prec \lambda_2 \,|\, \lambda_1, \lambda_2 < \alpha\}$ be the graph of successor and fundamental sequence relation, or *covering graph* of the ordinal α. For instance, a representation of $\mathcal{G}_{\omega^\omega}$ is given in Figure 1.

We first remark the finite out-degree of the covering graphs.

Lemma 3 *For any $\omega \Uparrow (n-1) < \alpha \leq \omega \Uparrow n$ and $n > 0$, the out-degree of $\mathcal{G}_\alpha$ is n.*

In the following, we refine this property to get a characterisation of an ordinal by the degree of its vertices. We define the *degree word* $u(\alpha)$ of a covering graph as follows. Consider the *greatest sequence* σ of $\mathcal{G}_\alpha$ starting from 0, i.e. $\sigma_0 = 0$ and for $k \geq 0$, σ_{k+1} is the greatest such that $\sigma_k \prec \sigma_{k+1}$. The previous lemma ensures that $\{\lambda \,|\, \sigma_k \prec \lambda\}$ is finite, so σ_{k+1} exists. Such a sequence may be finite.

The degree word $u(\alpha)$ is a finite or infinite word over $[0, n]$ when $\alpha \leq \omega \Uparrow n$, and its k^{th} letter is the out-degree of σ_k in $\mathcal{G}_\alpha$.

For instance, consider $u(\omega^\omega)$. Its greatest sequence is $(0, 1, \omega, \omega^2, \omega^3, \dots)$, where all have degree 2 in $\mathcal{G}_{\omega^\omega}$ except the first; so $u(\omega^\omega) = 12^\omega$. Now consider $u(\omega^3 + \omega^2)$: the sequence is now

$$0, 1, \omega, \omega^2, \omega^3, \omega^3 + 1, \omega^3 + \omega, \omega^3 + \omega + 1, \dots$$

which loops into $(\dots, \omega^3 + \omega.k, \omega^3 + \omega.k + 1, \dots)$ so $u(\omega^3 + \omega^2) = 12221(21)^\omega$.

Lemma 4 *For any $\alpha \leq \omega \Uparrow n$, if α is successor then $u(\alpha)$ is a finite word of $[0, n]^*$; otherwise $u(\alpha)$ is an ultimately periodic word of $[1, n]^\omega$.*

PROOF (SKETCH). If α is successor, then since the greatest sequence is unbounded, the predecessor of α is in it and the word is finite. Otherwise, we prove that $\alpha[k]$ is in the greatest sequence of α for all finite k. The sequence of degrees from 0 to $\alpha[0]$ forms the static part of the ultimately periodic word, whereas the sequences of degrees between $\alpha[k]$ and $\alpha[k+1]$ are always the same. ∎

Let $<^n_{lex}$ be the lexicographic ordering on words on $[0,n]$ based on standard order. Degree words differ for each ordinal.

Lemma 5 *If $\alpha < \alpha' \leq \omega \Uparrow n$, then $u(\alpha) <^n_{lex} u(\alpha')$.*

PROOF. Consider $n > 0$, otherwise its degree word of α is the empty word. As before, note that the greatest sequence is unbounded, and that $\sigma_0 = \sigma'_0 = 0$. Thus if $0 < \alpha < \alpha'$ and σ' is the greatest sequence of $\mathcal{G}_{\alpha'}$, there is a smallest $n > 0$ such that $\sigma_n \neq \sigma'_n$, or σ_n doesn't exist whereas σ'_n does. In both cases, the output degree of σ_{n-1} is less in $\mathcal{G}_\alpha$ than in $\mathcal{G}_{\alpha'}$, so $u(\alpha) <^n_{lex} u(\alpha')$. ∎

A ultimately periodic pattern can be captured by a monadic formula. This is the goal of the the following lemma.

Lemma 6 *For each finite or infinite word u over $[0,n]$ and a given ordinal α, there is a monadic formula φ^u such that $\mathcal{G}_\alpha \models \varphi^u$ iff $u = u(\alpha)$.*

PROOF. The fact that the degree word is finite or ultimately periodic permits to use a finite number of variables. We consider the ultimately periodic case, and $u(\alpha) = uv^\omega$.

To simplify the writing, we consider the following shortcuts :

- $\tau(p,q)$ stands if q is the greatest such that $p \prec q$;
- if the output degree of p is k, then $\partial_k(p)$ is true;
- root(X,p) and end(X,p) are true when p is co-accessible (resp. accessible) from each vertex of X, with the entire path in X; root(p) looks for a root of the whole graph;
- inline(X) checks that X is a finite or infinite path;
- size$_k(X)$ stands for $|X| = k$.

All these notations stand for monadic formulas. For instance, the inline(X) property is true when there is a root in X and each vertex has output degree 1, and each except the root has input degree 1.

Now we may write the formula φ^u. For this, we need two finite sets $p_1 \dots p_{|u|} \in U$ for the static part, $q_1 \dots q_{|v|} \in V'$ for the beginning of the periodic part and an infinite set V with $V' \subseteq V$. We check that p_1 is the general root 0, and q_1 the root of V, which is an infinite path. Formulas τ and ∂_k force the degree of the uv part. For the periodic part, each $q \in V$ there must be the root of a finite path $X_q \subseteq V$ of size $|v|+1$, which end has the same degree that q. ∎

The combination of Lemmas 5 and 6 yields the following theorem.

Theorem 7 *For $\alpha \neq \alpha'$ smaller than ε_0, we have* $\mathrm{MTh}(\mathcal{G}_\alpha) \neq \mathrm{MTh}(\mathcal{G}_{\alpha'})$.

As a consequence, there is no generic monadic interpretation (see next section for definition) from an ordinal greater than ω^ω to its covering graph. Below this limit, there is an interpretation, because it is possible to distinguish successive limit ordinals.

3 The pushdown hierarchy

In this section, the pushdown hierarchy will only be defined by monadic interpretations and the treegraph operation. For other definitions, see for instance [4]. In particular, each level can be defined as the set of transition graphs (up to some closure operation) of finite-state higher-order pushdown automata of level n (*n-hopda*), hence the name.

A major property shared by this class of graphs is the decidability of their monadic theories. Since it is also the case for countable ordinals [15, 3], it is natural to examine the intersection. Here, covering graphs and ordinals are located at each level of the hierarchy.

3.1 Definitions

A *monadic interpretation* I is a finite set $\{\varphi_a(x,y)\}_{a\in\Gamma}$ of monadic formulas with two free first order variables. The interpretation of a graph $G \subseteq V \times \Sigma \times V$ by I is a graph $I(G) = \{p \xrightarrow{a} q \mid p,q \in V \wedge G \vDash \varphi_a(p,q)\} \subseteq V \times \Gamma \times V$. It is helpful to have $\Gamma = \Sigma$ to allow iteration process. The set of monadic interpretations $\mathcal{I}$ is closed by composition.

A particular case of monadic interpretation is inverse rational mapping. The alphabet $\bar{\Sigma}$ is used to read the arcs backwards : $p \xrightarrow{\bar{a}} q$ iff $q \xrightarrow{a} p$. An inverse rational mapping is an interpretation such that $\varphi_a(p,q) := p \xrightarrow{L_a} q$ where L_a is a regular language over $\Sigma \cup \bar{\Sigma}$.

For instance, the transitive closure of R_a for a label a is a monadic interpretation. By Lemma 1, there is therefore an immediate monadic interpretation from $\mathcal{G}_\alpha$ to α. An important corollary of Lemma 7 is that the reverse cannot exist, or there would be a monadic formula identifying a specific ordinal smaller than ε_0, which is contradictory to the result of Büchi [3, Th. 4.9] cited in introduction.

For a more complex illustration of a monadic interpretation, we notice that the degree word allows the restriction from a greater ordinal.

Lemma 8 *If $\alpha < \alpha'$, there is a MSO interpretation I such that $\mathcal{G}_\alpha = I(\mathcal{G}_{\alpha'})$.*

PROOF. Following the definition, we look for an interpretation $I = \{\psi_\prec\}$. We use again the fact that the degree word is unique and MSO-definable. Defining the greatest sequence of $\mathcal{G}_\alpha$ provides a MSO marking on $\mathcal{G}'_\alpha$, which bounds the set of vertices. More precisely, let $\Psi^u(p)$ be an expression similar to φ^u of the Lemma 6 but where the part $\tau(p_i,p_{i+1}) \wedge \delta_{u_i}(p_i)$ has been replaced by $\tau_{u_i}(p_i,p_{i+1})$ meaning "p_{i+1} is the u_i^{th} such that $p_i \prec p_{i+1}$"; the same goes for the q_j and for $\tau(p_{|u|},q_1) \wedge \delta_{u_{|u|}}(p_{|u|})$. Also add the condition that p is a part of the sequence : $(\bigvee_i p = p_i) \vee p \in V$. Then $\Psi^u(p)$ is a marking of the greatest sequence associated to u. For a given α, I simply adds the condition of co-accessibility to a vertex marked by $\Psi^{u(\alpha)}$.

$$\begin{aligned} \psi_\prec(p,q) &:= p \xrightarrow{\prec} q \wedge \exists r\,(\Psi^{u(\alpha)}(r) \wedge q \xrightarrow{\prec^*} r) \\ \mathcal{G}_\alpha &= \{p \xrightarrow{\prec} q \mid p \xrightarrow{\prec} q \in \mathcal{G}_{\alpha'} \wedge \exists r\,(\Psi^{u(\alpha)}(r) \wedge q \xrightarrow{\prec^*} r)\} \end{aligned}$$

∎

The *treegraph* Treegraph(G) of a graph G is the set $\{p \xrightarrow{a} q\} \subseteq V_G^* \times (\Sigma_G \cup \{\#\}) \times V_G^*$ where $(p,q) \in V_G^*$ are sequences of vertices of G, and $a \in \Sigma_G$ either if $p = wu$, $q = wv$ and $u \xrightarrow{a} v \in G$, or if $a = \#$, $p = wu$ and $q = wuu$. One can also see the treegraph as the fixpoint of the operation which, to each vertex which is not starting point of an # arc, adds this arc leading to the location of this vertex in a copy of G. The starting graph is called the *root graph*.

One way to define the *pushdown hierarchy* (see [5] for details) is as follows.

- $\mathcal{H}_0$ is the class of graphs with finite support,

- $\mathcal{H}_n = \mathcal{I} \circ Treegraph(\mathcal{H}_{n-1})$.

For instance, $\mathcal{H}_1$ is the class of *prefix-recognizable* graphs [7] and further $\mathcal{H}_n$ classes have been proved to correspond to an extension of prefix-recognizability on higher-order stacks [4].

3.2 Building covering graphs

We note $p \xrightarrow{a^\bullet} q$ for the longest possible path labeled by a, and $p \xrightarrow{S} q$ a shortcut for the successor relation, i.e.

$$\begin{aligned} p \xrightarrow{a^\bullet} q &:= p \xrightarrow{a^*} q \wedge \neg\exists r\ (q \xrightarrow{a} r) \\ p \xrightarrow{S} q &:= p \xrightarrow{\prec} q \wedge \neg\exists r (p \xrightarrow{\prec} r \wedge r \xrightarrow{\prec^*} q). \end{aligned}$$

Now let $I = \{\varphi_\prec\}$ and $M(p)$ respectively be the interpretation and marking

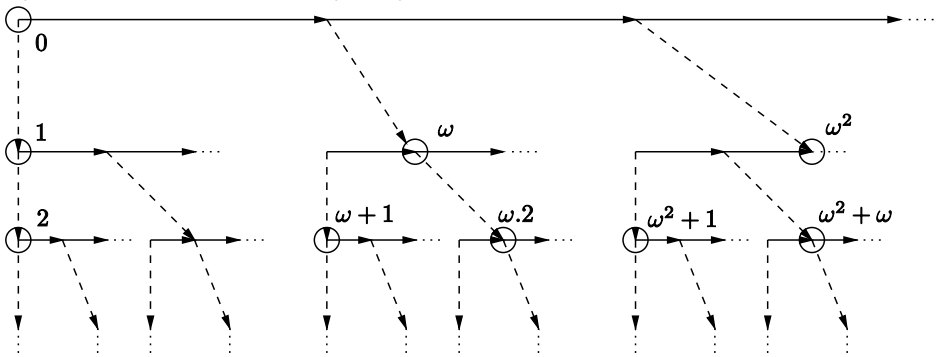
$$\begin{aligned} \varphi_\prec(p,q) &:= M(p) \wedge M(q) \wedge p \xrightarrow{\succ^\bullet \#} q \vee p \xrightarrow{\bar{\#}^\bullet S \#} q \vee p \xrightarrow{\bar{\#} \prec \#} q \\ M(p) &:= \exists r : \forall q\, (r \xrightarrow{(\prec + \# + \succ)^*} q) \wedge r \xrightarrow{\prec^* \# (\succ^* \#)^*} p \end{aligned}$$

The marking $M(p)$ allows to start anywhere on the root graph, but as soon as a #-arc has been followed, $\prec$-arcs can only be followed backwards. We consider only goals of a #-arc.

The $\varphi_\prec(p,q)$ formula states the relation on these vertices, leaving three choices : either to follow $\prec$-arcs as long as possible (in practice, until a copy of 0) and go down one #-arc; or on the contrary, to follow # backwards as long as possible, then take the successor and one #-arc; or just to follow one # backwards, one $\prec$ and one #.

Lemma 9 $\mathcal{G}_{\omega^\alpha} = I \circ \text{Treegraph}(\mathcal{G}_\alpha)$.

For instance consider $\mathcal{G}_\omega$, which is an infinite path. A representation of its treegraph is given below (plain lines for $\prec$, dotted lines for #). The circled vertices are the ones marked by M and therefore they are the only ones kept by the interpretation φ. We are allowed to go anywhere on the root $\mathcal{G}_\omega$ structure, but as soon as we follow # we can only go backwards. This reflects the construction of a power of ω as a decreasing sequence of ordinals : we may start by any, but afterwards we only may decrease.



Lemma 10 *If* $\alpha < \omega \Uparrow (n+1)$, *then* $\mathcal{G}_\alpha \in \mathcal{H}_n$.

PROOF. For any finite α, $\mathcal{G}_\alpha$ is in fact a finite path labeled by $\prec$ and is in $\mathcal{H}_0$. By Lemma 9 iterated n times, every $\omega^{\cdots^{\omega^k}}$ with n times ω and $1 < k < \omega$ is in $\mathcal{H}_n$. Smaller ordinals are captured by a restriction as in Lemma 8. ∎

This proves the decidability of the monadic theory of the covering graphs. By transitive closure (Lemma 1), ordinals are also captured.

Theorem 11 *If $\alpha < \omega \Uparrow (n+1)$, then $\alpha \in \mathcal{H}_n$.*

The decidability of the monadic theory of these ordinals is well-known, but this result also shows that ordinals below ε_0 can be expressed by finite objects, namely higher-order pushdown automata. Following the steps of a well-chosen automaton (up to an operation called the ε-closure) builds exactly an ordinal. This approach is explained in Section 5.

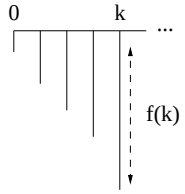
4 Strictness of the hierarchy for covering graphs

In this section, we strengthen Lemma 10 by proving that covering graphs cannot be in any level of the hierarchy. Let $\exp(x,n,k)$ be a tower of exponentiation of x of height n with power k on the top, where n and k are integers.

$$\begin{array}{rcll} \exp(x,n,k) & = & k & \text{if } n = 0 \\ & = & x^{\exp(x,n-1,k)} & \text{otherwise.} \end{array}$$

In the following section, this function will be used in the cases $x = 2$ and $x = \omega$.

We examine the tree $\mathcal{T}_n$ of trace (from the root) $\{a^n b^{\exp(2,n,k)}\}$. It has the form below with $f(k) = \exp(2,n,k)$. The horizontal arcs are labeled by a and the vertical arcs by b.



For any n, there is such a tree which is not in the level n of the hierarchy [2].

Proposition 1 *For $n \geq 1$, $\mathcal{T}_{3n} \notin \mathcal{H}_n$.*

Finding a monadic interpretation from $\mathcal{G}_\alpha$ to $\mathcal{T}_{3n}$ is therefore enough to prove $\mathcal{G}_\alpha \notin \mathcal{H}_n$. In fact, Lemma 8 already states that if $\omega \Uparrow 3n+1 \leq \alpha$, then there is an interpretation from $\mathcal{G}_\alpha$ to $\mathcal{G}_{\omega\Uparrow 3n+1}$; so the interpretation from $\mathcal{G}_{\omega\Uparrow 3n+1}$ to $\mathcal{T}_{3n}$ is enough for a whole class of ordinals. We sketch this interpretation.

Let C_n^k be the set of ordinals smaller than $\exp(\omega,n,k)$ where each coefficient in RCNF is at most 1, except for the top-most power :

- $[0,k-1] \in C_0^k$,
- $0 \in C_n^k$,
- if $\gamma_0,\ldots,\gamma_h$ are all distinct ordinals of C_{n-1}^k, then $\omega^{\gamma_0} + \cdots + \omega^{\gamma_h} \in C_n^k$.

For instance, $C_1^3 = \{0, 1, \omega, \omega+1, \omega^2, \omega^2+1, \omega^2+\omega, \omega^2+\omega+1\}$;

$$\begin{aligned} C_2^2 \;=\; & \{0, 1, \omega, \omega+1, \omega^\omega, \omega^\omega+1, \omega^\omega+\omega, \omega^\omega+\omega+1, \\ & \omega^{\omega+1}, \omega^{\omega+1}+1, \omega^{\omega+1,}+\omega, \omega^{\omega+1}+\omega+1, \\ & \omega^{\omega+1}+\omega^\omega, \omega^{\omega+1}+\omega^\omega+1, \omega^{\omega+1}+\omega^\omega+\omega, \omega^{\omega+1}+\omega^\omega+\omega+1\}. \end{aligned}$$

The following lemma is only a matter of cardinality of powersets.

Lemma 12 *The cardinality of the set C_n^k is* $\exp(2, n, k)$.

We abusively note $\alpha + C_n^k$ for the set $\{\alpha + \gamma \mid \gamma \in C_n^k\}$. The main difficulty of this section is to define a monadic formula for this set.

Lemma 13 *For $n > 0$, there is a monadic formula describing* $\exp(\omega, n, k) + C_n^k$ *in $\mathcal{G}_\alpha$, for α greater than* $\exp(\omega, n, k)$.2.

These ordinals are easy to capture by previous tools. The following lemma is a natural corollary of the proof of Lemma 4, since $\exp(\omega, n, k) \prec \exp(\omega, n, k+1)$.

Lemma 14 *The greatest sequence of $\omega \Uparrow (n+1)$ is ultimately the sequence* $(\exp(\omega, n, k))_{k \geq 1}$.

We may now state the main result of this section.

Theorem 15 *If $n > 0$ and $\alpha \geq \omega \Uparrow 3n+1$, then $\mathcal{G}_\alpha \notin \mathcal{H}_n$.*

PROOF (SKETCH). If we concatenate the previous lemmas, it appears that

- since the greatest sequence of α is interpretable from $\mathcal{G}_\alpha$, we can extract the sequence $(\exp(\omega, 3n, k))_{k \geq 1}$ from $\mathcal{G}_{\omega \Uparrow 3n+1}$, which will be the "horizontal path" of $\mathcal{T}_{3n}$;
- for each $\exp(\omega, 3n, k)$ we can also capture the associated set $\exp(\omega, 3n, k) + C_{3n}^k$ and arrange it in path. This yields the "vertical path" hanging from $\exp(\omega, 3n, k)$ and of length $\exp(2, 3n, k)$.

Eventually, the monadic interpretation builds exactly $\mathcal{T}_{3n}$, which is the expected result. ∎

The covering graph $\mathcal{G}_{\varepsilon_0}$ can be defined and has unbounded degree, but has still the property of Lemma 8 : it can give any smaller ordinal via monadic interpretation, which yields the following result.

Corollary 16 *$\mathcal{G}_{\varepsilon_0}$ does not belong to the hierarchy.*

From [2] we could actually extract the lower bound $\mathcal{T}_{2n} \notin \mathcal{H}_n$. The conjecture is that $\mathcal{T}_n \notin \mathcal{H}_n$, which would allow to locate exactly each covering graph in the hierarchy.

The Theorem 15 does not apply to ordinal themselves, since there we showed that there is no interpretation from ordinals to covering graphs. Therefore, the question is still open, which leads to Conjecture 1 at the end of this paper.

5 Higher-order stack description of ordinals

The graph on the level n of the hierarchy are also graphs (up to ε-closure) of higher-order pushdown automata of level n [5], i.e. automata which use nested stacks of stacks of depth n. The construction by monadic interpretations and unfolding could be translated into a pushdown automata description. Instead of doing so, we use the equivalent notion of prefix-recognizable relations [4] from scratch. This notion offers a natural encoding of ordinals by their Cantor normal form. Nonetheless, the associated proof is still heavy.

5.1 Short presentation

This section sketches a particular case of prefix-recognizable graphs. For a complete description, see [4]. We only consider 1-stacks (usual stacks) over an alphabet of size 1, *i.e.* integers. The empty 1-stack is therefore noted 0. For all $n > 1$, a n-stack is a non-empty finite sequence of $(n-1)$-stacks, noted $[a_1, \ldots, a_m]_n$. The operations Ops_1 on a 1-stack are

$$\begin{array}{rcl} \mathsf{push}_1(i) & := & i+1, \\ \mathsf{pop}_1(i+1) & := & i. \end{array}$$

For $n > 1$, the set Ops_n of operations on a n-stack include

$$\begin{array}{rcl} \mathsf{copy}_n([a_1, \ldots, a_m]_n) & := & [a_1, \ldots, a_m, a_m]_n \\ \mathsf{pop}_n([a_1, \ldots, a_m]_n) & := & [a_1, \ldots, a_{m-1}]_n \\ f([a_1, \ldots, a_m]_n) & := & [a_1, \ldots, f(a_m)]_n \end{array}$$

where f is any operation on k-stacks, $k < n$.

The 2-stack containing only 0 is noted $[\ ]_2$, and the n-stack containing only $[\ ]_{n-1}$ is noted $[\ ]_n$. Let also be an identity operation id defined on all stacks.

The set Ops_n forms a monoid with the composition operation. Let $Reg(Ops_n)$ the closure of the finite subsets of this monoid under union, product and iteration, *i.e.* the set of *regular expressions* on Ops_n. To each expression $E \in Reg(Ops_n)$ we associate the set of n-stacks $S(E) = E([\ ]_n)$ and the set of relations on stacks $R(E) = \{(s, s') | s' \in E(s)\}$.

Given E and a finite set $(E_a)_{a \in \Sigma}$ in $Reg(Ops_n)$, the graph of support $S(F)$ and arcs $s \xrightarrow{a} s'$ iff $(s, s') \in R(F_a)$ is a *prefix-recognizable graph of order n*. General prefix-recognizable graphs are exactly graphs of pushdown automata of the same order.

5.2 Towers of ω

We define the expressions dom and inc which respectively fix the domain of the structure and the order relation. In the following we also will need an expression dec to perform the symmetric of inc. In one word, we want the structure $\langle S(\mathsf{dom}(\alpha)), R(\mathsf{dec}(\alpha)), R(\mathsf{inc}(\alpha)) \rangle$ to be isomorphic to the structure $\langle \alpha, >, < \rangle$.

For ω, we consider the set of all 1-stacks (i.e. integers). In this case, $\mathsf{dom}(\omega)$ is obtained by iterating push_1 on the empty stack. The other operations are also straighforward.

$$\begin{array}{rcl} \mathsf{dom}(\omega) & := & \mathsf{push}_1^* \\ \mathsf{inc}(\omega) & := & \mathsf{push}_1^+ \\ \mathsf{dec}(\omega) & := & \mathsf{pop}_1^+ \end{array}$$

We consider now any ordinal α. Let n be the smallest value such that $\mathsf{dom}(\alpha), \mathsf{inc}(\alpha)$ and $\mathsf{dec}(\alpha)$ are all in $Reg(Ops_{n-1})$.

Let $\mathsf{tail}(\alpha) := \mathsf{copy}_n.(\mathsf{id} + \mathsf{dec}(\alpha))$. Informally, each ordinal $\gamma < \omega^\alpha$ is either 0 or may be written as $\gamma = \omega^{\gamma_0} + \cdots + \omega^{\gamma_k}$ with $\gamma_i < \alpha$; so we code γ as a sequence of stacks respectively coding $\gamma_0 \ldots \gamma_k$. The tail operation takes the last stack (representing γ_k) and adds a stack coding an ordinal $\leq \gamma_k$, so that the CNF constraint is respected. For the relation $<$, inc

either adds a decreasing sequence (by tail), or it first pops stacks, then increases a given one before adding a tail.

$$\begin{array}{rcl} \mathsf{dom}(\omega^\alpha) & := & \mathsf{dom}(\alpha).\mathsf{tail}(\alpha)^* \\ \mathsf{inc}(\omega^\alpha) & := & [\mathsf{pop}_n^*.\mathsf{inc}(\alpha) + \mathsf{tail}(\alpha)].\mathsf{tail}(\alpha)^* \\ \mathsf{dec}(\omega^\alpha) & := & \mathsf{pop}_n^*.[\mathsf{pop}_n + \mathsf{dec}(\alpha).\mathsf{tail}(\alpha)^*] \end{array}$$

We get this version of Theorem 11 restricted to towers of ω.

Theorem 17 *The graph of $\omega \Uparrow n$ is isomorphic to the prefix-recognizable graph of order n with support $S(\mathsf{dom}(\omega \Uparrow n))$ and one relation $R(\mathsf{inc}(\omega \Uparrow n))$.*

The proof of this proposition encodes exponentiation of ω, so the case of all ordinals smaller than ε_0 can be obtained by encoding also addition. This can be done with a greater starting alphabet and using markers to differentiate each part of the addition.

6 Perspectives

We have defined covering graphs as graphs of fundamental sequence and successor relations and shown the existence of a formula identifying a covering graph among others, via the degree word. Then, the covering graphs and the corresponding ordinals have been located in the pushdown hierarchy according to the size in terms of tower of ω, in a strict way for the covering graph case.

Theorem 11 raises the question of the strictness of the classification of ordinals in the hierarchy. Theorem 15 naturally suggests that if $\alpha \geq \omega \Uparrow n$, then α does not belong to $\mathcal{H}_{n-1}$, and therefore ε_0 is banned from the hierarchy.

Conjecture 1 *ε_0 does not belong to the hierarchy.*

If this were proved, ε_0 would actually be a good candidate for extending the hierarchy above the $\mathcal{H}_n$. Indeed, a current field of research is to capture as many structures with decidable monadic theory as possible. A way to do so would be to find an operation extending those used in this paper — interpretation and treegraph.

One can find definitions [16] of a canonical fundamental sequence for ordinals greater than ε_0 and therefore define covering graphs outside of the hierarchy. For instance, one can take $\varepsilon_0[n] = \omega \Uparrow (n+1)$. In this way, covering graphs may be defined for a large number of ordinals; but we conjecture that the Theorem 7 does not stand any more, i.e. for any definition of fundamental sequence, there are two ordinals whose covering graphs have the same monadic theories.

Also, the ability to differentiate covering graphs smaller than ε_0 leads to check this robustness for more difficult questions. One of them is selection in monadic theory, which is negative for ordinals greater than ω^ω [12].

In another direction, it would be interesting to remove the well-ordering property and to consider more general linear orderings. The orders of $\mathbb{Q}$ and $\mathbb{Z}$ are obviously prefix-recognizable. We would like to reach structures of more complex orders.

Acknowledgments

We would like to thank Didier Caucal and Arnaud Carayol for their constant help, and the anonymous referees for their useful comments.

References

[1] Stephen Bloom and Zoltán Ésik. Regular and algebraic words and ordinals. In Till Mossakowski, Ugo Montanari, and Magne Haveraaen, editors, *CALCO*, volume 4624 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007.

[2] Achim Blumensath. On the structure of graphs in the caucal hierarchy. *Theoretical Computer Science*, 400:19–45, 2008.

[3] Richard Büchi. The monadic theory of all countable ordinals. *Springer Lecture Notes in Mathematics*, 328:1x–217, 1973.

[4] Arnaud Carayol. Regular sets of higher-order pushdown stacks. In *LNCS*, volume 3618, pages 168–179, 2005.

[5] Arnaud Carayol and Stefan Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.

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

[7] Didier Caucal. On infinite transition graphs having a decidable monadic theory. *Theor. Comput. Sci.*, 290(1):79–115, 2003.

[8] Christian Delhommé. Automaticité des ordinaux et des graphes homogènes. *C. R. Acad. Sci. Paris*, Ser. I 339:5–10, 2004.

[9] Erich Grädel, Wolfgang Thomas, and Thomas Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*. Springer, 2002.

[10] Bakhadyr Khoussainov, Sasha Rubin, and Frank Stephan. Automatic linear orders and trees. *ACM Trans. Comput. Log.*, 6(4):675–700, 2005.

[11] David Muller and Paul Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37(1):51–75, 1985.

[12] Alexander Rabinovich and Amit Shomrat. Selection and uniformization problems in the monadic theory of ordinals: A survey. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 571–588. Springer, 2008.

[13] Judith Roitman. *Introduction to Modern Set Theory*. John Wiley and Sons, 1990.

[14] Joseph Rosenstein. *Linear orderings*. Academic Press Inc., 1982.

[15] Saharon Shelah. The monadic theory of order. *The Annals of Mathematics*, 102(3):379–419, 1975.

[16] Oswald Veblen. Continuous increasing functions of finite and transfinite ordinals. *Transactions of the American Mathematical Society*, 9(3):280–292, 1908.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Fractional Pebbling and Thrifty Branching Programs

Mark Braverman[1], Stephen Cook[2], Pierre McKenzie[3], Rahul Santhanam[4], Dustin Wehr[2]

[1] Microsoft Research
Cambridge, Massachussetts
mbraverm@cs.toronto.edu

[2] University of Toronto
Toronto
sacook@cs.toronto.edu, wehr@cs.toronto.edu

[3] Universite de Montreal
Montreal
mckenzie@iro.umontreal.ca

[4] University of Edinburgh
Edinburgh
rsanthan@inf.ed.ac.uk

ABSTRACT.
We study the branching program complexity of the *tree evaluation problem*, introduced in [3] as a candidate for separating **NL** from **LogCFL**. The input to the problem is a rooted, balanced d-ary tree of height h, whose internal nodes are labelled with d-ary functions on $[k] = \{1, \ldots, k\}$, and whose leaves are labelled with elements of $[k]$. Each node obtains a value in $[k]$ equal to its d-ary function applied to the values of its d children. The output is the value of the root.
Deterministic k-way branching programs as related to black pebbling algorithms have been studied in [3]. Here we introduce the notion of *fractional pebbling* of graphs to study non-deterministic branching program size. We prove that this yields non-deterministic branching programs with $\Theta(k^{h/2+1})$ states solving the Boolean problem "determine whether the root has value 1" for binary trees - this is asymptotically better than the branching program size corresponding to black-white pebbling. We prove upper and lower bounds on the fractional pebbling number of d-ary trees, as well as a general result relating the fractional pebbling number of a graph to the black-white pebbling number.
We introduce a simple semantic restriction called *thrifty* on k-way branching programs solving tree evaluation problems and show that the branching program size bound of $\Theta(k^h)$ is tight (up to a constant factor) for all $h \geq 2$ for deterministic thrifty programs. We show that the non-deterministic branching programs that correspond to fractional pebbling are thrifty as well, and that the bound of $\Theta(k^{h/2+1})$ is tight for non-deterministic thrifty programs for $h = 2, 3, 4$. We hypothesise that thrifty branching programs are optimal among k-way branching programs solving the tree evaluation problem - proving this for deterministic programs would separate **L** from **LogCFL** and proving it for non-deterministic programs would separate **NL** from **LogCFL**.

© Braverman, Cook, McKenzie, Santhanam, Wehr; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 109–120
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2311

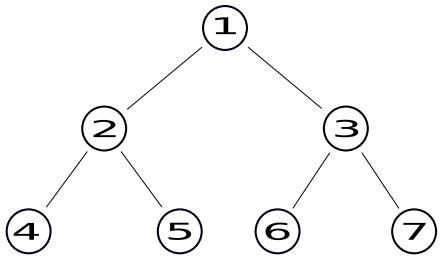


Figure 1: A height 3 binary tree T_2^3 with nodes numbered heap style.

1 Introduction

One of the fundamental problems in complexity theory is to separate **P** from **L**. In a recent paper [3], we propose the Tree evaluation problem as a candidate to separate these classes, and indeed to obtain the much tighter separation of **LogCFL** from **NL**.

The *Tree Evaluation problem* $FT_d(h,k)$ is defined as follows. The input to $FT_d(h,k)$ is a balanced d-ary tree of height h, denoted T_d^h.(see Fig. 1). Attached to each internal node i of the tree is some explicit function $f_i : [k]^d \to [k]$ specified as k^d integers in $[k] = \{1,\ldots,k\}$. Attached to each leaf is a number in $[k]$. Each internal tree node thus takes a value in $[k]$ by applying its attached function to the values of its children. The function problem $FT_d(h,k)$ is to compute the value of the root, and the Boolean problem $BT_d(h,k)$ is to determine whether this value is 1.

In [3], we show that $BT_d(h,k) \in$ **LogCFL**. To show that **LogCFL** $\not\subseteq$ **L** (resp. **LogCFL** $\not\subseteq$ **NL**), it's sufficient to get a super-polynomial lower bound on the deterministic (resp. non-deterministic) branching program complexity of the Tree evaluation problem. As we observe in [3], a lower bound of $\Omega(k^{r(h)})$ on the number of states in any deterministic (resp. non-deterministic) branching program solving $FT_d(h,k)$ or $BT_d(h,k)$ for *any* unbounded function $r(h)$ would yield the desired separation between **LogCFL** and **L** (resp. **NL**).

In this paper, we study the deterministic and non-deterministic branching program complexity of the tree evaluation problem, both from the perspective of upper bounds and lower bounds. In the context of branching programs we think of d and h as fixed, and we are interested in how the number of states required grows with k. To indicate this point of view we write the function problem $FT_d(h,k)$ as $FT_d^h(k)$ and the Boolean problem $BT_d(h,k)$ as $BT_d^h(k)$. For this it turns out that k-way branching programs are a more natural model than Boolean branching programs, since an input of $FT_d^h(k)$ or $BT_d^h(k)$ is naturally presented as a tuple of elements in $[k]$. Each non-final state in a k-way BP queries a specific element of the tuple, and branches k possible ways according to the k possible answers. Lower bounds for k-way BPs are at least as strong as lower bounds for Boolean BPs, while upper bounds can be smaller by at most a factor of k.

Our best upper bounds for k-way deterministic branching programs come from black pebbling algorithms for trees. There is a well-known generalisation of black pebbling called black-white pebbling which naturally models non-deterministic procedures. However we find we can often do better in terms of non-deterministic branching program complexity

than using black-white pebbling. For example, there is a k-way non-deterministic branching program of size $O(k^{5/2})$ which solves $BT_2^3(k)$ while the size of the branching program arising from the optimal black-white pebbling of T_2^3 is $O(k^3)$. These non-trivial upper bounds lead us to re-examine the notion of pebbling, and we come up with a more relaxed notion of pebbling called *fractional pebbling*, which corresponds to non-deterministic branching program complexity in a tighter way. For example, the tree T_2^3 can be fractionally pebbled with $5/2$ pebbles, which leads to non-deterministic branching programs of size $O(k^{5/2})$ for $BT_2^3(k)$. We show a general correspondence between fractional pebbling number and non-deterministic branching program complexity.

THEOREM 1. *If T_d^h can be fractionally pebbled with p pebbles, then non-deterministic branching programs can solve $BT_d^h(k)$ with $O(k^p)$ states*

We explore this new notion of pebbling, and prove a general result that fractional pebbling saves at most a factor of 2 over black-white pebbling. Getting tight bounds on the fractional pebbling number of trees turns out to be much more difficult than proving bounds for black-white pebbling. We do have some success though - we prove upper and lower bounds which are within $d/2 + 1$ of each other for degree d, using a non-trivial reduction to results of Klawe [11] for pyramid graphs. In addition, we get tight results for height-3 trees and the height-4 binary tree.

THEOREM 2. *The fractional pebbling number of T_d^h is at least $(d-1)h/2 - d/2$, and at most $(d-1)h/2 + 1$.*

We then turn our attention to lower bounds. In our previous paper [3], we proved tight lower bounds for the tree evaluation problem on height-3 trees. Here we try to obtain lower bounds for trees of arbitrary height, but this comes at a cost to generality in the model. We introduce a natural semantic restriction on BPs which solve the tree evaluation problem: A k-way BP is *thrifty* if it only queries the function $f(x_1, \ldots, x_d)$ associated with a node when $(x_1, \ldots, x_d)$ are the correct values of the children of the node. The deterministic BPs corresponding to black pebbling are thrifty; so are the non-deterministic BPs corresponding to fractional pebbling.

THEOREM 3. *If p is the minimum number of pebbles required to black-pebble T_2^h then every deterministic thrifty BP solving $BT_2^h(k)$ (or $FT_2^h(k)$) requires $\Omega(k^p)$ states.*

For the decision problem $BT_2^h(k)$ there is indeed a non-thrifty deterministic BP improving on the bound by a factor of $\log k$, and this is tight for $h = 3$ [3]. But we have not been able to improve on thrifty BPs for solving any function problem $FT_d^h(k)$.

We have been able to prove that thrifty non-deterministic BPs cannot beat fractional pebbling for binary trees of height $h = 4$ or less, but for general trees this is open. It is not hard to see that for black pebbling, fractional pebbles do not help. The difficulty of analysing fractional pebbling compared to black pebbling might explain why we have been able to prove tight bounds for deterministic thrifty BPs for all binary trees, but only for trees of height 4 or less for non-deterministic thrifty BPs.

We pose the following as another interesting open question:

Thrifty Hypothesis: Thrifty BPs are optimal among k-way BPs solving $FT_d^h(k)$.

Proving this for deterministic BPs would show $\mathbf{L} \neq \mathbf{LogDCFL}$, and for non-deterministic BPs would show $\mathbf{NL} \neq \mathbf{LogCFL}$. Disproving this would provide interesting new space-efficient algorithms and might point the way to new approaches for proving lower bounds.

1.1 Relation to previous work

Taitslin [16] proposed a problem similar to $BT_2^h(k)$ in which the functions attached to internal nodes are specific quasi groups, in an unsuccessful attempt to prove $\mathbf{NL} \neq \mathbf{P}$.

Gal, Koucky and McKenzie [8] proved exponential lower bounds on the size of restricted n-way branching programs solving versions of the problem GEN. Like our problems $BT_d^h(k)$ and $FT_d^h(k)$, the best known upper bounds for solving GEN come from pebbling algorithms.

As a concrete approach to separating $\mathbf{NC}^1$ from $\mathbf{NC}^2$, Karchmer, Raz and Wigderson [10] suggested proving that the circuit depth required to compose a Boolean function with itself h times grows appreciably with h. Edmonds, Impagliazzo, Rudich and Sgall [7] noted that the approach would in fact separate $\mathbf{NC}^1$ from $\mathbf{AC}^1$. They also coined the name *Iterated Multiplexor* for the most general computational problem considered in [10], namely composing in a tree-like fashion a set of explicitly presented Boolean functions, one per tree node. Our problem $FT_d^h(k)$ can be considered as a generalisation of the Iterated Multiplexor problem in which the functions map $[k]^d$ to $[k]$ instead of $\{0,1\}^d$ to $\{0,1\}$. This generalisation allows us to focus on getting lower bounds as a function of k when the tree is fixed.

1.2 Organization

The paper is organized as follows. Section 2 defines the main notions used in this paper, including branching programs and pebbling. Section 3 proves various upper and lower bounds on black, black-white and fractional pebbling. Section 4 relates branching programs and pebbling, and uses the results of Section 3 to prove upper bounds on the size of branching programs. Section 5 contains results for thrifty branching programs. Because of space constraints, proofs are omitted from this version of the paper.

2 Preliminaries

We assume some familiarity with complexity theory, such as can be found in [9]. We write $[k]$ for $\{1, 2, \ldots, k\}$. For $d, h \geq 2$ we use T_d^h to denote the balanced d-ary tree of height h.

Warning: Here the *height* of a tree is the number of levels in the tree, as opposed to the distance from root to leaf. Thus T_2^2 has just 3 nodes.

We number the nodes of T_d^h as suggested by the heap data structure. Thus the root is node 1, and in general the children of node i are (when $d = 2$) nodes $2i, 2i + 1$ (see Figure 1).

DEFINITION 4. *[Tree evaluation problems] Given: The tree T_d^h with each non-leaf node i independently labelled with a function $f_i : [k]^d \to [k]$ and each leaf node independently labelled with an element from $[k]$, where $d, h, k \geq 2$.*
Function evaluation problem $FT_d^h(k)$: Compute the value $v_1 \in [k]$ of the root 1 of T_d^h, where in general $v_i = a$ if i is a leaf labelled a and $v_i = f_i(v_{j_1}, \ldots, v_{j_d})$ if the children of i are $j_1, \ldots, j_d$.

Boolean problem $BT^h_d(k)$: *Decide whether* $v_1 = 1$.

2.1 Branching programs

We use the following definition of branching programs, inspired by Wegener [17, p. 239] and by the k-way branching program model of Borodin and Cook [2].

DEFINITION 5. *[Branching programs] A* non-deterministic k-way branching program B *computing a total function* $g : [k]^m \to R$*, where* R *is a finite set, is a directed rooted multi-graph whose nodes are called* states*. Every edge has a label from* $[k]$*. Every state has a label from* $[m]$*, except* $|R|$ final *sink states consecutively labelled with the elements from* R*. An input* $(x_1, \ldots, x_m) \in [k]^m$ *activates, for each* $1 \leq j \leq m$*, every edge labelled* x_j *out of every state labelled* j*. A* computation *on input* $\vec{x} = (x_1, \ldots, x_m) \in [k]^m$ *is a directed path consisting of edges activated by* $\vec{x}$ *which begins with the unique start state (the root), and either it is infinite, or it ends in the final state labelled* $g(x_1, \ldots, x_m)$*, or it ends in a non-final state labelled* j *with no out-edge labelled* x_j *(in which case we say the computation* aborts*). At least one such computation must end in a final state. The* size *of* B *is its number of states.* B *is* deterministic k-way *if every non-final state has precisely* k *out-edges labelled* $1, \ldots, k$*.* B *is* binary *if* $k = 2$*.*

We say that B *solves a decision problem (relation) if it computes the characteristic function of the relation.*

A k-way branching program computing the function $FT^h_d(k)$ requires k^d k-ary arguments for each internal node i of T^h_d in order to specify the function f_i, together with one k-ary argument for each leaf. Thus in the notation of Definition 4, $FT^h_d(k)$: $[k]^m \to R$ where $R = [k]$ and $m = \frac{d^{h-1}-1}{d-1} \cdot k^d + d^{h-1}$. Also $BT^h_d(k)$: $[k]^m \to \{0,1\}$.

For fixed d, h we are interested in how the number of states required for a k-way branching program to compute $FT^h_d(k)$ and $BT^h_d(k)$ grows with k. We define $\#\mathsf{detFstates}^h_d(k)$ (resp. $\#\mathsf{ndetFstates}^h_d(k)$) to be the minimum number of states required for a deterministic (resp. non-deterministic) k-way branching program to solve $FT^h_d(k)$. Similarly we define $\#\mathsf{detBstates}^h_d(k)$ and $\#\mathsf{ndetBstates}^h_d(k)$ to be the number of states for solving $BT^h_d(k)$.

The next lemma shows that the function problem is not much harder to solve than the Boolean problem.

LEMMA 6. *[3]*

$$\#\mathsf{detBstates}^h_d(k) \leq \#\mathsf{detFstates}^h_d(k) \leq k \cdot \#\mathsf{detBstates}^h_d(k)$$
$$\#\mathsf{ndetBstates}^h_d(k) \leq \#\mathsf{ndetFstates}^h_d(k) \leq k \cdot \#\mathsf{ndetBstates}^h_d(k)$$

Next we introduce thrifty programs, a restricted form of k-way branching programs for solving tree evaluation problems. Thrifty programs efficiently simulate pebbling algorithms, and implement the best known upper bounds for $\#\mathsf{ndetBstates}^h_d(k)$ and $\#\mathsf{detFstates}^h_d(k)$, and are within a factor of $\log k$ of the best known for $\#\mathsf{detBstates}^h_d(k)$. In Section 4 we prove tight lower bounds for deterministic thrifty programs which solve $BT^h_d(k)$ and $FT^h_d(k)$.

DEFINITION 7. *[Thrifty branching program] A deterministic k-way branching program which solves $FT_d^h(k)$ or $BT_d^h(k)$ is* thrifty *if during the computation on any input every query $f_i(\vec{x})$ to an internal node i of T_d^h satisfies the condition that $\vec{x}$ is the tuple of correct values for the children of node i. A non-deterministic such program is* thrifty *if for every input every computation which ends in a final state satisfies the above restriction on queries.*

Note that the restriction in the above definition is semantic, rather than syntactic. It somewhat resembles the semantic restriction used to define incremental branching programs in [8]. However we are able to prove strong lower bounds using our semantic restriction, but in [8] a syntactic restriction was needed to prove lower bounds.

2.2 Pebbling

The pebbling game for dags was defined by Paterson and Hewitt [15] and was used as an abstraction for deterministic Turing machine space in [5]. Black-white pebbling was introduced in [6] as an abstraction of non-deterministic Turing machine space (see [14] for a recent survey).

Here we define and use three versions of the pebbling game. The first is a simple 'black pebbling' game: A black pebble can be placed on any leaf node, and in general if all children of a node i have pebbles, then one of the pebbles on the children can be slid to i (this is a "black sliding move')'. Any black pebble can be removed at any time. The goal is to pebble the root, using as few pebbles as possible. The second version is 'whole' black-white pebbling as defined in [6] with the restriction that we do not allow "white sliding moves". Thus if node i has a white pebble and each child of i has a pebble (either black or white) then the white pebble can be removed. (A white sliding move would apply if one of the children had no pebble, and the white pebble on i was slid to the empty child. We do not allow this.) A white pebble can be placed on any node at any time. The goal is to start and end with no pebbles, but to have a black pebble on the root at some time.

The third is a new game called *fractional pebbling*, which generalises whole black-white pebbling by allowing the black and white pebble value of a node to be any real number between 0 and 1. However the total pebble value of each child of a node i must be 1 before the black value of i is increased or the white value of i is decreased. Figure 2 illustrates two configurations in an optimal fractional pebbling of the binary tree of height three using 2.5 pebbles.

Our motivation for choosing these definitions is that we want pebbling algorithms for trees to closely correspond to k-way branching program algorithms for the tree evaluation problem.

We start by defining fractional pebbling, and then define the other two notions as restrictions on fractional pebbling.

DEFINITION 8. *[Pebbling] A* fractional pebble configuration *on a rooted d-ary tree T is an assignment of a pair of real numbers $(b(i), w(i))$ to each node i of the tree, where*

$$0 \leq b(i), w(i) \tag{1}$$

$$b(i) + w(i) \leq 1 \tag{2}$$

Here $b(i)$ and $w(i)$ are the black pebble value *and the* white pebble value*, respectively, of i, and $b(i) + w(i)$ is the* pebble value *of i. The number of pebbles in the configuration is the sum over all nodes i of the pebble value of i. The legal pebble moves are as follows (always subject to maintaining the constraints (1), (2)): (i) For any node i, decrease $b(i)$ arbitrarily, (ii) For any node i, increase $w(i)$ so that $b(i) + w(i) = 1$, (iii) For every node i, if each child of i has pebble value 1, then decrease $w(i)$ to 0, increase $b(i)$ arbitrarily, and simultaneously decrease the black pebble values the children of i arbitrarily.*

A fractional pebbling *of T using p pebbles is any sequence of (fractional) pebbling moves on nodes of T which starts and ends with every node having pebble value 0, and at some point the root has black pebble value 1, and no configuration has more than p pebbles.*

A whole black-white pebbling *of T is a fractional pebbling of T such that $b(i)$ and $w(i)$ take values in $\{0,1\}$ for every node i and every configuration. A* black pebbling *is a black-white pebbling in which $w(i)$ is always 0.*

Notice that rule (iii) does not quite treat black and white pebbles dually, since the pebble values of the children must each be 1 before any decrease of $w(i)$ is allowed. A true dual move would allow increasing the white pebble values of the children so they all have pebble value 1 while simultaneously decreasing $w(i)$. In other words, we allow black sliding moves, but disallow white sliding moves. The reason for this (as mentioned above) is that non-deterministic branching programs can simulate the former, but not the latter.

We use $\#\mathsf{pebbles}(T)$, $\#\mathsf{BWpebbles}(T)$, and $\#\mathsf{FRpebbles}(T)$ respectively to denote the minimum number of pebbles required to black pebble T, black-white pebble T, and fractional pebble T. Bounds for these values are given in Section 3. For example for $d = 2$ we have $\#\mathsf{pebbles}(T_2^h) = h$, $\#\mathsf{BWpebbles}(T_2^h) = \lceil h/2 \rceil + 1$, and $\#\mathsf{FRpebbles}(T_2^h) \leq h/2 + 1$. In particular $\#\mathsf{FRpebbles}(T_2^3) = 2.5$ (see Figure 2).

3 Pebbling Bounds

3.1 Previous results

We start by summarizing what is known about whole black and black-white pebbling numbers as defined at the end of Definition 8 (i.e. we allow black sliding moves but not white sliding moves).

The following are minor adaptations of results and techniques that have been known since work of Loui, Meyer auf der Heide and Lengauer-Tarjan [13, 1, 12] in the late '70s. They considered pebbling games where sliding moves were either disallowed or permitted for both black and white pebble, in contrast to our results below.

We always assume $h \geq 2$ and $d \geq 2$.

THEOREM 9. *[3]* $\#\mathsf{pebbles}(T_d^h) = (d-1)h - d + 2$.

THEOREM 10. *For $d = 2$ and d odd:*

$$\#\mathsf{BWpebbles}(T_d^h) = \lceil (d-1)h/2 \rceil + 1 \tag{3}$$

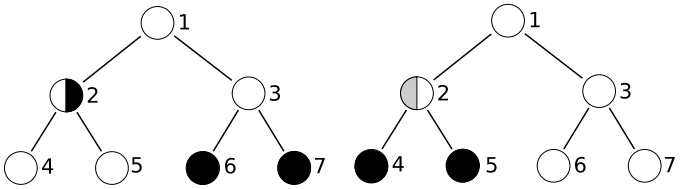


Figure 2: Two configurations from the pebbling of the height 3 binary tree with 2.5 pebbles

For d even:

$$\#\mathsf{BWpebbles}(T_d^h) \leq \lceil (d-1)h/2 \rceil + 1 \tag{4}$$

When d is odd, this number is the same as when white sliding moves are allowed.

3.2 Results for fractional pebbling

The concept of fractional pebbling is new. Determining the minimum number p of pebbles required to fractionally pebble T_d^h is important since $O(k^p)$ is the best known upper bound on the number of states required by a non-deterministic BP to solve $FT_d^h(k)$ (see Theorem 18). It turns out that proving fractional pebbling lower bounds is much more difficult than proving whole black-white pebbling lower bounds. We are able to get exact fractional pebbling numbers for the binary tree of height 4 and less, but the best general lower bound comes from a nontrivial reduction to a paper by Klawe [11] which proves bounds for the pyramid graph. This bound is within $d/2 + 1$ pebbles of optimal for degree d trees (at most 2 pebbles from optimal for binary trees).

Our proof of the exact value of $\#\mathsf{FRpebbles}(T_2^4) = 3$ led us to conjecture that any non-deterministic BP computing $BT_2^4(k)$ requires $\Omega(k^3)$ states. In Section 5 we provide evidence for that conjecture by proving that any non-deterministic *thrifty* BP requires $\Omega(k^3)$ states.

We start by presenting a general result showing that fractional pebbling can save at most a factor of two over whole black-white pebbling for any DAG (directed acyclic graph). (Here the pebbling rules for a DAG are the same as for a tree, where we require that every sink node (i.e. every 'root') must have a whole black pebble at some point.) We will not use this result, but it does provide a simple proof of weaker lower bounds than those given in Theorem 12 below.

THEOREM 11. *If a DAG D has a fractional pebbling using p pebbles, then it has a black-white pebbling using $2p$ pebbles.*

The next result presents our best-known bounds for fractionally pebbling trees T_d^h. Theorem 2 is the first part of this result.

THEOREM 12.

$$(d-1)h/2 - d/2 \leq \#\mathsf{FRpebbles}(T_d^h) \leq (d-1)h/2 + 1$$

$$\#\mathsf{FRpebbles}(T_d^3) = (3/2)d - 1/2$$

$$\#\mathsf{FRpebbles}(T_2^4) = 3$$

Theorem 12 is a consequence of the following lemmas.

LEMMA 13.

$$\#\mathsf{FRpebbles}(T_d^h) \leq (d-1)h/2 + 1$$

This lemma gives the upper bound for all degrees and heights.

LEMMA 14.

$$\#\mathsf{FRpebbles}(T_d^3) = (3/2)d - 1/2$$

This lemma gives the lower bound for height 3 and all degrees. It follows from the asymptotically tight lower bound on the number of states for non-deterministic BPs computing $BT_d^3(k)$ in [3] (Theorem 4.3 in that paper).

LEMMA 15.

$$\#\mathsf{FRpebbles}(T_2^4) \geq 3$$

This lemma gives the tight lower bound for binary height 4 trees.

LEMMA 16. *For any d and h,* $\#\mathsf{FRpebbles}(T_d^h \geq (d-1)(h-1)/2 - .5$

This lemma gives our general lower bound for all degrees and heights. We do not believe that this lower bound is tight. The proof of Lemma 16 requires the following result about optimal pebblings.

LEMMA 17. *For every finite DAG there is an optimal fractional B/W pebbling in which all pebble values are rational numbers. (This result is robust independent of various definitions of pebbling; for example with or without sliding moves, and whether or not we require the root to end up pebbled.)*

4 Pebbling and Branching Program Upper Bounds

In this section, we connect pebbling upper bounds with upper bounds for branching programs, and use the results of the previous section to derive tight bounds for branching program size of tree evaluation on trees of small height.

The first result connects pebbling upper bounds with upper bounds for thrifty branching programs. The second part is Theorem 1. Part (i) of this result without the thriftiness condition was proved in [3].

THEOREM 18. *(i) If T_d^h can be black pebbled with p pebbles, then deterministic thrifty branching programs with $O(k^p)$ states can solve $FT_d^h(k)$ and $BT_d^h(k)$. (ii) If T_d^h can be fractionally pebbled with p pebbles then non-deterministic thrifty branching programs can solve $BT_d^h(k)$ with $O(k^p)$ states.*

COROLLARY 19. $\#\mathsf{ndetFstates}_d^h(k) = O(k^{\#\mathsf{FRpebbles}(T_d^h)})$.

For every height $h \geq 2$ we prove upper bounds for deterministic *thrifty* programs which solve $FT_d^h(k)$ (Theorem 20, (5)), and show in Section 5 that these bounds are optimal for degree $d = 2$ even for the Boolean problem $BT_d^h(k)$ (Theorem 21). We prove upper bounds for non-deterministic thrifty programs solving $BT_d^h(k)$ in general, and show in Section 5 that these are optimal for binary trees of height 4 or less (Theorem 22 together with Theorem 4.3 in [3]).

For the non-deterministic case our best BP upper bounds for every $h \geq 2$ come from fractional pebbling algorithms via Theorem 18. For the deterministic case our best bounds for the function problem $FT_d^h(k)$ come from black pebbling via the same theorem, although we can improve on them for the Boolean problem $BT_2^h(k)$ by a factor of $\log k$ (for $h \geq 3$) [3].

THEOREM 20. *[BP Upper Bounds] For all $h, d \geq 2$*

$$\#\mathsf{detFstates}_d^h(k) = O(k^{(d-1)h-d+2}) \quad (5)$$

$$\#\mathsf{ndetBstates}_d^h(k) = O(k^{(d-1)(h/2)+1}) \quad (6)$$

These bounds are realized by thrifty programs.

5 Thrifty lower bounds

See Definition 7 for thrifty programs.

Theorem 21 below shows that the upper bound given in Theorem 20 (5) is optimal for deterministic thrifty programs solving the function problem $FT_d^h(k)$ for $d = 2$ and all $h \geq 2$. Theorem 22 shows that the upper bound given in Theorem 20 (6) is optimal for non-deterministic thrifty programs solving the Boolean problem $BT_d^h(k)$ for $d = 2$ and $h = 4$. Theorem 21 below is a re-statement of Theorem 3.

THEOREM 21. *For all $h \geq 2$ every deterministic thrifty program that solves $BT_2^h(k)$ has at least $0.5k^h$ states for sufficiently large k.*

Next we prove a lower bound on non-deterministic thrifty branching programs.

THEOREM 22. *Every non-deterministic thrifty branching program solving $BT_2^4(k)$ has $\Omega(k^3)$ states.*

6 Conclusion

The Thrifty Hypothesis states that thrifty branching programs are optimal among k-way BPs solving $FT_d^h(k)$. For the deterministic case, this says that the black pebbling method is optimal. Proving this would separate **L** from **P** .

Even disproving this would be interesting, since it would show that one can improve upon this obvious application of pebbling. One of the referees pointed out that if the functions at nodes are restricted to be integer polynomials, then for some parameter settings it is possible to obtain non-trivial branching programs that are not thrifty, by using Chinese remaindering.

Other accessible open problems are to generalise Theorem 22 to get general lower bounds for non-deterministic thrifty BPs solving BT_2^h, and to improve Theorem 12 to get tight bounds on the number of pebbles required to fractionally pebble T_d^h.

For a complete and combined treatment of the notions and results in this paper and in [3], please see [4].

Acknowledgement James Cook played a helpful role in the early parts of this research.

References

[1] F. Meyer auf der Heide. A comparison between two variations of a pebble game on graphs. Master's thesis, Universität Bielefeld, Fakultät für Mathematik, 1979.

[2] A. Borodin and S. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.*, 11(2):287–297, 1982.

[3] M. Braverman, S. Cook, P. McKenzie, R. Santhanam, and D. Wehr. Branching programs for tree evaluation. In *Proceedings of 34th International Symposium on Mathematical Foundations of Computer Science*, 2009.

[4] M. Braverman, S. Cook, P. McKenzie, R. Santhanam, and D. Wehr. Pebbles and branching programs for tree evaluation. On Steve's webpage, 2009.

[5] S. Cook. An observation on time-storage trade off. *J. Comput. Syst. Sci.*, 9(3):308–316, 1974.

[6] S. Cook and R. Sethi. Storage requirements for deterministic polynomial time recognizable languages. *J. Comput. Syst. Sci.*, 13(1):25–37, 1976.

[7] J. Edmonds, R. Impagliazzo, S. Rudich, and J. Sgall. Communication complexity towards lower bounds on circuit depth. *Computational Complexity*, 10(3):210–246, 2001. An abstract appeared in the *32nd IEEE FOCS (1991)*.

[8] A. Gál, M. Koucký, and P. McKenzie. Incremental branching programs. *Theory Comput. Syst.*, 43(2):159–184, 2008.

[9] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[10] M. Karchmer, R. Raz, and A. Wigderson. Super-logarithmic depth lower bounds via direct sum in communication complexity. *Computational Complexity*, 5:191–204, 1995. An abstract appeared in the *6th Structure in Complexity Theory Conference (1991)*.

[11] M. Klawe. A tight bound for black and white pebbles on the pyramid. *J. ACM*, 32(1):218–228, 1985.

[12] T. Lengauer and R. Tarjan. The space complexity of pebble games on trees. *Inf. Process. Lett.*, 10(4/5):184–188, 1980.

[13] M.C. Loui. The space complexity of two pebble games on trees. Technical Report LCS 133, MIT, Cambridge, Massachussetts, 1979.

[14] J. Nordström. New wine into old wineskins: A survey of some pebbling classics with supplemental results. Available on line at `http://people.csail.mit.edu/jakobn/research/`, 2009.

[15] M. Paterson and C. Hewitt. Comparative schematology. In *Record of Project MAC Conference on Concurrent Systems and Parallel Computations*, pages 119–128, 1970. (June 1970) ACM. New Jersey.

[16] M.A. Taitslin. An example of a problem from PTIME and not in NLogSpace. In *Proceedings of Tver State University*, volume 6(12) of *Applied Mathematics, issue 2, Tver State University, Tver*, pages 5–22, 2005.

[17] I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM Monographs on Discrete Mathematics and Applications. Soc. for Industrial and Applied Mathematics, Philadelphia, 2000.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

The Wadge Hierarchy of Max-Regular Languages

Jérémie Cabessa[1], **Jacques Duparc**[2], **Alessandro Facchini**[2,3], **Filip Murlak**[4]

[1] Grenoble Institute of Neuroscience, Joseph Fourier University, France

[2] Faculty of Business and Economics, University of Lausanne, Switzerland

[3] LaBRI, University of Bordeaux 1, France

[4] University of Warsaw, Poland

ABSTRACT. Recently, Mikołaj Bojańczyk introduced a class of max-regular languages, an extension of regular languages of infinite words preserving many of its usual properties. This new class can be seen as a different way of generalising the notion of regularity from finite to infinite words. This paper compares regular and max-regular languages in terms of topological complexity. It is proved that up to Wadge equivalence the classes coincide. Moreover, when restricted to $\mathbf{\Delta}_2^0$-languages, the classes contain virtually the same languages. On the other hand, separating examples of arbitrary complexity exceeding $\mathbf{\Delta}_2^0$ are constructed.

Introduction

Until recently, the notion of regularity for languages of infinite words developed by Büchi [2] seemed to be universally accepted. Büchi's class has various characterisations, most notably in terms of automata and monadic second order logic, and enjoys a multitude of elegant properties, like closure by Boolean operations (including negation). Nowadays however some doubt has been cast by Mikołaj Bojańczyk [1], who presented a richer class of *max-regular languages*, arguably as much regular as Büchi's languages. This new class has a characterisation via weak monadic second-order logic with the unbounding quantifier, and a suitable automaton model with decidable emptiness. It also exhibits the usual closure properties.

In this paper we would like to shed some more light on the relations between the two classes. A typical max-regular language is defined by the property "the distance between consecutive b's is unbounded",

$$K = \{a^{n_1}ba^{n_2}ba^{n_3}\ldots : \forall m\, \exists i\, n_i > m\}\,.$$

This language is not regular, but it is $\mathbf{\Pi}_2^0$-complete. In fact, as Bojańczyk notes, all max-regular languages are Boolean combinations of $\mathbf{\Sigma}_2^0$-sets, just like regular languages. Is this a coincidence, or does the similarity go further? How big is the new class? The ultimate tool for this kind of questions is the Wadge hierarchy [13, 14]. Ordering the sets based on the existence of continuous reductions (Wadge reductions) between them, the Wadge

© J. Cabessa, J. Duparc, A. Facchini, F. Murlak; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 121–132
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2312

hierarchy is the most refined complexity measure in descriptive set theory. For classical regular languages, it coincides exactly with automata-based Wagner hierarchy, and is well-understood [15]. Here we investigate the Wadge hierarchy of max-regular languages.

As was shown by Finkel's work on blind counter automata [10], adding very restricted counters already makes the Wadge hierarchy much richer. Surprisingly, even though max-automata do involve counters, the Wadge hierarchy they induce actually coincides with the Wagner hierarchy. In other words, for each max-regular language, there exists a Wadge-equivalent regular language. Topologically, Bojańczyk's extension is very conservative.

On the other hand, there is an abundance of separating languages: we provide one for each level beginning from ω. This shows that the difference between the two classes spans orthogonally to the topological complexity.

Below the level ω, which corresponds exactly to the languages complete for $\mathbf{\Pi}_2^0$ or $\mathbf{\Sigma}_2^0$, the levels contain the same languages. Hence, the exemplary language K is as simple as possible: every max-regular language strictly lower than K in the Wadge hierarchy is necessarily regular.

1 Preliminaries

1.1 Languages

A set of finite words is called a *language*, and a set of infinite words an *ω-language*. Given a finite set A, called the *alphabet*, then A^*, A^+, A^ω, and A^∞ denote respectively the sets of finite words, nonempty finite words, infinite words, and finite or infinite words, all of them over the alphabet A. The empty word is denoted by ε. Given a finite word u and a finite or infinite word v, we write uv to denote the concatenation of u and v. Given $X \subseteq A^*$ and $Y \subseteq A^\infty$, the concatenation of X and Y is defined by $XY = \{xy : x \in X \text{ and } y \in Y\}$, the finite iteration of X is $X^* = \{x_1 \cdots x_n : n \geq 0 \text{ and } x_1, \ldots, x_n \in X\}$, and the infinite iteration of X is $X^\omega = \{x_0 x_1 x_2 \cdots : x_i \in X, \text{ for all } i \in \mathbb{N}\}$. Given $u \in A^*$ and $X \subseteq A^\omega$, the set $u^{-1}X$ is defined as $u^{-1}X = \{x \in A^\omega : ux \in X\}$, and X_u is $u(u^{-1}X) = uA^\omega \cap X$.

The *ω-regular languages* are exactly the ones recognised by finite Büchi, or equivalently, by finite Muller automata. We refer to [11, p.15] for further details.

Finally, for any alphabet A, the set A^ω can be equipped with the product topology of the discrete topology on A. The open sets of A^ω are thus of the form WA^ω, for some $W \subseteq A^*$.

1.2 The Wadge hierarchy

The Wadge hierarchy is a very refined topological classification of ω-languages. This classification is obtained by means of Wadge (or continuous) reduction, which is a partial ordering defined via the Wadge games [13] presented below.

Let A and B be two finite alphabets, and let $X \subseteq A^\omega$ and $Y \subseteq B^\omega$. The *Wadge game* $\mathbb{W}((A, X), (B, Y))$ is a two-player infinite game with perfect information, where player I is in charge of the subset X and player II is in charge of the subset Y. Players I and II alternately play letters from the alphabets A and B, respectively. Player I begins. Player II is allowed to skip her turn, formally denoted by the symbol "$-$", provided she plays infinitely many letters, whereas player I is not allowed to do so. After ω turns, players I

and II have produced two infinite words, $\alpha \in A^\omega$ and $\beta \in B^\omega$ respectively. Player II wins $\mathbb{W}((A,X),(B,Y))$ if and only if $(\alpha \in X \Leftrightarrow \beta \in Y)$. From this point onward, the Wadge game $\mathbb{W}((A,X),(B,Y))$ will be denoted $\mathbb{W}(X,Y)$ and the alphabets involved will always be clear from the context. Along the play, the finite sequence of all previous moves of a given player is called the *current position* of this player. A *strategy* for player I is a mapping from $(B \cup \{-\})^*$ into A. A *strategy* for player II is a mapping from A^+ into $B \cup \{-\}$. A strategy is *winning* if the player following it must necessarily win, no matter what his opponent plays.

The *Wadge reduction* is defined via the Wadge game as follows: a set X is said to be *Wadge reducible* to Y, denoted by $X \leq_W Y$, if and only if player II has a winning strategy in $\mathbb{W}(X,Y)$. This relation $\leq_W$ is reflexive and transitive. The corresponding equivalence relation and strict reduction are defined by $X \equiv_W Y$ if and only if both $X \leq_W Y$ and $Y \leq_W X$ hold, and $X <_W Y$ if and only if $X \leq_W Y$ and $X \not\equiv_W Y$. In addition, the sets X and Y are said to be Wadge incomparable, denoted as $X \perp_W Y$, if and only if both $X \not\leq_W Y$ and $Y \not\leq_W X$. Besides, a set $X \subseteq A^\omega$ is called *self-dual* if $X \equiv_W X^c$, and *non-self-dual* if $X \not\equiv_W X^c$.

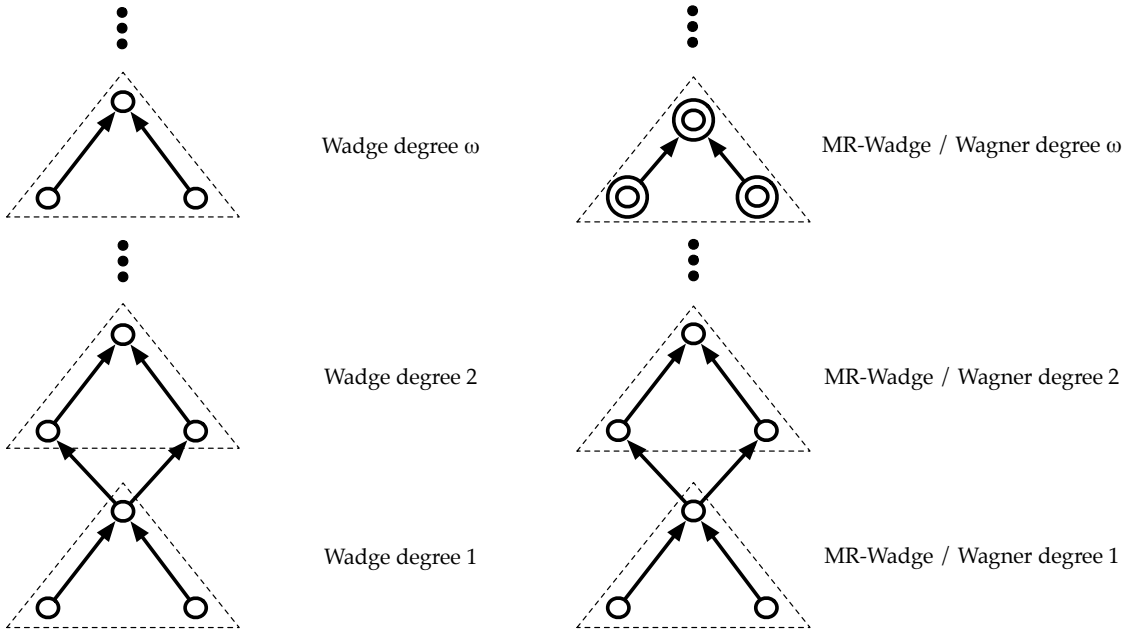
Let us point out that Wadge games were designed so that the Wadge reduction correspond precisely to the continuous reduction. Indeed, it holds that $X \leq_W Y$ if and only if there exists a continuous function $f : A^\omega \to B^\omega$ such that $f^{-1}(Y) = X$ [13].

The Wadge hierarchy consists of the collection of all ω-languages ordered by the Wadge reduction, and *the Borel Wadge hierarchy* is the restriction of the Wadge hierarchy to Borel ω-languages. As a consequence of Martin's Borel determinacy theorem, for any two Borel ω-languages X and Y, there exists a winning strategy for one of the players in $\mathbb{W}(X,Y)$. This key property induces the following strong consequences on the Borel Wadge hierarchy. First, the $\leq_W$-antichains have length at most 2, and the only incomparable ω-languages are, up to Wadge equivalence, of the form X and X^c, for X non-self-dual. Furthermore, the Wadge reduction is well-founded on Borel sets, meaning that there is no infinite strictly descending sequence of Borel ω-languages $X_0 >_W X_1 >_W X_2 >_W \dots$. These results ensure that, up to complementation and Wadge equivalence, the Borel Wadge hierarchy is actually a well ordering.

Therefore, there exist a unique ordinal, called the *height* of the Borel Wadge hierarchy, and a mapping d_W from the Borel Wadge hierarchy onto its height, called the *Wadge degree*, such that $d_W(X) < d_W(Y)$ if and only if $X <_W Y$, and $d_W(X) = d_W(Y)$ if and only if either $X \equiv_W Y$ or $X \equiv_W Y^c$, for every Borel ω-languages X and Y. Actually, it is usually convenient to consider another definition of the Wadge degree which makes the non-self dual sets and the first self dual ones that strictly reduce these latter always share the same degree, namely:

$$d_W(X) = \begin{cases} 1 & \text{if } X = \emptyset \text{ or } X = \emptyset^c, \\ \sup\{d_W(Y) + 1 : Y \text{ n.s.d. and } Y <_W X\} & \text{if } X \text{ is non-self-dual,} \\ \sup\{d_W(Y) : Y \text{ n.s.d. and } Y <_W X\} & \text{if } X \text{ is self-dual.} \end{cases}$$

Furthermore, it can be proved that the Borel Wadge hierarchy actually consists of an alternating succession of non-self-dual and self-dual sets with non-self-dual pairs at each limit level (provided finite alphabets are considered) [7, 13, 14]. Therefore, for any ordinal α below the height of the Borel Wadge hierarchy, there exist exactly three Wadge classes of degree α, namely two non-self-dual and one self-dual located precisely just one level above, as illustrated in Figure 1(a).



(a) The Wadge hierarchy: circles represent Wadge-equivalence classes and arrows stand for the strict Wadge reduction between those. The non-self dual sets and the self dual ones located just one level above share the same Wadge degree.

(b) The MR-Wadge and the Wagner hierarchy. On finite levels the classes coincide; above, MR-Wadge classes properly extend corresponding Wagner classes.

Figure 1: The hierarchies

The three Wadge classes are very closely related. In fact, any set $X \subseteq A^\omega$ that is complete for some Wadge class of degree α gives rise to two other sets $Y, Z \subseteq A^\omega$ that are respectively complete for the two remaining Wadge classes of same degree α. More precisely, if one starts with X self-dual such that $d_W(X) = \alpha$, then we know that there exists $u \in A^*$ such that $Y = u^{-1}X$ is non-self-dual and $d_W(Y) = \alpha$. It directly follows that $Z = (u^{-1}X)^c$ is also non-self-dual and $d_W(Z) = \alpha$. On the other hand, if one starts with X non-self-dual and $d_W(X) = \alpha$, then $Y = X^c$ is also non-self-dual, Wadge incomparable with X, and $d_W(Y) = \alpha$. Moreover, for any $a \in A$, the set $Z = aX \cup (A \setminus \{a\})X^c$ is self-dual with $d_W(Z) = \alpha$. All these results are folklore and can be found for instance in [7]. In the sequel we will also use the fact that the constructions above preserve regularity and max-regularity.

In this paper we are working only with the sets from $\mathrm{BC}(\mathbf{\Sigma}^0_2)$, the class of Boolean combinations of $\mathbf{\Sigma}^0_2$ sets, but in fact we need to go quite deep into the structure of the Wadge hierarchy in order to obtain the promised results. The proofs of all the facts we state below can be found in [7].

Let us start with the relation between the Borel classes and the Wadge degrees. The nth level of the Borel hierarchy corresponds to the Wadge degree "a tower of ω_1's of the height $n-1$". In particular, a language complete for $\mathbf{\Sigma}^0_2$ or $\mathbf{\Pi}^0_2$ has degree ω_1. This already shows how drastically the Borel Wadge hierarchy refines the Borel hierarchy! When we move to combinations of $\mathbf{\Sigma}^0_2$ sets, we get exactly the Wadge degrees strictly below ω_1^ω.

Important milestones on the way from ω_1 to ω_1^ω are the so-called *initialisable* sets. They are defined as those sets X, for which player II has a winning strategy in the II-imposed Wadge game $\mathbb{W}(X, X)$ where player I is allowed at any moment, but only once, to erase everything he has played before and start anew.

Let us remark that initialisable sets generalise prefix-independent sets, i.e., sets satisfying condition $u^{-1}X = X$ for all finite words u. Indeed, the winning strategy for player II in the corresponding game amounts to copying the letters played by player I, even after player I decides to erase everything and start again: the part of player II's word played before player I erased his word will not influence the outcome. Roughly speaking, initialisability is prefix-independence up to Wadge-equivalence.

Initialisable sets within $\mathrm{BC}(\mathbf{\Sigma}_2^0)$ are exactly those with Wadge degrees ω_1^n for some natural number n. Clearly, the empty set and the whole space are prefix-independent, and so initialisable. So is the well-known $\mathbf{\Pi}_2^0$-complete set $(1^*2)^\omega$. In fact, the parity languages with $n+1$ ranks correspond exactly to the degree ω_1^n. Showing that no other degree below ω_1^ω is initialisable requires a lot of technical effort. We refer the reader to [7] for the proof.

Let us finish this quick peek into the internal structure of $\mathrm{BC}(\mathbf{\Sigma}_2^0)$ with a fact that shows how simpler sets are hidden inside more complex ones. As already stated, $\mathrm{BC}(\mathbf{\Sigma}_2^0)$ sets have degrees strictly below ω_1^ω. Hence, if $X \subseteq A^*$ is $\mathrm{BC}(\mathbf{\Sigma}_2^0)$, its Wadge degree can be written in the Cantor normal form of base ω_1 as $d_W(X) = \omega_1^{n_k} \cdot p_k + \cdots + \omega_1^{n_0} \cdot p_0$, for some $k > 0$, some $\omega > n_k > \ldots > n_0 \geq 0$, and some $0 < p_i < \omega_1$ for all $0 \leq i \leq k$. Assume that one of the coefficients, say p_j, is not finite, i.e., $p_j \geq \omega$. Then for each $m > 0$ there exists a word $u \in A^*$ such that $d_W(X_u) = \omega_1^{n_k} \cdot p_k + \cdots + \omega_1^{n_j} \cdot m$. This fact is a special case of a more general result [8, Lemmas 33 and 39]. The following lemma follows easily.

LEMMA 1. *Let $X \subseteq A^*$ be a $\mathrm{BC}(\mathbf{\Sigma}_2^0)$ set such that the family $\{X_u : u \in A^*\}$ is finite up to Wadge equivalence. Then*

$$d_W(X) = \omega_1^{n_k} \cdot p_k + \cdots + \omega_1^{n_0} \cdot p_0\,,$$

for some $k > 0$, some $\omega > n_k > \ldots > n_0 \geq 0$, and some $0 < p_i < \omega$ for all $0 \leq i \leq k$.

1.3 The Wagner hierarchy

In 1979, Klaus Wagner described a classification of ω-regular sets in terms of the graph-theoretical structure automata known as the *the Wagner hierarchy* [15]. This hierarchy is a decidable pre-well-ordering of width 2 and height ω^ω. The Wagner degree of any given ω-regular language can be effectively computed by analysing the graph of a Muller automaton accepting this language [16].

In 1986, Simonnet proved that the Wagner hierarchy corresponds precisely to the restriction of the Wadge hierarchy to ω-regular languages. In our further explanations the following notion will be convenient. We say that a Wadge class is *inhabited* by a language if the language is complete for the Wadge class. In these terms, ω-regular languages inhabit exactly all Wadge classes with Wadge degrees of the form $\omega_1^{n_k} \cdot p_k + \cdots + \omega_1^{n_0} \cdot p_0$, where $\omega > n_k > \ldots > n_0 \geq 0$ and $0 < p_i < \omega$ for all $0 \leq i \leq k$. In addition, it can be shown that the Wagner reduction, which already coincides with the Wadge reduction, can also be

defined in terms of automata [11, Thm. 5.2, p. 209]. Similarly to the Wadge degree, the Wagner degree of an ω-regular language L can thus be defined as follows:

$$d_{\omega R}(L) = \begin{cases} 1 & \text{if } L = \emptyset \text{ or } L = \emptyset^c, \\ \sup\{d_{\omega R}(K) + 1 : K \text{ n.s.d. and } K <_W L\} & \text{if } L \text{ is non-self-dual}, \\ \sup\{d_{\omega R}(K) : K \text{ n.s.d. and } K <_W L\} & \text{if } L \text{ is self-dual}. \end{cases}$$

In consequence, the Wagner and the Wadge degrees of ω-regular languages are related as follows: for any ω-regular language L, if

$$d_{\omega R}(L) = \omega^{n_k} \cdot p_k + \cdots + \omega^{n_0} \cdot p_0\,,$$

for some $\omega > n_k > \ldots > n_0 \geq 0$ and $0 < p_i < \omega$ for all $0 \leq i \leq k$, then

$$d_W(L) = \omega_1^{n_k} \cdot p_k + \cdots + \omega_1^{n_0} \cdot p_0\,.$$

The Wagner hierarchy has been extensively investigated. Its complete set theoretical description in terms of Boolean expressions was given by Selivanov [12], and its algebraic counterpart was studied by various authors [3, 4, 5, 6, 9].

2 Max-regular languages

In [1], Bojańczyk introduces a new class of languages of infinite words called *max-regular languages*. This class is a proper extension of the class of ω-regular languages. It has two equivalent descriptions, one in terms of automata (max-automata), and the other in terms of logic (weak MSO with the unbounding quantifier). Here, we briefly recall the automata-theoretic one.

DEFINITION 2. *A max-automaton is a tuple $\mathcal{A} = (Q, A, \Gamma, q_0, E, \mathcal{T})$, where Q is a finite set of states, A a finite input alphabet, Γ a finite set of counters, q_0 an initial state, $\mathcal{T} \subseteq \mathcal{P}(\Gamma)$ is a specified collection of subsets of Γ, and $E \subseteq Q \times A \times Q \times (\bigcup_{c,c' \in \Gamma}\{inc_c, res_c, out_c, max_{c,c'}\})^*$ is a finite set of transitions, which, given a current state q and input letter a specifies a changing state and a sequence of counter operations. The operations inc_c, res_c, out_c, and $max_{c,c'}$ respectively mean set $c := c + 1$, set $c := 0$, output the current value of c, and set $c := \max(c, c')$.*

As usual, a deterministic max-automaton is defined by requiring the transition set E to be the graph of a partial function from $Q \times A$ into $Q \times (\bigcup_{c,c' \in \Gamma}\{inc_c, res_c, out_c, max_{c,c'}\})^*$.

For any counter $c \in \Gamma$ and any finite sequence of counter operations $o_0, \ldots, o_i$, the value of counter c after the successive performing of these operations will be denoted by $c(o_1 \cdots o_i)$.

A run of $\mathcal{A}$ is a sequence of consecutive transitions. Given an infinite run ρ, the infinite output sequence of counter c during ρ is denoted by ρ_c. An infinite word x is *accepted* by $\mathcal{A}$ if it admits a run ρ such that $\{c \in \Gamma : \rho_c \text{ is unbounded}\} \in \mathcal{T}$. In other words, the accepting conditions of max-automata are Boolean combinations of clauses of the form "the sequence ρ_c is bounded".

The set of infinite words accepted by $\mathcal{A}$ is the *language recognised by* $\mathcal{A}$ and is denoted by $L(\mathcal{A})$. An ω-language is called *max-regular* if it is recognised by a *deterministic* max-automaton.

Note that, as for Muller automata, up to adding a sink state together with the appropriate transitions and counter operations, we may assume without loss of generality that every deterministic max-automaton is complete. Hence, for any finite or infinite word, there exists exactly one corresponding finite or infinite run labelled by this word. From this point onwards, every max-automaton will be assumed to be deterministic and complete.

The following fact is taken from [1]. We sketch the proof for the sake of completeness.

LEMMA 3. *The class of max-languages is a proper extension of the class of ω-regular languages.*

PROOF. The language $K = \{a^{n_1}ba^{n_2}ba^{n_3}\ldots : \forall m\, \exists i\, n_i > m\}$ mentioned in the introduction separates the classes. Let us concentrate on showing that every ω-regular language is max-regular.

Let L be an ω-regular language, and let $\mathcal{A} = (Q, A, q_0, \delta, \mathcal{T})$ be a deterministic Muller automaton recognising it. We build a deterministic max-automaton $\mathcal{A}'$ recognising this same language. The automaton $\mathcal{A}' = (Q', A, \Gamma, q'_0, \delta', \mathcal{T}')$ is obtained by associating a counter c_q with each state q of $\mathcal{A}$ and by simulating the visit of each state of $\mathcal{A}$ by incrementing and outputting the corresponding counter of $\mathcal{A}'$. More precisely, we set $Q' = Q$, $\Gamma = \{c_q : q \in Q\}$, $q'_0 = q_0$, $\delta' = \{(q, a, q', (inc_{c'_q}, out_{c'_q})) : (q, a, q') \in \delta\}$, and $\mathcal{T}' = \{\{c_{q_1}, \ldots, c_{q_n}\} : \{q_1, \ldots, q_n\} \in \mathcal{T}\}$. In this way, a state of $\mathcal{A}$ is visited infinitely often iff the output sequence of its corresponding counter in $\mathcal{A}'$ is unbounded. The definition of $\mathcal{T}'$ then ensures that $\mathcal{A}$ and $\mathcal{A}'$ recognise the same ω-language. □

We now prove that if two infinite words induce converging runs, they are either both accepted or both rejected. This technical result will be very useful in the sequel. For finite words u and v we write $u \sim_{\mathcal{A}} v$ iff $\mathcal{A}$'s runs on u and v end in the same state.

LEMMA 4. *Let $\mathcal{A}$ be a deterministic max-automaton, and let u and v such that $u \sim_{\mathcal{A}} v$. Then $u^{-1}L(\mathcal{A}) = v^{-1}L(\mathcal{A})$.*

PROOF. Let A be the input alphabet of the automaton $\mathcal{A}$, and let $x = x_0x_1x_2\cdots$ be some infinite word of A^ω. Let also $\rho = \rho_0\rho_1\rho_2\cdots$ and $\rho' = \rho'_0\rho'_1\rho'_2\cdots$ be the two infinite runs of $\mathcal{A}$ labelled by ux and vx, respectively, and let $o_0o_1o_2\cdots$ and $o'_0o'_1o'_2\cdots$ be the two corresponding infinite sequences of counter operations performed during these respective runs. Since $u \sim_{\mathcal{A}} v$, there exist two integers m' and n' such that $\rho_{m'+i} = \rho'_{n'+i}$ for all $i \geq 0$, thus there also exist two integers m and n such that $o_{m+i} = o'_{n+i}$ for all $i \geq 0$. Now let $k = \max_{c\in\Gamma} |c(o_0\cdots o_m) - c(o'_0\cdots o'_n)|$. We prove by induction on $i \in \mathbb{N}$ that the relation $|c(o_0\cdots o_{m+i}) - c(o'_0\cdots o'_{n+i})| \leq k$ holds for all $c \in \Gamma$.

By definition of k, the claim holds for $i = 0$. Now let $i > 0$, and assume that for all $j \leq i$, the inequality $|c(o_0\cdots o_{m+j}) - c(o'_0\cdots o'_{n+j})| \leq k$ is true for all $c \in \Gamma$. Let $c \in \Gamma$, and consider the counter operation $o_{m+i+1} = o'_{n+i+1}$. We discuss the nature of this operation.

(1) If $o_{m+i+1} = o'_{n+i+1} = res_c$, then $|c(o_0\cdots o_{m+i+1}) - c(o'_0\cdots o'_{n+i+1})| = 0 \leq k$.
(2) If $o_{m+i+1} = o'_{n+i+1}$ is either inc_c or out_c, then by the induction hypothesis, it follows that $|c(o_0\cdots o_{m+i+1}) - c(o'_0\cdots o'_{n+i+1})| = |c(o_0\cdots o_{m+i}) - c(o'_0\cdots o'_{n+i})| \leq k$.

(3) If $o_{m+i+1} = o'_{n+i+1}$ concerns another counter than c, then by the induction hypothesis $|c(o_0 \cdots o_{m+i+1}) - c(o'_0 \cdots o'_{n+i+1})| = |c(o_0 \cdots o_{m+i}) - c(o'_0 \cdots o'_{n+i})| \leq k$.
(4) If $o_{m+i+1} = o'_{n+i+1} = max_{c,d}$, for some $d \in \Gamma$, four different cases need to be considered:
   (a) If $c(o_0 \cdots o_{m+i}) \leq d(o_0 \cdots o_{m+i})$ and $c(o'_0 \cdots o'_{n+i}) \leq d(o'_0 \cdots o'_{n+i})$, it follows that $c(o_0 \cdots o_{m+i+1}) := d(o_0 \cdots o_{m+i})$ and $c(o'_0 \cdots o'_{n+i+1}) := d(o'_0 \cdots o'_{n+i})$. Therefore by the induction hypothesis $|c(o_0 \cdots o_{m+i+1}) - c(o'_0 \cdots o'_{n+i+1})| = |d(o_0 \cdots o_{m+i}) - d(o'_0 \cdots o'_{n+i})| \leq k$.
   (b) The case $c(o_0 \cdots o_{m+i}) \geq d(o_0 \cdots o_{m+i})$ and $c(o'_0 \cdots o'_{n+i}) \geq d(o'_0 \cdots o'_{n+i})$ is symmetric.
   (c) If $c(o_0 \cdots o_{m+i}) \leq d(o_0 \cdots o_{m+i})$ but $c(o'_0 \cdots o'_{n+i}) \geq d(o'_0 \cdots o'_{n+i})$, it follows that $c(o_0 \cdots o_{m+i+1}) := d(o_0 \cdots o_{m+i})$ and $c(o'_0 \cdots o'_{n+i+1}) := c(o'_0 \cdots o'_{n+i})$. Thence $|c(o_0 \cdots o_{m+i+1}) - c(o'_0 \cdots o'_{n+i+1})| = |d(o_0 \cdots o_{m+i}) - c(o'_0 \cdots o'_{n+i})|$. Now the two following cases need to be distinguished:
      i. If $c(o'_0 \cdots o'_{n+i}) \leq d(o_0 \cdots o_{m+i})$, thence $|d(o_0 \cdots o_{m+i}) - c(o'_0 \cdots o'_{n+i})| = d(o_0 \cdots o_{m+i}) - c(o'_0 \cdots o'_{n+i}) \leq d(o_0 \cdots o_{m+i}) - d(o'_0 \cdots o'_{n+i}) \leq k$.
      ii. If $c(o'_0 \cdots o'_{n+i}) \geq d(o_0 \cdots o_{m+i})$, thence $|d(o_0 \cdots o_{m+i}) - c(o'_0 \cdots o'_{n+i})| = c(o'_0 \cdots o'_{n+i}) - d(o_0 \cdots o_{m+i}) \leq c(o'_0 \cdots o'_{n+i}) - c(o'_0 \cdots o'_{n+i}) \leq k$.
   (d) The case $c(o_0 \cdots o_{m+i}) \geq d(o_0 \cdots o_{m+i})$ but $c(o'_0 \cdots o'_{n+i}) \leq d(o'_0 \cdots o'_{n+i})$ is symmetric.

Now since $|c(o_0 \cdots o_{m+i}) - c(o'_0 \cdots o'_{n+i})| \leq k$ for all $i \geq 0$ and all $c \in \Gamma$, it follows that, for all $c \in \Gamma$, the output sequence ρ_c is bounded iff ρ'_c is also bounded. Therefore $ux \in L(\mathcal{A})$ iff $vx \in L(\mathcal{A})$ for all $x \in A^\omega$, or in other words, $u^{-1}L(\mathcal{A}) = v^{-1}L(\mathcal{A})$. □

3 The Wadge hierarchy of max-regular languages

The collection of all max-regular languages ordered by the Wadge reduction will be called the *MR-Wadge hierarchy*. The present section provides a description of this hierarchy. We prove that, although the class of max-regular languages properly extends the class of ω-regular languages, the MR-Wadge hierarchy and the Wagner hierarchy are equal up to Wadge equivalence.

THEOREM 5. *Max-regular languages inhabit exactly those self-dual and non-self-dual classes, which have the Wadge degree of the form*

$$\omega_1^{n_k} \cdot p_k + \cdots + \omega_1^{n_0} \cdot p_0$$

with $k > 0$, $\omega > n_k > \ldots > n_0 \geq 0$, *and* $0 < p_i < \omega$ *for all* $0 \leq i \leq k$.

In particular, the MR-Wadge hierarchy is a pre-well-ordering of width 2 *and height* ω^ω.

PROOF. Let α be an ordinal with Cantor normal form $\alpha = \omega_1^{n_k} \cdot p_k + \cdots + \omega_1^{n_0} \cdot p_0$, for some $k > 0$, some $\omega > n_k > \ldots > n_0 \geq 0$ and some $0 < p_i < \omega$ for all $0 \leq i \leq k$. In the Wagner hierarchy, there exist two ω-regular languages L and L' such that L is self-dual, L' is non-self dual, and $d_W(L) = d_W(L') = \alpha$. Lemma 3 guarantees that L and L' are also max-regular.

It remains to prove that no other Wadge class is inhabited by a max-regular language. Let L be a max-regular language over the alphabet A. The language L is recognised by a

finite state max-automaton, so from Lemma 4 it follows that the family $\{u^{-1}L : u \in A^*\}$ is finite. But then, up to Wadge equivalence, $\{L_u : u \in A^*\}$ is finite and the claim follows by Lemma 1. □

More precisely, the MR-Wadge hierarchy consists of an alternating succession of non-self-dual and self-dual Wadge classes with non-self-dual pairs at each limit level. The MR degree of a max-regular language L is now defined as

$$d_{MR}(L) = \begin{cases} 1 & \text{if } L = \emptyset \text{ or } L = \emptyset^c, \\ \sup\{d_{MR}(K) + 1 : K \text{ n.s.d. and } K <_W L\} & \text{if } L \text{ is non-self-dual,} \\ \sup\{d_{MR}(K) : K \text{ n.s.d. and } K <_W L\} & \text{if } L \text{ is self-dual.} \end{cases}$$

Once again, this definition of the MR degree ensures that the non-self dual languages and the self dual ones located just one level above in the MR-Wadge hierarchy always share the same degree. Therefore, the MR-Wadge and the Wadge degrees of max-regular languages are related as follows: for any max-regular languages L, if $d_{MR}(L) = \omega^{n_k} \cdot p_k + \cdots + \omega^{n_0} \cdot p_0$, for some $\omega > n_k > \ldots > n_0 \geq 0$ and $0 < p_i < \omega$ for all $0 \leq i \leq k$, then $d_W(L) = \omega_1^{n_k} \cdot p_k + \cdots + \omega_1^{n_0} \cdot p_0$.

4 The MR-Wadge and the Wagner hierarchies

We now provide a detailed comparison of the MR-Wadge and the Wagner hierarchies. In the previous section we have seen that the MR-Wadge and the Wagner hierarchies inhabit exactly the same Wadge classes.

THEOREM 6. *The MR-Wadge and the Wagner hierarchy are equal (up to Wadge equivalence).*

The following two results prove that the ω first classes of the MR-Wadge and the Wagner hierarchies contain exactly the same ω-languages, whereas every other MR-Wadge class is a proper extension of its Wagner counterpart (see Fig. 1(b)).

PROPOSITION 7. *For every natural number n the following conditions are equivalent:*
(1) L is ω-regular and $d_{\omega R}(L) = n$.
(2) L is max-regular and $d_{MR}(L) = n$.

PROOF. Let us first see that (1) implies (2). Let L be ω-regular with $d_{\omega R}(L) = n$. Then L is also max-regular. Moreover, the structure of the Wagner hierarchy ensures that $d_W(L) = n$. Hence, by Theorem 5, $d_{MR}(L) = n$.

Now, let us prove that (2) implies (1). Take a max-regular language L with $d_{MR}(L) = n$. We first show that L is ω-regular. Let $\mathcal{A} = (Q, A, \Gamma, q_0, \delta, \mathcal{T})$ be a max-automaton that recognises L. Let $\mathcal{C}_1, \ldots, \mathcal{C}_p$ be all (maximal) strongly connected components (s.c.c.) of the graph of the automaton $\mathcal{A}$. Given any infinite word x, we denote $\mathrm{scc}(x)$ the unique s.c.c. that contains all states visited infinitely often while reading x. In other words, $\mathrm{scc}(x)$ is the s.c.c. inside which the reading of the terminal part of x takes place. Consider the following equivalence relation between infinite words: $x \approx y$ iff $\mathrm{scc}(x) = \mathrm{scc}(y)$. We claim that $x \approx y$ implies that $(x \in L \Leftrightarrow y \in L)$. Towards a contradiction, assume that there exist $x \in L$ and

$y \notin L$ with $x \approx y$. Let $\mathrm{scc}(x) = \mathrm{scc}(y) = \mathcal{C}_i$ and let $u, v \in A^*$ be the shortest prefixes of x and y respectively such that there exist respectively $q_u, q_v \in \mathcal{C}_i$ with $q_0 \xrightarrow{u} q_u$ and $q_0 \xrightarrow{v} q_v$. Let x', y' be such that $x = ux'$ and $y = vy'$. Since $\mathcal{C}_i$ is a s.c.c., there exists a finite word w such that $q_u \xrightarrow{w} q_v$. Consider $Z = \{z \in uA^\omega : \mathrm{scc}(z) = \mathcal{C}_i\}$. We next prove the following facts:

(1) $Z \cap L$ is initialisable,

(2) both $\emptyset \leq_W Z \cap L$ and $\emptyset^c \leq_W Z \cap L$ hold,

(3) $Z \cap L \leq_W L$.

(1) Consider the II-imposed game $\mathbb{W}(Z \cap L, Z \cap L)$ where I may only once erase his play and start anew. We will provide a winning strategy for player II that guarantees that she *always remains inside* Z. As long as player I stays inside Z, player II should copy his actions. If player I exits Z, player II should play a finite word that reaches q_v, and then to play y'. If player I decides to erase everything he has played since the beginning, then player II can still catch up by playing any finite word that leads her back to q_u, and start copying again I's play, from the moment when I reaches q_u. If player I exits Z again, II should proceed like before. By Lemma 4 this provides a winning strategy. (2) $\emptyset \leq_W Z \cap L$ and $\emptyset^c \leq_W Z \cap L$ hold because playing $x = uwy'$ or ux', respectively, is winning for II in the corresponding Wadge games. (3) A winning strategy for player II in $\mathbb{W}(Z \cap L, L)$ amounts to copying player I's moves, as long as he stays in Z. If player I exits Z, player II should play a word reaching q_v (this is always possible, since so far player II has stayed inside Z) and then play y'.

Since $Z \cap L$ is a Boolean combination of $\mathbf{\Sigma}^0_2$ sets, by a result from [7], condition (1) yields $d_W(Z \cap L) = \omega_1^n$ for some natural n. Condition (2) ensures that $n > 0$, hence $d_W(Z \cap L) \geq \omega_1$. Finally, condition (3) implies that $d_W(L) \geq \omega_1$, but this is a contradiction. Hence, the claim holds.

Consider $\mathcal{A}' = (Q, A, q_0, \delta', F)$, the deterministic finite automaton with Büchi acceptance conditions where δ' is just δ with the operations on counters removed, and F is the set of states q for which there exists an infinite word $x \in L$ such that $q \in \mathrm{scc}(x)$. Then $\mathcal{A}'$ recognises L, which shows that L is ω-regular. Theorem 6, guarantees that $d_{MR}(L) = d_{\omega R}(L) = d_W(L) = n$. □

Before we move to the proof of our last result, let us show that the language

$$K = \{a^{n_1}ba^{n_2}ba^{n_3}b \cdots : \forall m \; \exists i \; n_i > m\}$$

is $\mathbf{\Pi}^0_2$-complete, as stated in the introduction. It is very easy to see that it is Wadge equivalent to the $\mathbf{\Pi}^0_2$-complete $L' = (a^*b)^\omega$. Indeed, player II has a winning strategy in the game $\mathbb{W}(L, L')$: every time player I produces a sequence of consecutive a's that is strictly longer than all previous ones, Player II should play a b. Otherwise, player II should play an a. Conversely, player II also has a winning strategy in the game $\mathbb{W}(L', L)$: every time player I plays a b, player II should play a sequence of consecutive a's that is strictly longer than all previously played, followed by b. Otherwise, she should play b alone.

PROPOSITION 8. *Let $\alpha = \omega_1^{n_k} \cdot p_k + \cdots + \omega_1^{n_0} \cdot p_0 \geq \omega_1$, where $\omega > n_k > \ldots > n_0 \geq 0$ and $0 < p_i < \omega$ for all $0 \leq i \leq k$. Then there exist max-regular languages L and L' such that L is self-dual, L' is non-self-dual, $d_W(L) = d_W(L') = \alpha$, and both L and L' are not ω-regular.*

PROOF. Without loss of generality we may assume that $A = \{a, b\}$. We first prove the existence of appropriate non-self-dual languages over A. If $\alpha = \omega_1$, then consider the lan-

guage K above. It is $\mathbf{\Pi}^0_2$-complete, which means that $d_W(K) = \omega_1 = \alpha$, as mentioned in Sect. 1.2. Now if $\alpha = \omega_1^{n_k} \cdot p_k + \cdots + \omega_1^{n_0} \cdot p_0 > \omega_1$, then there exists a non-self-dual ω-regular language $M \subseteq A^\omega$ such that $d_W(M) = \alpha$. Let $L = aM \cup bK$. The language L is non-self-dual and satisfies $L \equiv_W M$. Thus $d_W(L) = d_W(M) = \alpha$. In addition, since both M and K are max-regular, so is L. Finally, L is not ω-regular, for if it were so, then $b^{-1}L = K$ would also be ω-regular – a contradiction.

From the existence of an appropriate non-self-dual language, we deduce the existence of an appropriate self-dual language over A. Let $L \subseteq A^\omega$ be a non-self-dual max-regular language such that both $d_W(L) = \alpha$ and L is not ω-regular. Take $L' = aL \cup bL^c$. Then L' is also max-regular. Moreover, as mentioned in Sect. 1.2, L' is self-dual and $d_W(L') = \alpha$. Finally, L' is not ω-regular, for if it were so, the language $a^{-1}L' = L$ would also be ω-regular – a contradiction. □

Conclusion

We have given a precise comparison of the Wadge hierarchies for regular and max-regular languages. As the hierarchies coincide, Bojańczyk's extension does not increase the topological complexity. It does provide more variety though, as witnessed by the plethora of separating examples.

The results of this paper give a complete description of the Wadge hierarchy of max-regular languages. Alas, the description is not effective (unlike [10, 15]). What is missing is an algorithm to decide the Wadge degree of a given language. From the proof of Proposition 7 one could extract a partial decidability result. Using decidability of emptiness for max-automata, one can check if there are two words $x \in L(A)$ and $y \notin L(A)$, such that the runs on both of them are finally trapped in the same strongly connected component of A, thus deciding if $L(A)$ is at least on the level ω or not. If not, one can construct effectively an equivalent automaton without counters, and use the Wagner's characterisation to compute the exact degree. Obtaining decidability of higher levels would probably require much deeper analysis of the loop structure within strongly connected components. We point this out as a promising line of investigation.

As for the technical side of the paper, we would like to highlight the method used to prove that no other Wadge degrees are realised by max-regular languages (Theorem 5). Here, the argument relies on the fact that the family $\{w^{-1}L \colon w \in A^*\}$ is finite up to Wadge equivalence. A more involved version of this method, based on a generalisation of Lemma 1, has been successfully applied to deterministic push-down automata [8]. We believe that this technique can be useful for other models of computation as well.

Acknowledgement

The fourth author was supported by the Polish government grant no. N206 008 32/0810. Part of this work was carried out when the fourth author was postdoc at the University of Edinburgh, UK. The suggestions of the anonymous referees were very helpful in the preparation of the final version of this paper.

References

[1] M. Bojańczyk. Weak MSO with the unbounding quantifier. STACS 2009: 159–170.

[2] J. R. Büchi. On a decision method in restricted second order arithmetic. *Proc. 1960 Int. Congr. for Logic, Methodolog, and Philosophy of Science*, Stanford Univ. Press, 1962, 1–11.

[3] J. Cabessa, J. Duparc. A game theoretical approach to the algebraic counterpart of the Wagner hierarchy: Part i. *RAIRO-Theor. Inf. Appl.* 43(3): 443–461 (2009).

[4] J. Cabessa, J. Duparc. A game theoretical approach to the algebraic counterpart of the Wagner hierarchy: Part ii. *RAIRO-Theor. Inf. Appl.* 43(3): 463–515 (2009).

[5] O. Carton, D. Perrin. Chains and superchains for ω-rational sets, automata and semigroups. *Int. J. Algebra Comput.*, 7(6): 673–695 (1997).

[6] O. Carton, D. Perrin. The Wagner hierarchy. *Int. J. Algebra Comput.* 9(5):597–620 (1999).

[7] J. Duparc. Wadge hierarchy and Veblen hierarchy. Part i: Borel sets of finite rank. *J. Symb. Log.* 66(1): 56–86 (2001).

[8] J. Duparc. A hierarchy of deterministic context-free ω-languages. *Theor. Comput. Sci.* 290(3): 1253–1300 (2003).

[9] J. Duparc, M. Riss. The missing link for ω-rational sets, automata, and semigroups. *Int. J. Algebra Comput.* 16(1): 161–185 (2006).

[10] O. Finkel. An Effective Extension of the Wagner Hierarchy to Blind Counter Automata. CSL 2001: 369-383

[11] D. Perrin, J.-É. Pin. *Infinite words*. Volume 141 of *Pure and Applied Mathematics*, Elsevier, 2004.

[12] V. Selivanov. Fine hierarchy of regular ω-languages. *Theor. Comput. Sci.* 191(1-2): 37–59 (1998).

[13] W. W. Wadge. Degrees of complexity of subsets of the Baire space. *Notices A.M.S.*, pp. A714–A715, 1972.

[14] W. W. Wadge. *Reducibility and determinateness on the Baire space*. PhD thesis, University of California, Berkeley (1983).

[15] K. Wagner. On ω-regular sets. *Inform. and Control* 43(2): 123–177 (1979).

[16] T. Wilke, H. Yoo. Computing the Wadge degree, the Lifshitz degree, and the Rabin index of a regular language of infinite words in polynomial time. TAPSOFT 1995: 288–302.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Automata and temporal logic over arbitrary linear time

Julien Cristau
LIAFA — CNRS & Université Paris 7

ABSTRACT. Linear temporal logic was introduced in order to reason about reactive systems. It is often considered with respect to infinite words, to specify the behaviour of long-running systems. One can consider more general models for linear time, using words indexed by arbitrary linear orderings. We investigate the connections between temporal logic and automata on linear orderings, as introduced by Bruyère and Carton. We provide a doubly exponential procedure to compute from any LTL formula with Until, Since, and the Stavi connectives an automaton that decides whether that formula holds on the input word. In particular, since the emptiness problem for these automata is decidable, this transformation gives a decision procedure for the satisfiability of the logic.

1 Introduction

Temporal logic, in particular LTL, was proposed by Pnueli to specify the behaviour of reactive systems [12]. The model of time usually considered is the ordered set of natural numbers, and executions of the system are seen as infinite words on some set of atomic propositions. This logic was shown to have the same expressive power as the first order logic of order [11], but it provides a more convenient formalism to express verification properties. It is also more tractable: while the satisfiability problem of FO is non-elementary [17], it was shown in [16] that the decision problem of LTL with Until and Since on ω-words is PSPACE-complete. This logic has also strong ties with automata, with important work to provide efficient translations to Büchi automata, e.g. [9].

Within this time model, a number of extensions of the logic and the automata model have been studied. But one can also consider more general models of time: general linear time could be useful in different settings, including concurrency, asynchronous communication, and others, where using the set of integers can be too simplistic. Possible choices include ordinals, the reals, or even arbitrary linear orderings. In terms of expressivity, while LTL with Until and Since is expressively complete (*i.e.* equivalent to FO) on Dedekind-complete orderings (which includes the ordering of the reals as well as all ordinals), this does not hold in the general case. Two more connectives, the future and past Stavi operators, are necessary to handle gaps [10] when considering arbitrary linear orderings.

Over ordinals, LTL with Until and Since has been shown to have a PSPACE-complete satisfiability problem [7]. Over the ordering of the real numbers, satisfiability of LTL with until and since is PSPACE-complete, but satisfiability of MSO is undecidable. Over general linear time, first order logic has been shown to be decidable, as well as universal monadic second order logic. Reynolds shows in [14] that the satisfiability problem of temporal logic with only the Until connective is also PSPACE-complete, and conjectures that this might stay true when adding the Since connective. The upper bound in [7] is obtained by reducing the satisfiability of LTL formulae to the accessibility problem in an appropriate automata model,

© Julien Cristau; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 133–144
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2313

accepting words indexed by ordinals. In this paper, we focus on the general case of arbitrary linear orderings, using the full logic with Until, Since and both Stavi connectives. Our aim is to investigate the connections between LTL and automata in this setting.

Automata on linear orderings were introduced by Bruyère and Carton [2]. This model extends traditional finite automata using "limit" transitions to handle positions with no successor or predecessor, furthering Büchi's model of automata on words of ordinal length [4]. Carton showed in [5] that accessibility over scattered ordering is decidable in polynomial time, and in [13] it was shown that these automata can be complemented over countable scattered linear orderings. The accessibility result can be extended to arbitrary orderings [6].

From any formula in this logic, we define an automaton which determines whether the formula holds on its input word. Satisfiability of the formula is reduced to accessibility in this automaton, and that way we get decidability of the satisfiability problem of LTL with Until, Since and the Stavi operators for any rational subclass.

Section 2 presents some definitions about linear orderings, linear temporal logic, and the model of automata used. Section 3 introduces our main result, an algorithm to translate any LTL formula into a corresponding automaton. Section 4 discusses the expressivity of the logic and automata considered, and looks at some natural fragments.

2 Definitions

2.1 Linear orderings

We first recall some basic definitions about orderings, and introduce some notations. For a complete introduction to linear orderings, the reader is referred to [15]. A *linear ordering* J is a totally ordered set $(J, <)$ (considered modulo isomorphism). The sets of integers (ω), of rational numbers (η), and of real numbers with the usual orderings are all linear orderings.

Let J and K be two linear orderings. One defines the reversed ordering $-J$ as the ordering obtained by reversing the relation $<$ in J, and the ordering $J + K$ as the disjoint union $J \sqcup K$ extended with $j < k$ for any $j \in J$ and $k \in K$. For example, $-\omega$ is the ordering of negative integers. $-\omega + \omega$ is the usual ordering of $\mathbb{Z}$, also denoted by ζ.

A non-empty subset K of an ordering J is an *interval* if for any $i < j < k$ in J, if $i \in K$ and $k \in K$ then $j \in K$. In order to define the runs of an automaton, we use the notion of cut. A *cut* of an ordering J is a partition (K, L) of J such that for any $k \in K$ and $l \in L$, $k < l$. We denote by $\hat{J}$ the set of cuts of J. This set is equipped with the order defined by $(K_1, L_1) < (K_2, L_2)$ if $K_1 \subsetneq K_2$. This ordering can be extended to $J \cup \hat{J}$ in a natural way ($(K, L) < j$ iff $j \in L$). Notice that $\hat{J}$ always has a smallest and a biggest element, respectively $c_{\min} = (\emptyset, J)$ and $c_{\max} = (J, \emptyset)$. For example, the set of cuts of the finite ordering $\{0, 1, \ldots, n-1\}$ is the ordering $\{0, 1, \ldots, n\}$, and the set of cuts of ω is $\omega + 1$.

For any element j of J, there are two successive cuts c_j^- and c_j^+, respectively $(\{i \in J \mid i < j\}, \{i \in J \mid j \leq i\})$ and $(\{i \in J \mid i \leq j\}, \{i \in J \mid j < i\})$. A *gap* in an ordering J is a cut c which is not an extremity ($c_{\max}$ or $c_{\min}$), and has neither a successor nor a predecessor.

Given an alphabet Σ, a *word* of length J is a sequence $(a_j)_{j \in J}$ of elements of Σ indexed by J. For example, $(ab)^\omega$ is a word of length ω; the sequence $ab^\omega ab^\omega a$ is a word of length $\omega + \omega + 1$, and $(ab^\omega)^\omega$ is a word of length ω^2.

2.2 Temporal logic

We use words over linear orderings to model the behaviour of systems over linear time. To express properties of these systems, we consider linear temporal logic. The set of LTL formulae is defined by the following grammar, where p ranges over a set AP of atomic propositions: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\mathcal{U}\varphi \mid \varphi\mathcal{S}\varphi \mid \varphi\mathcal{U}'\varphi \mid \varphi\mathcal{S}'\varphi$

Besides the usual boolean operators, we have four temporal connectives. The $\mathcal{U}$ connective is called "Until", and $\mathcal{S}$ is called "Since". $\mathcal{U}'$ and $\mathcal{S}'$ are respectively the future and past Stavi connectives. Other usual connectives such as "Next" ($\mathcal{X}$), "Eventually" ($\mathcal{F}$), "Always" ($\mathcal{G}$) can be defined using these, as we see below.

These formulae are interpreted on words over the alphabet 2^{AP}. A letter in those words is the set of atomic propositions that hold at the corresponding position. Let $x = (x_j)_{j\in J}$ a word of length J. A formula φ is evaluated at a particular position i in x; we say that φ holds at position i in x, and we write $x, i \models \varphi$, using the following semantics:

$x, i \models p$ if $p \in x_i$

$x, i \models \neg\psi$ if $x, i \not\models \psi$

$x, i \models \psi_1 \vee \psi_2$ if $x, i \models \psi_1$ or $x, i \models \psi_2$

$x, i \models \psi_1\mathcal{U}\psi_2$ if there exists $j > i$ such that $x, j \models \psi_2$, and for any k such that $i < k < j$, we have $x, k \models \psi_1$

$x, i \models \psi_1\mathcal{S}\psi_2$ if $-x, i \models \psi_1\mathcal{U}\psi_2$ where $-x$ is the reversed word $(a_j)_{j\in -J}$

$x, i \models \psi_1\mathcal{U}'\psi_2$ if there exists a gap $c \in \hat{J}$ verifying three properties:

(1) $x, j \models \psi_1$ for any position j such that $i < j < c$

(2) there is no interval starting at c where ψ_1 is always true (*i.e.* $\forall c < k\ \exists c < j < k\ x, j \models \neg\psi_1$), and

(3) ψ_2 is always true in some interval starting at c

$x, i \models \psi_1\mathcal{S}'\psi_2$ if $-x, i \models \psi_1\mathcal{U}'\psi_2$ (it is the corresponding past connective)

Note that we use a "strict" semantic for the Until operator, contrary to a common definition, which would be:

$$x, i \models \psi_1\mathcal{U}^{ns}\psi_2 \quad \text{if} \quad \text{there exists } j \geq i \text{ such that } x, j \models \psi_2 \text{ and } x, k \models \psi_1 \text{ for any } i \leq k < j.$$

In the strict version, the current position i is not considered for either the ψ_1 or the ψ_2 part of the definition. Using the strict or non-strict version makes no difference when considering ω-words, but in the case of arbitrary orderings, the strict Until is more powerful, as noted by Reynolds in [14].

The formula "Next φ", or $\mathcal{X}\,\varphi$, is equivalent to $\bot\mathcal{U}\varphi$. "Eventually φ", noted $\mathcal{F}\,\varphi$, is $\varphi \vee (\top\mathcal{U}\varphi)$, and "always φ", noted $\mathcal{G}\,\varphi$, can be expressed as $\neg(\mathcal{F}(\neg\varphi))$.

Given a word x of length J, the *truth word* of φ on x is the word $v_\varphi(x)$ of length J over the alphabet $\{0, 1\}$ where the position j is labelled by 1 iff $x, j \models \varphi$. A formula is *valid* if its truth word on any input only has ones. A formula is *satisfiable* if there exists an input word such that the truth word contains a one.

Consider the formula $\varphi = \neg a \wedge (\mathcal{G} \neg \mathcal{X} a)$, with AP $= \{a\}$. If $x = (a\emptyset)^\omega$ (where a stands for $\{a\}$), then $v_\varphi(x) = 0^\omega$ (at every position, either a is true or a is true in the successor). On the other hand, if $x = a\emptyset^\omega a\emptyset^\omega a$, then $v_\varphi(x) = 01^\omega 01^\omega 0$: at positions 0, ω and at the last position, a is true so the formula doesn't hold; at all other positions, a is false, and there is no position in the input word where $\mathcal{X} a$ holds.

The *satisfiability problem* for a formula φ consists in deciding whether there exists a word w and a position i in w such that $w, i \models \varphi$. As FO is decidable, and every LTL formula can be expressed using first order, satisfiability of LTL is decidable. Note however that in terms of complexity FO is already non-elementary on finite words [17], which is not true of LTL.

2.3 Automata

On infinite words, Büchi automata can be used to decide satisfiability of LTL formulae. In the case of words over linear orderings, a model of automata has been introduced in [2]. Instead of accepting or rejecting each input word, as in the case of ω-words, we use these automata to compute the truth words corresponding to an LTL formula. Our model of automata thus has an output letter on each transition, so they are actually letter-to-letter transducers, which make composition easier (see Section 3.1).

An *automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, I, F)$ where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite output alphabet, I and F are subsets of Q, respectively the set of initial and final states, and $\delta \subseteq (Q \times \Sigma \times \Gamma \times Q) \cup (2^Q \times Q) \cup (Q \times 2^Q)$ is the set of transitions. We note:

- $p \xrightarrow{a|b} q$ if $(p, a, b, q) \in \delta$ (*successor* transition)
- $P \to q$ if $(P, q) \in \delta$ (*left limit* transition)
- $q \to P$ if $(q, P) \in \delta$ (*right limit* transition).

Consider a word $x = (q_j)_{j \in J}$ over Q. We define the left and right limit sets of x at position $j \in J$ as the sets of labels that appear arbitrarily close to j (respectively to its left and to its right). Formally:

$$\lim_{j^-} x = \{q \in Q \mid \forall k < j \; \exists i \; k < i < j \wedge q_i = q\}$$
$$\lim_{j^+} x = \{q \in Q \mid \forall k > j \; \exists i \; j < i < k \wedge q_i = q\}$$

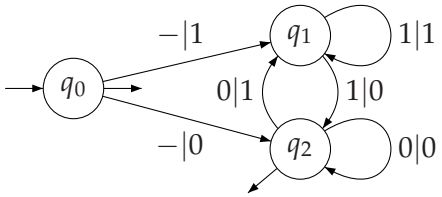
Note that $\lim_{j^-} x$ is non-empty if and only if the transition to j is a left limit, and similarly for $\lim_{j^+} x$ if the transition from j is a right limit. These sets help define the possible limit transitions in a run.

Given an automaton $\mathcal{A}$, an accepting run of $\mathcal{A}$ on a word $x = (x_j)_{j \in J}$ is a word ρ of length $\hat{J}$ over Q such that:

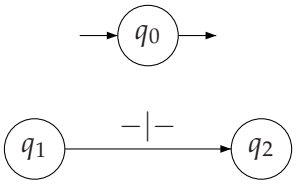
- $\rho_{c_{\min}} \in I$ and $\rho_{c_{\max}} \in F$;
- for each $i \in J$, there exists $y_i \in \Gamma$ such that $\rho_{c_i^-} \xrightarrow{x_i|y_i} \rho_{c_i^+}$;
- if $c \in \hat{J}$ has no predecessor, $\lim_{c^-} \rho \to \rho_c$, and if $c \in \hat{J}$ has no successor, $\rho_c \to \lim_{c^+} \rho$.

EXAMPLE 1. *The first automaton in Figure 1 outputs* 1 *at each position immediately followed by a* 1 *in the input word, and* 0 *at other positions.*

The second automaton accepts input words whose length is a linear ordering without first or last element, and without two consecutive elements (i.e. dense *orderings). The notation $P \to q_0, q_1$ means that there is a transition $P \to q_0$ and a transition $P \to q_1$.*



Limits: $P \to q_0$ $q_0 \to P$ $q_2 \to P$
for any $P \subseteq \{q_0, q_1, q_2\}$



Limits: $\{q_1, q_2\}, \{q_0, q_1, q_2\} \to q_0, q_1$
$q_1, q_2 \to \{q_1, q_2\}, \{q_0, q_1, q_2\}$

Figure 1: Example automata

In [5], Carton proves that the accessibility problem on these automata can be solved in polynomial time, when only considering scattered orderings. This result can be extended to arbitrary orderings [6] as it is done for rational expressions in [3]. The idea is to build an automaton over finite words which simulates the paths in the initial automaton and remembers their contents. In order to handle the general case (as opposed to only scattered orderings), the added operation is called "shuffle": $sh(w_1, \ldots, w_n) = \Pi_{j \in J} x_j$ where J is a dense and complete ordering without a first or last element, partitioned in dense suborderings $J_1 \ldots J_n$, such that $x_j = w_i$ if $j \in J_i$. Looking at automata, this means that if there are paths from p_1 to q_1 with content P_1, ..., from p_n to q_n with content P_n, and transitions from $P_1 \cup \cdots \cup P_n$ to each p_i, transitions from each q_i to $P_1 \cup \cdots \cup P_n$, a transition from p to $P_1 \cup \cdots \cup P_n$ and a transition from $P_1 \cup \cdots \cup P_n$ to q, then there is a path from p to q.

3 Translation between formulae and automata

Over ω-words, problems on temporal logics are commonly solved using tableau methods [20], or automata-based techniques [19]. In this work we extend the correspondence between LTL and automata to words over linear orderings. Our main result is Theorem 2.

THEOREM 2. *For every LTL formula φ, there is an automaton $\mathcal{A}_\varphi$ which given any input word x outputs the truth word $v_\varphi(x)$.*

Moreover, this automaton $\mathcal{A}_\varphi$ can be effectively computed, and has a number of states exponential in the size of φ. Because we can compute the product of $\mathcal{A}_\varphi$ with any given automaton and check for its emptiness, we get Corollary 3, which states that given a temporal formula and a rational property (*i.e.* an automaton on words over linear orderings), we can check whether there exists a model of the formula which is accepted by the automaton.

COROLLARY 3. *The satisfiability problem for any rational subclass is decidable.*

The idea is to build $\mathcal{A}_\varphi$ by induction on the formula. We construct an elementary automaton for each logical connective. We use composition and product operations to build inductively the automaton of any LTL formula from elementary automata. All automata used in this proof have the particular property that there exists exactly one accepting run for each possible input word, *i.e.* they are non-deterministic, but also non-ambiguous. This property is preserved by composition and product.

The structure of the proof is the following: we define the composition and product operators on automata, then we present the elementary automata that are needed to encode logical connectives. Finally, we give the inductive method to build the automaton corresponding to a formula from elementary ones.

3.1 Product, composition and elementary automata

Let $\mathcal{A}_1 = (Q_1, \Sigma, \Gamma, \delta_1, I_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma', \Delta, \delta_2, I_2, F_2)$ be two automata. The product consists in running both automata with the same input alphabet in parallel, and outputting the combination of their outputs. If $\mathcal{A}_1$'s output alphabet and $\mathcal{A}_2$'s input alphabet are the same, the composition consists in running $\mathcal{A}_2$ over $\mathcal{A}_1$'s output. We use the notation $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$ for the first and second projections.

DEFINITION 4. *Suppose that $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same input alphabet, i.e. $\Sigma = \Sigma'$. The* product *of $\mathcal{A}_1$ and $\mathcal{A}_2$ is the automaton $\mathcal{A}_1 \times \mathcal{A}_2 = (Q_1 \times Q_2, \Sigma, \Gamma \times \Delta, \delta, I_1 \times I_2, F_1 \times F_2)$, where δ contains the following transitions:*

- $(q_1, q_2) \xrightarrow{a|b,c} (q_1', q_2')$ *if* $q_1 \xrightarrow{a|b} q_1'$ *and* $q_2 \xrightarrow{a|c} q_2'$,
- $(q_1, q_2) \to P$ *if* $q_1 \to \pi_1(P)$ *and* $q_2 \to \pi_2(P)$,
- $P \to (q_1, q_2)$ *if* $\pi_1(P) \to q_1$ *and* $\pi_2(P) \to q_2$.

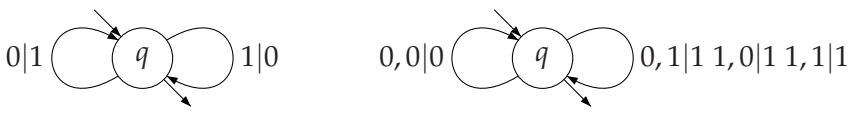
DEFINITION 5. *Suppose now that the output alphabet of $\mathcal{A}_1$ is the input alphabet of $\mathcal{A}_2$, i.e. $\Gamma = \Sigma'$. The* composition *of $\mathcal{A}_1$ and $\mathcal{A}_2$ is the automaton $\mathcal{A}_2 \circ \mathcal{A}_1 = (Q_1 \times Q_2, \Sigma, \Delta, \delta, I_1 \times I_2, F_1 \times F_2)$. The transitions in δ are:*

- $(q_1, q_2) \xrightarrow{a|c} (q_1', q_2')$ *if* $q_1 \xrightarrow{a|b} q_1'$ *and* $q_2 \xrightarrow{b|c} q_2'$,
- $(q_1, q_2) \to P$ *if* $q_1 \to \pi_1(P)$ *and* $q_2 \to \pi_2(P)$,
- $P \to (q_1, q_2)$ *if* $\pi_1(P) \to q_1$ *and* $\pi_2(P) \to q_2$.

Recall that LTL formulae are given by $\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \mathcal{U} \varphi \mid \varphi \mathcal{U}' \varphi \mid \varphi \mathcal{S} \varphi \mid \varphi \mathcal{S}' \varphi$. For each atomic proposition p we construct an automaton $\mathcal{A}_p$ which, given a word x, outputs $v_p(x)$. For each logical connective of arity n, we construct an automaton with input alphabet $\{0,1\}^n$, and output alphabet $\{0,1\}$. The input word is the tuple of truth words of the connective's variables, the output is the truth word of the complete formula. For temporal connectives, we only describe the automata corresponding to $\mathcal{U}$ and $\mathcal{U}'$. For the "past" connectives, the automata are the same with all transitions (successor and limits) reversed, and initial and final states swapped.

For any $p \in \mathrm{AP}$, the automaton $\mathcal{A}_p$ is $(\{q\}, 2^{\mathrm{AP}}, \{0,1\}, \delta, \{q\}, \{q\})$ where $\delta = \{(q \xrightarrow{a|0} q \mid p \notin a\} \cup \{q \xrightarrow{a|1} q \mid p \in a\} \cup \{q \to \{q\}, \{q\} \to q\}$. This automaton simply outputs 1 at positions where p is true, and 0 everywhere else. Note that the run is uniquely determined by the input word; such a transducer is called non-ambiguous.

Figures 2(a) and 2(b) show the automata corresponding to the negation ($\neg$) and disjunction ($\vee$) connectives. Their limit transitions are $\{q\} \to q$ and $q \to \{q\}$. Again, these automata admit exactly one run for each input word.



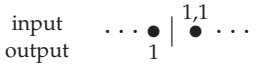
(a) Automaton for negation (b) Automaton for disjunction

3.2 Automaton for $\mathcal{U}$

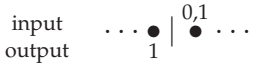
The difficulty starts with the "Until" connective ($\mathcal{U}$). We recall that $\varphi\mathcal{U}\psi$ holds at position i in a word w if there exists $j > i$ such that ψ holds at j, and such that φ holds at every position k such that $i < k < j$.

We build an automaton $\mathcal{A}_\mathcal{U}$ with input alphabet $\{0,1\}^2$ and output alphabet $\{0,1\}$. On an input word of the form $(v_\varphi(w), v_\psi(w))$ for some word w, we want the output to be $v_{\varphi\mathcal{U}\psi}(w)$. Let $J = |w|$, and $c \in \hat{J}$. We distinguish five different situations. For each of them the figure describes an example, where "|" represents the cut c, and each • represents a position in the input word.

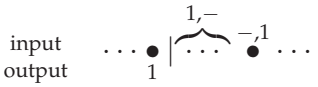
0. c is followed by a position where φ and ψ are true.



1. $c = c_j^-$, and j is such that φ is false and ψ is true.



2. other cases where $\varphi\mathcal{U}\psi$ is true at c.



3. c is followed by a position where both φ and ψ are false.

4. other cases where $\varphi\mathcal{U}\psi$ is false at c. If $c = c_j^-$ then the input at position j is $(1,0)$.

The structure of the automaton $\mathcal{A}_\mathcal{U}$ and the limit transitions are given by Figure 2. This automaton has five states q_0 to q_4 corresponding to the situations described above. Given any two states q and q' there exists a transition $q \to q'$ except from q_2 to q_3 or q_4 and from q_4 to q_0, q_1 or q_2. The input label of successor transitions is determined by the origin node: $(1,1)$ for q_0, $(0,1)$ for q_1, $(0,0)$ for q_3, and $(1,0)$ for q_2 and q_4. The output label is 1 on transitions leading to q_0, q_1 or q_2, and 0 on transitions leading to q_3 or q_4. All states are initial, while q_4 is the only final state.

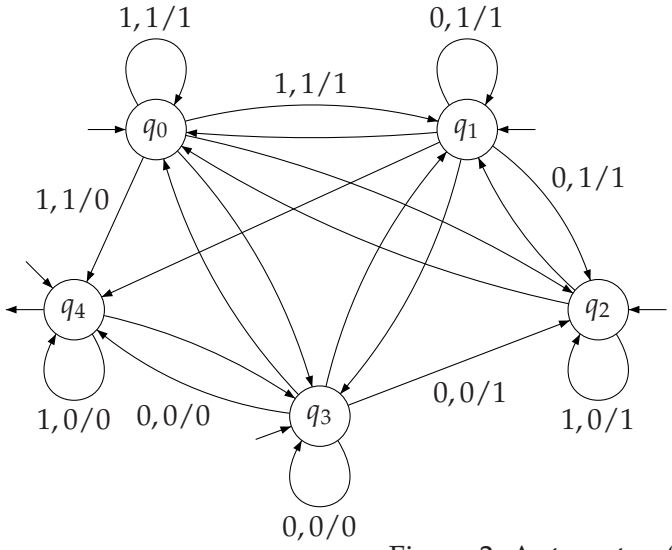
LEMMA 6. *Let φ and ψ two formulae. Let x and y be the truth words of φ and ψ on a word w of length J. The output of $\mathcal{A}_\mathcal{U}$ on (x,y) is the truth word of $\varphi\mathcal{U}\psi$ on w.*

PROOF. Let ρ be the word of length $\hat{J}$ on Q defined by

- if $x_j = y_j = 1$, then $\rho(c_j^-) = q_0$;
- if $x_j = 0$ and $y_j = 1$ then $\rho(c_j^-) = q_1$;
- if $x_j = y_j = 0$ then $\rho(c_j^-) = q_3$;
- otherwise, if there exists $j > c$ such that $y_j = 1$ and for all i such that $c < i < j$, $x_i = 1$, then $\rho(c) = q_2$;
- otherwise, $\rho(c) = q_4$.

We show that ρ is a run of $\mathcal{A}_\mathcal{U}$ on the input (x,y), that this run is unique, and that the corresponding output word is indeed the truth word of $\varphi\mathcal{U}\psi$ on w.

By definition, ρ ends in the final state q_4. Let $c \in \hat{J}$. If $\rho(c)$ is q_0, q_1 or q_3, then $c = c_j^-$ for some j and the successor transition from c to the next cut is allowed by the automaton. If $\rho(c) = q_2$, and $c = c_j^-$ for some j, then $x_j = 1$ and $y_j = 0$, and $\rho(c_j^+)$ is q_0, q_1 or q_2. Similarly



$$
\begin{array}{rcl}
P & \to & q_0, q_1, q_2, q_3, q_4 \\
 & & \text{if } q_0, q_1 \text{ or } q_3 \in P \\
\{q_2\} & \to & q_0, q_1, q_2 \\
\{q_4\} & \to & q_3, q_4 \\
q_2 & \to & \{q_0\}, \{q_2\}, \{q_0, q_2\} \\
q_4 & \to & P \text{ if } q_1 \text{ or } q_3 \in P \\
q_4 & \to & \{q_4\}
\end{array}
$$

Figure 2: Automaton for $\mathcal{U}$

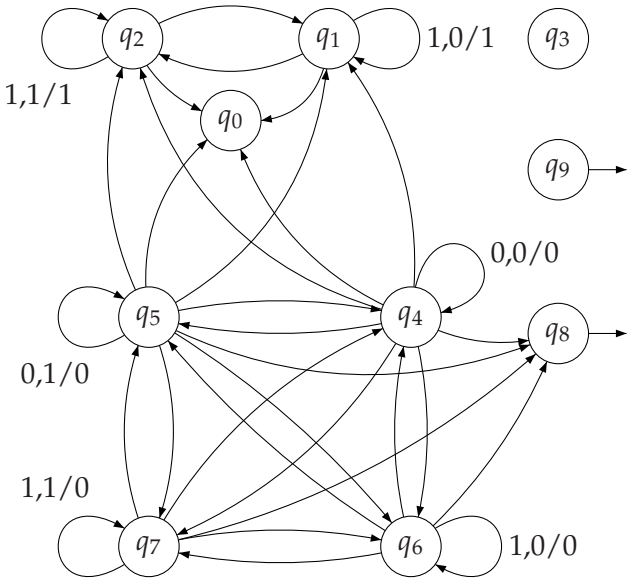
if $\rho(c_j^-) = q_4$, then $x_j = 1$ and $y_j = 0$, and $\rho(c_j^+)$ can be q_3 or q_4. Every successor transition in ρ is thus allowed by $\mathcal{A}_\mathcal{U}$.

We now prove the same for limit transitions. If a left limit transition leads to a cut c, then either ψ is true arbitrarily close to the left of c (in which case the corresponding limit set contains q_0 or q_1), or it is always false (and the limit set is $\{q_2\}$ or a subset of $\{q_3, q_4\}$). If the limit set contains q_0, q_1 or q_3, any state for c is allowed. If the limit set is $\{q_2\}$, the cut c can't be labelled by q_3 or q_4 without violating the definition of ρ. Conversely, if the limit set is $\{q_4\}$, $\rho(c)$ is necessarily q_3 or q_4. Let's now consider a right limit transition starting at a cut c. The label of this cut can only be q_2 or q_4. In the first case, φ must be true everywhere in the limit set, which is thus a subset of $\{q_0, q_2\}$. In the second case, either φ is false infinitely often in the limit, or ψ is always false. This means that the limit set contains q_1 or q_3, or is restricted to $\{q_4\}$. This shows that ρ is a run of $\mathcal{A}_\mathcal{U}$ on the input (x, y).

We now show that a run on $\mathcal{A}_\mathcal{U}$ is uniquely determined by the input word. Let γ a run of $\mathcal{A}_\mathcal{U}$ on (x, y). Because of the constraints on the successor transitions, a cut c is labelled by q_0, q_1 or q_3 in γ if and only if it is labelled by the same state in ρ.

Let's suppose that a cut c is labelled by q_2 in γ. Since q_2 is not final, there exists $c' > c$ labelled by some other state. If there is a first such cut, its label is necessarily q_0 or q_1 (by a successor transition from q_2 or a limit transition from $\{q_2\}$). Otherwise, there is a transition of the form $q_2 \to \{q_0\}$ or $q_2 \to \{q_0, q_2\}$. In both cases, c satisfies the condition for cuts labelled by q_2 in the definition of ρ. A similar argument shows that a cut labelled by q_4 in γ has the same label in ρ. The run of $\mathcal{A}_\mathcal{U}$ on a given input word is thus unique.

Finally, we show that the output word is really the truth word of $\varphi \mathcal{U} \psi$. Let j an element of J. First, suppose that $w, j \models \varphi \mathcal{U} \psi$. If j has a successor k, and ψ is true at k, then $y_k = 1$, and $\mathcal{A}_\mathcal{U}$ outputs 1 at position j. Otherwise, there exists $k > j$ such that $w, k \models \psi$ (i.e. $y_k = 1$), and $x_\ell = 1$ whenever $j < \ell < k$. Thus, c_j^+ is labelled with q_2, and $\mathcal{A}_\mathcal{U}$ once again outputs 1 at position j. Suppose now that $w, j \not\models \varphi \mathcal{U} \psi$. If j has a successor k and $x_k = y_k = 0$, then c_j^+ is labelled by q_3, so the output at position j is 0. Otherwise, c_j^+ is labelled by q_4, and once again $\mathcal{A}_\mathcal{U}$ outputs 0. ∎



- $P \to q_0, q_1, q_2$ if $P \cap \{q_4, q_5\} \neq \emptyset$ or $P \subseteq \{q_0, q_1, q_2\}$
- $P \to q_3$ if $P \subseteq \{q_0, q_1, q_2\}$
- $P \to q_4, q_5, q_6, q_7$ if $P \not\subseteq \{q_0, q_1, q_2\}$
- $P \to q_8$ if $P \cap \{q_4, q_5\} \neq \emptyset$
- $P \to q_9$ if $P \cap \{q_4, q_5\} = \emptyset$ and $P \not\subseteq \{q_0, q_1, q_2\}$
- $q_0 \to P$ if $P \subseteq \{q_0, q_1, q_2\}$
- $q_3 \to P$ if $P \cap \{q_1, q_4, q_6\} = \emptyset$ and $q_5 \in P$
- $q_8 \to P$ if $P \cap \{q_4, q_5, q_6, q_7\} \neq \emptyset$
- $q_9 \to P$ if $P \cap \{q_4, q_5, q_6, q_7\} \neq \emptyset$ and either $P \cap \{q_4, q_5\} = \emptyset$ or P intersects $\{q_1, q_4, q_6\}$

Figure 3: Automaton for the future Stavi operator

3.3 Automaton for the future Stavi connective ($\mathcal{U}'$)

Let's recall that $\varphi \mathcal{U}' \psi$ holds at position i if there exists a gap $c > i$ such that φ holds at every position $i < j < c$, the property ψ holds at every position in some interval starting at x, and $\neg\varphi$ holds at positions arbitrarily close to c to the right.

The central point in this definition is the gap c, which corresponds to state q_3 in the automaton. States q_0, q_1 and q_2 follow the positions, before q_3, where the formula holds. States q_4, q_5, q_6, q_7, q_8 follow the positions where the formula doesn't hold. If a run reaches q_0, q_1 or q_2, it has to leave this region through q_3, and all successor transitions until then have input label $(1,0)$ or $(1,1)$. The structure of this automaton is depicted in Figure 3. All states except q_3 and q_9 are initial; q_8 and q_9 are final. Transitions from q_1 and q_7 have input label $(1,1)$, transitions from q_2 and q_6 have input label $(1,0)$, transitions from q_4 have input label $(0,0)$, and transitions from q_5 have input label $(0,1)$. The output is 1 for transitions to q_0, q_1 and q_2, and 0 for transitions to q_4, q_5, q_6, q_7 and q_8.

We define a labelling ρ of the cuts of a word w on $\{0,1\}^2$ using the states of the automaton as follows. A cut c is labelled with:

- q_0 if it has no successor, $\varphi \mathcal{U}' \psi$ is true
- q_1 if it is followed by a position labelled $(1,0)$, $\varphi \mathcal{U}' \psi$ is true
- q_2 if it is followed by a position labelled $(1,1)$, $\varphi \mathcal{U}' \psi$ is true
- q_3 if it is a gap, $\varphi \mathcal{U}' \psi$ is true before it and false afterwards
- q_4 if it is followed by a position labelled $(0,0)$, $\varphi \mathcal{U}' \psi$ is false
- q_5 if it is followed by a position labelled $(0,1)$, $\varphi \mathcal{U}' \psi$ is false
- q_6 if it is followed by a position labelled $(1,0)$, $\varphi \mathcal{U}' \psi$ is false
- q_7 if it is followed by a position labelled $(1,1)$, $\varphi \mathcal{U}' \psi$ is false
- q_8 if it has no successor, φ doesn't hold in the left limit if it has no predecessor, and $\varphi \mathcal{U}' \psi$ is false
- q_9 if it is a gap or is the last cut, $\varphi \mathcal{U}' \psi$ is false, and φ is true in some interval to the left

LEMMA 7. *ρ defines the unique run of the automaton on its input word. If the input is $(v_\varphi(w), v_\psi(w))$ for some word w, then the output of this run is $v_{\varphi\mathcal{U}'\psi}(w)$.*

PROOF. We first show that ρ is a run. Successor transitions correspond almost directly to the definitions of the labelling ρ, so let's look at limit transitions. For left limits, the following cases need to be considered:

- if a transition $P \to q_0$ is taken at a cut c, then either φ is true in the limit, and so $\varphi\mathcal{U}'\psi$ is too, and $P \subseteq \{q_0, q_1, q_2\}$, or it's not, and either q_4 or q_5 appear in the limit
- the same reasoning applies for q_1 and q_2
- if c is labelled q_3 then the incoming transition has to come from a subset of $\{q_0, q_1, q_2\}$ since $\varphi\mathcal{U}'\psi$ is true in the limit.
- if a transition $P \to q_4$ is used, then $\varphi\mathcal{U}'\psi$ is not true in the limit (otherwise it would still be true), and so $P \not\subseteq \{q_0, q_1, q_2\}$; the same applies for q_5, q_6, q_7, q_8 and q_9
- if c is a left limit and is labelled q_8 then the incoming transition comes from a set P intersecting $\{q_4, q_5\}$ because $\neg\varphi$ is repeated
- if c is labelled q_9 then q_4 and q_5 can't appear in the left limit set (φ is true)

If c is a right limit cut, it can be labelled q_0, q_3, q_8 or q_9. The right-limit transition can be:

- if c is labelled q_0, the limit transition has to go to a subset of $\{q_0, q_1, q_2\}$ since $\varphi\mathcal{U}'\psi$ holds in the limit;
- if c is labelled with q_3, the limit transition to its right leads necessarily to a set P not including q_1, q_4 and q_6 since ψ is always true, and including q_5 because $\neg\varphi$ is repeated;
- if c is labelled q_8 or q_9, the right limit set can't be a subset of $\{q_0, q_1, q_2\}$ otherwise c would have been labelled q_0;
- if c is labelled q_9 we have the additional condition that either φ holds in the limit (and neither q_4 nor q_5 appears) or ψ doesn't (and one of q_1, q_4 and q_6 is in the limit).

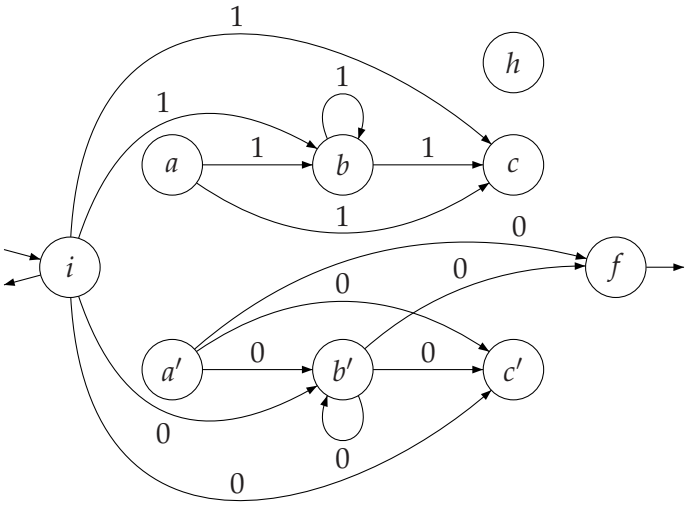
The labelling of cuts defined above is thus a path of the automaton, and we only need to show that it's the only one, using the same method as for the $\mathcal{A}_\mathcal{U}$. Moreover, the definition of ρ means that the output is 1 whenever $\varphi\mathcal{U}'\psi$ holds, and 0 at all other positions. ∎

3.4 Construction of $\mathcal{A}_\varphi$

Now that we have the basic blocks for our construction, we can build an automaton for any formula φ. If φ is an atomic proposition p, we have $\mathcal{A}_p$ as in Section 3.1. If $\varphi = \neg\psi$, then $\mathcal{A}_\varphi = \mathcal{A}_\neg \circ \mathcal{A}_\psi$. If $\varphi = \psi_1 \vee \psi_2$, then $\mathcal{A}_\varphi = \mathcal{A}_\vee \circ (\mathcal{A}_{\psi_1} \times \mathcal{A}_{\psi_2})$. If $\varphi = \psi_1 \mathcal{U} \psi_2$, then $\mathcal{A}_\varphi = \mathcal{A}_\mathcal{U} \circ (\mathcal{A}_{\psi_1} \times \mathcal{A}_{\psi_2})$. The same can be done for $\mathcal{U}'$ and for the past connectives.

The number of states of the resulting automaton is the product of the number of states of all the elementary automata, and is thus exponential in the size of the formula. The actual size of the automaton includes limit transitions, so can be doubly exponential in the size of the formula, if those transitions are represented explicitly.

To check whether the formula φ is satisfiable by a model which is recognized by an automaton $\mathcal{B}$, we can compute the product of the automaton $\mathcal{A}_\varphi$ with $\mathcal{B}$, and check whether a transition where $\mathcal{A}_\varphi$ outputs 1 is accessible and co-accessible. This ensures that there exists a successful run of the product automaton going through that transition, meaning that the corresponding input word is accepted by $\mathcal{B}$ and there is a position where φ holds.



Limit transitions:

$P \to a, h$ if $P \subseteq \{a, b, c, h\}$

$c \to P$ if $P \subseteq \{a, b, c, h\}$

$P \to a', f$ for any P

$c' \to P$ if $P \cap \{a, b, c, h\} = \emptyset$

Figure 4: Automaton checking whether a gap exists in the future

4 Discussion

Logical characterization of automata. We have shown that any LTL, and thus FO, formula can be represented as a non-ambiguous automaton with output. But one can also build such an automaton where the output is the truth word of a property which can't be expressed in the first order. The automaton shown on Figure 4 outputs 1 whenever "there is a gap somewhere in the future" is true; that formula can't be expressed in FO. It would be interesting to find a logical characterisation of the properties that can be expressed using such automata.
Computational complexity. The exact complexity of the satisfiability problem for LTL on arbitrary orderings remains open. We give a 2EXPSPACE procedure to compute an automaton from a formula, whose emptiness can then be checked efficiently. A classical optimization in similar problems is to compute the automaton on the fly, which saves a lot of complexity, so an algorithm using this technique for LTL on arbitrary orderings would be interesting.
Expressive power. On finite and ω-words, LTL restricted to the unary operators ($\mathcal{X}$, $\mathcal{F}$, and their past counterparts) is equivalent to first-order logic restricted to two variables, $\mathrm{FO}^2(<, +1)$ [8]. Restricting even further to $\mathcal{F}$ and its reverse, we get a logic expressively equivalent to $\mathrm{FO}^2(<)$. In the case of finite words, $\mathrm{FO}^2(<)$ corresponds to "partially ordered" two-way automata [18]. The proof of equivalence between unary temporal logic and FO^2 can be easily extended to the case of arbitrary linear orderings. It would be interesting to find such a correspondence for arbitrary orderings as well, and to see if these restrictions provide lower complexity results.
Mosaics technique. In his work on LTL($\mathcal{U}$), Reynolds uses "mosaics" to keep track of the subformulas that need to be satisfied in particular intervals, and to find a decomposition that shows the satisfiability of the initial formula. Unfortunately it is not clear if and how this can be extended to handle a larger fragment of the logic.

5 Conclusion

We investigate linear temporal order with Until, Since, and the Stavi connectives over general linear time, and its relationship with automata over linear orderings. We provide a translation from LTL to a class of non-ambiguous automata with output, giving a 2EXPSPACE

procedure to decide satisfiability of a formula in any rational subclass. This leaves a number of immediate questions, starting with the actual complexity for the satisfiability problem for LTL, but also for some of its fragments, where some operators are excluded. While the full class of automata over linear orderings is not closed under complementation [1], it might still be possible to find a logical characterization for some interesting subclasses.

References

[1] N. BEDON, A. BÈS, O. CARTON, AND C. RISPAL. Logic and rational languages of words indexed by linear orderings. *CSR'08*: 76–85.

[2] V. BRUYÈRE AND O. CARTON. Automata on linear orderings. *MFCS'01*: 236–247.

[3] A. BÈS AND O. CARTON. A Kleene theorem for languages of words indexed by linear orderings. *Int. J. Found. Comput. Sci.*, 17(3):519–542, 2006.

[4] J. R. BÜCHI. Transfinite automata recursions and weak second order theory of ordinals. *Proc. Int. Congress Logic, Methodology, and Philosophy of Science* 2–23, 1965.

[5] O. CARTON. Accessibility in automata on scattered linear orderings. *MFCS'02*: 155–164.

[6] O. CARTON, 2009. Private communication.

[7] S. DEMRI AND A. RABINOVICH. The complexity of temporal logic with until and since over ordinals. *LPAR'07*: 531–545.

[8] K. ETESSAMI, M. Y. VARDI, AND T. WILKE. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.

[9] P. GASTIN AND D. ODDOUX. Fast LTL to Büchi automata translation. *CAV'01*: 53–65.

[10] D. M. GABBAY, A. PNUELI, S. SHELAH, AND J. STAVI. On the temporal basis of fairness. *POPL'80*: 163–173.

[11] H. W. KAMP. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, 1968.

[12] A. PNUELI. The temporal logic of programs. *FOCS'77*: 46–57.

[13] C. RISPAL AND O. CARTON. Complementation of rational sets on countable scattered linear orderings. *Int. J. Found. Comput. Sci.*, 16(4):767–786, 2005.

[14] M. REYNOLDS. The complexity of the temporal logic with "until" over general linear time. *J. Comput. Syst. Sci.*, 66(2):393–426, 2003.

[15] J. G. ROSENSTEIN. *Linear Orderings*. Academic Press, New York, 1982.

[16] A. P. SISTLA AND E. M. CLARKE. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[17] L. J. STOCKMEYER. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, MIT, 1974.

[18] T. SCHWENTICK, D. THÉRIEN, AND H. VOLLMER. Partially-ordered two-way automata: A new characterization of DA. *DLT'01*: 239–250.

[19] M. Y. VARDI AND P. WOLPER. An automata-theoretic approach to automatic program verification (preliminary report). *LICS'86*: 332–344.

[20] P. WOLPER. The tableau method for temporal logic: an overview. *Logique et Analyse*: 28–119, 1985.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Graph Isomorphism for $K_{3,3}$-free and K_5-free graphs is in Log-space

Samir Datta[1], **Prajakta Nimbhorkar**[2], **Thomas Thierauf**[3], **Fabian Wagner**[4]*

[1]Chennai Mathematical Institute
sdatta@cmi.ac.in

[2]The Institute of Mathematical Sciences
prajakta@imsc.res.in

[3]Fak. Elektronik und Informatik, HTW Aalen

[4]Institut für Theoretische Informatik,
Universität Ulm, 89073 Ulm
{thomas.thierauf,fabian.wagner}@uni-ulm.de

ABSTRACT. Graph isomorphism is an important and widely studied computational problem with a yet unsettled complexity. However, the exact complexity is known for isomorphism of various classes of graphs. Recently, [8] proved that planar isomorphism is complete for log-space. We extend this result further to the classes of graphs which exclude $K_{3,3}$ or K_5 as a minor, and give a log-space algorithm.

Our algorithm decomposes $K_{3,3}$ minor-free graphs into biconnected and those further into triconnected components, which are known to be either planar or K_5 components [20]. This gives a triconnected component tree similar to that for planar graphs. An extension of the log-space algorithm of [8] can then be used to decide the isomorphism problem.

For K_5 minor-free graphs, we consider 3-connected components. These are either planar or isomorphic to the four-rung mobius ladder on 8 vertices or, with a further decomposition, one obtains planar 4-connected components [9]. We give an algorithm to get a unique decomposition of K_5 minor-free graphs into bi-, tri- and 4-connected components, and construct trees, accordingly. Since the algorithm of [8] does not deal with four-connected component trees, it needs to be modified in a quite non-trivial way.

1 Introduction

The graph isomorphism problem GI consists of deciding whether there is a bijection between the vertices of two graphs, preserving the adjacencies among vertices. It is an important problem with a yet unknown complexity. The problem is clearly in NP and is also in SPP [2]. It is unlikely to be NP-hard [5, 16], because otherwise the polynomial time hierarchy collapses to the second level. As far as lower bounds are concerned, GI is hard for DET [18], the class of problems NC^1-reducible to the determinant [6].

*Supported by DFG grants TO 200/2-2.

© S. Datta, P. Nimbhorkar, T. Thierauf, F. Wagner; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 145–156
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2314

While this enormous gap has motivated a study of isomorphism in *general* graphs, it has also induced research in isomorphism restricted to special cases of graphs where this gap can be reduced. Tournaments are an example of directed graphs where the DET lower bound is preserved [21], while there is a quasi-polynomial time upper bound [4]. The complexity of isomorphism is settled for trees [11, 12], partial 2-trees [1], and for planar graphs [8]. We extend the result of [8] to isomorphism of $K_{3,3}$ and K_5 minor-free graphs. The previously known upper bound for these graph classes is P due to [14]. Both of these graph classes include planar graphs, and hence are considerably larger than the class of planar graphs.

We consider undirected graphs without parallel edges and loops, also known as *simple* graphs. For directed graphs or graphs with loops and parallel edges, there are log-space many-one reductions to simple undirected graphs (cf. [10]). Our log-space algorithm relies on the following properties of $K_{3,3}$ and K_5 minor-free graphs:

- The 3-connected components of $K_{3,3}$ minor-free graphs are either planar graphs or complete graphs on 5 vertices i.e. K_5's [20].
- The 3-connected components of K_5 minor-free graphs are either planar or V_8's (where V_8 is a four-rung mobius ladder on 8 vertices) or the following holds. The 4-connected components of the remaining non-planar 3-connected components are planar [9].

There is a related result [17] where reachability in $K_{3,3}$ and K_5 minor-free graphs are reduced to reachability in planar graphs under log-space many-one reductions. The basic idea is that the non-planar components are transformed into new planar components. This technique preserves the reachability properties but not the isomorphism. We give a log-space algorithm to get these decompositions in a *canonical* way, and construct the biconnected and triconnected component trees for $K_{3,3}$ minor-free graphs. Then we extend the log-space algorithm of [8] for isomorphism testing and canonization of two such graphs. The isomorphism of K_5 minor-free graphs is more complex, as in addition it has bi-, tri- *and* four-connected component trees. This needs considerable modifications and new ideas.

The rest of the paper is organized as follows: Section 2 gives the necessary definitions and background. Section 3 gives the decomposition of $K_{3,3}$ and K_5 minor-free graphs and proves the uniqueness of such decompositions. In Section 4 we give a log-space algorithm for isomorphism and canonization of $K_{3,3}$ and K_5 minor-free graphs. We omit some proofs due to space constraints and refer to full version of the paper for those proofs.

2 Definitions and Notations

For $U \subseteq V$ let $G \setminus U$ be the *induced subgraph* of G on $V \setminus U$. Let $S \subseteq V$ with $|S| = k$. S is a *k-separating set* if $G \setminus S$ is not connected. The vertices of a k-separating set are called *articulation point (or cut vertex)* for $k = 1$, *separating pair* for $k = 2$, and *separating triple* for $k = 3$. G is *k-connected* if it contains no $(k-1)$-separating set, i.e. there are k vertex-disjoint paths between any pair in G. A 1-connected graph is simply called *connected* and a 2-connected graph *biconnected*. The connected components of $G \setminus S$ are called the *split components of* S.

DEFINITION 1. The biconnected component tree. *We define nodes for the biconnected components and articulation points. There is an edge between a* biconnected component

node *and an* articulation point node *if the articulation point is contained in the corresponding component. The resulting graph is a tree, the* biconnected component tree $\mathcal{T}^{\mathrm{B}}(G)$.

A graph is *triconnected* if it is either 3-connected, a cycle or a 3-bond. A k-bond is a pair of vertices connected by k edges. A separating pair $\{a, b\}$ is called 3-*connected* if there are three vertex-disjoint paths between a and b. In the rest of the paper a separating pair is always considered to be a 3-connected separating pair.

DEFINITION 2. The triconnected component tree. *Define nodes for the triconnected components and (3-connected) separating pairs for a biconnected graph G. There is an edge between a* triconnected component node *and a* separating pair node *if the separating pair is contained in the corresponding component. In a triconnected component, the vertices of a separating pair are connected by a* virtual edge. *If a separating pair is connected in the original graph G then there is a node for a 3-bond connected to the separating pair node. The resulting graph is a tree, the* triconnected component tree $\mathcal{T}^{\mathrm{T}}(G)$.

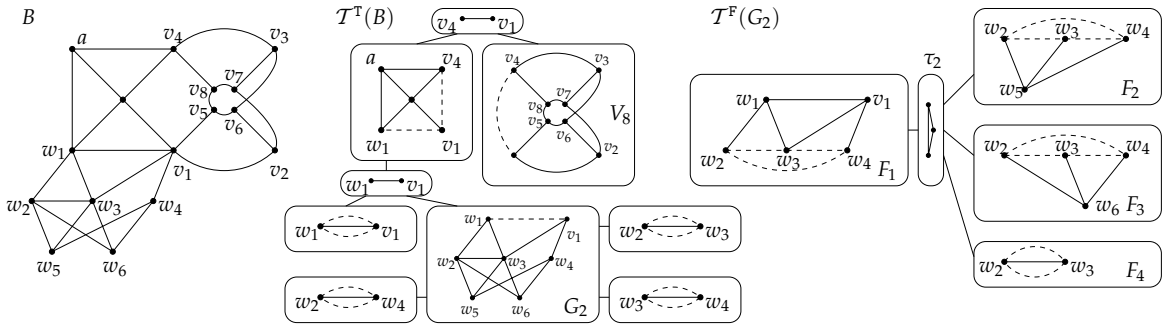


Figure 1: Decomposition of biconnected component B into triconnected components, and G_2 further into four-connected components. Virtual edges are indicated by dashed lines. There is a 3-bond connected to τ_2 because $\{w_2, w_3\}$ is an edge in G_2.

For a component tree T, the *size of an individual component node* C of T is the number of nodes in C. The vertices of the separating sets are counted in in every component where they occur. The *size of the tree* T, denoted by $|T|$, is the sum of the sizes of its component nodes. The size of T is at least as large as the number of vertices in $\mathsf{graph}(T)$, the graph corresponding to the component tree T. Let T_C be T when rooted at node C. A child of C is called a *large child* if $|T_C| > |T|/2$. #C denotes the number of children of C.

A graph H is a minor of a graph G if and only if H can be obtained from G by a finite sequence of edge-removal and edge-contraction operations. A $K_{3,3}$*-free graph* (K_5*-free graph*) is an undirected graph which does not contain a $K_{3,3}$ (or K_5) as a minor.

For two isomorphic graphs we write $G \cong H$. A *canon* for G is a sorted list of edges with renamed vertices $f(G)$, such that for all graphs G, H we have $G \cong H \Leftrightarrow f(G) = f(H)$. We also use *canon* with respect to some fixed starting edge. A *code* of G is the lexicographically sorted list of edges when given an arbitrary labeling of vertices.

By L we denote the languages computable by a log-space bounded Turing machine.

3 Decomposition into triconnected components

3.1 Decomposition of $K_{3,3}$-free graphs

We consider the decomposition of biconnected $K_{3,3}$-free graphs into triconnected components. The decomposition is unique [19] and has the following form.

LEMMA 3. [3] *Each triconnected component of a $K_{3,3}$-free graph is either planar or exactly the graph K_5.*

We state a more general result below, which is used in our decomposition:

LEMMA 4. *In a simple undirected biconnected graph G, the removal of 3-connected separating pairs gives a unique decomposition, irrespective of the order in which they are removed. This decomposition can be computed in log-space.*

Miller and Ramachandran [13] showed that the triconnected component tree of a $K_{3,3}$-free graph can be computed in NC^2. Thierauf and Wagner [17] describe a construction that works in log-space. This now follows from Lemma 4:

COROLLARY 5. *For a biconnected $K_{3,3}$-free graph, the triconnected planar components and K_5 components can be computed in log-space.*

3.2 Decomposition of K_5-free graphs

We decompose the given K_5-free graph into 3-connected and 4-connected components. It follows from a theorem of Wagner [22] that besides planar components we obtain the following non-planar components that way:

- the four-rung Mobius ladder (also called V_8), a 3-connected graph on 8 vertices, which is non-planar because it contains a $K_{3,3}$.
- The remaining 3-connected non-planar components are further decomposed into 4-connected components which are all planar.

Khuller [9] described a decomposition of K_5-free graphs with a clique-sum operation. If two graphs G_1 and G_2 each contain cliques of equal size, the *clique-sum* of G_1 and G_2 is a graph G formed from their disjoint union by identifying pairs of vertices in these two cliques to form a single shared clique, and then possibly deleting some of the clique edges. A *k-clique-sum* is a clique-sum in which both cliques have at most k vertices.

If G can be constructed by repeatedly taking k-clique-sums starting from graphs isomorphic to members of some graph class $\mathcal{G}$, then we say $G \in \langle \mathcal{G} \rangle_k$. The class of K_5-free graphs can be decomposed as follows.

THEOREM 6. [22] *Let $\mathcal{C}$ be the class of all planar graphs together with the four-rung Mobius ladder V_8. Then $\langle \mathcal{C} \rangle_3$ is the class of all graphs with no K_5-minor.*

Theorem 6 and the following observations lead to Corollary 7:

- If we build the 3-clique-sum of two planar graphs, then the three vertices of the joint clique are a separating triple in the resulting graph. Hence, the 4-connected components of a graph which is built as the 3-clique-sum of planar graphs must all be planar.

- The V_8 is non-planar and 3-connected and cannot be part of a 3-clique sum, because it does not contain a triangle as subgraph.

COROLLARY 7.(cf. [9]) *A non-planar 3-connected component of a K_5-free undirected graph is either the V_8 or its 4-connected components are all planar.*

Similar to the decomposition algorithm of Vazirani [20], we decompose the K_5-free graph into triconnected components. That is, we first decompose it into biconnected components and then the biconnected components further into triconnected components.

Unique decomposition of 3-connected K_5-free graphs. Let $G \neq V_8$ be a non-planar 3-connected component of a K_5-free graph, which needs to be decomposed into 4-connected components. The decomposition by [17] is not unique up to isomorphism. Therefore we describe a different way of decomposition. The main idea is to decompose G at only those separating triples which cause the non-planarity.

DEFINITION 8. *Let G be a 3-connected component of a graph G^* and let $\tau \subseteq V(G)$ be a separating triple. Then τ is called* 3-divisive *if in $G^* \setminus \tau$ the component G is split into at least three connected components.*

Intuitively, to see that a 3-divisive separating triple τ causes always non-planarity, collapse the split components of τ to single vertices and a $K_{3,3}$ is obtained. If G is not the $K_{3,3}$ then we can split G at one 3-divisive separating triple and all the other 3-divisive separating triples remain. We prove now that the $K_{3,3}$ is the only special case where this is different.

DEFINITION 9. *Let G be an undirected K_5-free 3-connected graph. Two 3-divisive separating triples $\tau \neq \tau'$ are* conflicting *if τ is no 3-divisive separating triple in a 3-connected component of $(G \setminus \tau') \cup \tau$.*

In general there are no conflicting 3-divisive separating triples except for the $K_{3,3}$. This is important to obtain a decomposition for G which is unique up to isomorphism.

LEMMA 10. *Let G be an undirected and 3-connected graph. There is a conflicting pair of 3-divisive separating triples in G if and only if G is the $K_{3,3}$.*

The four-connected component tree. If we fix one 3-divisive separating triple as root then we get a unique decomposition for G up to isomorphism, also if G is the $K_{3,3}$. We decompose the given graph G at 3-divisive separating triples and obtain *four-connected components*. Two vertices u, v belong to a *four-connected component* if for all 3-divisive separating triples τ the following is true: At least one of u, v belongs to τ *or* there is a path from u to v in $G \setminus \tau$. Note, a four-connected component is planar and 3-connected.

We define a graph with nodes for the four-connected components and 3-divisive separating triples. A *four-connected component node* is connected to a *3-divisive separating triple node* τ if the vertices of τ are also contained in the corresponding four-connected component. The resulting graph is a tree, the *four-connected component tree* $\mathcal{T}^{\mathrm{F}}(G)$. This unique decomposition can be computed in log-space, because every computation step can be queried to the reachability problem in undirected graphs which is in log-space [15].

THEOREM 11. *A unique decomposition of a 3-connected non-planar K_5-free graph (not the V_8) into four-connected components can be computed in log-space.*

The triconnected component tree of K_5-free graphs. From Lemma 4, it follows that the triconnected component tree of a K_5-free graph is log-space computable (also see [8, 17]). For technical reasons we make some changes to this tree structure. Let B be a biconnected K_5-free graph with G_0 a triconnected non-planar component node in $\mathcal{T}^{\mathrm{T}}(B)$. In $\mathcal{T}^{\mathrm{T}}(B)$ there is a separating pair node s for each edge which is part of a 3-divisive separating triple in G. In $\mathcal{T}^{\mathrm{T}}(B)$ the node s is connected to the node G. We call s a *leaf separating pair* of $\mathcal{T}^{\mathrm{T}}(B)$ if it is connected to only one component node. It can be seen that the set of leaf separating pairs can be computed in log-space.

4 Canonization of $K_{3,3}$-free and K_5-free graphs

4.1 Isomorphism ordering and canonization of $K_{3,3}$-free graphs

We decompose $K_{3,3}$-free graphs as in Section 3.1 and extend [8] for K_5-components.

Isomorphism ordering for K_5-components. For a K_5 component we have a node in the triconnected component tree. There are 5! ways of labeling the vertices of a K_5. The first two vertices are from the parent separating pair. There remain $2 \cdot 3! = 12$ ways of labelling the vertices, e.g. $a = 1, b = 2, c = 3, d = 4, e = 5$ is one possibility to label the vertices a, b, c, d, e. The canonical description of the graph is then $(1,2)(1,3),(1,4),(1,5),(2,1),(2,3),\ldots,(5,4)$. The canonical descriptions of all these labelings are candidates for the canon of the K_5. To keep notation short, we say *code* instead of *candidate for a canon*.

For each code, the isomorphism ordering algorithm compares two codes edge-by-edge, thereby going into recursion at child separating pairs and comparing their subtrees. If the subtrees are not isomorphic, the larger code is eliminated. The comparison and the elimination of codes is done similarly as for the planar triconnected components in Datta et.al. [8]. The comparison takes $O(1)$ space on the work-tape to keep counters for the not eliminated codes. The orientation of a K_5-component given to its parent depends on the direction of $\{a, b\}$ in the codes.

CLAIM 12. *Let G_0 be a K_5-node in a triconnected component tree and let $V(G_0) = \{a, b, c, d, e\}$ and (a, b) be the parent separating pair of G_0. Either all minimum remaining codes start with (a, b) (or reverse, (b, a)) or there is an equal number of codes starting with (a, b) and (b, a).*

Once we can canonize K_5-components, we can use the algorithm of [8] to check the isomorphism ordering of triconnected and biconnected component trees.

THEOREM 13. *A $K_{3,3}$-free graph can be canonized in log-space.*

4.2 Isomorphism order of K_5-free graphs

Isomorphism order of K_5-free 3-connected graphs

The isomorphism order of two triconnected component trees S and T rooted at separating pairs $s = \{a, b\}$ and $t = \{a', b'\}$ is defined the same way as for planar graphs in [8] with one

difference. When comparing nodes of the tree we first distinguish between the new types of nodes. We define planar triconnected components $<_T$ V_8-components $<_T$ non-planar 3-connected components. The isomorphism order for planar components is as in [8] and refine it now for the new types of non-planar components.

Isomorphism order of subtrees rooted at V_8-components: Consider S_{G_i} and T_{H_j} rooted at V_8-component nodes G_i and H_j. We construct the codes of G_i and H_j and compare them bit-by-bit. To canonize the V_8-components, we traverse it starting from the parent separating pairs $\{a, b\}$ and $\{a', b'\}$ and then traversing the components as follows. We define the codes with help of Hamiltonian cycles of the V_8-component. We define E' to be the set of edges which are contained in four Hamiltonian cycles.

LEMMA 14. *Each directed edge of a V_8 appears in two or four Hamiltonian cycles.*

Basically, there are two possible traversals of each Hamiltonian cycle starting from $\{a, b\}$, one in each direction. We define the code for G_i and the starting edge (a, b) in this direction as follows. We distinguish the situation whether $\{a, b\} \in E'$ or not. We will fix one Hamiltonian cycle starting with (a, b). We rename the vertices in that order of their first occurrence in the fixed Hamiltonian cycle exactly in that order. The code is then the list of edges in lexicographical order with the new labels.

Isomorphism order of the 3-connected non-planar components

Let S_{G_i} and T_{H_j} be trees rooted at 3-connected non-planar component nodes G_i and H_j which are different to the V_8. Let $s = \{a, b\}$ and $t = \{a', b'\}$ be the parent separating pairs of G_i and H_j, respectively. We are interested in the orientation given to s and t. After this, we discuss the comparison algorithm of G_i with H_j. We further partition G_i and H_j into their four-connected components and consider their trees $\mathcal{T}^{\mathtt{F}}(G_i)$ and $\mathcal{T}^{\mathtt{F}}(H_j)$.

Overview of the steps in the isomorphism order.

DEFINITION 15. *For a four-connected component tree T, the* size *of an individual component node C of T is the number n_C of vertices in C. The separating triple nodes are counted in every component where they occur. The* size *of the tree T, denoted by $|T|$, is the sum of the sizes of its component nodes.*

The isomorphism order of two four-connected component trees S and T rooted at 3-divisive separating triples τ and τ' where given an order $\mathsf{order}(\tau)$ and $\mathsf{order}(\tau')$ is defined $S_\tau \leq_{\mathtt{F}} T_{\tau'}$ if:

1. $|S_\tau| < |T_{\tau'}|$ or
2. $|S_\tau| = |T_{\tau'}|$ but $\#\tau < \#\tau'$ or
3. $|S_\tau| = |T_{\tau'}|$, $\#\tau = \#\tau' = k$, but $(S_{F_1}, \ldots, S_{F_k}) <_{\mathtt{F}} (T_{F'_1}, \ldots, T_{F'_k})$ lexicographically, where we assume that $S_{F_1} \leq_{\mathtt{F}} \ldots \leq_{\mathtt{T}} S_{F_k}$ and $T_{F'_1} \leq_{\mathtt{T}} \ldots \leq_{\mathtt{T}} T_{F'_k}$ are the ordered subtrees of S_τ and $T_{\tau'}$, respectively. For the isomorphism order between the subtrees S_{F_i} and $T_{F'_i}$ we compare lexicographically the codes of F_i and F'_i and *recursively* the subtrees rooted at the children of F_i and F'_i. Note, that these children are again separating triple nodes.
4. $|S_\tau| = |T_{\tau'}|$, $\#\tau = \#\tau' = k$, $(S_{F_1} \leq_{\mathtt{F}} \ldots \leq_{\mathtt{F}} S_{F_k}) =_{\mathtt{F}} (T_{F'_1} \leq_{\mathtt{F}} \ldots \leq_{\mathtt{F}} T_{F'_k})$, but the following holds. For all i, the return value from the recursion of S_{F_i} with $T_{F'_i}$ is an *orientation*

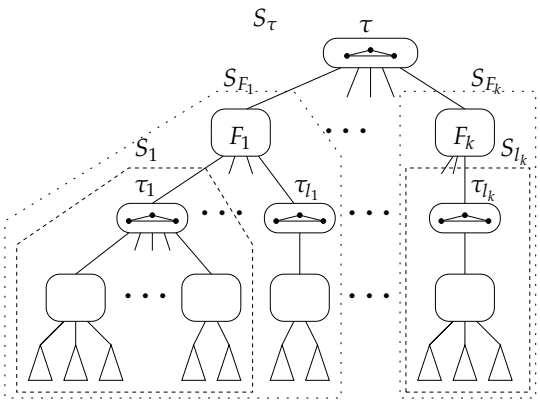
graph X_i and X'_i with $V(X_i) = \tau$ and $V(X'_i) = \tau'$ and colored edges, respectively. We compute a *reference orientation graph* X and X' from all the X_i and X'_i. We compare lexicographically whether X with $\mathsf{order}(\tau) < X'$ with $\mathsf{order}(\tau')$. We describe the notion of (reference) orientation graph and $\mathsf{order}(\tau)$ below in more detail.

We say that two four-connected component trees S_τ and $T_{\tau'}$ are *equal according to the isomorphism order*, denoted by $S_\tau =_{\mathrm{F}} T_{\tau'}$, if neither $S_\tau <_{\mathrm{F}} T_{\tau'}$ nor $T_{\tau'} <_{\mathrm{F}} S_\tau$ holds.

Orientation given to the parent separating pair by a non-planar component, not the V_8. Given two non-planar 3-connected components G_i and H_j and their trees $\mathcal{T}^{\mathrm{F}}(G_i)$ and $\mathcal{T}^{\mathrm{F}}(H_j)$. There is a set of candidates for root separating triples such that we obtain the minimum codes when the trees are rooted at them. For the canonization algorithm, the isomorphism ordering algorithm is used as a sub-routine. For the isomorphism ordering procedure, we give distinct colors to the parent separating pair and the parent articulation point in the trees. We also have colors for the child separating pairs and child articulation points, according to their isomorphism order. We recompute these colors by interrupting the current isomorphism ordering procedure and going into recursion at the corresponding subtrees.

Finally, we just consider the first occurrence of the parent separating pair in all the minimum codes. If the first occurrence is (a, b) in this direction in all the codes, then G_i gives this orientation to the parent. Similarly for (b, a). If both (a, b) and (b, a) occur first in different minimum codes, then there is no orientation given to the parent.

Isomorphism order of four-connected component trees. We describe what is different between isomorphism ordering for four-connected and triconnected component trees in Section 4.2 (also see [8]). Instead of separating pairs we have 3-divisive separating triples. In the isomorphism order algorithm for two triconnected component trees there was one task, where the orientations of the separating pairs were compared. An orientation of a pair $\{a, b\}$ in a triconnected component tree $T_{\{a,b\}}$ is the set of permutations which partially map $T_{\{a,b\}}$ to its canon. This is a subgroup of the symmetric group $Sym(\{a, b\})$. For a four-connected component tree S_τ, we consider the set of permutations of the triple $\tau = \{a, b, c\}$. This set contains all partial automorphisms which partially map S_τ to its canon. This is a subgroup of the symmetric group $Sym(\{a, b, c\})$. Instead of 3-connected planar components we have four-connected planar components in S_τ and $T_{\tau'}$.



Isomorphism order of two subtrees rooted at four-connected component nodes. We consider the isomorphism order of two subtrees S_{F_i} and $T_{F'_j}$ rooted at four-connected component nodes F_i and F'_j, respectively. We construct the codes of F_i and F'_j and compare them bit-by-bit. To canonize F_i, we use the log-space algorithm from [7]. Besides F_i, the algorithm gets as input a starting edge and a combinatorial embedding ρ of F_i. There are three choices of selecting a starting edge $\{a, b\}, \{b, c\}, \{a, c\}$ and two choices for the direction of each edge, e.g. for $\{a, b\}$ we have (a, b) and (b, a). Further, a 3-connected planar graph has two planar

combinatorial embeddings [23]. There are 12 possible ways to canonize G_i.

We start the canonization of G_i and H_j in all the possible ways and compare these codes bit-by-bit. Let C and C' be two codes to be compared. The base case is that F_i and F'_j are leaf nodes and therefore contain no further virtual edges. In this case we use the lexicographic order between C and C'. If G_i and H_j contain a virtual edge, then this belongs to a child separating triple and is treated in a special way when comparing C and C':

First, if a virtual edge is traversed in the codes C or C' but not in the other, then we define the one without the virtual edge to be the *smaller code.*

Second, if C and C' encounter the virtual edges $\{u, v\}$ and $\{u', v'\}$ then we consider only the child separating triples which do not have virtual edges considered earlier in the codes C and C'. We order these triples according to the positions of all their virtual edges in the codes. We call this order the *position-order*. W.l.o.g. let τ_{i_0} (in C) and τ'_{j_0} (in C') be the separating triples which come first in this position-order. For τ_{i_0} and τ'_{j_0}, we will define below the *reference orientation graphs* X and X' with $V(X) = \tau_{i_0}$ and $V(X') = \tau'_{j_0}$, respectively. For all pairs in $\tau_{i_0} = \{u, v, w\}$ and $\tau'_{j_0} = \{u', v', w'\}$ we have virtual edges in C and C'. We compare X and X' with respect to the ordering of these virtual edges $\mathsf{order}(\tau_{i_0})$ and $\mathsf{order}(\tau'_{j_0})$ in the codes C and C', respectively. This is described below in more detail. If we find an inequality, say $X < X'$ then C is defined to be the *smaller code*. Proceed with the next separating triples in the position-order until we ran through all of them.

We eliminate the codes which were found to be the larger codes at least once. In the end, the codes that are not eliminated are the *minimum codes*. If we have minimum codes for both F_i and F'_j then we define $S_{F_i} =_{\mathtt{F}} T_{F'_j}$. The construction of the codes also defines an isomorphism between the subgraphs associated to S_{F_i} and $T_{F'_j}$, i.e. $\mathsf{graph}(S_{F_i}) \cong \mathsf{graph}(T_{F'_j})$. For a single four-connected component this follows from [7]. If the trees contain several components, then our definition of $S_{F_i} =_{\mathtt{F}} T_{F'_j}$ guarantees that we can combine the isomorphisms of the components to an isomorphism between $\mathsf{graph}(S_{F_i})$ and $\mathsf{graph}(T_{F'_j})$.

Finally, we define the *orientation given to the parent separating triple of* F_i and F'_j as follows.

- We compute an *orientation graph* X_i with $V(X_i) = \tau$.
- For each pair in τ when taken as starting edge for the canonization of S_{F_i} which leads to a minimum code (among all the codes for these edges) we have a directed edge in $E(X_i)$ with color (1).
- Also for the r-th minimum codes we have a directed edge in $E(X_i)$ with color (r), for all $1 \leq r \leq 6$. Here, 6 is the number of directed edges in τ.

We define a new graph X_i with $V(X_i) = \tau$ and X'_j with $V(X'_j) = \tau'$. For each of the remaining minimum codes we have a unique starting edge which is also contained as a directed edge in X_i or X'_j, respectively. Every subtree rooted at a four-connected component node gives an orientation graph to the parent separating triple. If the orientation is consistent, then we define $S_\tau =_{\mathtt{F}} T_{\tau'}$ and show that the corresponding graphs are isomorphic.

Isomorphism order of two subtrees rooted at separating triple nodes. We consider the subtrees $S_{F_1}, \ldots, S_{F_k}$ and $T_{F'_1}, \ldots, T_{F'_k}$. We order them $S_{F_1} \leq_{\mathtt{F}} \cdots \leq_{\mathtt{F}} S_{F_k}$ and $T_{F'_1} \leq_{\mathtt{F}} \cdots \leq_{\mathtt{F}} T_{F'_k}$, and verify that $S_{F_i} =_{\mathtt{F}} T_{F'_i}$ for all i. If we find an inequality then the one with the smallest index i defines the order between S_τ and $T_{\tau'}$. For all i, assume now that $S_{F_i} =_{\mathtt{F}}$

$T_{F_i'}$ and, inductively, the corresponding split components are isomorphic, i.e. $\mathsf{graph}(S_{F_i}) \cong \mathsf{graph}(T_{F_i'})$. The next comparison concerns the orientation of τ and τ'. We already explained above the orientation given by each of the S_{F_i}'s to τ. We define a *reference orientation* for the root nodes τ and τ' which is given by their children. This is done as follows. We partition $(S_{F_1}, \ldots, S_{F_k})$ into classes of isomorphic subtrees, say $I_1 <_{\mathtt{F}} \ldots <_{\mathtt{F}} I_p$ for some $p \leq k$, and similarly we partition $(T_{F_1'}, \ldots, T_{F_k'})$ into $I_1' <_{\mathtt{F}} \ldots <_{\mathtt{F}} I_p'$. It follows that I_j and I_j' contain the same number of subtrees for every j.

Consider the orientation given to τ by an isomorphism class I_j: For each child F_i which belongs to I_j we compute an *orientation graph* X_i with vertices $V(X_i) = \tau$. The orientation graph is defined as above but with the following changes. Instead of colors $(1), \ldots, (6)$ we have the colors $(j,1), \ldots, (j,6)$ for the edges. The *reference orientation given to* τ is defined as follows. We define the orientation graph X with vertices $V(X) = \tau$ and edges $E(X) = \bigcup_{1 \leq i \leq k} E(X_i)$ the disjoint union of the edges of the orientation graphs from all children of τ. Thus, X has multiple edges. We call X the *reference orientation graph for* τ.

Comparison of two orientation graphs. For τ and τ', the isomorphism ordering algorithm compares X and X' for isomorphism. Assume now τ and τ' have isomorphic subtrees and the nodes F and F' as parents. In this situation we return from recursion with $=_{\mathtt{F}}$ and give the orientation graphs X and X' to the parent. We went into recursion because the virtual edges of τ and τ' appeared in the same positions in the codes of the parent. In these codes, we have a complete ordering on the vertices of τ and τ'. Let $V(X) = \{u, v, w\}$ and let $\mathsf{order}(\tau) = u < v < w$ be an ordering of τ. We compute a list of counters for $(X, \mathsf{order}(\tau))$:

- We order the edges of X according to the order of their vertices, lexicographically. That is, $(u,v) < (u,w) < (v,u) < (v,w) < (w,u) < (w,v)$.
- Among directed edges with the same ends, we order them according to their color. That is, an edge with color (i_1, i_2) comes before an edge with color (j_1, j_2) if $(i_1, i_2) < (j_1, j_2)$ lexicographically.
- We define a counter for the number of edges with the same ends and the same color. For the edge (u,v) we have the counters $c_{(u,v),1}, \ldots, c_{(u,v),6p}$. Note, we have at most $6p$ colors because there are 6 colors for edges from orientation graphs of one isomorphism class and there are p isomorphism classes.
- We order the counters according to the order of the edges. That is, we have a list of counters $L(X, \mathsf{order}(\tau)) = (c_{(u,v),1}, c_{(u,v),2}, \ldots, c_{(u,v),6p}, \ldots, c_{(w,v),1}, \ldots, c_{(w,v),6p})$.

Note, among isomorphic graphs, there must be edges having the same color up to a permutation of them. Counting the colored edges allows to combine the orientations of all isomorphic subtrees. Note, if an orientation graph X_i for τ has two equal colored edges then there is an automorphism that maps the one edge to the other same colored edge in τ. The permutation of one edge completely fixes the whole automorphism of τ. Hence, also when counting the edges from different orientation graphs X_1 and X_2, if w.l.o.g. there are the edges (u,v) with colors 1 and 2 then the mapping of (u,v) to other edges completely fixes the whole automorphism among τ and whether X_1 and X_2 are swapped. With an inductive argument, this can be generalized to the whole orientation graph X. Let X' be the corresponding reference orientation graph for τ'. We define the isomorphism order $(X, \mathsf{order}(\tau)) < (X', \mathsf{order}(\tau'))$ exactly when $L(X, \mathsf{order}(\tau)) < L(X', \mathsf{order}(\tau'))$ lexicographically. The preceding discussion leads to the following theorem.

THEOREM 16. *The 3-connected non-planar graphs G and H which contained 3-divisive separating triples are isomorphic if and only if there is a choice of separating triples τ, τ' in G and H such that $S_\tau =_F T_{\tau'}$ when rooted at τ and τ', respectively.*

4.3 Complexity of the isomorphism order algorithm and canonization

For a log-space implementation, there are two tasks: We limit the number of choices for roots of triconnected and four-connected component trees, and ensure that nothing is stored on the work-tape while recursing on a large child. For the first task, we modify the algorithm of [8] to accommodate non-planar 3-connected components. The two root finding procedures are interdependent. We bound the number of roots for the four-connected component tree with respect to the number of child separating pairs and child articulation points of tri- and biconnected subtrees. The algorithm is based on an intricate case analysis which has to be extended to work with respect to the three tree structures.

For the second task, we extend the ideas from [8], because for the analysis of large children we consider the bi- tri- and four-connected component trees simultaneously. In the trees, the recursion goes from depth d to $d+2$ and large children are handled a priori at any level.For the space requirement of our algorithm we get:

$$\mathcal{S}(N) = \max_j \mathcal{S}\left(\tfrac{N}{k_j}\right) + O(\log k_j),$$

where $k_j \geq 2$ (for all j) is the number of bi-, tri- or four-connected subtrees having the same size. Hence, $\mathcal{S}(N) = O(\log N)$. It is helpful to imagine that we have three work-tapes, which are used when we go into recursion at articulation point nodes, separating pair nodes, and separating triple nodes respectively. We canonize K_5-free graphs exactly the same way as planar graphs. Thus we get

THEOREM 17. *The isomorphism order between two K_5-free graphs can be computed in log-space. The canonization of K_5-free graphs can be done in log-space.*

Acknowledgment We thank V. Arvind, Bireswar Das, Raghav Kulkarni, Nutan Limaye, Meena Mahajan and Jacobo Torán for helpful discussions.

References

[1] V. Arvind, Bireswar Das, and Johannes Köbler. A logspace algorithm for partial 2-tree canonization. In *Computer Science Symposium in Russia (CSR)*, 2008.

[2] V. Arvind and Piyush P. Kurur. Graph isomorphism is in SPP. *Information and Computation*, 204(5), 2006.

[3] Tetsuo Asano. An approach to the subgraph homeomorphism problem. *Theoretical Computer Science*, 38, 1985.

[4] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the 15th annual ACM symposium on Theory of computing (STOC)*, 1983.

[5] R. B. Boppana, J. Hastad, and S. Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 25(2), 1987.

[6] Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1-3), 1985.

[7] Samir Datta, Nutan Limaye, and Prajakta Nimbhorkar. 3-connected planar graph isomorphism is in log-space. In *Proceedings of the 28th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2008.

[8] Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. Technical Report TR09-052, Electronic Colloquium on Computational Complexity (ECCC), 2009.

[9] Samir Khuller. Parallel algorithms for K_5-minor free graphs. Technical Report TR88-909, Cornell University, Computer Science Department, 1988.

[10] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The Graph Isomorphism Problem*. Birkhäuser, 1993.

[11] Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proceedings of the 24th annual ACM Symposium on Theory of Computing (STOC)*, 1992.

[12] Pierre McKenzie, Birgit Jenner, and Jacobo Torán. A note on the hardness of tree isomorphism. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity (CCC)*. IEEE Computer Society, 1998.

[13] Gary L. Miller and V. Ramachandran. A new graph triconnectivity algorithm and its parallelization. In ACM, editor, *Proceedings of the nineteenth annual ACM Symposium on Theory of Computing*, pages 335–344, 1987.

[14] Ilia N. Ponomarenko. The isomorphism problem for classes of graphs closed under contraction. *Journal of Mathematical Sciences (JSM, formerly Journal of Soviet Mathematics)*, 55, 1991.

[15] Omer Reingold. Undirected st-connectivity in log-space. In *Proceedings of the 37th annual ACM Symposium on Theory of Computing (STOC)*, 2005.

[16] Uwe Schöning. Graph isomorphism is in the low hierarchy. *Journal on Computing and System Sciences*, 37(3), 1988.

[17] Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free graphs and K_5-free graphs is in unambiguous log-space. Technical Report TR09-029, Electronic Colloquium on Computational Complexity (ECCC), 2009.

[18] Jacobo Torán. On the hardness of graph isomorphism. *SIAM Journal on Computing*, 33(5), 2004.

[19] William T. Tutte. *Connectivity in graphs*. University of Toronto Press, 1966.

[20] Vijay V Vazirani. NC algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. *Information and Computation*, 80, 1989.

[21] Fabian Wagner. Hardness results for tournament isomorphism and automorphism. In *32nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2007.

[22] Klaus Wagner. Über eine Eigenschaft der ebenen Komplexe. In *Mathematical Annalen*, volume 114, 1937.

[23] Hassler Whitney. A set of topological invariants for graphs. *American Journal of Mathematics*, 55, 1933.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Domination Problems in Nowhere-Dense Classes of Graphs

Anuj Dawar[1], **Stephan Kreutzer**[2]

[1]University of Cambridge Computer Lab, U.K.
anuj.dawar@cl.cam.ac.uk

[2]Oxford University Computing Laboratory, U.K.
kreutzer@comlab.ox.ac.uk

ABSTRACT. We investigate the parameterized complexity of generalisations and variations of the dominating set problem on classes of graphs that are nowhere dense. In particular, we show that the distance-d dominating-set problem, also known as the (k,d)-centres problem, is fixed-parameter tractable on any class that is nowhere dense and closed under induced subgraphs. This generalises known results about the dominating set problem on H-minor free classes, classes with locally excluded minors and classes of graphs of bounded expansion. A key feature of our proof is that it is based simply on the fact that these graph classes are uniformly quasi-wide, and does not rely on a structural decomposition. Our result also establishes that the distance-d dominating-set problem is FPT on classes of bounded expansion, answering a question of Nešetřil and Ossona de Mendez.

1 Introduction

The dominating-set problem is among the most well-studied problems in algorithmic graph theory and complexity theory. Given a graph G and an integer k, we are asked to determine whether G contains a set X of at most k vertices such that every vertex of G is either in X or adjacent to a vertex in X. This is a classical NP-complete problem that has been intensively studied from the point of view of approximation algorithms and fixed-parameter tractability. A number of generalisations and variations of the dominating set problem have also been studied in this context. In particular, the *distance-d dominating-set problem* is one where we are given a graph G and integer parameters d and k and we are to determine whether G contains a set X of at most k vertices such that every vertex in G has distance at most d to a vertex in X. This problem, also known as the (k,d)-centre problem, has for instance been studied in [5] in connection with network centres and other clustering problems (see the references in [10]). It is clear that in the case $d = 1$, this is just the dominating set problem. A number of other domination problems are considered in Section 5.

We are interested in investigating these problems from the point of view of fixed-parameter tractability. That is we are interested in algorithms for these problems that run in time $f(k) \cdot n^c$ (or $f(k+d) \cdot n^c$ in the case of the distance-d dominating-set problem) where n is the order of the graph G, c is independent of the parameters k and d and f is any computable function. Such algorithms are unlikely to exist in general, since the dominating-set problem is $W[2]$-complete (see [13, 15] for a general introduction to parameterized complexity, including definitions of FPT and $W[2]$). However, if we restrict the class of graphs under

© Dawar, Kreutzer; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 157–168
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2315

consideration, we can obtain efficient algorithms in the sense of fixed-parameter tractability, even though the problem remains NP-complete on the restricted class. We are interested in knowing how general we can make our graph classes while retaining fixed-parameter tractability. In this paper, we push the tractability frontier forward by showing that the distance-d dominating-set problem as well as a number of other domination problems, are FPT on *nowhere-dense* classes of graphs. This generalises known results about the dominating set problem on H-minor free graphs, classes of graphs of *bounded expansion* and classes with *locally excluded minors*. Moreover, while the latter results relied heavily on graph structure theory, our proof depends on a simple combinatorial property of nowhere-dense classes and thus affords a great simplification to the proof. In the sequel, we will use the term *efficient algorithm* always to mean efficient in the sense of fixed-parameter tractability.

Classes on which efficient algorithms have previously been obtained for the dominating set problem include planar graphs where an algorithm with running time $\mathcal{O}(8^k n)$-time is given in [2] and graphs of genus g, where an $\mathcal{O}((4g+40)^k n^2)$-time algorithm is given in [14]. Improvements to the algorithms on planar graphs have subsequently been made, to $\mathcal{O}(4^{\sqrt[6]{34k}} n)$ in [1], to $\mathcal{O}(2^{27\sqrt{k}} n)$ in [18] and to $\mathcal{O}(2^{15.13\sqrt{k}} k + n^3 + k^4)$ in [16]. Efficient algorithms for distance-d dominating sets are also known for planar graphs and map-graphs [10]. For the dominating set problem, efficient algorithms were shown for H-minor free graphs in [11]. The latter generalises the result for classes of graphs of bounded genus. More recently, Alon and Gutner gave a linear time parameterized algorithm for dominating sets on d-degenerate graphs running in time $k^{\mathcal{O}(dk)} n$ [3]. This is a further generalisation beyond H-minor-free classes. Another generalisation is obtained by Philip et al. [23] who show that the problem is fixed-parameter tractable on graphs that exclude $K_{i,j}$ as a subgraph. It should be noted that while all other classes mentioned above also admit an efficient algorithm for the distance-d dominating-set problem, this is not the case for classes of degenerate graphs. Indeed, this problem is $W[2]$-hard, already on the class of 2-degenerate graphs [17].

Other generalisations of H-minor free classes that have been considered in the literature are classes with locally excluded minors [9] and classes of bounded expansion [20]. For the former, it follows from [9] that the distance-d dominating-set problem is FPT. This is because the problem can be specified by a first-order formula (depending on d and k), and any property so specified is FPT on classes that locally exclude a minor. For classes of bounded expansion, Nešetřil and Ossona de Mendez [22] show that the dominating set problem is solvable in fixed-parameter linear time, while the question of whether the distance-d dominating-set problem is FPT on such classes is left open. Indeed, they point out that their method cannot be used to show that the distance-2 dominating-set problem is FPT on classes of bounded expansion. Our result settles this question as it implies the existence of an efficient algorithm for distance-d dominating-set on bounded-expansion classes.

Our main results concern classes of graphs that are *nowhere dense*. This is a concept introduced by Nešetřil and Ossona de Mendez [19, 21] that generalises both locally excluded minors and bounded expansion in the sense that any class of graphs that either locally excludes a minor or has bounded expansion is also nowhere dense. Nešetřil and Ossona de Mendez show that nowhere-dense classes can be characterised by the property of being *uniformly quasi-wide* (see Section 2 for the defintions). The latter is a property introduced by Dawar [7, 8] in the study of preservation theorems in finite model theory. In the present pa-

per we show that this property is by itself sufficient to establish that a class of graphs admits an efficient parameterized algorithm for distance-d dominating set. The great advantage here is that this is a combinatorial property that is easy to state and yields a transparently simple algorithm. This should be contrasted with the algorithms [10, 11] on H-minor free graphs that heavily rely on graph structure theory.

We begin by establishing some basic terminology and notation in Section 2, and introduce nowhere-dense classes and uniformly quasi-wide classes of graphs. In Section 3 we examine the relationship between these two notions and extract the algorithmic content of the equivalence between them. This allows us, in Section 4, to exhibit an efficient parameterized algorithm for the distance-d dominating set problem on nowhere dense classes. In Section 5, we explain how the same ideas can be carried over to a number of other parameterized problems that are defined in terms of domination in graphs.

2 Preliminaries

For a graph G and vertices $u, v \in V(G)$, we write $\text{dist}^G(u, v)$ for the distance (i.e. the length of the shortest path) from u to v. We write $N_d^G(v)$ for the *d-neighbourhood* of v, i.e. the set of vertices u in $V(G)$ with $\text{dist}^G(u, v) \leq d$. Where the meaning is clear from context, we drop the superscript G. For positive integers $i < j$, we write $[i, j]$ for the set $\{k : i \leq k \leq j\}$.

For a graph G and a set of vertices $X \subseteq V(G)$, we write $G - X$ for $G[V(G) \setminus X]$, i.e. the subgraph of G induced by the vertices $V(G) \setminus X$.

DEFINITION 1. *Let G be a graph and $d \in \mathbb{N}$.*

1. *A set $X \subseteq V(G)$ is* d-scattered *if for $u \neq v \in X$, $N_d(u) \cap N_d(v) = \varnothing$.*
2. *A set $X \subseteq V(G)$* d-dominates *a set $W \subseteq V(G)$ if for all $u \in W$ there is a $v \in X$ such that $u \in N_d(v)$.*
3. *A set $X \subseteq V(G)$ is a* d-dominating set *if it d-dominates $V(G)$.*

We say that a graph H is a *minor* of G (written $H \preceq G$) if H can be obtained from a subgraph of G by contracting edges. An equivalent characterization (see [12]) states that H is a minor of G if there is a map that associates to each vertex v of H a non-empty *connected* subgraph G_v of G such that G_u and G_v are disjoint for $u \neq v$ and whenever there is an edge between u and v in H there is an edge in G between some node in G_u and some node in G_v. The subgraphs G_v are called *branch sets*.

We say that H is a *minor at depth r* of G (and write $H \preceq_r G$) if H is a minor of G and this is witnessed by a collection of branch sets $\{G_v \mid v \in V(H)\}$, each of which induces a graph of radius at most r. That is, for each $v \in V(H)$, there is a $w \in V(G_v)$ such that $G_v \subseteq N_r^{G_v}(w)$.

The definition of nowhere-dense classes is due to Nešetřil and Ossona de Mendez [19, 21]. We are particularly interested in classes where the excluded minors are computable and for this purpose we introduce the notion of *effectively nowhere-dense* classes.

DEFINITION 2. *[nowhere dense classes] A class of graphs $\mathcal{C}$ is said to be* nowhere dense *if for every $r \geq 0$ there is a graph H_r such that $H_r \not\preceq_r G$ for all $G \in \mathcal{C}$. We say $\mathcal{C}$ is* effectively nowhere dense *if the map $r \mapsto H_r$ is computable.*

It is immediate from the definition that if $\mathcal{C}$ excludes a minor then it is nowhere dense. It is also not difficult to show that classes of bounded expansion and classes that locally

exclude minors are also nowhere dense. Nešetřil and Ossona de Mendez [19] show an interesting relationship between nowhere dense classes and a property of classes of structures introduced by Dawar [7, 8] called *quasi-wideness*. Again, we are interested in effective versions of this concept.

DEFINITION 3.*[quasi-wide classes] Let $s : \mathbb{N} \to \mathbb{N}$ be a function. A class $\mathcal{C}$ of graphs is* quasi-wide with margin s *if for all $r \geq 0$ and $m \geq 0$ there exists an $N \geq 0$ such that if $G \in \mathcal{C}$ and $|G| > N$ then there is a set $S \subseteq V(G)$ with $|S| < s(r)$ such that $G - S$ contains an r-scattered set of size at least m.*

We say that $\mathcal{C}$ is quasi-wide *if there is some s such that $\mathcal{C}$ is quasi-wide with margin s. We say that $\mathcal{C}$ is* effectively quasi-wide *if s and $N(r, m)$ are computable.*

We occasionally refer to a set S as in the above definition as a *bottleneck* of G.

It turns out that if $\mathcal{C}$ is closed under taking induced subgraphs, then it is nowhere dense if, and only if, it is quasi-wide. For such classes, quasi-wideness is equivalent to the notion of uniform quasi-wideness defined below, which is the notion we will use in the present paper.

DEFINITION 4.*[uniformly quasi-wide classes] A class $\mathcal{C}$ of graphs is* uniformly quasi-wide *with margin s if for all $r \geq 0$ and all $m \geq 0$ there exists an $N \geq 0$ such that if $G \in \mathcal{C}$ and $W \subseteq V(G)$ with $|W| > N$ then there is a set $S \subseteq V(G)$ with $|S| < s(r)$ such that W contains an r-scattered set of size at least m in $G - S$. $\mathcal{C}$ is* effectively uniformly quasi-wide *if s and $N(r, m)$ are computable.*

We often write $s_{\mathcal{C}}$ for the margin of the class $\mathcal{C}$, and $N_{\mathcal{C}}(r, m)$ for the value of N it guarantees for this class. We are only interested in classes $\mathcal{C}$ for which these functions are *computable*, and we tacitly make this assumption from now on. We can now state the equivalence of the two central notions.

THEOREM 5.*[19] Any class $\mathcal{C}$ of graphs that is closed under taking induced subgraphs is quasi-wide if, and only if, it is nowhere dense.*

In Section 3, we will exhibit the algorithmic content of this equivalence by showing that in any nowhere-dense class, there is an efficient (in the sense of fixed-parameter tractability) algorithm that can find the bottleneck S and the scattered set induced by its removal. In particular this implies that a class $\mathcal{C}$ closed under subgraphs is effectively nowhere dense if, and only if, it is effectively quasi-wide. We end this section with some examples of quasi-wide classes.

EXAMPLE 6.

1. Bounded-degree graphs. *The class of graphs $\mathcal{D}_d$ of valence at most d is quasi-wide with margin 1 and $N_{\mathcal{D}_d}(r, m) = (d-1)^r + d + 1$.*
2. H-minor free graphs. *The class of graphs excluding H as a minor is quasi-wide with margin $|H| - 1$.*

3 Computing Bottlenecks and Scattered Sets

In this section, our aim is to extract the computational content of Theorem 5 stating the equivalence between nowhere dense classes and uniformly quasi-wide classes. In particu-

lar, we show that in any nowhere dense class $\mathcal{C}$, we can efficiently extract bottlenecks and scattered sets in any graph.

The first step is to show that in any uniformly quasi-wide class with margin s, we can compute, from $s(r)$ and $N_{\mathcal{C}}$, a bound on the order of the graphs that are excluded as minors of depth r.

LEMMA 7. *If $\mathcal{C}$ is a uniformly quasi-wide class with margin s and $h > N_{\mathcal{C}}(r+1, s(r+1)+1)$, then $K_h \not\preceq_r G$ for any $G \in \mathcal{C}$.*

PROOF. Suppose, for contradiction, that $K_h \preceq_r G$ and let $u_1, \ldots, u_h$ be such that the neighbourhoods $N_r^G(u_i)$ contain branch sets $G_1, \ldots, G_h$ witnessing this. Then, by the choice of h and the definition of quasi-wideness, there is a set $S \subseteq V(G)$ with $|S| < s(r+1)$ such that $\{u_1, \ldots, u_h\}$ contains an $r+1$-scattered set A of size $s(r+1)+1$ in $G - S$. Thus, since the branch sets are pairwise disjoint, there must be two distinct vertices $u_i, u_j \in A$ such that $S \cap G_i = S \cap G_j = \varnothing$. There is an edge between some vertex in G_i and some vertex in G_j (since they are branch sets witnessing $K_h \preceq_r G$). We thus have that $N_{r+1}(u_i) \cap N_{r+1}(u_j) \neq \varnothing$ even in $G - S$, contradicting the fact that A is $r+1$-scattered. ∎

The other direction is based on the following theorem, stated in [8], though the proof is extracted from that of a weaker statement proved in [4].

THEOREM 8. *[8] For any $h, r, m \geq 0$ there is an $N \geq 0$ such that if G is a graph with more than N vertices then*

1. *either $K_h \preceq_{r+1} G$; or*
2. *there is a set $S \subseteq V(G)$ with $|S| \leq h-2$ such that $G - S$ contains an r-scattered set of size m.*

Indeed, the bound N is computable as a function of h, r and m. To be precise, let R be the function guaranteed by Ramsey's theorem so that for any set A with $|A| > R(x, y, z)$ any colouring of the y-tuples from A with x distinct colours yields a homogeneous subset of size at least z. Let $b_h(x) = R(h+1, h, (h-2)(x+1))$ and let $c_h(x) = R(2, 2, b_h^{h-2}(x))$ where $b_h^i(x)$ denotes the function b_h iterated i times. Then, it follows from the construction in [4] that taking $N(h, r, m) = c_h^r(m)$ (i.e. c_h iterated r times) suffices for the proof of Theorem 8.

It follows from the above that if $\mathcal{C}$ is a nowhere-dense class of graphs with a computable function h such that $K_{h(r)} \not\preceq_r G$ for any $G \in \mathcal{C}$, then $\mathcal{C}$ is quasi-wide with computable margin s and a computable function $N_{\mathcal{C}}$. We now show that in this case, we can compute rather more. That is, given a graph $G \in \mathcal{C}$ and a set $W \subseteq V(G)$ with $|W| > N(h(r), r, m)$, we can find, in time $\mathcal{O}(|G|^2)$, a set S and a subset $A \subseteq W$ of at least m elements so that in $G - S$, A is r-scattered. This is formalised in the lemma below, which relies on extracting the algorithmic content of the proofs in [4].

LEMMA 9. *Let $\mathcal{C}$ be a nowhere-dense class of graphs and h be the function such that $K_{h(r)} \not\preceq_r G$ for all $G \in \mathcal{C}$. The following problem is solvable in time $\mathcal{O}(|G|^2)$.*

Input:	$G \in \mathcal{C}, r, m \in \mathbb{N}, W \subseteq V(G)$ with $\|W\| > N(h(r), r, m)$
Problem:	compute a set $S \subseteq V(G)$, $\|S\| \leq h(r) - 2$ and a set $A \subseteq W$ with $\|A\| \geq m$, such that in $G - S$, A is r-scattered.

PROOF. In what follows, we write h for $h(r)$ and N for $N(h, r, m)$. The proof constructs sequences of sets of vertices $W_0 \supseteq W_1 \supseteq \cdots \supseteq W_r$ and $S_0 \subseteq S_1 \subseteq \cdots \subseteq S_r = S$ such that for all i,

1. $|S_i| < h - 1$
2. W_i is i-scattered in $G - S_i$
3. $c_h^{r-i}(m) < |W_i|$
4. for all $v \in S_i$ and $u \in W_i$ there is a $w \in N_i^{G-S_i}(u)$ such that $\{v, w\} \in E(G)$.

For $i = 0$, we take $S_0 = \varnothing$ and $W_0 = W$. It is clear that all four conditions are satisfied. Suppose that S_i and W_i have been constructed. We define a graph G' on the set of vertices W_i by putting an edge between u and v if there is an edge in $E(G)$ between some vertex in $N_i^{G-S_i}(u)$ and $N_i^{G-S_i}(v)$. Since $K_h \not\preceq_i G$, G' cannot contain a h-clique and thus as $|W_i| > c_h^{r-i}(m) = R(2, 2, b_h^{h-2}(c_h^{r-(i+1)}(m)))$, G' contains an independent set I with $|I| > b_h^{h-2}(c_h^{r-(i+1)}(m))$, which can be found by a greedy algorithm. Note that G' can be constructed from G in linear time, thus I is found in quadratic time.

The proof of Lemma 5.2 in [4] then guarantees that as long as $K_h \not\preceq_{i+1} G$ we can find $W_{i+1} \subseteq I$ and S_{i+1} satisfying the four conditions above. This is because the condition $K_h \not\preceq_{i+1} G$ guarantees that there is a (possibly empty) set of vertices Z in $G - S_i$ with $|S_i \cup Z| < h - 1$ and a set $J \subseteq I$ with $|J| > c_h^{r-(i+1)}(m)$ such that $N_{i+1}^{G-S_i}(u) \cap N_{i+1}^{G-S_i}(v) = Z$ for each $u, v \in J$. Moreover, the choice of size bounds ensures that Z can be found by a greedy algorithm. We start by taking $Z_0 := \varnothing$ and $I_0 := I$. Once Z_j has been constructed (for $j < h - 2$), we check if there is a vertex z such that there are more than $b_h^{h-2-j}(c_h^{r-(i+1)}(m))$ elements $v \in I_j$ such that $z \in N_{i+1}^{G-(S_i \cup Z_j)}(v)$. If there is, we take I_{j+1} to be the set of such elements v and $Z_{j+1} := Z_j \cup \{z\}$. This process is guaranteed to halt within at most $h - 2$ steps, at which point a greedy algorithm can find a set J with at least $c_h^{r-(i+1)}(m)$ vertices with $N_{i+1}^{G-(S_i \cup Z_j)}(u) \cap N_{i+1}^{G-(S_i \cup Z_j)}(v) = \varnothing$, as otherwise we will have found K_h as a minor of G at depth $i + 1$. Thus, we take $S_{i+1} = S_i \cup Z$ and $W_{i+1} = J$ to satisfy the four conditions above. ∎

The algorithm for distance-d dominating set we present in Section 4 below makes repeated use of the procedure defined above to recursively reduce the problem of finding a distance-d-dominating set of size k in a graph down to the size $N := N_C(d, (k + 2)(d + 1)^s)$, at which point an exhaustive search is performed. The running time of the algorithm is thus exponential in N (which only depends on the parameters), and cubic in $|G|$. On the other hand, the exact parameter dependence of the algorithm depends on the function h, which is determined by the class of structures $\mathcal{C}$. However, even for simple classes $\mathcal{C}$, where h is linear, or constant, N may be a rather fast-growing function of k and d, as it is defined in terms of iterations of the Ramsey function R. On the other hand, as we saw in Example 6, there are quasi-wide classes, such as classes of graphs of bounded degree, where N can be bounded by a simple exponential.

The property of being quasi-wide can be seen as stating the existence of weak separators. Recall that a set S is a separator of a set of vertices W in a graph G if in the graph $G - S$, W is split into non-empty disjoint parts with no path between them. It is known, for instance, that if G is a graph of treewidth at most h, for any set W of vertices, we can find a sep-

arator S of W with $|S| \leq h+1$. Now, nowhere-dense graphs are a generalisation of classes of H-minor free graphs which include, in particular, classes of bounded treewidth. While we cannot hope for the separator property of the form that holds on bounded treewidth classes to hold in nowhere-dense classes, uniform quasi-wideness does show us that we can find a small set S that splits W into parts so that there are no *short* paths between them.

4 Distance-d-Dominating Set

In this section, we show that the distance-d-dominating set problem is fixed-parameter tractable on any nowhere-dense class $\mathcal{C}$ of graphs, with parameter $d+k$. Throughout the remainder of this section, fix a class $\mathcal{C}$ that is uniformly quasi-wide with margin s_C and let $N_C(r,m)$ be as in Definition 4.

We consider a more general form of the problem. We are given a graph G and a set $W \subseteq V(G)$ of vertices and we are asked to determine whether there is a set X in G of at most k vertices that d-dominates W. We begin with the observation that this problem, when parameterized by k, d and *the size of* W is FPT on the class of all graphs.

LEMMA 10. *The following problem is fixed parameter tractable.*

Input:	A graph G, $W \subseteq V(G)$, $k, d \geq 0$
Parameter:	$k + d + \|W\|$
Problem:	Are there $x_1, \ldots, x_k \in V(G)$ such that $W \subseteq \bigcup_i N_d(x_i)$?

PROOF. Consider any partition of W into k sets $W_1, \ldots, W_k$. For each $i \in [1,k]$, define the set $X_i := \bigcap_{w \in W_i} N_d(w)$. That is, X_i is the set of vertices that individually d-dominate the set W_i. Now, if each X_i is non-empty, then we can find the dominating set we are looking for by choosing x_i to be any element of X_i. Conversely, any set $\{x_1, \ldots, x_k\}$ that d-dominates W determines a partition $W_1, \ldots, W_k$ such that $x_i \in X_i$.

The algorithm proceeds by considering each partition of W into k sets in turn (note that the number of such partitions is less than $k^{|W|}$). For each partition, we compute the sets X_i by computing $N_d(w)$ for each $w \in W$ and taking appropriate intersections. This takes time $\mathcal{O}(d \cdot |W| \cdot |G|)$. The total running time is therefore $\mathcal{O}(d \cdot |W| \cdot k^{|W|} \cdot |G|)$ ∎

Now we want to consider the case where the size of W is not part of the parameter, but G is chosen from the nowhere-dense class $\mathcal{C}$. We show that in this case, we can find a set $W' \subseteq W$ whose size is bounded by a function of the parameters k and d such that G contains a set of size k that d-dominates W if, and only if, it contains such a set that d-dominates W'. This will then allow us to use Lemma 10 to get the desired result.

For now, fix k and d, and let $s := s_C(d)$ and $N := N_C(d, (k+2)(d+1)^s)$. That is, for any $G \in \mathcal{C}$ and $W \subseteq V(G)$ with $|W| > N$, we can find $S \subseteq V(G)$ and $A \subseteq W$ such that $|S| \leq s$, $|A| \geq (k+2)(d+1)^s$ and A is d-scattered in $G - S$. We claim that, in this case, we can efficiently find an element $a \in A$ such that G contains a set of size k that d-dominates W if, and only if, there is such a set that d-dominates $W \setminus \{a\}$. We formalise this statement in the lemma below.

LEMMA 11. *There is an algorithm, running in time $f(k,d)|G|^2$ for some computable function f, that given $G \in \mathcal{C}$ and $W \subseteq V(G)$ with $|W| > N$ returns a vertex $w \in W$ such that for any set $X \subseteq V(G)$ with $|X| \leq k$, X d-dominates W if, and only if, X d-dominates $W \setminus \{w\}$.*

PROOF. By Lemma 9, we can find, in time $\mathcal{O}(|G|^2)$, $S \subseteq V(G)$ and $A \subseteq W$ such that $|S| \leq s$, $|A| \geq (k+2)(d+1)^s$ and A is d-scattered in $G - S$. Let $S = \{t_1, \ldots, t_s\}$ and, for each $a \in A$, we compute the *distance vector* $\boldsymbol{v}_a = (v_1, \ldots, v_s)$ where $v_i = \operatorname{dist}(a, t_i)$ if this distance is at most d and $v_i = \infty$ otherwise. Note that there are, by construction, at most $(d+1)^s$ distinct distance vectors. Since $|A| \geq (k+2)(d+1)^s$, there are $k+2$ distinct elements $a_1, \ldots, a_{k+2} \in A$ which have the same distance vector. We claim that taking $w := a_1$ satisfies the lemma.

CLAIM 12. *For any set $X \subseteq V(G)$ with $|X| \leq k$, X d-dominates W if, and only if, X d-dominates $W \setminus \{a_1\}$.*

The direction from left to right is obvious. Now, suppose X d-dominates $W \setminus \{a_1\}$. Consider the sets $A_i := N_d^{G-S}(a_i)$ for $i \in [2, k+2]$. These sets are, by construction, mutually disjoint. Since there are $k+1$ of them, at least one, say A_j, does not contain any element of X. However, since $a_j \in W \setminus \{a_1\}$ there is a path of length at most d from some element x in X to a_j. This path must, therefore, go through an element of S. Since $\boldsymbol{v}_{a_1} = \boldsymbol{v}_{a_j}$, we conclude that there is also a path of length at most d from x to a_1 and therefore X d-dominates W.

For the complexity bounds, note that all the distance vectors can be computed in time $\mathcal{O}(|S| \cdot |A| \cdot |G|)$. This is $f(k,d)|G|$ for a computable f. Adding this to the $\mathcal{O}(|G|^2)$ time to find S and A gives us the required bound. ∎

We now state the main result of this section.

THEOREM 13. *The following is fixed-parameter tractable for any effectively nowhere-dense class $\mathcal{C}$ of graphs.*

DISTANCE-d-DOMINATING SET	
Input:	A graph $G \in \mathcal{C}$, $W \subseteq V(G)$, $k, d \geq 0$
Parameter:	$k + d$
Problem:	Determine whether there is a set $X \subseteq V(G)$ of k vertices which d-dominates W.

PROOF. The algorithm proceeds as follows. Compute $s := s_C(d)$ and $N := N_C(d, (k+2)(d+1)^s)$. As long as $|W| > N$, use the procedure from Lemma 11 to choose an element $w \in W$ that may be removed. Once $|W| \leq N$, use the procedure from Lemma 10 to determine whether the required dominating set exists. ∎

This concludes the proof of Theorem 13. The following corollaries are immediate.

COROLLARY 14. *The dominating set problem is fixed-parameter tractable on any effectively nowhere-dense class.*

This generalises the known results for the dominating set problem on classes of bounded expansion and classes that locally exclude a minor. This corollary is also obtained as a consequence of a result in [23].

COROLLARY 15. *The distance-d dominating set problem is fixed-parameter tractable with parameter $k + d$ on any class of graphs of bounded expansion, where k is the size of the solution.*

This answers a question of Nešetřil and Ossona de Mendez who show that the dominating set problem is fixed-parameter tractable on such classes and ask whether the same could be true for the distance-2 dominating set problem.

5 Other Domination Problems

Among problems that are fixed-parameter intractable, dominating set and its variants play an important role. For instance, in the Compendium of Parameterized Problems [6], a number of problems are given which are known to be hard in general, but tractable on planar graphs. Virtually all of them are variations on the theme of finding dominating sets. In this section we show that all of these problems and, in many cases, their harder "distance-d" versions remain fixed-parameter tractable on nowhere-dense classes of graphs, which greatly generalises the results on planar graphs. We refer to [6] for formal definitions of the problems below and references to the literature where they were first studied.

The first type of problems we look at are dominating set problems with additional requirements for connectivity, such as CONNECTED DOMINATING SET where we are to compute a dominating set which induces a connected sub-graph. We study its generalisation to d-domination defined as follows.

CONNECTED DISTANCE-d-DOMINATING SET
Input: Graph $G, k, d \in \mathbb{N}$
Parameter: $k + d$
Problem: Is there a subset $X \subseteq V(G)$ with $|X| = k$ such that X d-dominates G and $G[X]$ is connected?

We are able to show that this problem is FPT on nowhere-dense classes of graphs by adapting the proof of Lemma 10 to show that the following problem is FPT.

Input: A graph $G, W \subseteq V(G), k, d \geq 0$
Parameter: $k + d + |W|$
Problem: Are there $x_1, \ldots, x_k \in V(G)$ such that $W \subseteq \bigcup_i N_d(x_i)$ and $G[x_1, \ldots, x_k]$ is connected?

Similar methods can be used to show that the problem d-CONNECTED DISTANCE-d-DOMINATING SET is FPT on nowhere-dense classes. This is the problem of deciding if there is a d-dominating set X of k vertices which is d-connected. A set is said to be d-connected in a graph G if it induces a connected subgraph in the graph G^d obtained from G by putting an edge between any two vertices that have distance at most d in G. The same method shows that EFFICIENT DOMINATING SET is in FPT on nowhere-dense graph classes.

EFFICIENT d-DOMINATING SET
Input: Graph $G, k, d \in \mathbb{N}$
Parameter: $k + d$
Problem: Is there a subset $X \subseteq V(G)$ with $|X| = k$ such that X is a d-dominating set and, in addition, all pairs $x \neq y \in X$ have distance at least $2d + 1$?

Further variations of domination problems studied in the literature are ANNOTATED DOMINATING SET and RED-BLUE DOMINATING SET. Annotated domination means that we are given a graph G and $W \subseteq V(G)$ and want to find a set dominating W. The distance-d-version of this problem is what is solved by Theorem 13. Red-Blue Domination means that we are given a graph G whose vertex set is partitioned into blue and red vertices and we want to dominate the blue vertices using red vertices only. Again its distance-d version can be solved by the methods presented in Section 4.

Finally, we look at problems such as ROMAN DOMINATION, MAXIMUM MINIMAL DOMINATING SET, PERFECT CODE and DIGRAPH KERNEL. If we are not interested in their distance-d-version than an even simpler algorithm than the one presented above can be used to show that these problems are in FPT on nowhere-dense classes of graphs, which we demonstrate using the ROMAN DOMINATION problem.

ROMAN DOMINATION
Input: Graph G, $k \in \mathbb{N}$
Parameter: k
Problem: Is there a Roman domination function R such that $\sum_{v \in V(G)} R(v) \leq k$?

A Roman domination of G is a function $R : V(G) \to \{0, 1, 2\}$ such that for all $v \in V(G)$ if $R(v) = 0$ then there exists an $x \in N(v)$ such that $R(x) = 2$. To solve the problem on nowhere-dense classes of graphs we first compute a set $S \subseteq G$ such that $G \setminus S$ contains a 2-scattered set A of size $2k + 1$. Clearly, for at least $k + 1$ vertices $v \in A$ we must have $R(v) = 0$ and hence one of their neighbours must be labelled by 2. However, this implies that at least one vertex in S must be labelled by 2. As $|S|$ only depends on the parameter we can use this to define a recursive procedure whose depth and width only depend on the parameter.

The following theorem sums up what we have established so far. It is easily seen that INDEPENDENT SET and INDEPENDENT DOMINATING SET are FPT on nowhere-dense classes and our procedures presented before readily solve the problems. We refer to [6] for precise definitions of the problems.

THEOREM 16. *The following problems are fixed-parameter tractable on effectively nowhere-dense classes of graphs:* CONNECTED DOMINATING SET, CONNECTED d-DOMINATING SET, c-CONNECTED d-DOMINATING SET, ANNOTATED d-DOMINATING SET, EFFICIENT d-DOMINATING SET, MAXIMUM MINIMAL DOMINATING SET, ROMAN DOMINATION, RED-BLUE DOMINATING SET, INDEPENDENT SET, INDEPENDENT DOMINATING SET, PERFECT CODE, *and* DIGRAPH KERNEL.

These examples show that the distance-d-dominating set problem that we showed is tractable on nowhere-dense graph classes is actually representative of a whole class of similar problems which become tractable in this case. Several of these are known to be intractable on graph classes of bounded degeneracy. To give an example of a problem which remains hard on nowhere-dense classes, consider the DIRECTED DISJOINT PATHS problem.

DIRECTED DISJOINT PATHS
Input: Directed graph G, pairs $(s_1, t_1), \ldots, (s_k, t_k) \in V(G)^2$
Parameter: k
Problem: Does G contain k vertex disjoint paths $P_1, \ldots, P_k$ such that P_i links s_i and t_i?

This is known to be $W[1]$-hard even on acyclic digraphs and it is easy to see that it can be reduced to the directed disjoint paths problem on graphs of degree at most 4 as follows. Let G be a digraph and let $v \in V(G)$ be a vertex with in-neighbours $u_1, \ldots, u_l$ where $l > 1$. Let T be a directed rooted tree of degree at most 3 which has l leaves and where all edges are oriented towards the root. Now eliminate all incoming edges to v and add the tree T to G identifying v with the root of T and $u_1, \ldots, u_l$ with the leaves of T. A similar procedure is used to eliminate outgoing edges of v. Applying this to all vertices in G yields a graph G' of degree at most 4 but which has k disjoint paths between the pairs $(s_1, t_1), \ldots, (s_k, t_k)$ if, and only if, such paths exist in G. Since the class of graphs of degree at most 4 is nowhere dense, this shows the problem is hard on such classes as in the general case.

6 Conclusion and Further Work

The aim of the paper is to initiate an algorithmic study of graph classes which are nowhere dense. The examples above, including the dominating set problem and the more general distance-d dominating set, or (k, d)-centre, problem, demonstrate that a certain class of important algorithmic problems become fixed-parameter tractable on classes which are nowhere dense. One of the main advantages of these results over known algorithms for these problems on classes excluding a fixed minor is that our algorithms are elementary and do not rely on deep results and methods from graph minor theory.

One direction for further research is to investigate what other problems might become tractable on nowhere-dense classes of graphs. Also, it would be interesting to compare nowhere-dense classes of graphs to graph classes of bounded degeneracy. The two concepts are incomparable but both generalise classes excluding a fixed minor and we have already seen that there are examples of problems that become fixed-parameter tractable on nowhere-dense classes of graphs which are intractable on classes of graphs of bounded degeneracy.

Finally, it would be interesting to explore the extent of the algorithmic theory of nowhere-dense classes of graphs in terms of algorithmic meta-theorems. In particular, it would be very interesting if model-checking for first-order logic was FPT on nowhere-dense classes of graphs. This would establish a rich algorithmic theory of such classes. However, establishing such a result would require novel methods as we do not have a decomposition theory for nowhere-dense classes along the lines of what is used to establish the tractability of first-order logic in classes with locally excluded minors.

References

[1] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.

[2] J. Alber, H. Fan, M.R. Fellows, H. Fernau, R. Niedermeier, F. Rosamond, and U. Stege. A refined search tree technique for dominating set on planar graphs. *Journal of Comput. Syst. Sci.*, 71(4):385–405, 2005.

[3] N. Alon and S. Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. In *COCOON*, pages 394–405, 2007.

[4] A. Atserias, A. Dawar, and Ph. G. Kolaitis. On preservation under homomorphisms and unions of conjunctive queries. *Journal of the ACM*, 53(2):208–237, 2006.

[5] J. Bar-Ilan, G. Kortsarz, and D. Peleg. How to allocate network centers. *J. Algorithms*, 15(3):385–415, 1993.

[6] Marco Cesati. http://bravo.ce.uniroma2.it/home/cesati/research/compendium/, September 15 2006.

[7] A. Dawar. Finite model theory on tame classes of structures. In *MFCS*, volume 4708 of *LNCS*, pages 2–12. Springer, 2007.

[8] A. Dawar. Homomorphism preservation on quasi-wide classes. to appear in JCSS, 2008. see also arXiv:0811.4497v1 [cs.LO].

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

[10] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Trans. on Algorithms*, 1(1):33–47, 2005.

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

[12] R. Diestel. *Graph Theory*. Springer-Verlag, 3rd edition, 2005.

[13] R. Downey and M. Fellows. *Parameterized Complexity*. Springer, 1998.

[14] J.A. Ellis, H. Fan, and M.R. Fellows. The dominating set problem is fixed parameter tractable for graphs of bounded genus. *J. Algorithms*, 52(2):152–168, 2004.

[15] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[16] F. V. Fomin and D. M. Thilikos. Dominating sets in planar graphs: branch-width and exponential speed-up. In *SODA*, pages 168–177, 2003.

[17] P. A. Golovach and Y. Villanger. Parameterized complexity for domination problems on degenerate graphs. In *WG*, volume 5344 of *LNCS*, pages 195–205. Springer, 2008.

[18] I. A. Kanj and L. Perkovic. Improved parameterized algorithms for planar dominating set. In *MFCS*, pages 399–410, 2002.

[19] J. Nešetřil and P. Ossona de Mendez. First-order properties of nowhere dense structures. to appear in Journal of Symbolic Logic, 2008.

[20] J. Nešetřil and P. Ossona de Mendez. Grad and classes with bounded expansion I–III. *European Journal of Combinatorics*, 29, 2008. Series of 3 papers appearing in volumes (3) and (4).

[21] J. Nešetřil and P. Ossona de Mendez. On nowhere dense graphs. to appear in European Journal of Combinatorics, 2008.

[22] J. Nešetřil and P. Ossona de Mendez. Structural properties of sparse graphs. In M. Grötschel and G.O.H. Katona, editors, *Building Bridges Between Mathematics and Computer Science*, volume 19. Springer, 2008.

[23] G. Philip, V. Raman, and S. Sikdar. Solving dominating set in larger classes of graphs: FPT algorithms and polynomial kernels. In *ESA*, volume 5757 of *LNCS*, pages 694–705. Springer, 2009.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Simulation based security in the applied pi calculus*

Stéphanie Delaune[1], Steve Kremer[1], and Olivier Pereira[2]

[1]LSV, ENS Cachan & CNRS & INRIA, France

[2]Université catholique de Louvain, B-1348 Belgium

ABSTRACT. We present a symbolic framework for refinement and composition of security protocols. The framework uses the notion of ideal functionalities. These are abstract systems which are secure by construction and which can be combined into larger systems. They can be separately refined in order to obtain concrete protocols implementing them. Our work builds on ideas from the "trusted party paradigm" used in computational cryptography models. The underlying language we use is the applied pi calculus which is a general language for specifying security protocols. In our framework we can express the different standard flavours of simulation-based security which happen to all coincide. We illustrate our framework on an authentication functionality which can be realized using the Needham-Schroeder-Lowe protocol. For this we need to define an ideal functionality for asymmetric encryption and its realization. We show a joint state result for this functionality which allows composition (even though the same key material is reused) using a tagging mechanism.

1 Introduction

Symbolic techniques showed to be a very useful approach for the modeling and analysis of security protocols: for twenty years, they improved our understanding of security protocols, allowed discovering flaws [17], and provided support for protocol design [9]. These techniques also resulted in the creation of powerful automated analysis tools (e.g. [3]), and impacted on several protocol standards used every day, e.g., [8].

Until now, symbolic techniques mostly concentrated on specifying and proving confidentiality and correspondence properties, i.e., showing which symbols are kept secret, and on which session parameters participants agree when a protocol session completes. However, such specifications do not provide any information about the behavior of protocols when they are used in composition with other protocols, and surprising behaviors are well know to happen in such contexts [7]. Moreover, protocols are often expected to provide more sophisticated security guarantees, which may be difficult to formalize.

In this paper, we present a symbolic framework for refinement and composition of security protocols, in which protocols are defined in terms of the behavior of trusted parties, or ideal functionalities, following the general outline of simulation-based security [5, 12, 4]. A lower-level protocol is said to securely emulate a higher-level protocol, or ideal functionality, if any behavior that can be observed from the interaction of an adversary with the lower-level protocol can also be observed from the interaction of another adversary (called

*This work has been supported by the ANR-07-SESU-002 project. O. Pereira is a Research Associate of the Belgian Fund for Scientific Research FRS-FNRS. Part of this work was done when he was visiting ENS Cachan.

© Delaune, Kremer, Pereira; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 169–180
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2316

the simulator) with the higher-level protocol. As a result, ideal functionalities can be successively refined into more concrete protocols, but also composed to build more complex protocols. Functionalities have been proposed for a wide range of protocol tasks, including general secure multi-party computation [5]. In the spi-calculus [2], Abadi and Gordon also present the idea of a protocol being equivalent to an idealized version. This is however more restrictive as they do not have the notion of a simulator.

Simulation-based security frameworks can typically be decomposed into two "layers": (i) a foundational layer that provides a general model for concurrent computation, and (ii) a security layer that provides general security definitions, most importantly the notion of secure protocol emulation to be used. While the security layer is essentially common to all frameworks [4, 5, 6, 14, 18], including this paper, the foundational layer varies widely. Those variations typically lie in the concurrency model (from the most common token-passing mechanism to the use of schedulers with various powers) and in the definition of computational bounds. These differences typically result in incomparable security notions.

Defining simulation-based security while choosing the applied pi calculus [1] as the foundational layer brings the main benefits of this approach into the symbolic world:

- it provides a powerful machinery that can be used to specify a wide range of sophisticated protocol tasks in terms of the behavior of functionalities, and
- general composition theorems guarantee that protocols keep behaving as expected when executed in arbitrary contexts.

While we tried to stick to the common definitions from the security layer of simulation-based security frameworks, the use of the applied pi calculus as foundational layer raised interesting challenges.

First, at the most fundamental level, one has to adopt a notion of indistinguishability of processes. While the symmetric notions of computational indistinguishability and observational equivalence are commonly used in the cryptographic and symbolic worlds, the symmetry of such relations appeared to be too restrictive for our purpose. For instance, a symmetric equivalence relation makes the addition of an adversary that simply acts as a relay visible. The resulting undesired behaviors motivate the introduction of new notions of observational preorder and labelled simulation relations in the applied pi calculus.

Next, our attempts at translating ideal functionalities from the computational world into the symbolic world showed to be a non immediate task. For instance, traditional ideal functionalities for asymmetric encryption produce ciphertexts by encrypting random strings. An association table (plaintext/ciphertext) is then necessary to perform decryption. In our symbolic setting we avoid such a table by using two layers of encryption and a secure key.

Eventually, we investigate the statement of general composition theorems, and of a specific joint state composition theorem for our asymmetric encryption functionality, as this functionality is typically expected to be used in several protocol sessions. While these theorems appear to be the natural counterpart of their computational versions [4, 5, 6, 16], the joint state composition theorem brings message tagging constraints that, interestingly, are consistent with those obtained by using a completely different symbolic approach (e.g. [13]).

Because of lack of space the proofs are omitted, but can be found in [10].

2 The applied pi calculus

2.1 Syntax and informal semantics

To describe processes, one starts with a set of *names* (which are used to name communication channels or other atomic data), a set of *variables*, and a *signature* Σ which consists of the *function symbols* which will be used to define *terms*. In the case of security protocols, typical function symbols will include enc for encryption, and dec for decryption. Terms are defined as names, variables, and function symbols applied to other terms. Terms and function symbols are sorted. While the details of the sort systems are not essential it is important to distinguish sorts of base types and sorts of channel type. Function symbols can only be applied and return terms of base type. By the means of an equational theory E we describe the equations which hold on terms. We denote $=_{\mathsf{E}}$ the equivalence relation induced by E.

Example 1 *In the equational theory* $\{\mathsf{dec}(\mathsf{enc}(x,k),k) = x,\ \mathsf{test}(\mathsf{enc}(x,y),y) = \mathsf{ok}\}$*, we have that* $\mathsf{test}(\mathsf{dec}(\mathsf{enc}(\mathsf{enc}(n,k_1),k_2),k_2),k_1) =_{\mathsf{E}} \mathsf{ok}$.

In the applied pi calculus, one has *plain processes* and *extended processes*. Plain processes (P, Q, R) are built up in a similar way to processes in the pi calculus, except that messages can contain terms (rather than just names). Extended processes add *active substitutions* and restriction on variables. Below, M is a term, n is a name, and x a variable.

$$A, B, C := P \mid A \mid B \mid \nu n.A \mid \nu x.A \mid \{^{M}/_{x}\}$$

Active substitutions generalise "let". The process $\nu x.(\{^{M}/_{x}\} \mid P)$ corresponds exactly to the process "let $x = M$ in P". As usual, names and variables have scopes, which are delimited by restrictions and by inputs. We write $fv(A)$, $bv(A)$, $fn(A)$ and $bn(A)$ for the sets of free and bound variables and free and bound names of A, respectively. We also assume that, in an extended process, there is at most one substitution for each variable, and there is exactly one when the variable is restricted. We say that an extended process is *closed* if all its variables are either bound or defined by an active substitution.

Active substitutions are useful because they allow us to map an extended process A to its *frame* $\phi(A)$ by replacing every plain process in A with 0. A frame is an extended process built up from 0 and active substitutions by parallel composition and restriction. The frame $\phi(A)$ can be viewed as an approximation of A that accounts for the static knowledge A exposes to its environment, but not A's dynamic behaviour. The *domain* of a frame φ, denoted by $\mathrm{dom}(\varphi)$, is the set of variables for which φ defines a substitution (those variables that are not under a restriction). An *evaluation context* $C[\_]$ is an extended process with a hole instead of an extended process. A context $C[\_]$ *closes* A when $C[A]$ is closed.

2.2 Semantics

The operational semantics of processes in the applied pi calculus is defined by structural rules defining two relations: *structural equivalence*, denoted $\equiv$, and *internal reduction*, denoted $\rightarrow$. Structural equivalence takes into account some basic structural rules, e.g. associativity and commutativity of the parallel operator. Internal reduction $\rightarrow$ is the smallest relation on extended processes closed under structural equivalence and application of evaluation contexts such that:

$$\begin{array}{lllll}
\textsc{Comm} & \mathsf{out}(a,M).P \mid \mathsf{in}(a,x).Q & \to & P \mid Q\{^M/_x\} & \\
\textsc{Then} & \text{if } M = N \text{ then } P \text{ else } Q & \to & P & \text{where } M =_{\mathsf{E}} N \\
\textsc{Else} & \text{if } M = N \text{ then } P \text{ else } Q & \to & Q & \\
 & \text{for any terms } M \text{ and } N \text{ without variable such that } M \neq_{\mathsf{E}} N & & &
\end{array}$$

The operational semantics is extended by a *labelled* operational semantics enabling us to reason about processes that interact with their environment. Labelled operational semantics defines the relation $\xrightarrow{\alpha}$ where α is either an input $\mathsf{in}(a,M)$ (a is a channel name and M is a term that can contain names and variables), or $\nu x.\mathsf{out}(a,x)$ (x is a variable of base type), or $\mathsf{out}(a,c)$ or $\nu c.\mathsf{out}(a,c)$ (c is a channel name).

$$\textsc{In}\quad \mathsf{in}(a,x).P \xrightarrow{in(a,M)} P\{^M/_x\} \qquad \textsc{Out-Ch}\quad \mathsf{out}(a,c).P \xrightarrow{out(a,c)} P$$

$$\textsc{Open-Ch}\quad \frac{A \xrightarrow{out(a,c)} A' \quad c \neq a}{\nu c.A \xrightarrow{\nu c.out(a,c)} A'} \qquad \textsc{Out-T}\quad \mathsf{out}(a,M).P \xrightarrow{\nu x.out(a,x)} P \mid \{^M/_x\} \quad x \notin fv(P) \cup fv(M)$$

$$\textsc{Scope}\quad \frac{A \xrightarrow{\alpha} A' \quad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}$$

$$\textsc{Par}\quad \frac{A \xrightarrow{\alpha} A' \quad bn(\alpha) \cap fn(B) = \emptyset \quad bv(\alpha) \cap fv(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

$$\textsc{Struct}\quad \frac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad A' \equiv B'}{A \xrightarrow{\alpha} A'}$$

Our rules differ slightly from those described in [1]. It is proved in [11] that labelled bisimulation (see Section 2.3) in our system coincides with labelled bisimulation in [1].

Example 2 *Consider the following process P:*
$\nu k.\ (\mathsf{in}(io_1,x).\mathsf{out}(net,\mathsf{enc}(x,k)) \mid \mathsf{in}(net,y).\ \mathit{if}\ \mathsf{test}(y,k) = \mathsf{ok}\ \mathit{then}\ \mathsf{out}(io_2,\mathsf{dec}(y,k))\ \mathit{else}\ 0)$.

The first component receives a message x on the channel io_1 and sends its encryption with the key k on the channel net. The second one is waiting for an input y on net, uses the secret key k to decrypt it. If the decryption succeeds, then it forwards the resulting plaintext on io_2. We have that:

$$\begin{array}{l}
P \xrightarrow{in(io_1,s)} \nu k.(\mathsf{out}(net,\mathsf{enc}(s,k)) \mid \mathsf{in}(net,y).\ \mathit{if}\ \mathsf{test}(y,k) = \mathsf{ok}\ \mathit{then}\ \mathsf{out}(io_2,\mathsf{dec}(y,k))\ \mathit{else}\ 0) \\
\quad \longrightarrow^* \quad \nu k.\mathsf{out}(io_2,s) \xrightarrow{\nu x.out(io_2,x)} \nu k.\{^s/_x\}
\end{array}$$

Let A be the resulting process. We have that $\phi(A) \equiv \nu k.\{^s/_x\}$.

2.3 Indistinguishability relations

In [1], it is shown that observational equivalence coincides with labelled bisimilarity. This relation is like the usual definition of bisimilarity, except that at each step one additionally requires that the processes are statically equivalent.

DEFINITION 1. *Two terms M and N are* equal in the frame ϕ, *written $(M =_{\mathsf{E}} N)\phi$, if, and only if there exists $\tilde{n}$ and a substitution σ such that $\phi \equiv \nu\tilde{n}.\sigma$, $M\sigma =_{\mathsf{E}} N\sigma$, and $\tilde{n} \cap (fn(M) \cup fn(N)) = \emptyset$. Two frames ϕ_1 and ϕ_2 are* statically equivalent, *$\phi_1 \approx_s \phi_2$, when $\mathrm{dom}(\phi_1) = \mathrm{dom}(\phi_2)$, and for all terms M, N we have that $(M =_{\mathsf{E}} N)\phi_1$ if and only if $(M =_{\mathsf{E}} N)\phi_2$.*

Example 3 *Let $\varphi_0 = \nu s.\{^{\mathsf{enc}(s,k)}/_x\}$ and $\varphi_1 = \nu r.\{^r/_x\}$ where k, s and r are names and E be the theory given in Example 1. We have that $(\mathsf{test}(x,k) =_{\mathsf{E}} \mathsf{ok})\varphi_0$ but not $(\mathsf{test}(x,k) =_{\mathsf{E}} \mathsf{ok})\varphi_1$, thus $\varphi_0 \not\approx_s \varphi_1$. However, we have that $\nu k.\varphi_0 \approx_s \varphi_1$.*

Now, we introduce the notion of a *barb*. Given an extended process A and a channel name a, we write $A \Downarrow a$ when $A \to^* C[\mathsf{out}(a,M).P]$ for some term M, plain process P, and

evaluation context $C[\_]$ that does not bind a.

DEFINITION 2. Observational preorder *($\preceq$) (resp.* equivalence *($\approx$)) is the largest (resp. largest symmetric) relation on extended processes with same domain s.t. $A\ \mathcal{R}\ B$ implies*

1. *if $A \Downarrow a$ then $B \Downarrow a$;*
2. *if $A \rightarrow^* A'$, then $B \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$ for some B';*
3. *$C[A]\ \mathcal{R}\ C[B]$ for all closing evaluation contexts $C[\_]$.*

DEFINITION 3. *A relation $\mathcal{R}$ on closed extended processes is a* simulation *if $A\ \mathcal{R}\ B$ implies*

1. *$\phi(A) \approx_s \phi(B)$,*
2. *if $A \rightarrow A'$, then $B \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$ for some B',*
3. *if $A \xrightarrow{\alpha} A'$ and $fv(\alpha) \subseteq \mathrm{dom}(A)$ and $bn(\alpha) \cap fn(B) = \emptyset$, then $B \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$ for some B'.*

If $\mathcal{R}$ and $\mathcal{R}^{-1}$ are both simulations we say that $\mathcal{R}$ is a bisimulation. Labelled similarity *($\preceq_\ell$), resp.* labelled bisimilarity *($\approx_\ell$), is the largest simulation, resp. bisimulation relation.*

Observational preorder and similarity were not introduced in [1]. However, these definitions seem natural for our purposes. Obviously we have that $\approx \subset \preceq$ and $\approx_\ell \subset \preceq_\ell$. We now show that labelled similarity is a precongruence.

PROPOSITION 4. *Let A and B be two extended processes such that $A \preceq_\ell B$. We have that $C[A] \preceq_\ell C[B]$ for all closing evaluation context $C[\_]$.*

From this proposition it follows that $\preceq_\ell \subseteq \preceq$. Hence, we can use labelled similarity as a convenient proof technique for observational preorder. We actually expect the two relations to coincide but did not prove it as we did not need it. We have also the following lemma:

LEMMA 5. *Let P and Q be two closed plain processes. We have that: (i) if $P \preceq_\ell Q$ then $!P \preceq_\ell !Q$; (ii) $!(P \mid Q) \preceq_\ell !P \mid !Q$ and $!P \mid !Q \preceq_\ell !(P \mid Q)$.*

3 Simulation based security

3.1 Basic definitions

The simulation-based security approach classically distinguishes between *input-output channels*, which yield the internal interface of a protocol or *functionality* to its environment and *network channels*, which allow the adversary to interact with the functionality. We suppose that all channels are of one of these two sorts: IO or NET. Moreover the sort system ensures that names of sort NET can never be conveyed as data on a channel, i.e. these channels can never be transmitted. We write $\mathsf{fnet}(P)$ for $fn(P) \cap \mathsf{NET}$.

DEFINITION 6. *A* functionality *$\mathcal{F}$ is a closed plain process. An* adversary *for $\mathcal{F}$ is an evaluation context $\mathcal{A}[\_]$ of the form: $\nu\widetilde{net}_1.(A_1 \mid \nu\widetilde{net}_2.(A_2 \mid \ldots \mid \nu\widetilde{net}_k.(A_k \mid \_)\ldots))$ where each A_i $(1 \leq i \leq k)$ is a closed plain process, $\mathsf{fnet}(\mathcal{F}) \subseteq \bigcup_{1\leq i\leq k} \widetilde{net}_i \subseteq \mathsf{NET}$, and $fn(\mathcal{A}[\_]) \cap \mathsf{IO} = \emptyset$.*

One may note that the nested form of the adversary process allows to express any arbitrary context while expliciting the restricted names whose scope ranges on the hole. We also note that if $\mathcal{A}[\_]$ is an adversary for $\mathcal{F}$ then $\mathsf{fnet}(\mathcal{A}[\_]) = \mathsf{fnet}(\mathcal{A}[\mathcal{F}])$.

LEMMA 7. *Let $\mathcal{F}$ be a functionality and $\mathcal{A}_1[\_]$ be an adversary for $\mathcal{F}$. Then $\mathcal{A}_1[\mathcal{F}]$ is a functionality. If $\mathcal{A}_2[\_]$ is an adversary for $\mathcal{A}_1[\mathcal{F}]$, then $\mathcal{A}_2[\mathcal{A}_1[\_]]$ is an adversary for $\mathcal{F}$.*

While adversaries control the communication of functionalities on NET channels, IO *contexts* model the environment of functionalities, providing inputs and collecting outputs.

DEFINITION 8. *An* IO context *is an evaluation context $C_{io}[\_]$ of the form $\nu\widetilde{io}_1.(C_1 \mid \nu\widetilde{io}_2.(C_2 \mid \ldots \mid \nu\widetilde{io}_k.(C_k \mid \_)\ldots))$ where each C_i ($1 \leq i \leq k$) is a closed plain process, and $\bigcup_{1\leq i\leq k} \widetilde{io}_i \subseteq \mathsf{IO}$*

Note that if $\mathcal{F}$ is a functionality and $C_{io}[\_]$ is an IO context, then $C_{io}[\mathcal{F}]$ is a functionality.

3.2 Strong simulatability

The notion of strong simulatability [15], which is probably the simplest secure emulation notion used in simulation-based security, can be formulated in our setting.

DEFINITION 9. *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two functionalities. $\mathcal{F}_1$* emulates *$\mathcal{F}_2$ in the sense of* strong simulatability*, written $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$, if there exists an adversary $\mathcal{S}$ for $\mathcal{F}_2$ (the simulator) such that $\mathsf{fnet}(\mathcal{F}_1) = \mathsf{fnet}(\mathcal{S}[\mathcal{F}_2])$ and $\mathcal{F}_1 \preceq \mathcal{S}[\mathcal{F}_2]$.*

The definition ensures that any behavior of $\mathcal{F}_1$ can be matched by $\mathcal{F}_2$ executed in the presence of a specific adversary $\mathcal{S}$. Hence, there are no more attacks on $\mathcal{F}_1$ than attacks on $\mathcal{F}_2$. Moreover, the presence of $\mathcal{S}$ allows abstract definitions of higher-level functionalities, which are independent of a specific realization. One may also note that $0 \leq^{\mathsf{SS}} \mathcal{F}$ for any functionality $\mathcal{F}$. This seems natural in a simulation based framework which only aims at preserving security. Non-triviality conditions may be imposed independently [5].

Example 4 *Let $\mathcal{F}_{\mathsf{cc}} = \mathsf{in}(io_1, x_s).\mathsf{out}(net_{\mathsf{cc}}, \mathsf{ok}).\mathsf{in}(net_{\mathsf{cc}}, x)$. if $x = \mathsf{ok}$ then $\mathsf{out}(io_2, x_s)$. The functionality models a confidential channel and takes a potentially secret value s as input on channel io_1. The adversary is notified via channel net_{cc} that this value is to be transmitted. If the adversary agrees the value is output on channel io_2. This functionality can be realized by the process described in Example 2. Let $\mathcal{S} = \nu net_{\mathsf{cc}}.\mathsf{in}(net_{\mathsf{cc}}, x).\nu r.\mathsf{out}(net, r).\mathsf{in}(net, x)$. if $x = r$ then $\mathsf{out}(net_{\mathsf{cc}}, \mathsf{ok}) \mid \_)$. We indeed have that $P \preceq_\ell \mathcal{S}[\mathcal{F}_{\mathsf{cc}}]$ and $\mathsf{fnet}(P) = \mathsf{fnet}(\mathcal{S}[\mathcal{F}_{\mathsf{cc}}])$.*

In order to examine the properties of strong simulatability in our specific setting, we now define a particular adversary $D[\_]$ which is called a *dummy adversary*: intuitively, it acts as a relay which forwards all messages. The formal definition is technical because $D[\_]$ needs to both restrict all names in $\mathsf{fnet}(\mathcal{F})$ and ensure that $\mathsf{fnet}(\mathcal{F}) = \mathsf{fnet}(D[\mathcal{F}])$. It therefore relies on two internal channels $sim_j^{i/o}$ for inputs, resp. outputs, for each channel in $\mathsf{fnet}(\mathcal{F})$.

DEFINITION 10. *Let $\mathcal{F}$ be a functionality. The* dummy adversary *for $\mathcal{F}$ is the adversary $D[\_] = \nu\widetilde{sim}.(D_1 \mid \nu\widetilde{net}.(D_2 \mid \_))$ where $\widetilde{net} = \mathsf{fnet}(\mathcal{F}) = \{net_1, \ldots, net_n\}$; $\widetilde{sim} = \{sim_1^i, \ldots, sim_n^i, sim_1^o, \ldots, sim_n^o\} \subseteq \mathsf{NET}$; and*

- $D_1 = !\mathsf{in}(net_1, x).\mathsf{out}(sim_1^i, x) \mid \ldots \mid !\mathsf{in}(net_n, x).\mathsf{out}(sim_n^i, x) \mid$
  $!\mathsf{in}(sim_1^o, x).\mathsf{out}(net_1, x) \mid \ldots \mid !\mathsf{in}(sim_n^o, x).\mathsf{out}(net_n, x)$;
- $D_2 = !\mathsf{in}(sim_1^i, x).\mathsf{out}(net_1, x) \mid \ldots \mid !\mathsf{in}(sim_n^i, x).\mathsf{out}(net_n, x) \mid$
  $!\mathsf{in}(net_1, x).\mathsf{out}(sim_1^o, x) \mid \ldots \mid !\mathsf{in}(net_n, x).\mathsf{out}(sim_n^o, x)$.

By construction we have that $\mathsf{fnet}(D[\mathcal{F}]) = \mathsf{fnet}(\mathcal{F})$.

LEMMA 11. *Let $\mathcal{F}$ be a functionality and $D[\_]$ be the dummy adversary for $\mathcal{F}$: $\mathcal{F} \preceq D[\mathcal{F}]$.*

However, we do not have that $\mathcal{F} \approx D[\mathcal{F}]$, since $D[\mathcal{F}]$ has more non-determinism than $\mathcal{F}$. A direct consequence of this lemma is that $\mathcal{F}_1 \preceq \mathcal{F}_2$ and $\mathsf{fnet}(\mathcal{F}_1) = \mathsf{fnet}(\mathcal{F}_2)$ implies that $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$: as $\mathcal{F}_2 \preceq D[\mathcal{F}_2]$ we have by transitivity that $\mathcal{F}_1 \preceq D[\mathcal{F}_2]$. We use these observations to show that $\leq^{\mathsf{SS}}$ is a preorder (Lemma 12) , which is closed under application of IO contexts (Proposition 13) and parallel composition (Proposition 14).

LEMMA 12. *(i)* $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_1$; *(ii)* $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$ *and* $\mathcal{F}_2 \leq^{\mathsf{SS}} \mathcal{F}_3 \Rightarrow \mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_3$.

PROPOSITION 13. *Let* $\mathcal{F}_1, \mathcal{F}_2$ *be two functionalities and* C_{io} *be an IO context.*

$$\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2 \implies C_{io}[\mathcal{F}_1] \leq^{\mathsf{SS}} C_{io}[\mathcal{F}_2].$$

PROPOSITION 14. *Let* $\mathcal{F}_1, \mathcal{F}_2$ *and* $\mathcal{F}_3$ *be three functionalities. We have that:* *(i)* $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2 \Rightarrow \mathcal{F}_1 \mid \mathcal{F}_3 \leq^{\mathsf{SS}} \mathcal{F}_2 \mid \mathcal{F}_3$*; and (ii)* $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2 \Rightarrow !\mathcal{F}_1 \leq^{\mathsf{SS}} !\mathcal{F}_2$.

While, (i) is a direct consequence of Proposition 13 (notice that $\_ \mid \mathcal{F}_3$ is an IO-context) the proof of (ii) is more involved and given in [10].

3.3 Other notions of simulation based security

Several other notions of simulation based security appear in the literature. We present them, and show that they all coincide in our setting. This coincidence is regarded as highly desirable [15, 14], while it does not hold in most simulation-based security frameworks [5, 4].

DEFINITION 15. *Let* $\mathcal{F}_1$ *and* $\mathcal{F}_2$ *be two functionalities and* $\mathcal{A}$ *be any adversary for* $\mathcal{F}_1$.

- $\mathcal{F}_1$ *emulates* $\mathcal{F}_2$ *in the sense of* black box simulatability, $\mathcal{F}_1 \leq^{\mathsf{BB}} \mathcal{F}_2$, *iff* $\exists \mathcal{S}. \forall \mathcal{A}. \mathcal{A}[\mathcal{F}_1] \preceq \mathcal{A}[\mathcal{S}[\mathcal{F}_2]]$ *where* $\mathcal{S}$ *is an adversary for* $\mathcal{F}_2$ *with* $\mathsf{fnet}(S[\mathcal{F}_2]) = \mathsf{fnet}(\mathcal{F}_1)$.
- $\mathcal{F}_1$ *emulates* $\mathcal{F}_2$ *in the sense of* universally composable simulatability, $\mathcal{F}_1 \leq^{\mathsf{UC}} \mathcal{F}_2$, *iff* $\forall \mathcal{A}. \exists \mathcal{S}. \mathcal{A}[\mathcal{F}_1] \preceq \mathcal{S}[\mathcal{F}_2]$ *where* $\mathcal{S}$ *is an adversary for* $\mathcal{F}_2$ *s.t.* $\mathsf{fnet}(\mathcal{A}[\mathcal{F}_1]) = \mathsf{fnet}(\mathcal{S}[\mathcal{F}_2])$.
- $\mathcal{F}_1$ *emulates* $\mathcal{F}_2$ *in the sense of* universally composable simulatability with dummy adversary, $\mathcal{F}_1 \leq^{\mathsf{UCDA}} \mathcal{F}_2$, *iff* $\exists \mathcal{S}. D[\mathcal{F}_1] \preceq \mathcal{S}[\mathcal{F}_2]$ *where* D *is the dummy adversary for* $\mathcal{F}_1$ *and* $\mathcal{S}$ *is an adversary for* $\mathcal{F}_2$ *such that* $\mathsf{fnet}(\mathcal{S}[\mathcal{F}_2]) = \mathsf{fnet}(D[\mathcal{F}_1])$.

THEOREM 16. *We have that* $\leq^{\mathsf{SS}} = \leq^{\mathsf{BB}} = \leq^{\mathsf{UC}} = \leq^{\mathsf{UCDA}}$.

The above security notions can also be defined replacing observational preorder by observational equivalence denoted $\leq^{\mathsf{SS}}_{\approx}, \leq^{\mathsf{BB}}_{\approx}, \leq^{\mathsf{UC}}_{\approx}$ and $\leq^{\mathsf{UCDA}}_{\approx}$. Surprisingly, the use of observational equivalence turns out to be too strong, ruling out natural secure emulation cases: for instance, the $\leq^{\mathsf{SS}}_{\approx}$ relation is not reflexive, due to the extra non-determinism that the simulator introduces. While symbolic analysis techniques typically rely on bisimulation relations, this is however consistent with the definitions proposed in the task-PIOA framework [6], which also allows non-deterministic executions for simulation based security.

4 Applications

We illustrate our framework by showing the secure emulation of a mutual authentication functionality by the Needham-Shroeder-Lowe (NSL) protocol [17]. As the NSL protocol uses public key encryption we first introduce in Section 4.1 functionalities for asymmetric

$$
\begin{array}{lcl}
\mathcal{P}_{\mathsf{pke}} & := & \mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\nu sk.\mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{KEY}, \mathsf{pk}(sk)\rangle). \\
& & (\text{let } io^i_{\mathsf{pke}} = io^1_{\mathsf{pke}} \text{ in } !\mathcal{P}_{\mathsf{enc}} \mid \text{let } io^i_{\mathsf{pke}} = io^2_{\mathsf{pke}} \text{ in } !\mathcal{P}_{\mathsf{enc}} \mid !\mathcal{P}_{\mathsf{dec}}) \\
\mathcal{P}_{\mathsf{enc}} & := & \mathsf{in}(io^i_{\mathsf{pke}}, \langle = \text{ENC}, m\rangle). \\
& & \nu r_2.\ \text{let } menc = \mathsf{aenc}(\langle \text{TAG}_0, m\rangle, \mathsf{pk}(sk), r_2) \text{ in } \mathsf{out}(io^i_{\mathsf{pke}}, \langle \text{CIPHER}, menc\rangle) \\
\mathcal{P}_{\mathsf{dec}} & := & \mathsf{in}(io^1_{\mathsf{pke}}, \langle = \text{DEC}, m\rangle). \\
& & \text{let } \langle = \text{TAG}_0, m_1\rangle = \mathsf{adec}(m, sk) \text{ in } \mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{PLAIN}, m\rangle)
\end{array}
$$

Figure 1: Real encryption functionality

encryption. Then, we briefly present the mutual authentication functionality and its realization through the NSL protocol. Finally we use the joint state composition result in Section 4.3 to obtain a result for an unbounded number of concurrent sessions.

4.1 Asymmetric encryption with joint state

We introduce a functionality for asymmetric encryption together with a *joint state composition result* which is crucial for composition of protocols that share key material. Even though encryption in a Dolev-Yao model is already idealized we will see that by introducing an *ideal functionality* for encryption we are able to obtain a joint state composition result. Throughout this section we rely on the following equational theory allowing us to model randomized asymmetric encryption:

$$\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y), z), y) = x \qquad \mathsf{testdec}(\mathsf{aenc}(x, \mathsf{pk}(y), z), y) = \mathsf{ok}.$$

Real encryption. This functionality (decribed in Figure 1) receives a channel name io^1_{pke}, which will be used for all sensitive information exchanges. A fresh private key sk is generated and the corresponding public key, i.e. $\mathsf{pk}(sk)$, is sent on io^1_{pke}. Then the process is ready to receive encryption or decryption requests. Note that encryption requests can be sent on the sensitive channel io^1_{pke} or on the public channel io^2_{pke} which is the channel the environment will typically use. Decryption requests are only available through the sensitive channel io^1_{pke} and thus will not be used by the attacker. Each time a decryption request is received on the channel io^1_{pke}, it tries to decrypt the ciphertext and checks whether the tag is TAG_0. If so, it outputs the plaintext on the channel io^1_{pke}. Otherwise, it does nothing.

Ideal functionality. We now propose, in Figure 2, an idealized version $\mathcal{F}_{\mathsf{pke}}$ of the real encryption functionality, which guarantees that the confidentiality of messages is preserved independently of any cryptanalytic effort that could be performed on ciphertexts from the knowledge of public keys. In various cryptographic settings [4, 5, 16], this is achieved by computing ciphertexts as the encryption of random messages instead of the actual plaintext. To be able to perform decryption, a table for plaintext/ciphertext associations is maintained. The burden of this association table is avoided in our symbolic specification by using two layers of encryption: messages are first encrypted using a secure key $\mathsf{pk}(ssk)$, then tagged and encrypted with the public key $\mathsf{pk}(sk)$ that is published during the initialization step. We stress that neither $\mathsf{pk}(ssk)$ nor ssk are ever transmitted by $\mathcal{F}_{\mathsf{pke}}$, guaranteeing that it is impossible to decrypt such a ciphertext outside the functionality, even if the key sk is adversarially chosen, which will be a crucial feature for our joint state composition theorem.

$$
\begin{array}{lcl}
\mathcal{F}_{\mathsf{pke}} & := & \mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\mathsf{out}(net, \text{INIT}).\mathsf{in}(net, \langle = \text{ALGO}, sk, tag\rangle).\mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{KEY}, \mathsf{pk}(sk)\rangle).\\
& & \nu ssk.\ (\text{let } io^i_{\mathsf{pke}} = io^1_{\mathsf{pke}} \text{ in } !\mathcal{F}_{\mathsf{enc}} \mid \text{let } io^i_{\mathsf{pke}} = io^2_{\mathsf{pke}} \text{ in } !\mathcal{F}_{\mathsf{enc}} \mid !\mathcal{F}_{\mathsf{dec}})\\
\mathcal{F}_{\mathsf{enc}} & := & \mathsf{in}(io^i_{\mathsf{pke}}, \langle = \text{ENC}, m\rangle).\nu r_1.\nu r_2.\\
& & \text{let } alea = \mathsf{aenc}(m, \mathsf{pk}(ssk), r_1) \text{ in let } menc = \mathsf{aenc}(\langle tag, alea\rangle, \mathsf{pk}(sk), r_2) \text{ in}\\
& & \mathsf{out}(io^i_{\mathsf{pke}}, \langle \text{CIPHER}, menc\rangle)\\
\mathcal{F}_{\mathsf{dec}} & := & \mathsf{in}(io^1_{\mathsf{pke}}, \langle = \text{DEC}, m\rangle).\ \text{let } \langle = tag, m_1\rangle = \mathsf{adec}(m, sk) \text{ in}\\
& & \text{if } \mathsf{testdec}(m_1, ssk) = \mathsf{ok} \text{ then } \mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{PLAIN}, \mathsf{adec}(m_1, ssk)\rangle)\\
& & \qquad\qquad \text{else } \mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{PLAIN}, m_1\rangle)
\end{array}
$$

Figure 2: Ideal encryption functionality

During the initialization, the attacker chooses the secret key sk and the tag that will be added in each encryption. Then a secure key ssk is generated and now the process is ready to receive encryption or decryption requests. Each time the process receives an encryption request, it computes the corresponding ciphertext and outputs the corresponding ciphertext. As explained above, the plaintext m is first encrypted using $\mathsf{pk}(ssk)$ before being tagged and encrypted with $\mathsf{pk}(sk)$. When the process receives a decryption request, it tries to decrypt the ciphertext and checks if the tag is the tag provided during the initialization. Then, it checks if the resulting plaintext is encrypted under $\mathsf{pk}(ssk)$. If so, this means that this ciphertext has been produced by the encryption functionality and thus has to be decrypted twice. Otherwise, the ciphertext has been produced by the attacker.

Realization. The real encryption functionality realizes the ideal one, i.e., $\mathcal{P}_{\mathsf{pke}} \leq^{\mathsf{SS}} \mathcal{F}_{\mathsf{pke}}$. This is witnessed by $\mathcal{A}_{\mathsf{pke}} = \nu net.(\mathsf{in}(net, = \text{INIT}).\, \nu sk.\, \mathsf{out}(net, (\text{ALGO}, sk, \text{TAG}_0)) \mid \_)$.

Composition with joint state. While $\leq^{\mathsf{SS}}$ is stable under replication this is not always sufficient to obtain composition guarantees. Indeed replication of a process also replicates all key generation operations. In order to obtain self-composition and inter-protocol composition with common key material we need a *joint state functionality*, i.e. a functionality that realizes $!\mathcal{F}_{\mathsf{pke}}$ while reusing the same key material. We actually consider the functionality $\underline{\mathcal{F}_{\mathsf{pke}}}$, which is a variant of $\mathcal{F}_{\mathsf{pke}}$ in which each message is tagged. More precisely, the process $\underline{\mathcal{F}_{\mathsf{pke}}}$ is defined as $\mathcal{F}_{\mathsf{pke}}$, except that: (i) the functionality begins with the instructions $\mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\mathsf{in}(io^1_{\mathsf{pke}}, sid)$ instead of $\mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}})$, (ii) each input of the form $\mathsf{in}(c, m)$ is replaced by $\mathsf{in}(c, \langle = sid, m\rangle)$, and (iii) each output of the form $\mathsf{out}(c, m)$ is replaced by $\mathsf{out}(c, \langle sid, m\rangle)$.

The joint state functionality $\mathcal{P}_{\mathsf{js}}[\underline{\mathcal{F}_{\mathsf{pke}}}]$ (see Figure 3) uses a single instance of $\underline{\mathcal{F}_{\mathsf{pke}}}$ for all protocol sessions. All the requests to the joint state functionality are received on the public channel io_{pke} in process $\mathcal{P}^1_{\mathsf{js}}$. They are then forwarded using the private IO channel *cont* to $\mathcal{P}^2_{\mathsf{js}}$. The process $\mathcal{P}^2_{\mathsf{js}}$ shares the private channel io_{pke} with $\underline{\mathcal{F}_{\mathsf{pke}}}$ and forwards all the requests after concatenating the session identifier to the plaintext. Then the response is again forwarded to the process $\mathcal{P}^1_{\mathsf{js}}$ which outputs the result on the public channel io_{pke}.

We now observe that the following joint state composition result holds. One instance of the encryption functionality can be used to emulate an unbounded number of such instances using the joint state process: $\mathcal{P}_{\mathsf{js}}[\mathcal{F}_{\mathsf{pke}}] \leq^{\mathsf{SS}} !\underline{\mathcal{F}_{\mathsf{pke}}}$.

$$
\begin{array}{lcl}
\mathcal{P}_{\mathsf{js}} & := & \nu cont.(\mathcal{P}^1_{\mathsf{js}} \mid \nu io_{\mathsf{pke}}, io^2_{\mathsf{pke}}.(P^2_{\mathsf{js}} \mid \_)) \\
\mathcal{P}^1_{\mathsf{js}} & := & \mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\mathsf{in}(io_1, sid).\mathsf{out}(cont, \langle sid, \text{INIT}\rangle). \\
 & & \mathsf{in}(cont, (= \text{KEY}, pk)).\mathsf{out}(io^1_{\mathsf{pke}}, \langle sid, \text{KEY}, pk\rangle). \\
 & & (\text{let } io^i_{\mathsf{pke}} = io^1_{\mathsf{pke}} \text{ in } !\mathcal{P}^1_{\mathsf{js-enc}} \mid \text{let } io^i_{\mathsf{pke}} = io^2_{\mathsf{pke}} \text{ in } !\mathcal{P}^1_{\mathsf{js-enc}} \mid !\mathcal{P}^1_{\mathsf{js-dec}} \mid \\
 & & !\mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\mathsf{in}(io_1, sid).\mathsf{out}(io^1_{\mathsf{pke}}, \langle sid, \text{KEY}, pk\rangle). \\
 & & (\text{let } io^i_{\mathsf{pke}} = io^1_{\mathsf{pke}} \text{ in } !\mathcal{P}^1_{\mathsf{js-enc}} \mid \text{let } io^i_{\mathsf{pke}} = io^2_{\mathsf{pke}} \text{ in } !\mathcal{P}^1_{\mathsf{js-enc}} \mid !\mathcal{P}^1_{\mathsf{js-dec}})) \\
\mathcal{P}^1_{\mathsf{js-enc}} & := & \mathsf{in}(io^i_{\mathsf{pke}}, \langle = sid, = \text{ENC}, m\rangle).\ \mathsf{out}(cont, \langle sid, \text{ENC}, m\rangle). \\
 & & \mathsf{in}(cont, \langle = \text{CIPHER}, c\rangle).\ \mathsf{out}(io^i_{\mathsf{pke}}, \langle sid, \text{CIPHER}, c\rangle) \\
\mathcal{P}^1_{\mathsf{js-dec}} & := & \mathsf{in}(io^1_{\mathsf{pke}}, \langle = sid, = \text{DEC}, c\rangle).\ \mathsf{out}(cont, \langle sid, dec, c\rangle). \\
 & & \mathsf{in}(cont, \langle = \text{PLAIN}, m\rangle).\ \mathsf{out}(io_{\mathsf{pke}}, \langle sid, \text{PLAIN}, m\rangle) \\
\mathcal{P}^2_{\mathsf{js}} & := & \mathsf{in}(cont, \langle sid, = \text{INIT}\rangle).\nu io'_{\mathsf{pke}}.\mathsf{out}(io_{\mathsf{pke}}, io'_{\mathsf{pke}}). \\
 & & \mathsf{in}(io'_{\mathsf{pke}}, \langle = \text{KEY}, pk\rangle).\mathsf{out}(cont, \langle \text{KEY}, pk\rangle).\ (!\mathcal{P}^2_{\mathsf{js-enc}} \mid !\mathcal{P}^2_{\mathsf{js-dec}}) \\
\mathcal{P}^2_{\mathsf{js-enc}} & := & \mathsf{in}(cont, \langle = sid, = \text{ENC}, m\rangle).\ \mathsf{out}(io'_{\mathsf{pke}}, \langle \text{ENC}, \langle sid, m\rangle\rangle). \\
 & & \mathsf{in}(io'_{\mathsf{pke}}, \langle = \text{CIPHER}, c\rangle).\ \mathsf{out}(cont, \langle \text{CIPHER}, c\rangle) \\
\mathcal{P}^2_{\mathsf{js-dec}} & := & \mathsf{in}(cont, \langle = sid, = \text{DEC}, c\rangle).\ \mathsf{out}(io'_{\mathsf{pke}}, \langle \text{DEC}, c\rangle). \\
 & & \mathsf{in}(io'_{\mathsf{pke}}, \langle = \text{PLAIN}, m\rangle).\ \mathsf{out}(cont, \langle \text{PLAIN}, m\rangle)
\end{array}
$$

Figure 3: Joint state IO-context

$$
\begin{array}{lcl}
\mathcal{A}_{\mathsf{js}} & := & \nu cs.(\mathcal{A}^1_{\mathsf{js}} \mid \nu net.(\mathcal{A}^2_{\mathsf{js}} \mid \_)) \\
\mathcal{A}^1_{\mathsf{js}} & := & \mathsf{in}(cs, \text{INIT}).\mathsf{out}(net, \text{INIT}).\mathsf{in}(net, \langle = \text{ALGO}, sk, tag\rangle).\mathsf{out}(cs, \langle \text{ALGO}, sk, tag\rangle) \\
\mathcal{A}^2_{\mathsf{js}} & := & \mathsf{in}(net, \langle sid, = \text{INIT}\rangle).\ \mathsf{out}(cs, \text{INIT}). \\
 & & \mathsf{in}(cs, \langle = \text{ALGO}, sk, tag\rangle).\ \mathsf{out}(net, \langle sid, \text{ALGO}, sk, \langle sid, tag\rangle\rangle). \\
 & & !\mathsf{in}(net, \langle sid', = \text{INIT}\rangle).\mathsf{out}(net, \langle sid', \text{ALGO}, sk, \langle sid', tag\rangle\rangle)
\end{array}
$$

Figure 4: Joint state adversary

This relation is witnessed by the adversary $\mathcal{A}_{\mathsf{js}}$ described in Figure 4 as we have that: $\mathcal{P}_{\mathsf{js}}[\mathcal{F}_{\mathsf{pke}}] \preceq \mathcal{A}_{\mathsf{js}}[!\underline{\mathcal{F}_{\mathsf{pke}}}]$. This adversary launches several functionalities with the same key sk. However, note that the session identifier sid used to tag each encryption associated could be different. The value of these session identifiers is selected by the attacker.

Note that it is crucial to introduce an ideal encryption functionality. We indeed have that $\mathcal{P}_{\mathsf{js}}[\mathcal{P}_{\mathsf{pke}}] \leq^{\mathsf{SS}} \mathcal{P}_{\mathsf{js}}[\mathcal{F}_{\mathsf{pke}}] \leq^{\mathsf{SS}} !\underline{\mathcal{F}_{\mathsf{pke}}}$ as well as $!\underline{\mathcal{P}_{\mathsf{pke}}} \leq^{\mathsf{SS}} !\underline{\mathcal{F}_{\mathsf{pke}}}$ (where $\underline{\mathcal{P}_{\mathsf{pke}}}$ is defined from $\mathcal{P}_{\mathsf{pke}}$ in the same way as $\underline{\mathcal{F}_{\mathsf{pke}}}$ from $\mathcal{F}_{\mathsf{pke}}$). However, $\mathcal{P}_{\mathsf{js}}[\mathcal{P}_{\mathsf{pke}}] \not\leq^{\mathsf{SS}} !\underline{\mathcal{P}_{\mathsf{pke}}}$. In particular $!\underline{\mathcal{P}_{\mathsf{pke}}}$ will provide multiple public keys while $\mathcal{P}_{\mathsf{js}}[\mathcal{P}_{\mathsf{pke}}]$ only provides a single one. Taking the more abstract ideal functionality allows this to be avoided by a simulator that chooses the same secret key for each instance of the functionality.

4.2 Mutual authentication

Because of lack of space we only briefly sketch the mutual authentication functionality. The details of the functionalities and the simulator are given in [10].

Ideal functionality for mutual authentication. The functionality $\mathcal{F}_{\mathsf{auth}}$ roughly works as follows. Both the initiator ($\mathcal{F}_{\mathsf{init}}$) and the responder ($\mathcal{F}_{\mathsf{resp}}$) receive a request for mutual authentication on their *io* channel. They forward this request to the adversary and, if both parties are honest, to a trusted host $\mathcal{F}_{\mathsf{th}}$ which compares these requests and authorizes going further if they match. Eventually, when the adversary asks to finish the protocol, then both participants complete the protocol session.

Realization of mutual authentication. The functionality $\mathcal{F}_{\mathsf{auth}}$ can be realized by a functionality $\mathcal{P}_{\mathsf{nsl}}$ implementing the well-known Needham-Schroeder-Lowe protocol [17]. We have that $\mathcal{P}_{\mathsf{nsl}} \leq^{\mathsf{SS}} \mathcal{F}_{\mathsf{auth}}$ by showing that $\mathcal{P}_{\mathsf{nsl}} \preceq \mathcal{S}[\mathcal{F}_{\mathsf{auth}}]$ for some $\mathcal{S}$.

4.3 From one to many sessions

We have that $\mathcal{P}_{\mathsf{nsl}} \leq^{\mathsf{SS}} \mathcal{F}_{\mathsf{auth}}$. This result only shows that $\mathcal{P}_{\mathsf{nsl}}$ is as secure as $\mathcal{F}_{\mathsf{auth}}$ for a single session of the protocol. By Proposition 14 we have that $!\mathcal{P}_{\mathsf{nsl}} \leq^{\mathsf{SS}} !\mathcal{F}_{\mathsf{auth}}$ but this does not correspond to the expected security for an unbounded number of sessions, as each session uses a different key. To show that $!\mathcal{F}_{\mathsf{auth}}$ can be realized with shared key material we use our joint state result. To apply this result we need the following technical lemma which allows pushing the replication under the restricted channel c.

LEMMA 17. *Let n be a name and c be a channel name such that $c \notin fn(P) \cup fn(Q)$.*

$$\nu c.\,!\,[\nu n.(\mathsf{out}(c,n) \mid P) \mid \mathsf{in}(c,x).Q] \;\preceq_\ell\; !\,\nu c.[\nu n.(\mathsf{out}(c,n) \mid P) \mid \mathsf{in}(c,x).Q].$$

This lemma allows us to apply the joint state result and obtain a result for an unbounded number of sessions sharing keys. Note that the joint state context uses a tagging mechanism.

5 Conclusions

This paper proposes a symbolic framework for the analysis of security protocols along the lines of the simulation based security approach, while adopting the applied pi calculus as its basic layer. We state central definitions and security notions, show general composition theorems and specific joint-state composition results for asymmetric encryption, and illustrate their use in the analysis of a mutual authentication protocol.

This framework brings the benefits of the secure composition theorems associated to simulation based security into the symbolic world, and opens the path to the analysis of more sophisticated protocols that can naturally be specified by the behavior of an ideal functionality. At a more fundamental level, we use preorder notions, which can be established by labeled simulation. While the use of labeled bisimulations is quite common in the applied pi calculus and has been integrated in automatic provers, the automation of proofs relying on labeled simulation appears as an interesting challenge for future works. Another direction for future work is to give a precise characterization of what properties are preserved by strong simulatability.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th ACM Symp. on Principles of Programming Languages (POPL'01)*. ACM, 2001.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. Technical Report 149, SRC, 1998.

[3] A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *Proc. 17th Int. Conference on Computer Aided Verification (CAV'05)*, LNCS. Springer, 2005.

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

[5] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symp. on Foundations of Computer Science (FOCS'01)*, 2001.

[6] R. Canetti, L. Cheung, D. Kaynar, N. Lynch, and O. Pereira. Compositional security for Task-PIOAs. In *Proc. 20th Computer Security Foundations Symposium (CSF'07)*, 2007.

[7] R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. Theory of Cryptography Conference (TCC'06)*, LNCS. Springer, 2006.

[8] I. Cervesato, A. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and fixing public-key kerberos. *Information and Computation*, 206(2-4):402–424, 2008.

[9] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Abstraction and refinement in protocol derivation. In *Proc. 17th Comp. Security Foundations Workshop (CSFW'04)*, 2004.

[10] S. Delaune, S. Kremer, and O. Pereira. Simulation based security in the applied pi calculus. Cryptology ePrint Archive, Report 2009/267. `http://eprint.iacr.org/`.

[11] S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi-calculus. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, LNCS. Springer, 2007.

[12] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game: A completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC'87)*. ACM Press, 1987.

[13] J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proc. 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, 2000.

[14] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW'06)*, 2006.

[15] R. Küsters, A. Datta, J. C. Mitchell, and A. Ramanathan. On the relationships between notions of simulation-based security. *Journal of Cryptology*, 21(4):492–546, 2008.

[16] R. Küsters and M. Tuengerthal. Joint State Theorems for Public-Key Encryption and Digitial Signature Functionalities with Local Computation. In *Proc. 21st IEEE Computer Security Foundations Symposium (CSF'08)*, 2008.

[17] G. Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[18] P. Mateus, J. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time calculus. In *Proc. 14th Conference on Concurrency Theory (CONCUR'03)*, LNCS. Springer, 2003.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

The Covering and Boundedness Problems for Branching Vector Addition Systems

Stéphane Demri[1], **Marcin Jurdziński**[2], **Oded Lachish**[2], **Ranko Lazić**[2]

[1]LSV, ENS Cachan, CNRS, INRIA Saclay, France

[2]DIMAP, Department of Computer Science, University of Warwick, UK

ABSTRACT. The covering and boundedness problems for branching vector addition systems are shown complete for doubly-exponential time.

1 Introduction

Vector addition systems (shortly, VAS), or equivalently Petri nets (e.g., [14]), are a fundamental model of computation, which is more expressive than finite-state machines and less than Turing-powerful. Decidability and complexity of a variety of problems have been extensively studied ([6] is a comprehensive survey).

A k-dimensional VAS consists of an initial vector of non-negative integers, and a finite set of vectors of integers, all of dimension k. Let us call the initial vector *axiom*, and the other vectors *rules*. A computation can then be thought of as a *derivation*: it starts with the axiom, and at each step, the next vector is derived from the current one by adding a rule. The vectors of interest are the ones derived *admissibly*, i.e. at the end of a derivation which is such that none of the vectors derived during it contains a negative entry.

Covering and boundedness are two central decision problems for VAS. The former asks whether a vector that is pointwise greater than or equal to a given vector can be admissibly derived, and the latter asks whether the set of all admissibly derived vectors is finite. In a landmark article [12], Rackoff showed that covering and boundedness for VAS are in EXPSPACE, matching Lipton's lower bound of EXPSPACE-hardness [10].* Considering the expressively equivalent VAS with states (shortly, VASS), Rosier and Yen refined the proofs of Lipton and Rackoff to obtain almost matching lower and upper bounds in terms of three parameters: the dimension, the binary size of the maximum absolute value of an entry in a rule, and the number of states [15]. Lipton's result was also extended by Mayr and Meyer to reversible Petri nets, which are equivalent to commutative semigroups [11]. Building further on Rosier and Yen's work, Habermehl showed that space exponential in the size of the system and polynomial in the size of the formula suffices for model checking the propositional linear-time μ-calculus on VASS, and he obtained a matching lower bound already for LTL on BPP [7].

*We recommend `http://rjlipton.wordpress.com/2009/04/08/an-expspace-lower-bound/`.

©Demri, Jurdziński, Lachish, Lazić; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 181–192
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2317

The following is a natural extension of VAS: instead of linearly, computation proceeds from the leaves to the root of a tree. For each node which is not a leaf, its vector is derived by summing the vectors derived at its children and adding a rule vector. The same condition of admissibility applies, i.e. no derived vector may contain a negative entry. This model of computation is branching VAS (shortly, BVAS).

In recent years, it has turned out that BVAS have interesting connections to a number of formalisms:

- BVAS correspond to a class of linear index grammars in computational linguistics [13];
- reachability (i.e. admissible derivability) for BVAS is decidable iff provability in multiplicative exponential linear logic is decidable [4];
- Verma and Goubault-Larrecq have extended the computation of Karp and Miller trees [8] to BVAS, and used it to draw conclusions about a class of equational tree automata which are useful for analysing cryptographic protocols [17];
- if first-order logic with 2 variables on finite data trees (which has applications to the XPath query language for XML) is decidable, then so is reachability for BVAS [1].

Covering and boundedness for BVAS are decidable easily using the branching extension of Karp and Miller's procedure [17]. However, the resulting algorithms do not operate in primitive recursive time or space, even in the linear case [16].

The main results we report are that, by switching from VAS to BVAS, covering and boundedness move two notches up the complexity hierarchy, to 2EXPTIME-complete.

For the 2EXPTIME-memberships, consider the following simple-minded idea for transferring knowledge about VAS derivations to the branching case:

✶ Every simple path from a leaf to the root in a BVAS derivation is a VAS derivation.

We show that the idea can give us mileage, but only after the following new insight, which is needed because the subderivations that grow off the simple path and hence contribute summands to it make the resulting VAS contain rules with unbounded positive entries.

☞ For VAS, we can obtain similar upper bounds to Rackoff's, but which depend only on the dimension and the minimum negative entry in a rule, i.e. not on the maximum positive entry in a rule.

The insight is at the centre of our proofs. In the case of covering, we show it essentially by inspecting carefully a proof of Rackoff, but in the case of boundedness, it relies on proving a new result on small solutions of integer programming problems, which extends a classical theorem of Borosh and Treybig and may also be a contribution of wider interest. To complete the proofs of the 2EXPTIME-memberships, we provide arguments for reducing the heights of appropriate BVAS derivations to at most doubly-exponential, and for why resulting small witnesses can be guessed and verified by alternating Turing machines in exponential space.

To obtain 2EXPTIME-hardness for covering and boundedness for BVAS, we extend the proof of Lipton to show that computations of alternating machines of size N with counters bounded by 2^{2^N} can be simulated in reverse by BVAS of size $O(N^2)$. Although universal branchings of alternating counter machines copy counter valutations whereas BVAS sum vectors derived at children nodes, the inner workings of Lipton's construction enable us to add a bit of machinery by which the BVAS can simulate the copying. We remark that, as is the case with Lipton's result, the lower bound is shown already for BVAS whose rules contain only entries -1, 0 or 1.

After fixing notations and making some preliminary observations in the next section, that covering and boundedness are in 2EXPTIME is shown in Sections 3 and 4, respectively. We then argue in Section 5 that both problems are 2EXPTIME-hard.

2 Preliminaries

Numbers, vectors and matrices. We write $\mathbb{N}_+$, $\mathbb{N}$ and $\mathbb{Z}$ for the sets of all positive, non-negative and arbitrary integers, respectively. Since we shall only work with integers, let the open interval (a, b) denote $(a, b) \cap \mathbb{Z}$, and analogously for half-open and closed intervals.

Given a dimension $k \in \mathbb{N}$, let $\mathbf{0}$ denote the zero vector and, for each $i \in [1, k]$, $\mathbf{e}_i$ denote the ith unit vector. For $\mathbf{v}, \mathbf{w} \in \mathbb{Z}^k$ and $B \in \mathbb{Z}$, we write:

- $\mathbf{v}(1), \ldots, \mathbf{v}(k)$ for the entries of $\mathbf{v}$;
- $\mathrm{supp}(\mathbf{v})$ for the set of all $i \in [1, k]$ such that $\mathbf{v}(i) \neq 0$;
- $\mathbf{v} \leq \mathbf{w}$ iff $\mathbf{v}(i) \leq \mathbf{w}(i)$ for all $i \in [1, k]$, and $\mathbf{v} < \mathbf{w}$ iff $\mathbf{v} \leq \mathbf{w}$ and $\mathbf{v} \neq \mathbf{w}$;
- $\min(B, \mathbf{v})$ for the vector $\langle \min\{B, \mathbf{v}(1)\}, \ldots, \min\{B, \mathbf{v}(k)\} \rangle$, and analogously for max;
- $\mathbf{v}^-$ for the vector $-\min(0, \mathbf{v})$, and $\mathbf{v}^+$ for the vector $\max(0, \mathbf{v})$.

For $\mathbf{v} \in \mathbb{N}^k$, let $\max(\mathbf{v}) = \max\{\mathbf{v}(1), \ldots, \mathbf{v}(k)\}$, where in case $k = 0$, we have $\max(\langle\rangle) = \max \emptyset = 0$. For finite $R \subseteq \mathbb{Z}^k$, let $\max(R^{-/+})$ denote $\max\{\max(\mathbf{r}^{-/+}) : \mathbf{r} \in R\}$, respectively.

Let $S^{k \times n}$ denote the set of all matrices with k rows, n columns and entries from S. Conveniently albeit slightly eccentrically, we use $-i$ for an index i to denote all rows or columns other than the ith, and $\bullet$ to denote all rows or columns. For example, $\mathbf{A}_{i\bullet}$ is row i of $\mathbf{A}$, and $\mathbf{A}_{\bullet(-j)}$ is $\mathbf{A}$ with column j removed.

Trees. A finite binary tree $\mathcal{T}$, which may contain nodes with one child, is a non-empty finite subset of $\{1, 2\}^*$ such that, for all $n \in \{1, 2\}^*$ and $i \in \{1, 2\}$, $n \cdot 2 \in \mathcal{T}$ implies $n \cdot 1 \in \mathcal{T}$, and $n \cdot i \in \mathcal{T}$ implies $n \in \mathcal{T}$. The nodes of $\mathcal{T}$ are its elements. The root of $\mathcal{T}$ is ε, the empty word. All notions such as parent, first child, second child, subtree and leaf, have their standard meanings. The height of $\mathcal{T}$ is the length, i.e. the number of nodes, of the longest simple path from the root to a leaf.

BVAS. The systems we define are equivalent to the branching vector addition systems with states [17] and the vector addition tree automata [4, 1]. To simplify our technical life, we work with stateless systems. In the linear case, it is well-known that states can be eliminated in logarithmic space, e.g. by adding the number of states to the dimension. For branching systems, the same is true, but computation steps that join two vectors by addition need to be generalised so that a vector from a fixed finite set (which may contain negative entries) is added also. Since we are not studying the systems as recognisers of languages, we do not have to work with alphabets either. Another simplification which costs only a logarithmic amount of space is in relation to the VATA [4], where branching up to a fixed finite arity was permitted. Hence, adopting a proof-theoretic terminology like that of Verma and Goubault-Larrecq [17], a system will consist of finite sets of axioms, unary rules and binary rules, all of which are simply integral vectors. The unary rules are present for easy compatibility with the linear case.

Let a *branching vector addition system* (*BVAS*) be a tuple $\mathcal{B} = \langle k, A_0, R_1, R_2 \rangle$, where:

- $k \in \mathbb{N}$ is the dimension;
- $A_0 \subseteq \mathbb{N}^k$ is a non-empty finite set of axioms;
- $R_1, R_2 \subseteq \mathbb{Z}^k$ are finite sets of unary and binary rules, respectively.

A derivation starts with a number of integral vectors, proceeds by applying the rules, and finishes with a single vector. Applying a unary rule means adding it to a derived vector, and applying a binary rule means adding it to the sum of two derived vectors. For a vector to be considered produced by the system, it needs to be derived by a derivation which starts with the axioms and whose derived vectors are all non-negative.

Formally, a *derivation* of $\mathcal{B}$ is a labelling $\mathcal{D} : \mathcal{T} \to \mathbb{Z}^k$ such that:

- $\mathcal{T}$ is a finite binary tree;
- if n has one child in $\mathcal{T}$, then $\mathcal{D}(n) \in R_1$;
- if n has two children in $\mathcal{T}$, then $\mathcal{D}(n) \in R_2$.

The vectors that are derived at every node are obtained recursively as follows:

- if n is a leaf in $\mathcal{T}$, then $\widehat{\mathcal{D}}(n) = \mathcal{D}(n)$;
- if n has one child n' in $\mathcal{T}$, then $\widehat{\mathcal{D}}(n) = \mathcal{D}(n) + \widehat{\mathcal{D}}(n')$;
- if n has two children n' and n'' in $\mathcal{T}$, then $\widehat{\mathcal{D}}(n) = \mathcal{D}(n) + \widehat{\mathcal{D}}(n') + \widehat{\mathcal{D}}(n'')$.

Now, we say that $\mathcal{D}$:

- is *initialised* iff, for each leaf n of $\mathcal{T}$, we have $\mathcal{D}(n) \in A_0$;
- is *admissible* iff, for each node n of $\mathcal{T}$, we have $\widehat{\mathcal{D}}(n) \in \mathbb{N}^k$;
- *derives* $\widehat{\mathcal{D}}(\varepsilon)$, which is the vector derived at the root.

For $\mathbf{v} \in \mathbb{N}^k$, we say that $\mathcal{B}$ *produces* $\mathbf{v}$ iff some initialised admissible derivation of $\mathcal{B}$ derives $\mathbf{v}$.

Substitutions and contractions. For finite binary trees $\mathcal{T}$ and $\mathcal{T}'$, and a node n of $\mathcal{T}$, let $\mathcal{T}[n \leftarrow \mathcal{T}']$ denote the tree obtained by replacing with $\mathcal{T}'$ the subtree of $\mathcal{T}$ rooted at n. To extend the notation to derivations, for $\mathcal{D} : \mathcal{T} \to \mathbb{Z}^k$ and $\mathcal{D}' : \mathcal{T}' \to \mathbb{Z}^k$, and a node n of $\mathcal{T}$, let $\mathcal{D}[n \leftarrow \mathcal{D}'] : \mathcal{T}[n \leftarrow \mathcal{T}'] \to \mathbb{Z}^k$ denote the derivation obtained by replacing with $\mathcal{D}'$ the subderivation of $\mathcal{D}$ rooted at n. Observe that the vector derived at node $n^\dagger$ in $\mathcal{D}[n \leftarrow \mathcal{D}']$ is:

- $\widehat{\mathcal{D}'}(n')$, if $n^\dagger$ corresponds to the node n' of $\mathcal{D}'$;
- $\widehat{\mathcal{D}}(n^\dagger) - \widehat{\mathcal{D}}(n) + \widehat{\mathcal{D}'}(\varepsilon)$, if $n^\dagger$ is an ancestor of n;
- $\widehat{\mathcal{D}}(n^\dagger)$, otherwise.

When $\mathcal{D}'$ has only one leaf n, we write $\mathcal{D}; \mathcal{D}'$ instead of $\mathcal{D}'[n \leftarrow \mathcal{D}]$.

For a derivation $\mathcal{D}$ and its nodes n and n' such that n is an ancestor of n', we write $\mathcal{D}[n \leftarrow n']$ instead of $\mathcal{D}[n \leftarrow \mathcal{D}']$, where $\mathcal{D}'$ is the subderivation of $\mathcal{D}$ rooted at n'. We call such substitutions *contracting*. For two derivations $\mathcal{D}^\dagger$ and $\mathcal{D}^\ddagger$, we say that $\mathcal{D}^\ddagger$ is a *contraction* of $\mathcal{D}^\dagger$ iff $\mathcal{D}^\ddagger$ is obtained from $\mathcal{D}^\dagger$ by a finite sequence of contracting substitutions.

VAS. The classical vector addition systems can be defined as BVAS of the form $\mathcal{V} = \langle k, \{\mathbf{a}\}, R, \emptyset \rangle$, i.e. with one axiom and no binary rules. We may write them as just $\langle k, \mathbf{a}, R \rangle$.

All the definitions for BVAS apply to VAS, but they simplify. For each derivation $\mathcal{D} : \mathcal{T} \to \mathbb{Z}^k$, its underlying tree $\mathcal{T}$ is a sequence.

Restrictions and bounds. For k-dimensional X, and $I \subseteq [1,k]$, we write $X(I)$ for the "restriction of X to the set of places I", e.g.: $\mathbf{v}(I)$ is the vector obtained from $\mathbf{v}$ by removing the entries in places outside of I; $\langle k, \mathbf{a}, R\rangle(I)$ is the $|I|$-dimensional VAS obtained from $\langle k, \mathbf{a}, R\rangle$ by replacing $\mathbf{a}$ with $\mathbf{a}(I)$, and by replacing every rule $\mathbf{r} \in R$ with $\mathbf{r}(I)$; and $\mathcal{D}(I)$ is the derivation obtained from $\mathcal{D}$ by replacing, for every node n, the label $\mathcal{D}(n)$ of n with $\mathcal{D}(n)(I)$.

For $\mathbf{v} \in \mathbb{Z}^k$ and $B \in \mathbb{N}$, we say that $\mathbf{v}$ is *B-bounded* iff $\mathbf{v} \in [0, B-1]^k$. We regard a derivation *B-bounded* iff all the vectors derived at its nodes are B-bounded. Thus, B-boundedness implies admissibility.

For a k-dimensional vector or derivation X, and $I \subseteq [1,k]$, we say that X is *I-B-bounded* iff $X(I)$ is B-bounded.

Decision problems. We study the complexity of the following problems. As is standard, the input sizes are with respect to binary representations of integers.

Covering Given a BVAS $\mathcal{B}$ and a target non-negative vector $\mathbf{t}$ of the same dimension, does $\mathcal{B}$ produce some $\mathbf{v}$ such that $\mathbf{v} \geq \mathbf{t}$?

Boundedness Given a BVAS, is the set of all vectors that it produces finite?

THEOREM 1. *[10, 12] Covering and boundedness for VAS are* EXPSPACE*-complete.*

THEOREM 2. *[17] Covering and boundedness for BVAS are decidable.*

3 Upper bound for the covering problem

We say that a derivation $\mathcal{D}$ of a BVAS $\mathcal{B}$ is a *covering* of a vector $\mathbf{t}$ iff the vector that $\mathcal{D}$ derives is at least $\mathbf{t}$, i.e. $\widehat{\mathcal{D}}(\varepsilon) \geq \mathbf{t}$. Thus, the covering problem asks whether there exists an initialised admissible covering.

For VAS, Rackoff [12] established EXPSPACE-membership of the covering problem by showing that, if an initialised admissible covering exists, then there must exist one of at most doubly-exponential length. Such a "short" covering can be guessed and verified in non-deterministic exponential space, and determinism is regained by Savitch's Theorem.

More precisely, Rackoff proved:

LEMMA 3. *[12, Section 3] If a VAS $\langle k, \mathbf{a}, R\rangle$ has an initialised admissible covering of $\mathbf{t} \in \mathbb{N}^k$, then it has one whose length is at most $2^{(3L)^{k+1}}$, where $L = \max\{\text{size}(R), \text{size}(\mathbf{t})\}$.*

Now, the following proof scheme suggests itself for showing that, if a k-dimensional BVAS $\mathcal{B}$ has an initialised admissible covering $\mathcal{D}$ of $\mathbf{t}$, then it has one of at most doubly-exponential height:

(i) If $\mathcal{D}$ has an excessively high leaf n, let $\mathcal{V}$ be the VAS whose axiom is $\mathcal{D}(n)$ and whose rules R are all the vectors:
   - $\mathcal{D}(n')$, such that n' is on the path π from n to the root, and has one child;
   - $\mathcal{D}(n') + \widehat{\mathcal{D}}(n'')$, such that n' is on π, and n'' is a child of n' not on π.

   Hence, the sequence obtained from π by relabelling the nodes with two children as specified is a derivation $\mathcal{D}^\dagger$ of $\mathcal{V}$. The vectors derived along $\mathcal{D}^\dagger$ are the same as the vectors derived along π in $\mathcal{D}$, so $\mathcal{D}^\dagger$ is an initialised admissible covering of $\mathbf{t}$.

(ii) By Lemma 3, $\mathcal{V}$ has an initialised admissible covering $\mathcal{D}^\ddagger$ of $\mathbf{t}$ with length at most $2^{(3L)^{k+1}}$, where $L = \max\{\text{size}(R), \text{size}(\mathbf{t})\}$.

(iii) Let $\mathcal{D}'$ be a derivation of $\mathcal{B}$ obtained from $\mathcal{D}^\ddagger$ by undoing the linearisation done in (i), i.e. by unfolding each rule in $\mathcal{D}^\ddagger$ which is not a unary rule of $\mathcal{B}$ into a binary rule of $\mathcal{B}$ and a subderivation of $\mathcal{D}$. It is straightforward to check that $\mathcal{D}'$ is also an initialised admissible covering of $\mathbf{t}$. We repeat from (i) with $\mathcal{D}'$ instead of $\mathcal{D}$, until there are no excessively high leaves.

There are, unfortunately, two obstacles:

- Since the definition of R in (i) involves adding derived vectors (the ones at the nodes one edge away from the path π), we have no bound on $\text{size}(R)$ in terms of $\text{size}(\mathcal{B})$ and $\text{size}(\mathbf{t})$, and therefore neither on L in (ii).
- Even if we manage to bound L, Lemma 3 gives us no guarantees about the shape of $\mathcal{D}^\ddagger$ in (ii) in relation to the shape of $\mathcal{D}^\dagger$. Hence, although the length of $\mathcal{D}^\ddagger$ is bounded, we are not able to deduce that after the unfolding in (iii), $\mathcal{D}'$ has fewer excessively high leaves than $\mathcal{D}$.

However, the key to overcoming both obstacles is observing that essentially Rackoff's proof of Lemma 3 shows more than is stated in that result! Firstly, any initialised admissible covering has a contraction which is a short initialised admissible covering, and secondly, the length of the latter is bounded by the sizes of the target vector and only the negative entries in the rules of the VAS. More precisely, we have:

LEMMA 4. *If a VAS $\langle k, \mathbf{a}, R\rangle$ has an initialised admissible covering $\mathcal{D}$ of $\mathbf{t} \in \mathbb{N}^k$, then it has one which is a contraction of $\mathcal{D}$ and whose length is at most* $(\max(R^-) + \max(\mathbf{t}) + 2)^{(3k)!}$.

We are now in a position to show that, indeed, if a given BVAS has an initialised admissible covering of a given vector of non-negative integers, then it has one of at most doubly-exponential height. Although that is all that is required in this article, the actual statement is stronger for the record.

LEMMA 5. *If a BVAS $\langle k, A_0, R_1, R_2\rangle$ has an initialised admissible covering $\mathcal{D}$ of $\mathbf{t} \in \mathbb{N}^k$, then it has one which is a contraction of $\mathcal{D}$ and whose height is at most* $(\max((R_1 \cup R_2)^-) + \max(\mathbf{t}) + 2)^{(3k)!}$.

Therefore, to decide the covering problem, it suffices to search for an initialised admissible covering of at most doubly-exponential height. Note, however, that the size of a binary tree of doubly-exponential height can be triply-exponential, and hence vectors derived in a derivation of doubly-exponential height may contain triply-exponential entries. In order to prove the main result of this section, i.e., that the covering problem for is in 2EXPTIME, we need to avoid having to manipulate such large numbers. That is achieved by our next result, Proposition 6, which shows that for a large enough bound B, whether a derivation is admissible and a covering can be verified accurately even if entries in the derived vectors are truncated to be at most B.

For a derivation $\mathcal{D} : \mathcal{T} \to \mathbb{Z}^k$ and $B \in \mathbb{N}$, we define the *B-truncated* derived vectors by:

- if n is a leaf in $\mathcal{T}$, then $\widehat{\mathcal{D}}^B(n) = \min(B, \mathcal{D}(n))$;
- if n has one child n' in $\mathcal{T}$, then $\widehat{\mathcal{D}}^B(n) = \min(B, \mathcal{D}(n) + \widehat{\mathcal{D}}^B(n'))$;
- if n has two children n' and n'' in $\mathcal{T}$, then $\widehat{\mathcal{D}}^B(n) = \min(B, \mathcal{D}(n) + \widehat{\mathcal{D}}^B(n') + \widehat{\mathcal{D}}^B(n''))$.

PROPOSITION 6. *Suppose $\mathcal{B} = \langle k, A_0, R_1, R_2\rangle$ is a BVAS, $\mathbf{t} \in \mathbb{N}^k$, $\mathcal{D}$ is a derivation in $\mathcal{B}$ of height at most H, and $B \geq H \cdot \max((R_1 \cup R_2)^-) + \max(\mathbf{t})$. Then $\mathcal{D}$ is an admissible covering of $\mathbf{t}$ iff, for each node n in $\mathcal{D}$, $\widehat{\mathcal{D}}^B(n) \geq \mathbf{0}$, and $\widehat{\mathcal{D}}^B(\varepsilon) \geq \mathbf{t}$.*

THEOREM 7. *Covering for BVAS is in* 2EXPTIME.

PROOF. Let $\mathcal{B} = \langle k, A_0, R_1, R_2\rangle$ be a BVAS and $\mathbf{t} \in \mathbb{N}^k$. Let $N = \text{size}(\mathcal{B}) + \text{size}(\mathbf{t})$. If $\ell = \max((R_1 \cup R_2)^-) + \max(\mathbf{t}) + 2$ then $\ell \leq 2^N$, and without any loss of generality we can assume that $3k \leq N$.

Lemma 5 implies that if there is an initialised admissible covering of $\mathbf{t}$ in $\mathcal{B}$ then there is one of height at most $\ell^{(3k)!} \leq (2^N)^{N!} \leq 2^{2^{C_1 N \log N}}$, for some constant $C_1 > 1$. If we set $H = 2^{2^{C_1 N \log N}}$ and $B = H^2$, then from Proposition 6 it follows that in order to establish existence of an initialised admissible covering of $\mathbf{t}$ in $\mathcal{B}$, it suffices to:

- guess an initialised derivation $\mathcal{D}$ in $\mathcal{B}$ of height at most H;
- guess the B-truncated derived vectors at all nodes in $\mathcal{D}$, and for every node and its children, verify that they satisfy the equations defining B-truncated derived vectors, and that they are non-negative;
- verify that the B-truncated derived vector at the root covers $\mathbf{t}$.

We argue that the guessing and verification of such a structure of at most triply-exponential size can be carried out by an alternating Turing machine with exponential space, and hence the covering problem is in 2EXPTIME [3]. The alternating Turing machine starts at the root of the derivation, it uses non-deterministic states to guess the rules labelling the current node and its children, and their B-truncated derived vectors, and it uses universal states to proceed with the guessing and verification process to both children (for nodes labelled by binary rules) in parallel. All those tasks can indeed be carried out by a Turing machine with only exponential space because it can represent—in binary—and manipulate numbers of doubly-exponential magnitude. ∎

4 Upper bound for the boundedness problem

Let us say that a derivation $\mathcal{D}$ is *self-covering* iff, for some node n, the vector derived at n is less than or equal to the one at the root, and less in at least one place, i.e. $\widehat{\mathcal{D}}(n) < \widehat{\mathcal{D}}(\varepsilon)$.

The following fact tells us that boundedness is equivalent to non-existence of an initialised admissible self-covering derivation. The "if" part is easy. The "only if" part was inferred by Verma and Goubault-Larrecq, using the properties of their extension of Karp and Miller's procedure.

THEOREM 8. *[17] A BVAS produces infinitely many vectors iff it has an initialised admissible self-covering derivation.*

In the simpler setting of VAS, to conclude that boundedness is in EXPSPACE, Rackoff showed that if an initialised admissible self-covering derivation exists, then there exists one of at most doubly-exponential length:

LEMMA 9. *[12, Section 4] If a VAS $\mathcal{V} = \langle k, \mathbf{a}, R\rangle$ has an initialised admissible self-covering derivation, then it has one whose length is at most $2^{2^{\mathsf{C}_2 L \log L}}$, where $L = \mathrm{size}(R)$ and C_2 is some constant.*

Encouraged by our eventual success in Section 3, consider the following scheme for proving that, if a BVAS $\mathcal{B} = \langle k, A_0, R_1, R_2\rangle$ has an initialised admissible self-covering derivation $\mathcal{D}$, then it has one of at most doubly-exponential height:

(I) Let node n be such that $\widehat{\mathcal{D}}(n) < \widehat{\mathcal{D}}(\varepsilon)$, and pick a simple path π in $\mathcal{D}$ which is from a leaf to the root and passes through n. Let $\mathcal{V}$ be the VAS defined as in (i) in Section 3, i.e. its axiom is the label of the leaf of π and its rules R are obtained by linearising the binary rules on π. Thus, $\mathcal{V}$ has a derivation $\mathcal{D}^\dagger$ whose sequence of derived vectors is the same as the sequence of derived vectors along π in $\mathcal{D}$. In particular, $\mathcal{D}^\dagger$ is initialised, admissible and self-covering.

(II) By Lemma 9, $\mathcal{V}$ has an initialised admissible self-covering derivation $\mathcal{D}^\ddagger$ whose length is at most $2^{2^{\mathsf{C}_2 L \log L}}$, where $L = \mathrm{size}(R)$.

(III) Let $\mathcal{D}'$ be a derivation of $\mathcal{B}$ obtained from $\mathcal{D}^\ddagger$ by undoing the linearisation done in (I), as in (iii) in Section 3, and let π' be the path in $\mathcal{D}'$ that is from a leaf to the root and corresponds to $\mathcal{D}^\ddagger$. It is straightforward to check that $\mathcal{D}'$ is also initialised, admissible and self-covering.

(IV) Let H be the length of π', which equals the length of $\mathcal{D}^\ddagger$. For each node n' that is one edge away from π' in $\mathcal{D}'$ (i.e., that was attached in (III)), the subderivation of $\mathcal{D}'$ rooted at n' is an initialised admissible covering of $\min((H-1)\cdot\max(R^-)+1, \widehat{\mathcal{D}'}(n'))$. By Lemma 5, $\mathcal{B}$ has an initialised admissible covering $\mathcal{D}^*_{n'}$ of the same vector, whose height is at most

$$\begin{aligned}\Big(\max((R_1\cup R_2)^-) + \max\Big(\min\Big((H-1)\cdot\max(R^-)+1, \widehat{\mathcal{D}'}(n')\Big)\Big) + 2\Big)^{(3k)!} \\ \leq \big(\max((R_1\cup R_2)^-) + (H-1)\cdot\max(R^-) + 3\big)^{(3k)!} \\ \leq \big(H\cdot\max((R_1\cup R_2)^-) + 3\big)^{(3k)!}.\end{aligned}$$

Let $\mathcal{D}''$ be obtained from $\mathcal{D}'$ by performing each substitution $[n' \leftarrow \mathcal{D}^*_{n'}]$. The truncating threshold $(H-1)\cdot\max(R^-)+1$ is such that $\mathcal{D}''$ is still admissible and self-covering, certainly it is still initialised, and $H + (H\cdot\max((R_1\cup R_2)^-)+3)^{(3k)!}$ bounds its height.

Of course, we have the same problem as the first one in Section 3: we have no bound on $\mathrm{size}(R)$ in terms of $\mathrm{size}(\mathcal{B})$, and therefore neither on H in (IV). Seeking therefore a refinement of Lemma 9, we find that the key ingredient in its proof is:

LEMMA 10. *[12, Lemma 4.5] Suppose $\mathcal{V} = \langle k, \mathbf{a}, R\rangle$ is a VAS, $I \subseteq [1,k]$ and $B > 1$. If $\mathcal{V}$ has an initialised I-B-bounded self-covering derivation, then it has one whose length is at most $B^{(\mathrm{size}(R))^{\mathsf{C}_3}}$, where C_3 is some constant.*

In turn, at the centre of the proof of Lemma 10, Rackoff invokes the following theorem of Borosh and Treybig on small solutions of integer linear programming problems. Recall that the interval notations denote sets of integers.

THEOREM 11. *[2] Let* $\mathbf{A} \in (-m, m)^{k \times n}$ *and* $\mathbf{b} \in (-m, m)^k$*, where* $k, n, m \in \mathbb{N}$*. If there exists* $\mathbf{x} \in \mathbb{N}^n$ *such that* $\mathbf{Ax} \geq \mathbf{b}$*, then there exists* $\mathbf{y} \in [0, (\max\{n, m\})^{\mathsf{C}_4 k}]^n$ *such that* $\mathbf{Ay} \geq \mathbf{b}$*, where* C_4 *is some constant.*

When we examine feeding a VAS $\langle k, \mathbf{a}, R \rangle$ for which we have a bound on $\max(R^-)$ but not on $\max(R^+)$ into Rackoff's proof of Lemma 10, we discover that Theorem 11 is invoked for bounded k, unbounded n, $\mathbf{A}$ whose entries are bounded below but not above, and $\mathbf{b}$ whose entries are bounded above but not below. Surprisingly, this is where we can make progress. We now show that, if we can afford roughly one exponential more, small solutions exist for $\mathbf{A}$ and $\mathbf{b}$ which are only one-sidedly bounded by m. Moreover, the number of non-zero entries in the small solutions and their values are bounded only in terms of k and m.

THEOREM 12. *Let* $\mathbf{A} \in (-m, \infty)^{k \times n}$ *and* $\mathbf{b} \in (-\infty, m)^k$*, where* $k, n, m \in \mathbb{N}$*. If there exists* $\mathbf{x} \in \mathbb{N}^n$ *such that* $\mathbf{Ax} \geq \mathbf{b}$*, then there exists* $\mathbf{y} \in [0, L]^n$ *such that* $|\mathrm{supp}(\mathbf{y})| \leq L$ *and* $\mathbf{Ay} \geq \mathbf{b}$*, where* $L = m^{2^{\mathsf{C}_5 k^2}}$ *and* C_5 *is some constant.*

In order to reformulate Theorem 12 so that it becomes appropriate for a proof by induction on k (cf. Lemma 14), we define $F_k(m)$, for all integers $k \geq 1$ and $m \geq 2$, by:

$$F_k(m) = \begin{cases} m & \text{if } k = 1, \\ \big(F_{k-1}(2m)\big)^{4\mathsf{C}_4 k^2} & \text{if } k > 1, \end{cases}$$

where C_4 is the constant from Theorem 11, which we can assume is at least 1.

PROPOSITION 13. *For all integers* $k \geq 1$ *and* $m \geq 2$*, we have* $F_k(m) \leq m^{(4\mathsf{C}_4)^k \cdot (2k)!}$.

Observe that there is a constant C_5 such that, for all integers $k \geq 1$ and $m \geq 2$, we have $F_k(m) \leq m^{(4\mathsf{C}_4)^k \cdot (2k)!} \leq m^{2^{\mathsf{C}_5 k^2}}$. Hence, and since Theorem 12 is true trivially when $k = 0$ or $m \leq 1$, Theorem 12 follows from the following lemma.

LEMMA 14. *Let* $\mathbf{A} \in (-m, \infty)^{k \times n}$ *and* $\mathbf{b} \in (-\infty, m)^k$*, where* $k \geq 1$*,* n *and* $m \geq 2$ *are integers. If there exists* $\mathbf{x} \in \mathbb{N}^n$ *such that* $\mathbf{Ax} \geq \mathbf{b}$*, then there exists* $\mathbf{y} \in [0, F_k(m)]^n$ *such that* $|\mathrm{supp}(\mathbf{y})| \leq F_k(m)$ *and* $\mathbf{Ay} \geq \mathbf{b}$.

PROOF. We can assume without any loss of generality that, for each $j \in [1, n]$, there exists $\mathbf{x} \in \mathbb{N}^n$ such that $\mathbf{Ax} \geq \mathbf{b}$ and $\mathbf{x}(j) \geq 1$. Otherwise, consider $\mathbf{A}' = \mathbf{A}_{\bullet(-j)}$, where there exists no $\mathbf{x} \in \mathbb{N}^n$ such that $\mathbf{Ax} \geq \mathbf{b}$ and $\mathbf{x}(j) \geq 1$.

The proof is by induction on k. First we consider the base case when $k = 1$. If $\mathbf{b} \leq 0$ then $\mathbf{Ay} \geq \mathbf{b}$ for $\mathbf{y} = \mathbf{0}$. If, however, $\mathbf{b} > 0$ then the existence of $\mathbf{x} \in \mathbb{N}^n$ such that $\mathbf{Ax} \geq \mathbf{b}$ implies that there must be $i \in [1, n]$ such that $\mathbf{A}(1, i) > 0$. Then, we have $\mathbf{Ay} \geq \mathbf{b}$ for $\mathbf{y} = m \cdot \mathbf{e}_i$.

For the inductive step we consider the following three cases. Essentially, if either $\mathbf{b}$ contains a large negative entry or $\mathbf{A}$ contains a large positive entry, then we remove that row of $\mathbf{A}$ and argue by the inductive hypothesis and the largeness of the entry. Otherwise, we have a lower bound for all entries of $\mathbf{b}$ and an upper bound for all entries of $\mathbf{A}$, and we invoke Theorem 11.

Case 1. *There exists $i \in [1,k]$ such that $\mathbf{b}(i) \leq -m \cdot (F_{k-1}(m))^2$.* Let $\mathbf{A}' = \mathbf{A}_{(-i)\bullet}$ and let $\mathbf{b}' = \mathbf{b}_{-i}$. By the inductive hypothesis, there exists $\mathbf{y} \in [0, F_{k-1}(m)]^n$—and hence $\mathbf{y} \in [0, F_k(m)]^n$—such that $|\mathrm{supp}(\mathbf{y})| \leq F_{k-1}(m) < F_k(m)$ and $\mathbf{A}'\mathbf{y} \geq \mathbf{b}'$. The assumption that $\mathbf{A}(i,j) > -m$ for all $j \in [1,n]$ then implies that $\mathbf{A}_{i\bullet}\mathbf{y} > -m \cdot (F_{k-1}(m))^2 \geq \mathbf{b}(i)$, and hence we have $\mathbf{A}\mathbf{y} \geq \mathbf{b}$.

Case 2. *There exist $i \in [1,k]$ and $j \in [1,n]$ such that $\mathbf{A}(i,j) \geq 2m \cdot (F_{k-1}(2m))^2$, and there exists $\mathbf{x} \in \mathbb{N}^n$ such that $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ and $\mathbf{x}(j) \geq 1$.* Let $\mathbf{A}' = \mathbf{A}_{(-i)\bullet}$, let $\mathbf{b}' = \mathbf{b}_{-i}$, and let $\mathbf{b}'' = \mathbf{b}' - \mathbf{A}_{(-i)j}$. Note that $\mathbf{A}'(\mathbf{x} - \mathbf{e}_j) \geq \mathbf{b}''$ and that, since $\mathbf{x}(j) \geq 1$, we have $\mathbf{x} - \mathbf{e}_j \in \mathbb{N}^n$. Observe also that $\mathbf{b}'' \in (-\infty, 2m)^{k-1}$ and hence, by the inductive hypothesis, there exists $\mathbf{y} \in [0, F_{k-1}(2m)]^n$ such that $|\mathrm{supp}(\mathbf{y})| \leq F_{k-1}(2m)$ and $\mathbf{A}'\mathbf{y} \geq \mathbf{b}''$.

Let $\mathbf{z} = \mathbf{y} + \mathbf{e}_j$. Note that then $\mathbf{z} \in [0, F_{k-1}(2m) + 1]^n \subseteq [0, F_k(m)]^n$ and $|\mathrm{supp}(\mathbf{y})| \leq F_{k-1}(2m) + 1 \leq F_k(m)$, and hence we only need to establish that $\mathbf{A}\mathbf{z} \geq \mathbf{b}$. We have:

$$(\mathbf{A}\mathbf{z})(i) = \mathbf{A}_{i\bullet}(\mathbf{y} + \mathbf{e}_j) \geq \mathbf{A}(i,j) - m \cdot (F_{k-1}(2m))^2 \geq m \cdot (F_{k-1}(2m))^2 \geq m \geq \mathbf{b}(i),$$

where the first inequality follows from $\mathbf{A} \in (-m, \infty)^{k \times n}$, from $\mathbf{y} \in [0, F_{k-1}(2m)]$, and from $|\mathrm{supp}(\mathbf{y})| \leq F_{k-1}(2m)$; and the second inequality follows from the assumption that $\mathbf{A}(i,j) \geq 2m \cdot (F_{k-1}(2m))^2$. Moreover, we have:

$$(\mathbf{A}\mathbf{z})_{-i} = \mathbf{A}'(\mathbf{y} + \mathbf{e}_j) = \mathbf{A}'\mathbf{y} + \mathbf{A}_{(-i)j} \geq \mathbf{b}'' + \mathbf{A}_{(-i)j} = \mathbf{b}' = \mathbf{b}_{-i}.$$

Case 3. *Neither Case 1 nor Case 2 applies.* Observe that, in this case, every column of $\mathbf{A}$ is in $[-m, 2m \cdot (F_{k-1}(2m))^2]^k$, and $\mathbf{b} \in [-m \cdot (F_{k-1}(m))^2, m]^k$. The number of distinct columns of $\mathbf{A}$ is therefore at most $(3m \cdot (F_{k-1}(2m))^2)^k \leq (F_{k-1}(2m))^{4k}$, and so without loss of generality we may assume $n \leq (F_{k-1}(2m))^{4k}$. By Theorem 11, there exists $\mathbf{y} \in [0, F_{k-1}(2m)^{4\mathsf{C}_4 k^2}]^n = [0, F_k(m)]^n$ such that $|\mathrm{supp}(\mathbf{y})| \leq (F_{k-1}(2m))^{4k} \leq F_k(m)$ and $\mathbf{A}\mathbf{y} \geq \mathbf{b}$. ∎

By substituting the use of Theorem 11 in Rackoff's proof of Lemma 10 by a use of Theorem 12, we obtain:

LEMMA 15. *Suppose $\mathcal{V} = \langle k, \mathbf{a}, R \rangle$ is a VAS, $I \subseteq [1,k]$ and $B > 1$. If $\mathcal{V}$ has an initialised I-B-bounded self-covering derivation, then it has one of length at most $((\max(R^-) + 1) \cdot B)^{2^{\mathsf{C}_6 k^2}}$, where C_6 is some constant.*

The final step in obtaining a revision of Lemma 9 that we can apply to VAS whose rules are bounded below but not above is to substitute in its proof uses of Lemma 10 by uses of Lemma 15. That yields the following result, which shows that we could indeed afford the extra exponential in Theorem 12. Although it filters through to Lemma 15, it gets swallowed by the steps of Rackoff's inductive proof of Lemma 9.

LEMMA 16. *If a VAS $\mathcal{V} = \langle k, \mathbf{a}, R \rangle$ has an initialised admissible self-covering derivation, then it has one of length at most $(2(\max(R^-) + 1))^{2^{\mathsf{C}_7 k^3}}$, where C_7 is some constant.*

THEOREM 17. *Boundedness for BVAS is in* 2EXPTIME.

5 Lower bounds

Let a *counter machine* consist of finite sets of states, counters and transitions. Each transition changes state, and either increments a counter, or checks that a counter is positive and decrements it, or checks that a counter is zero. We consider alternating counter machines, where the set of states is partitioned into non-deterministic and universal. Without loss of generality, we restrict to at most binary branching. A computation of such a machine is a binary tree of configurations, each of which is a state together with a non-negative integer for every counter.

To establish lower bounds for the covering and boundedness problems for BVAS, we reduce from the following problem. Its AEXPSPACE-hardness is an easy consequence of standard translations from Turing machines to counter machines (e.g., by simulating the tape by two stacks and encoding the latter by counters), and so it is 2EXPTIME-hard [3].

Doubly-exponential halting Given an alternating counter machine of size N with an initial state and a halting state, does it have an initialised 2^{2^N}-bounded halting computation, i.e. whose root is the initial state with 0 for every counter, in which every counter value is less than 2^{2^N}, and which is finite and such that the state of each leaf is halting?

We argue that, given an alternating counter machine $\mathcal{M}$ of size N, a BVAS $\mathcal{B}_\mathcal{M}$ which simulates $\mathcal{M}$ and is of size $O(N^2)$ is computable:

- For simulating the operations on counters, we employ Lipton's construction [10] (cf. the nice presentation by Esparza [5, Section 7]), in which each counter c of $\mathcal{M}$ is represented by two places p_c and $\overline{p_c}$ of $\mathcal{B}_\mathcal{M}$, and it is an invariant in all initialised admissible derivations of $\mathcal{B}_\mathcal{M}$ that the sum of p_c and $\overline{p_c}$ is 2^{2^N}. Increments and decrements of c are easy, but to simulate checking that c is zero, $\mathcal{B}_\mathcal{M}$ uses implementations of two auxiliary counters bounded by $2^{2^{N-1}}$ to decrement $\overline{p_c}$ exactly $2^{2^{N-1}} \cdot 2^{2^{N-1}} = 2^{2^N}$ times. The implementations of the two auxiliary counters in turn require two auxiliary counters bounded by $2^{2^{N-2}}$ etc.
- The simulation is performed in reverse, so that $\mathcal{B}_\mathcal{M}$ guesses and verifies an initialised 2^{2^N}-bounded halting computation of $\mathcal{M}$. To verify a universal branching, where the two child configurations of $\mathcal{M}$ are represented by two derived vectors $\mathbf{v}$ and $\mathbf{v}'$, $\mathcal{B}_\mathcal{M}$ derives $\mathbf{v}''$ from $\mathbf{v}'$ by transferring each pair of places that represents a counter of $\mathcal{M}$ to a separate pair of places which is reserved for that purpose. Then, $\mathcal{B}_\mathcal{M}$ joins $\mathbf{v}$ and $\mathbf{v}''$ by performing a binary rule, verifies that the values of each counter of $\mathcal{M}$ were the same in $\mathbf{v}$ and $\mathbf{v}'$, and empties the auxiliary places.
- Since $\mathcal{B}_\mathcal{M}$ can simulate checking that every counter of $\mathcal{M}$ is zero, it can guess and verify that the configuration that it represents is initial.

To reduce to the covering problem, we use the target vector to specify that the reverse simulation has reached the initial configuration of $\mathcal{M}$. To reduce to the boundedness problem, we amend $\mathcal{B}_\mathcal{M}$ so that upon guessing and verifying that the configuration of $\mathcal{M}$ is initial, it becomes unbounded by deriving an infinite sequence of increasing vectors.

THEOREM 18. *Covering and boundedness for BVAS are* 2EXPTIME*-hard.*

6 Concluding remarks

The extra work in this article in relation to the proofs of Lipton and Rackoff [10, 12], and the recent result that reachability for BVAS is 2EXPSPACE-hard [9] (the highest known lower bound for VAS is Lipton's), indicate that BVAS are not a trivial extension of VAS.

We would like to thank Serge Haddad (LSV, Cachan) for numerous discussions about VAS and their extensions, Sylvain Schmitz (LSV, Cachan) for pointing us to [13], and Alexander Schrijver (CWI, Amsterdam) for correspondence about integer linear programming.

References

[1] M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3), 2009.

[2] I. Borosh and L. B. Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Proc. AMS*, 55:299–304, 1976.

[3] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[4] P. de Groote, B. Guillaume, and S. Salvati. Vector addition tree automata. In *LICS*, pages 64–73. IEEE, 2004.

[5] J. Esparza. Decidability and complexity of Petri net problems — an introduction. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lect. Notes Comput. Sci.*, pages 374–428. Springer, 1998.

[6] J. Esparza and M. Nielsen. Decidability issues for Petri nets — a survey. *Bull. EATCS*, 52:244–262, 1994.

[7] P. Habermehl. On the complexity of the linear-time mu-calculus for Petri nets. In *ICATPN*, volume 1248 of *Lect. Notes Comput. Sci.*, pages 102–116. Springer, 1997.

[8] R. M. Karp and R. E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.

[9] R. Lazić. The branching reachability problem requires doubly-exponential space. Manuscript, 2009.

[10] R. J. Lipton. The reachability problem requires exponential space. Technical Report 62, Dep. Comput. Sci., Yale Univ., Jan. 1976.

[11] E. W. Mayr and A. R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Adv. Math.*, 46:305–329, 1982.

[12] C. Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6(2):223–231, 1978.

[13] O. Rambow. Multiset-valued linear index grammars: imposing dominance constraints on derivations. In *ACL*, pages 263–270. Morgan Kaufmann, 1994.

[14] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *Monographs in Theor. Comput. Sci. An EATCS Series*. Springer, 1985.

[15] L. Rosier and H.-C. Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *J. Comput. Syst. Sci.*, 32:105–135, 1986.

[16] R. Valk and G. Vidal-Naquet. Petri nets and regular languages. *J. Comput. Syst. Sci.*, 23(3):299–325, 1981.

[17] K. N. Verma and J. Goubault-Larrecq. Karp-Miller trees for a branching extension of VASS. *Discr. Math. and Theor. Comput. Sci.*, 7:217–230, 2005.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Subexponential Algorithms for Partial Cover Problems

Fedor V. Fomin[1], Daniel Lokshtanov[1], Venkatesh Raman[2] and Saket Saurabh[2]

[1] Department of Informatics, University of Bergen,
N-5020 Bergen, Norway.
{fedor.fomin|daniello}@ii.uib.no

[2] The Institute of Mathematical Sciences,
Chennai, India.
{vraman|saket}@imsc.res.in

ABSTRACT. Partial Cover problems are optimization versions of fundamental and well studied problems like VERTEX COVER and DOMINATING SET. Here one is interested in covering (or dominating) the maximum number of edges (or vertices) using a given number (k) of vertices, rather than covering all edges (or vertices). In general graphs, these problems are hard for parameterized complexity classes when parameterized by k. It was recently shown by Amini et. al. [*FSTTCS 08*] that PARTIAL VERTEX COVER and PARTIAL DOMINATING SET are fixed parameter tractable on large classes of sparse graphs, namely H-minor free graphs, which include planar graphs and graphs of bounded genus. In particular, it was shown that on planar graphs both problems can be solved in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$.

During the last decade there has been an extensive study on parameterized subexponential algorithms. In particular, it was shown that the classical VERTEX COVER and DOMINATING SET problems can be solved in subexponential time on H-minor free graphs. The techniques developed to obtain subexponential algorithms for classical problems do not apply to partial cover problems. It was left as an open problem by Amini et al. [*FSTTCS 08*] whether there is a subexponential algorithm for PARTIAL VERTEX COVER and PARTIAL DOMINATING SET. In this paper, we answer the question affirmatively by solving both problems in time $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$ not only on planar graphs but also on much larger classes of graphs, namely, apex-minor free graphs. Compared to previously known algorithms for these problems our algorithms are significantly faster and simpler.

1 Introduction and Motivation

A generic instance of a covering problem consists of a family of sets over an universe and the objective is to cover the universe with as few sets from the family as possible. Covering problems are basic problems not only in combinatorial optimization and algorithms but occur naturally in variety of applications. One of the prominent covering problems is the classical SET COVER problem. Other classical problems in the framework of covering include well known problems like VERTEX COVER, DOMINATING SET, FACILITY LOCATION, k-MEDIAN, k-CENTER problems, on which hundreds of papers have been written.

As the name suggests, in partial cover problems one is interested in covering as much of the universe, if not the entire universe. This makes the partial cover problems natural

© Fomin, Lokshtanov, Raman and Saurabh; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 193–201
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2318

generalizations of the well known covering problems. More precisely, in the partial covering problem, for a given integer $t \geq 0$, we want to cover at least t elements using as few objects (vertices or edges) as possible. For an example, in PARTIAL VERTEX COVER (PVC), the goal is to cover at least t edges with the minimum number of vertices while in PARTIAL DOMINATING SET (PDS) the goal is to dominate at least t vertices of the input graph with the minimum number of vertices.

Partial cover problems have been investigated extensively and are well understood in the context of polynomial time approximation [2, 4, 3, 5, 16, 18] and parameterized complexity [1, 4, 24, 25, 23, 27]. In this paper we study partial cover problems defined on graphs namely PARTIAL VERTEX COVER and PARTIAL r-DOMINATING SET from the view point of parameterized algorithms. PARTIAL VERTEX COVER is defined as follows.

> PARTIAL VERTEX COVER (PVC): Given a graph $G = (V, E)$ and positive integers k and t, check whether there exists a set of vertices $C \subseteq V$ such that $|C| \leq k$ and there are at least t edges incident to C.

The PARTIAL r-DOMINATING SET is a generalization of DOMINATING SET and is defined as follows.

> PARTIAL r-DOMINATING SET (P-r-DS): Given a graph $G = (V, E)$ and positive integers k, r and t, determine whether there exists a set of vertices $D \subseteq V$ such that $|D| \leq k$ and there are at least t vertices at distance at most r from some vertex in D.

In parameterized algorithms, for decision problems with input size n, and a parameter k, the goal is to design an algorithm with runtime $f(k) \cdot n^{O(1)}$, where f is a function of k alone. Problems having such an algorithm are said to be fixed parameter tractable (FPT). There is also a theory of hardness using which one can identify parameterized problems that are not amenable to such algorithms. This hardness hierarchy is represented by $W[i]$ hierarchy for $i \geq 1$. For an introduction and more recent developments see the books [13, 14, 29]. In this paper, we always parameterize a problem by the size of the cover, that is, the positive integer k.

Most of the research on partial cover problems in parameterized complexity has considered the number of objects to be covered (t) as a parameter rather than the the size of the cover (k). Bläser [4] initiated the study of partial cover problems parameterized by t and obtained a randomized algorithm with running time $5.45^t n^{\mathcal{O}(1)}$ for PDS. Kneis et al. [25] improved this algorithm and obtained a randomized algorithm with running time $(4+\epsilon)^t n^{\mathcal{O}(1)}$ for every fixed $\epsilon > 0$. Recently, Koutis and Williams [27] obtained an even faster randomized algorithm for PDS, which runs in time $2^t n^{\mathcal{O}(1)}$. Kneis et al. [24] studied the PVC problem when parameterized by the number edged to be covered (t) and obtained a randomized algorithm running in time $2.0911^t n^{\mathcal{O}(1)}$. The algorithm for PVC was recently improved by Kneis et al. [23]. They obtain a randomized algorithm with running time $1.2993^t n^{\mathcal{O}(1)}$ and a deterministic algorithm with running time $1.396^t n^{\mathcal{O}(1)}$ for PVC. When parameterized by the size of cover k, PVC is known to be $W[1]$-complete [17]. The P-r-DS problem being a generalization of DOMINATING SET is also known to be $W[2]$-hard on general graphs when parameterized by the cover size. Amini et al. [1] considered these problems with the size of the cover k being the parameter and initiated a study of these problem on sparse graphs namely planar graphs, apex minor free graphs and H-minor

free graphs. They obtained algorithms with running time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ for PVC and P-r-DS and left an open question of whether these problems have an algorithm with running time $2^{o(k)}n^{\mathcal{O}(1)}$, like their non partial counterpart on planar graphs or more generally on H-minor free graphs. In this paper we answer this question in affirmative and obtain algorithms with running time $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$ for PVC and P-r-DS on planar graphs and more general classes of sparse graphs, namely, apex-minor free graphs.

Most of the known sub-exponential time algorithms on planar graphs, graphs of bounded genus, apex minor free graphs and H-minor free graphs are based on the meta-algorithmic theory of bidimensionality, developed by Demaine et al. [7]. The bidimensionality theory is based on algorithmic and combinatorial extensions to various parts of Graph Minors Theory of Robertson and Seymour [30] and provides a simple criteria for checking whether a parameterized problem is solvable in subexponential time on sparse graphs. The theory applies to the graph problems that are *bidimensional* in the sense that the value of the solution for the problem in question on $k \times k$ grid or "grid like graph" is at least $\Omega(k^2)$ and the value of solution decreases while contracting or sometime deleting the edges. Problems that are bidimensional include k-FEEDBACK VERTEX SET, k-EDGE DOMINATING SET, k-LEAF SPANNING TREE, k-PATH, k-rDOMINATING SET, k-VERTEX COVER and many others. We refer to surveys by Demaine and Hajiaghayi [10] and Dorn et al. [12] for further details on bidimensionality and subexponential parameterized algorithms. But neither PVC nor P-r-DS are bidimensional problems. This is because an optimum solution to PVC or P-r-DS need not cover all the edges (or the vertices respectively) of a $k \times k$ grid, and hence its value need not be large on such a grid. Hence this theory is *not amenable* to our problems.

Our subexponential time algorithms for PVC and P-r-DS are based on a technique used to solve the classical DISJOINT PATH problem in the Graph Minors Theory of Robertson and Seymour [31], called *irrelevant vertex* argument. The technique can be described as follows, in polynomial time we find a vertex which is irrelevant for the solution and hence can be deleted and when we can not find an irrelevant vertex, we show that the reduced instance has bounded treewidth. This technique has recently been used to solve several problems around finding disjoint paths [19, 20, 21, 22, 26]. To obtain subexponential time algorithms for PVC and P-r-DS we introduce a notion of "lexicographically smallest" solution and use its properties to obtain an irrelevant vertex in the graph. When we can not find any irrelevant vertex then we are able to show that that the treewidth of the reduced graph is at most $\mathcal{O}(\sqrt{k})$. Once we have a sublinear bound on the treewidth of the input graph, we can solve the problem in $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$ time using dynamic programming over graphs of bounded treewidth. Our results are based on a simple but powerful observation relating lexicographically least solutions and r-dominating sets of size at most k.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph where V is the set of vertices and E is the set of edges. We denote the number of vertices by n and number of edges by m. For a subset $V' \subseteq V$, by $G[V']$ we mean the subgraph of G induced by V'. By $N(u)$ we denote (open) neighborhood of u that is set of all vertices adjacent to u and by $N[u] = N(u) \cup \{u\}$. Similarly, for a subset $D \subseteq V$, we define $N[D] = \cup_{v \in D} N[v]$. The *distance* $d_G(u, v)$ between two vertices u and v of

G is the length of the shortest path in G from u to v. For a given vertex $v \in V$ by $\partial(v)$ we denote the set of edges which are incident with v. For a subset $X \subseteq V$, $\partial(S) = \cup_{v \in S} \partial(v)$.

Given an edge $e = (u, v)$ of a graph G, the graph G/e is obtained by contracting the edge (u, v) that is we get G/e by identifying the vertices u and v and removing all the loops and duplicate edges. A *minor* of a graph G is a graph H that can be obtained from a subgraph of G by contracting edges. A graph class $\mathcal{C}$ is *minor closed* if any minor of any graph in $\mathcal{C}$ is also an element of $\mathcal{C}$. A minor closed graph class $\mathcal{C}$ is *H-minor-free* or simply *H-free* if $H \notin \mathcal{C}$. A graph H is called an apex graph if the removal of one vertex makes it a planar graph.

A *tree decomposition* of a graph $G = (V, E)$ is a pair (X, T) where T is a tree on vertex set $V(T)$ whose vertices we call *nodes* and $X = (\{X_i \mid i \in V(T)\})$ is a collection of subsets of V such that

1. $\bigcup_{i \in V(T)} X_i = V$,
2. for each edge $(v, w) \in E$, there is an $i \in V(T)$ such that $\{v, w\} \in X_i$, and
3. for each $v \in V$ the set of nodes $\{i \mid v \in X_i\}$ forms a subtree of T.

The *width* of a tree decomposition $(\{X_i \mid i \in V(T)\}, T)$ equals $\max_{i \in V(T)}\{|X_i| - 1\}$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G. We use notation $\mathbf{tw}(G)$ to denote the treewidth of a graph G.

Given a graph $G = (V, E)$ a set of vertices D of V is called an *r-dominating set* for G if $N_r(D) = V$. For $r = 1$ the set D is called a *dominating set*. In the r-DOMINATING SET problem, we are given a graph $G = (V, E)$ and the objective is to find the smallest sized D such that $N_r(D) = V$.

3 Subexponential algorithm for Partial Vertex Cover

In this section we consider the PVC problem. In fact we will solve a slightly more general problem, that is, given an undirected graph, a non negative integer k, we find the *maximum* number of edges that can be covered by a subset of at most k vertices. The decision version of the problem is precisely PVC. If the maximum number of edges covered by any vertex set of size at most k is at least t then we return "yes" else we return "no".

The key idea of the algorithm is to identify a set of *irrelevant* vertices, I, which can be deleted without destroying at least one set $C \subseteq V$ such that $|C| \leq k$ and $|\partial(C)| \geq t$, if such a set exists. Then we will show that the $\mathbf{tw}(G[V \setminus I]) \leq \mathcal{O}(\sqrt{k})$ and hence the dynamic programming over graphs of bounded treewidth can be applied. To identify a set of irrelevant vertices we introduce the notion of *lexicographically smallest solution*.

Definition 1 *Given a graph $G = (V, E)$, an ordering $\sigma = v_1 \dots v_n$ of the vertices in V and subsets X and Y of V, if X is lexicographically smaller than Y then we denote it by $X \leq_\sigma Y$. We call a set $C \subseteq V$ the* lexicographically smallest solution *for PVC if for any other solution C' for the PVC we have that $C \leq_\sigma C'$.*

Let $\sigma = v_1 v_2 \dots v_n$ be an ordering of the vertices such that the vertices are in non increasing order of their degrees, with ties being broken arbitrarily. That is,

$$d(v_1) \geq d(v_2) \cdots \geq d(v_{n-1}) \geq d(v_n).$$

Throughout this section, we will assume that the vertex set of the input graph is ordered by *this* fixed ordering σ and denote the graph by $G = (V_\sigma, E)$ to *emphasize* the fact that the

vertex set is order with respect to σ. By V_σ^i we denote the vertex set $v_1 \dots v_i$. Our goal will be to find the lexicographically smallest solution for PVC. The algorithm is based on the following properties of the lexicographically smallest solution for PVC.

LEMMA 1. *Let $G = (V_\sigma, E)$ be a yes instance to* PVC*, $C = \{u_{i_1}, \dots, u_{i_k}\}$ be the lexicographically smallest solution for* PVC *and $u_{i_k} = v_j$ for some j. Then C is a dominating set of size at most k for $G[V_\sigma^j]$.*

PROOF. Let us assume to the contrary that C is not a dominating set for $G[V_\sigma^j]$. Then there exists a vertex v_i, $1 \leq i < j$ such that $N[v_i] \cap C = \emptyset$. Set $C' := C \setminus \{v_j\} \cup \{v_i\}$. We claim that C' covers at least as many edges as are covered by C. That is, $|\partial(C')| \geq |\partial(C)|$. Since $d(v_i) \geq d(v_j)$, we have that

$$|\partial(C')| \geq |\partial(C)| - d(v_j) + d(v_i) \geq |\partial(C)|.$$

This is because the edges covered by v_i are not covered by any element of $C - \{v_j\}$. Hence, $|C'| = |C|$, C' is lexicographically smaller than C and $|\partial(C')| \geq |\partial(C)|$ a contradiction to the choice of C. ∎

We also need the following results for our algorithm.

LEMMA 2. *Let G be a n-vertex graph excluding an apex graph H as a minor. If G has an r-dominating set of size at most k, then G has treewidth at most $c_H r\sqrt{k} = \mathcal{O}(r\sqrt{k})$, where c_H is a constant depending only on the size of H.*

Lemma 2 follows from the fact that the size of r-dominating set is a "contraction bidimensional" parameter and that if a contraction bidimensional parameter has value at most k on a graph G which excludes an apex graph H as a minor then $\mathbf{tw}(G) \leq \mathcal{O}(r\sqrt{k})$ [6, 8, 15] . We will use the following known algorithm to solve PVC on graphs of bounded treewidth.

LEMMA 3. *[28] Let G be an undirected graph such that the treewidth of G is at most w. Then in time $2^w n^{\mathcal{O}(1)}$ we can find a subset C of at most k vertices that cover the maximum number of edges of G.*

For our proof we also need the following result by Demaine and Hajiaghayi to obtain a polynomial time approximation scheme (PTAS) for r-DOMINATING SET.

LEMMA 4. *[9] There is a PTAS for r-*DOMINATING SET *on apex minor free graphs.*

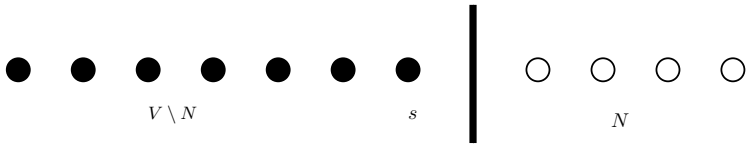


Figure 1: The Algorithmic Schema

The basic schema of the algorithm is as follows. We start with the vertex set V_σ and scan the vertices in the *reverse* order of $\sigma = v_1 v_2 \dots v_n$. That is, we scan the vertices in the order $v_n v_{n-1} \dots v_2 v_1$. The algorithm can be viewed as having a stick, initially positioned to the

ALGO-PC($G = (V_\sigma, E), k, \epsilon, N$)
(Here G is a graph with vertices ordered in non increasing order σ of their degrees , k a non negative integer, $\epsilon > 0$ is an arbitrary fixed constant, N is a set of vertices (initially $\emptyset$), and the goal is to find a subset of $V \setminus N$ of size at most k that covers the maximum number of edges of $G = (V, E)$.)

1. Let $p := n$.
2. While there does not exist a dominating set of size at most $(1+\epsilon)k$ for $G[V_\sigma^p]$ (determined using Lemma 4)
   - set $N := N \cup \{v_p\}$ and $p := p - 1$.

   endwhile
3. Let $I = \{u \mid u \in N, N(u) \subseteq N\}$ and set $V' = V \setminus I$. Find a tree-decomposition (U, T) of $G[V']$ using the constant factor approximation algorithm of Demaine et al. [11] for computing the treewidth of H-minor free graph.
4. Apply Lemma 3 to find a subset C' of size at most k of $G[V']$ which covers the maximum number of edges.

Figure 2: Description of the partial cover Algorithm

right of v_n which we *slide* towards its left if the vertex to its left satisfies certain properties. See Figure 1. At any intermediate stage, we have a vertex set N which are the vertices in the original order σ, to the right of the stick. The vertex set s is the first vertex to the left of the stick. The stick represents the fact that the lexicographically smallest solution C we are looking for lies completely in $V \setminus N$, that is, $C \subseteq V \setminus N$. To slide the stick we do as follows. Let $s = v_j$ for some j. Now we check whether $G[V_\sigma^j]$ has a dominating set of size "roughly k". If not, we slide the stick to one position left. Else we find an appropriate induced subgraph $G' = (V', E')$ of G such that $\mathbf{tw}(G') \leq \mathcal{O}(\sqrt{k})$ and G has a set C of size at most k such that $|\partial(C)| \geq t$ if and only if there exists a set $C' \subseteq V'$ such that $|C'| \leq k$ and $|\partial(C')| \geq t$. A formal description of our algorithm for partial vertex cover is given in Figure 2. The ALGO-PC is called with the parameter $(G = (V_\sigma, E), k, \epsilon, \emptyset)$. Now we state our main theorem for this section.

THEOREM 5. *Let $G = (V, E)$ a graph that excludes an apex graph H as a minor and k and t be a positive integers. Then in $2^{\mathcal{O}(\sqrt{k})} n^{\mathcal{O}(1)}$ time we can determine whether there exists a subset $C \subseteq V$ of size at most k such that $|\partial(C)| \geq t$.*

PROOF. We argue the correctness of the algorithm. In the first part of the algorithm we try to identify the subset N of vertices such that it does not intersect with the lexicographically least solution C we are looking for. We iteratively run through the vertices in the reverse order and try to maintain the invariant that N is a subset of the vertices that does not intersect with the lexicographically least solution. Initially N is empty, so the invariant trivially holds. The set N only grows if in any step, the PTAS algorithm of Lemma 4 finds a dominating set of $G[V \setminus N]$ of size more than $(1+\epsilon)k$. Let v_p be the largest indexed vertex in $V \setminus N$, that is, v_p is to the left of the set N in the ordering σ. Now by Lemma 1, we know that if $v_p \in C$ then $G[V \setminus N]$ has a dominating set of size at most k and hence the PTAS from Lemma 4 would find an approximate dominating set of size at most $(1+\epsilon)k$. This implies that $v_p \notin C$ and hence we can safely place v_p in N. This proves the correctness of the first part.

Note that edges in $G[N]$ will not be covered by C, and hence vertices in N that have neighbors only in N are collected in the set I and deleted at the end. The set I is the irrelevant set of vertices we were looking for. Let $V' = V \setminus I$. Thus we have shown that G has a set C of size at most k such that $|\partial(C)| \geq t$ if and only if there exists a set $C' \subseteq V'$ such that $|C'| \leq k$ and $|\partial(C')| \geq t$. Now applying Lemma 3 we find a subset C' of size at most k of $G[V']$ which covers the maximum number of edges. So if $|\partial(C')| \geq t$ then we return "yes" else we return "no". The correctness of this step follows from Lemma 3.

Now we analyze the time complexity of the algorithm. We know that when the algorithm exits the while loop, $G[V \setminus N]$ has a dominating set of size at most $(1+\epsilon)k$. Let D be a dominating set of $G[V \setminus N]$ of size at most $(1+\epsilon)k$. This implies that D is a 2-*dominating set* of $G[V']$ as every vertex $v \in (N \cap V')$ has a neighbor in $V \setminus N$. Hence by Lemma 2, $\mathbf{tw}(G') \leq \mathcal{O}(\sqrt{(1+\epsilon)k}) = \mathcal{O}(\sqrt{k})$. Now using the constant factor approximation algorithm of Demaine et al. [11] for computing the treewidth of H-minor free graph, we find a tree-decomposition of $G[V']$ of width $\mathcal{O}(\sqrt{k})$ in time $n^{\mathcal{O}(1)}$. Finally, the dynamic programming algorithm mentioned in Lemma 3 runs in time $2^w n^{\mathcal{O}(1)}$ on graphs of treewidth w and hence our algorithm has running time $2^{\mathcal{O}(\sqrt{k})} n^{\mathcal{O}(1)}$. ∎

4 Partial dominating set problems

In this section we consider PARTIAL r-DOMINATING SET problem. We first modify Lemma 1 to prove the following.

LEMMA 6. *Let $G = (V, E)$ be a graph and let σ be the ordering of the vertices in non increasing order of their sizes of $N_r(v)$, that is, if $v_i < v_j$ in σ, then $|N_r(v_i)| \geq |N_r(v_{i+1})|$ with ties being broken arbitrarily. Let $G = (V_\sigma, E)$ be a yes instance to P-r-DS, $C = \{u_{i_1}, \ldots, u_{i_k}\}$ be the lexicographically smallest solution for P-r-DS and $u_{i_k} = v_j$ for some j. Then C is a $2r$-dominating set of size at most k for $G[V_\sigma^j]$.*

PROOF. Let $N_r(C) = \bigcup_{s \in C} N_r(s)$ be the set of vertices of V_σ^j that are r-dominated by C, and suppose that C is not a $2r$-dominating set of V. Let v_i, $i < j$ be a vertex of V_σ^j that is not $2r$-dominated by C ($v_i \notin N_{2r}(C)$). Then $N_r(v_i) \cup N_r(s) = \emptyset$ for every $s \in C$ as otherwise if for some vertex $s \in C$, the intersection is non empty, then v_i will be $2r$ dominated by s. Let $C' = C - v_j \cup \{v_i\}$, then $|C'| = |C|$, C' is lexicographically smaller than C and $|N_r(C')| \geq |N_r(C)| + |N_r(v_i)| - |N_r(v_j)| \geq N_r(C)$ a contradiction to the choice of C. ∎

We also need a lemma similar to Lemma 3 which we state below.

LEMMA 7. *[7] Let G be an undirected graph such that the treewidth of G is at most w. Then in time $(2r+1)^{1.5w} n^{\mathcal{O}(1)}$ we can find a subset C of at most k vertices that r-dominate the maximum number of vertices of G.*

With all these ingredients, the subexponential algorithm for the P-r-DS is very similar to our algorithm for PVC. The only difference is in the while loop where instead of finding a dominating set of size $(1+\epsilon)k$, we find a $2r$-dominating set of size $(1+\epsilon)k$, and in the final step, use the dynamic programming algorithm of Lemma 7 to find a subset C of at most k vertices that r-dominate the maximum number of vertices of G. Thus we have

THEOREM 8. *Let $G = (V, E)$ a graph that excludes an apex graph H as a minor and k and t be a positive integers. Then in $2^{\mathcal{O}(r(\log r)\sqrt{k})}n^{\mathcal{O}(1)}$ time we can determine whether there exists a subset $C \subseteq V$ of size at most k such that $|N_r(C)| \geq t$.*

5 Conclusion

We have given the first subexponential algorithms for PARTIAL VERTEX COVER and PARTIAL r-DOMINATING SET problems on planar and apex minor free graphs, answering an open problem in [1]. Our results were based on a simple but powerful observation relating lexicographically least solutions and r-dominating sets of size at most k. This allowed us to significantly improve the running time of several algorithm presented in [1] in an elegant way. Through this process, we have also expanded the list of problems tractable using the irrelevant vertex argument and it would be nice to apply this technique for other problems in planar and other classes of sparse graphs.

References

[1] O. Amini, F. V. Fomin, and S. Saurabh. Implicit branching and parameterized partial cover problems (extended abstract). In *FSTTCS*, 2008.

[2] S. Arora and G. Karakostas. A 2+epsilon approximation algorithm for the -mst problem. In *SODA*, pages 754–759, 2000.

[3] R. Bar-Yehuda. Using homogenous weights for approximating the partial cover problem. In *SODA*, pages 71–75, 1999.

[4] M. Bläser. Computing small partial coverings. *Inf. Process. Lett.*, 85(6):327–331, 2003.

[5] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In *SODA*, pages 642–651, 2001.

[6] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Bidimensional parameters and local treewidth. *SIAM J. Discrete Math.*, 18(3):501–511, 2004.

[7] 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.

[8] E. D. Demaine and M. Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008.

[9] E. D. Demaine and M. T. Hajiaghayi. Bidimensionality: new connections between fpt algorithms and ptass. In *SODA*, pages 590–601, 2005.

[10] E. D. Demaine and M. T. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Computer Journal*, 51(3):292–302, 2008.

[11] E. D. Demaine, M. T. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *FOCS*, pages 637–646, 2005.

[12] F. Dorn, F. V. Fomin, and D. M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008.

[13] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Springer-Verlag, New York, 1999.

[14] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.
[15] F. V. Fomin, P. A. Golovach, and D. M. Thilikos. Contraction bidimensionality: the accurate picture. In *ESA 09*, LNCS, Berlin Heidelberg, 2009. Springer-Verlag.
[16] R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53(1):55–84, 2004.
[17] J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of vertex cover variants. *Theory Comput. Syst.*, 41(3):501–520, 2007.
[18] E. Halperin and A. Srinivasan. Improved approximation algorithms for the partial vertex cover problem. In K. Jansen, S. Leonardi, and V. V. Vazirani, editors, *APPROX*, volume 2462 of *Lecture Notes in Computer Science*, pages 161–174. Springer, 2002.
[19] K. Kawarabayashi. An improved algorithm for finding cycles through elements. In *IPCO*, volume 5035 of *Lecture Notes in Computer Science*, pages 374–384, 2008.
[20] K. Kawarabayashi and Y. Kobayashi. The induced disjoint paths problem. In *IPCO*, volume 5035 of *Lecture Notes in Computer Science*, pages 47–61, 2008.
[21] K. Kawarabayashi and B. A. Reed. A nearly linear time algorithm for the half integral disjoint paths packing. In *SODA*, pages 446–454, 2008.
[22] K. Kawarabayashi and B. A. Reed. A nearly linear time algorithm for the half integral parity disjoint paths packing problem. In *SODA*, pages 1183–1192, 2009.
[23] J. Kneis, A. Langer, and P. Rossmanith. Improved upper bounds for partial vertex cover. In *WG*, volume 5344 of *Lecture Notes in Computer Science*, pages 240–251, 2008.
[24] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Intuitive algorithms and t-vertex cover. In *ISAAC*, volume 4288 of *Lecture Notes in Computer Science*, pages 598–607, 2006.
[25] J. Kneis, D. Mölle, and P. Rossmanith. Partial vs. complete domination: t-dominating set. In *SOFSEM (1)*, volume 4362 of *Lecture Notes in Computer Science*, pages 367–376, 2007.
[26] Y. Kobayashi and K. Kawarabayashi. Algorithms for finding an induced cycle in planar graphs and bounded genus graphs. In *SODA*, pages 1146–1155, 2009.
[27] I. Koutis and R. William. Limits and applications of group algebras for parameterized problems. In *ICALP 09*, LNCS, Berlin Heidelberg, 2009. Springer-Verlag.
[28] H. Moser. Exact Algorithms for Generalizations of Vertex Cover. Master's thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany, 2005.
[29] R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.
[30] N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory Series B*, 62:323–348, 1994.
[31] N. Robertson and P. D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

On the Tightening of the Standard SDP for Vertex Cover with ℓ_1 Inequalities

Konstantinos Georgiou, Avner Magen, Iannis Tourlakis

Department of Computer Science, University of Toronto

ABSTRACT. We show that the integrality gap of the standard SDP for VERTEX COVER on instances of n vertices remains $2 - o(1)$ even after the addition of *all* hypermetric inequalities. Our lower bound requires new insights into the structure of SDP solutions behaving like ℓ_1 metric spaces when one point is removed. We also show that the addition of all ℓ_1 inequalities eliminates any solutions that are not convex combination of integral solutions. Consequently, we provide the strongest possible separation between hypermetrics and ℓ_1 inequalities with respect to the tightening of the standard SDP for VERTEX COVER.

1 Introduction

A *vertex cover* for a graph $G = (V, E)$ is a subset of the vertices touching all edges. The minimum VERTEX COVER problem (VC) is to find a minimal vertex cover for a graph. While the corresponding decision version for VERTEX COVER is a classic NP-hard problem, the exact approximability of the minimum VERTEX COVER problem remains one of the outstanding open problems in approximation algorithms.

In terms of lower bounds, Dinur and Safra [6] show that VERTEX COVER is NP-hard to approximate within a factor better than 1.36. Assuming Khot's [15] Unique Games conjecture holds, Khot and Regev [16] show that computing a $2 - \Omega(1)$ approximation is NP-hard. As for upper bounds, a very simple argument based on maximal matchings shows that VERTEX COVER admits a polynomial time 2 approximation. The best approximation algorithm known is due to Karakostas [13] and has an approximation ratio of $2 - \Omega(\sqrt{1/\log n})$.

Closing the gap between the known upper and lower bounds on VERTEX COVER's approximability (or obtaining a tight lower bound without relying on the Unique Games Conjecture) has proved particularly difficult. As a result researchers have focused on studying how well we can approximate VERTEX COVER using algorithmic techniques proven successful for other optimization problems. One such family of algorithms arises by first formulating the optimization problem as a (intractable) quadratic integer problem and then relaxing the integrality constraint to obtain a semidefinite program (SDP) that can be solved in polynomial time up to any desired precision. This approach was first introduced by Goemans and Williamson [11] and used to obtain a breakthrough 0.878-approximation algorithm for MAX CUT. Subsequently, many SDP-based algorithms have been discovered and which yield the best approximation algorithms known for several optimization problems [2, 14, 13].

The quality of an SDP relaxation is typically measured by its integrality gap, namely, the ratio between the true optimal solution and the relaxed SDP solution. It is generally accepted that a lower bound on the integrality gap is a lower bound on the approximation

© K. Georgiou, A. Magen, I. Tourlakis; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 203–214
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2319

ratio achievable by any algorithm based on the SDP relaxation. Unfortunately, so far no SDP relaxation for VERTEX COVER has been found whose integrality gap is not $2 - o(1)$.

Indeed, Kleinberg and Goemans [10] show that the obvious "standard" SDP for VERTEX COVER (defined in Section 2.2) has integrality gap $2 - o(1)$. But can this standard SDP be tightened with further constraints to reduce the integrality gap? A series of papers studies whether so-called ℓ_1 inequalities can decrease the integrality gap. The use of ℓ_1 inequalities is motivated by the fact that solutions to the standard quadratic programming formulation for VERTEX COVER lie in an ℓ_1 metric space. Further motivation comes from a paper by Hatami et al. [12] showing that adding *all* ℓ_1 inequalities to the standard SDP for VERTEX COVER yields true optimal solutions. Now, adding all ℓ_1 inequalities yields an intractable SDP relaxation. The natural question that then emerges is whether there is a subset of ℓ_1 inequalities which decreases the integrality gap while keeping the program tractable. Indeed such subsets have been useful for other optimization problems: For instance, the simplest ℓ_1 inequality, the triangle inequality, is crucial in the Arora-Rao-Vazirani SDP algorithm for SPARSEST CUT [2] and subsequently in the best tractable SDP formulation for VERTEX COVER [13]. Avis and Umemoto [3] used k-gonal inequalities (a family of ℓ_1 inequalities generalizing the triangle inequality) to design a PTAS for MAX CUT on certain sparse graph families. However, results for VERTEX COVER have so far all been negative: A series of papers [4, 12, 9] culminates in showing that adding so-called hypermetric inequalities (the most well known canonical family of ℓ_1 inequalities, and a generalization of k-gonal inequalities) of *bounded support* does not reduce the integrality gap. The latter is also motivated by the fact that, as k grows, the k-gonal inequalities become increasingly stronger. This will be discussed in Section 2.1.

In this paper, we bring this series of results to its "completion" by showing, somewhat surprisingly, that hypermetrics *never* help for VERTEX COVER:

THEOREM 1. *The integrality gap of the standard SDP relaxation for* VERTEX COVER *tightened with* all *hypermetric inequalities is* $2 - o(1)$.

Theorem 1 may provide further evidence of the true inapproximability of the VERTEX COVER problem. It was consistent with previous results that tightening the standard SDP relaxation for VERTEX COVER with hypermetrics of sufficiently large support (note that such SDPs might not be "tractable": they would only be computable in time polynomial in the number of constraints added) might give an integrality gap of $2 - \Omega(1)$.

Our result extends several ideas from [9]. Indeed the graph instances and our SDP vector construction is similar to the one used in [9] (and related to those used in [8]). Our improvement relies on some new insights we develop for controlling the value of certain "hypermetric-like" inequalities on ℓ_1 embeddable metrics. (An in-depth comparison to previous constructions can be found in Section 3.4.)

But if hypermetrics don't help, can we hope that the family of all ℓ_1 inequalities helps? The answer seems to depend on the problem; for example, in the MINIMUM MULTICUT problem [1] the addition of all ℓ_1 inequalities does not yield integrality gap 1. In contrast, we show that for VERTEX COVER the opposite is true:

THEOREM 2. *Consider a vector solution of the standard SDP for* VERTEX COVER *that along with (at least) one antipode vector satisfies all* ℓ_1 *inequalities. Then the solution is in the*

integral hull, and therefore the integrality gap is 1.

In particular, Theorems 1 and 2 together show that to reduce the integrality gap one *must* employ "unnatural" ℓ_1-inequalities. As mentioned above, Hatami et al. [12] prove a similar result to Theorem 2 showing that strengthening the standard SDP for VERTEX COVER with all ℓ_1 inequalities yields an SDP with no integrality gap. However, we emphasize that their result, which is essentially proved by exploiting the optimality of the SDP solution, does not rule out the possibility of a feasible SDP solution outside the integral hull.

Relations to Lift-and-project systems Lift-and-project procedures, such as those defined by Lovász and Schrijver [18] and Lasserre [17], take an initial LP or SDP relaxation and then systematically derive (over successive rounds) *all* inequalities valid for the integral hull. Relaxations for VERTEX COVER in the Lovász-Schrijver hierachy are incomparable to those studied here (see [9]); the VERTEX COVER SDP relaxation produced after k rounds of Lasserre's tightening satisfies all ℓ_1 inequalities of support k. Strong integrality gaps for lift-and-project derived SDPs (but incomparable to those proved here) are proved by Georgiou et al. [8] and Schoenebeck [19] for the Lovász-Schrijver and Lasserre systems, respectively.

2 Preliminaries

2.1 Metric Spaces, and ℓ_1 and Hypermetric Inequalities

A finite metric space (X, d) is ℓ_1 *embeddable*, or simply an ℓ_1*-metric*, if there exists a mapping $f : X \to \mathbb{R}^n$ such that for all $x, y \in X$ we have $d(x, y) = \|f(x) - f(y)\|_1$. The mapping f is called an *isometry*. We now survey those facts about ℓ_1 metric spaces we will need. For proofs see [5].

Fix a finite set of points X of size n which we will denote by $[n]$. For each $S \subseteq X$ define the *cut metric* $\delta_S : [n] \times [n] \to \{0, 1\}$ such that $\delta_S(i, j) = 1$ if $|S \cap \{i, j\}| = 1$ and 0 otherwise. Cut metrics are clearly ℓ_1 embeddable, and moreover, every ℓ_1 embeddable metric space d can be represented as a convex combination of cut metrics, namely $d(i, j) = \sum_S \lambda_S \delta_S(i, j)$, where $\lambda_S \geq 0$. We then say that (X, d) is *realized* by $\{\lambda_S\}_{S \subseteq X}$ (realization is not unique in general). An ℓ_1 inequality is an inequality $\sum_{ij} B_{ij} x_{ij} \leq 0$ that holds for all ℓ_1 embeddable metrics d, that is $\sum_{ij} B_{ij} d(i, j) \leq 0$. It is possible to show that $\sum_{ij} B_{ij} x_{ij} \leq 0$ is an ℓ_1 inequality if and only if it satisfies $\sum_{ij} B_{ij} d(i, j) \leq 0$ for all cut metrics d.

A canonical discrete class of ℓ_1 inequalities is the class of *hypermetric inequalities*.

DEFINITION 3. *For any* $\mathbf{b} \in \mathbb{Z}^n$ *with* $\sum_{i=1}^n b_i = 1$*, the inequality* $\sum_{ij} b_i b_j x_{ij} \leq 0$ *is a* hypermetric inequality. *The support of a hypermetric inequality is the support of* $\mathbf{b} = (b_1, \ldots, b_n)$.

It is well known that hypermetric inequalities are ℓ_1-inequalities, that is $\sum_{ij} b_i b_j d(i, j) \leq 0$ for all ℓ_1-metrics d (this also follows as a corollary from Lemma 5 below). Note that the hypermetrics include the triangle inequality (by taking $b_i = b_j = 1$, $b_k = -1$ and $\mathbf{b}$ is 0 elsewhere), and all other k-gonal inequalities (e.g., the pentagonal inequality) which are simply those hypermetrics where each b_i is ± 1 or 0.

Both hypermetric inequalities and ℓ_1 inequalities define convex cones. The cone of hypermetric inequalities is contained in the cone of ℓ_1 inequalities, and the containment

is strict for dimension at least 7. The cone of hypermetric inequalities is polyhedral, and many of its facets define facets in the ℓ_1 cone. Since no hypermetric inequality is a positive multiple of another, it follows that only finitely many hypermetrics define facets of the cut cone. A canonical example of such facets are the k-gonal inequalities defined above. It is important to note that k-gonal inequalities are stronger the larger k is in the following sense: for every $k > 1$ there exists a metric on n-points satisfying all $(2t+1)$-gonal inequalities, $0 < t < k$ while the $(2k+1)$-inequality is violated (Corollary 28.3.3 in [5]).

2.2 SDP Formulations for Vertex Cover and Integrality Gap Constructions

Let $G = (V, E)$ be a graph with $V = [n]$. The standard SDP relaxation for VERTEX COVER is

$$\begin{array}{lll} \min & \sum_{i \in V} \|\mathbf{z}_i + \mathbf{z}_0\|_2^2/4 & \\ \text{s.t.} & \|\mathbf{z}_i - \mathbf{z}_0\|_2^2 + \|\mathbf{z}_j - \mathbf{z}_0\|_2^2 = \|\mathbf{z}_i - \mathbf{z}_j\|_2^2 & \forall ij \in E \\ & \|\mathbf{z}_i\|_2^2 = 1 & \forall i \in \{0\} \cup V, \end{array} \tag{1}$$

where the $\mathbf{z}_i$ are vectors*. Note that any vector solution $\{\mathbf{z}_i\}_{i \in \{0\} \cup V}$ of (1) induces a distance function $d(i,j) = \|\mathbf{z}_i - \mathbf{z}_j\|_2^2$.

The SDP relaxation (1) is in general stronger than the standard LP relaxation for VERTEX COVER. Unlike the standard LP, showing that (1) has an integrality gap of $2 - o(1)$ is non-trivial [10]. The graph instances witnessing the integrality gap rely on a powerful combinatorial theorem due to Frankl and Rödl [7] that shows that there cannot be a large family of sets of certain cardinality, all of whose pairwise intersections satisfy a certain condition.

DEFINITION 4. *Given $\gamma > 0$, the* Frankl-Rödl graph *G_m^γ is the graph on the 2^m vertices of the m-dimensional hypercube $\{-1,1\}^m$ having edges between those vertices with Hamming distance exactly $(1-\gamma)m$.*

The theorem of Frankl and Rödl [7] implies that for any constant $\gamma > 0$, a vertex cover of the graphs G_m^γ has size $2^m - o(2^m)$. In fact, it follows from their work that G_m^γ enjoys these properties even for sufficiently large subconstant γ; this was made explicit in [8] showing that one can set $\gamma = \Omega(\sqrt{\log m / m})$ to ensure that no small vertex covers exist.

To appreciate the theorem, notice that for $\gamma = 0$ the graph G_m^γ is just a perfect matching, and hence has a vertex cover of size only half the graph. But by making γ only slightly positive the minimum vertex cover of the obtained graph "jumps" in size to be almost all the vertices!

Frankl-Rödl graphs have been used to prove all the tight integrality gap results [10, 4, 12, 8, 9] for VERTEX COVER SDPs mentioned in the introduction. Most of these papers study (implicitly or explicitly) whether there exists some small enough subset of ℓ_1-inequalities that can be added to the standard SDP relaxation (1) to reduce the integrality gap. Let us briefly explain the role of ℓ_1 inequalities in this context. The metric induced by an integral solution of (1) is (a scalar multiple of) the cut metric associated with the vertex cover. Therefore, ℓ_1 inequalities are valid for all integral solutions. In the extreme, adding *all* ℓ_1-inequalities eliminates the integrality gap [12], and thus focusing on this family of inequalities seems natural.

*Note that the edge constraint can be equivalently written (and is perhaps more well-known) as $(\mathbf{z}_0 - \mathbf{z}_i) \cdot (\mathbf{z}_0 - \mathbf{z}_j) = 0$.

In this paper we analyze the performance of the standard SDP for VERTEX COVER strengthened with hypermetric inequalities, namely, the SDP (1) strengthened by

$$\sum_{ij} b_i b_j \|\mathbf{z}_i - \mathbf{z}_j\|_2^2 \leq 0, \qquad \forall \mathbf{b} \in \mathbb{Z}^{n+1} \text{ such that } \sum_{i}^{n+1} b_i = 1. \tag{2}$$

In the above, if we only use integer vectors $\mathbf{b}$ of support k we obtain the SDP for VERTEX COVER strengthened by all hypermetrics of support k.

Charikar [4] was the first to show tight integrality gaps when we add triangle inequality (a hypermetric of support three) to SDP (1). In [12] a similar result was shown when pentagonal inequalities are added. The strongest negative result analyzing the effect of ℓ_1-inequalities on the standard SDP for VERTEX COVER is due to Georgiou et al. [9] where it is shown that the addition of hypermetrics with support $O(\sqrt{\log n / \log\log n})$ cannot reduce the integrality gap below $2 - o(1)$.

3 Hypermetrics Cannot Strengthen the Standard SDP for VC

3.1 Preparatory Observations about Hypermetric Inequalities

How can we show that a certain metric d satisfies all hypermetric inequalities? Of course the simplest way would be to take an ℓ_1 embeddable metric d that "automatically" satisfies all such inequalities. But by [12] we know that if the solution metric is ℓ_1 embeddable then the value of the SDP will be the same as the integral optimum. However, this type of reasoning is still useful: our solution d will be "almost" ℓ_1 embeddable: if we remove the point associated with v_0 the rest of the points will in fact be ℓ_1 embeddable; nevertheless, we will pick our solution so that we have an integrality gap as large as $2 - o(1)$. Next we present some simple lemmas that will help in analyzing hypermetric inequalities for such "almost-ℓ_1" metrics.

We start by analyzing a generalization of the notion of hypermetric inequalities (hypermetrics correspond to the case $q = 1$).

LEMMA 5. *Let (X, d), be an ℓ_1-metric on n points realized by $\{\lambda_S\}_{S \subseteq X}$. Let $b_1, \ldots, b_n \in \mathbb{Z}$ be such that $\sum_i^n b_i = q$. Then $\sum_{1 \leq i < j \leq n} b_i b_j d(i,j) \leq \lfloor (q/2)^2 \rfloor \sum_S \lambda_S$.*

PROOF.

$$\sum_{1 \leq i < j \leq n} b_i b_j d(i,j) = \sum_{1 \leq i < j \leq n} b_i b_j \sum_S \lambda_S \delta_S(i,j) = \sum_S \lambda_S \sum_{1 \leq i < j \leq n} b_i b_j \delta_S(i,j)$$
$$= \sum_S \lambda_S \sum_{i \in S, j \notin S} b_i b_j = \sum_S \lambda_S \left(\sum_{i \in S} b_i \right) \left(q - \sum_{i \in S} b_i \right) \leq \sum_S \lambda_S (\lfloor (q/2)^2 \rfloor).$$

The last inequality follows from the geometric-mean arithmetic-mean inequality for integers.

We next show that when an ℓ_1-metric has a unit representation, that is, points are vectors in $\mathbb{R}^n$ of unit ℓ_2^2 norm, then it is sometimes possible to bound the sum of the cut coefficients. We say that an ℓ_1-metric with a unit representation has *large diameter* if it has diameter 4. (Notice that the diameter of any metric with unit representation is at most 4.)

LEMMA 6. *Let d be an ℓ_1-metric with unit representation that has large diameter. Then $\sum_S \lambda_S = 4$.*

PROOF. Having a large diameter is equivalent to having two unit vectors in the representation that are antipodes. Without loss of generality, let $\mathbf{z}_1 = -\mathbf{z}_2$. Also, since any $S \subseteq X$ induces a cut, we may assume that $\{\lambda_S\}_{S \subseteq X}$ are non-zero only for sets S that contain 1. Now note that $4 = \|\mathbf{z}_1 - \mathbf{z}_2\|^2 = d_{12} = \sum_{S \not\ni 2} \lambda_S$ so our task is to show that $\lambda_S = 0$ whenever $2 \in S$. Let $i \in [n]$. Then $\|\mathbf{z}_1 - \mathbf{z}_i\|^2 + \|\mathbf{z}_i - \mathbf{z}_2\|^2 = 4 - 2(\mathbf{z}_1\mathbf{z}_i + \mathbf{z}_2\mathbf{z}_i) = 4 = \|\mathbf{z}_1 - \mathbf{z}_2\|^2$. Since for every S, $\delta_S(1,i) + \delta_S(2,i) \geq \delta_S(1,2)$ and since $\|\mathbf{z}_1 - \mathbf{z}_i\|^2 + \|\mathbf{z}_i - \mathbf{z}_2\|^2 = \|\mathbf{z}_1 - \mathbf{z}_2\|^2$, we know that whenever $\lambda_S > 0$ we must have $\delta_S(1,i) + \delta_S(2,i) = \delta_S(1,2)$. But for S that contains 1 and 2 the right hand side is 0, and hence the left hand side is too and $i \in S$. This is true for all i, and hence $S = X$ which makes it a trivial cut that can be ignored.

COROLLARY 7. *Let $\tilde{d}$ be an ℓ_1-metric space with large diameter unit representation, and let d be the restriction of $\tilde{d}$ on a subset of the points. Further, let $b_1, \ldots, b_n \in \mathbb{Z}$ be such that $\sum_i^n b_i = q$. Then $\sum_{i,j} b_i b_j d(i,j) \leq 4\lfloor (q/2)^2 \rfloor$.*

PROOF. Let λ_S be a realization of $\tilde{d}$. By Lemma 6 we have $\sum_S \lambda_S = 4$. We now apply Lemma 5 to $\tilde{d}$ to get $\sum_{i,j} b_i b_j \tilde{d}(i,j) \leq \sum_S \lambda_S \left(\lfloor q/2 \rfloor\right)^2 = 4\left(\lfloor q/2 \rfloor\right)^2$. Since d is a restriction of $\tilde{d}$ the corollary follows.

3.2 The Vector Solution

Our construction is based on tensored vectors. Recall that the tensor product $\mathbf{u} \otimes \mathbf{v}$ of vectors $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{v} \in \mathbb{R}^m$ is the vector in $\mathbb{R}^{nm}$ indexed by ordered pairs from $n \times m$ and assuming the value $u_i v_j$ at coordinate (i,j). Define $\mathbf{u}^{\otimes d}$ to be the vector in $\mathbb{R}^{n^d}$ obtained by tensoring $\mathbf{u}$ with itself d times. Let $P(x) = c_1 x^{t_1} + \ldots + c_q x^{t_q}$ be a polynomial with nonnegative coefficients. Then T_P is the function that maps a vector $\mathbf{u}$ to the vector $T_P(\mathbf{u}) = (\sqrt{c_1}\mathbf{u}^{\otimes t_1}, \ldots, \sqrt{c_q}\mathbf{u}^{\otimes t_q})$. Polynomial tensoring can be used to manipulate inner products in the sense that $T_P(\mathbf{u}) \cdot T_P(\mathbf{v}) = P(\mathbf{u} \cdot \mathbf{v})$.

Recall Definition 4 of the graphs G_m^γ for which we want to build a vector solution for SDP (1) strengthened by (2). For $\gamma > 0$ where $1/\gamma$ is even, our SDP solution will be the result of the tensoring polynomial $P(x) = c_2(x+1)x^{2m/\gamma} + c_1 x^{1/\gamma} + (1 - (c_1 + 2c_2))x$ applied on the normalized m-dimensional hypercube $\{-1,1\}^m$, where all c_1, c_2 and $1 - (c_1 + 2c_2)$ are non-negative. Note that regardless of c_1, c_2, we have $P(1) = 1$. Let $\mathbf{u}_i$ be the normalized vectors of the hypercube, namely $\{\pm 1/\sqrt{m}\}^m$. Our solution vectors are then

$$\begin{aligned} &\mathbf{w}_i = (18\gamma, \sqrt{1 - (18\gamma)^2} T_P(\mathbf{u}_i)),\ i = 1, \ldots, 2^m, \\ &\mathbf{w}_0 = (1, 0, \ldots, 0). \end{aligned} \tag{3}$$

Regardless of the exact choice of P, the value of the objective with the vectors $\{\mathbf{w}_i\}$ in (1) is $2^m(1/2 + 9\gamma)$. To achieve a big integrality gap, we will use the smallest possible value of γ that ensures that no small vertex covers exist, namely $\gamma = \Theta(\sqrt{\log m / m})$.

The following lemma whose proof is deferred to the appendix shows that there exist appropriate constants c_1, c_2 such that the vector solution both satisfies the standard SDP and the triangle inequality.

LEMMA 8. *For sufficiently big m, there exist positive c_1, c_2 (both of order $\Theta(\gamma)$), such that for G_m^γ, the vectors* (3) *satisfy the standard SDP* (1) *strengthened with the triangle inequality. Moreover, $c_2 > 9\gamma$.*

An analogous lemma (Lemma 3) with different bounds on the constants c_1 and c_2 was proved in [8], and the proof is very similar. Indeed, the precise constraints on c_1, c_2 given by Lemma 8 will be crucial for our analysis here and are not implied by Lemma 3 in [8].

Lemma 8 immediately implies that the integrality gap of SDP (1) is at least $\frac{2^m - o(2^m)}{2^m(1/2+9\gamma)}$, which is of course $2 - o(1)$. Therefore Theorem 1 will follow if we additionally show that the vectors (3) satisfy any hypermetric inequality (2). This is taken care of in Section 3.3.

3.3 Proof of Theorem 1

Let $\sum_{ij} B_{ij}x_{ij} \leq 0$ be a hypermetric inequality, with $B_{ij} = b_i b_j$, $b_i \in \mathbb{Z}$, $i = 0, \ldots, n$. Our goal is to show that for the vectors (3), $\sum_{0 \leq i < j \leq n} B_{ij}\|\mathbf{w}_i - \mathbf{w}_j\|_2^2 \leq 0$. By definition, for $i, j \geq 1$,

$$\|\mathbf{w}_i - \mathbf{w}_j\|_2^2 = 2 - 2((18\gamma)^2 + (1-(18\gamma)^2)P(\mathbf{u}_i \cdot \mathbf{u}_j)) = (1-(18\gamma)^2)\|T_P(\mathbf{u}_i) - T_P(\mathbf{u}_j)\|_2^2 \ ,$$

and $\|\mathbf{w}_i - \mathbf{w}_0\|_2^2 = 2(1-18\gamma)$. Hence,

$$\sum_{0 \leq i < j \leq n} B_{ij}\|\mathbf{w}_i - \mathbf{w}_j\|_2^2 = 2(1-18\gamma)\sum_{i=1}^{n} B_{0i} + (1-(18\gamma)^2) \sum_{1 \leq i < j \leq n} B_{ij}\|T_P(\mathbf{u}_i) - T_P(\mathbf{u}_j)\|_2^2$$

Therefore, we need to show

$$\sum_{i=1}^{n} B_{0i} + (1+18\gamma)\frac{1}{2} \sum_{1 \leq i < j \leq n} B_{ij}\|T_P(\mathbf{u}_i) - T_P(\mathbf{u}_j)\|_2^2 \leq 0 \ . \tag{4}$$

Let now (Y, d) be a metric defined as $Y = \{1, \ldots, n\}$, and $d(i,j) = \|T_P(\mathbf{u}_i) - T_P(\mathbf{u}_j)\|_2^2$. All points $T_P(\mathbf{u}_i)$ are normalized sign vectors. By considering all points $T_P(\mathbf{u}_i)$ along with their antipodes $-T_P(\mathbf{u}_i)$ we can obtain the metric $(\tilde{Y}, \tilde{d})$, where again $\tilde{d}$ is the square Euclidean distance of the vectors. Clearly, d is a restriction of $\tilde{d}$ on a subset of points (recall that the tensoring polynomial P is not odd).

CLAIM 9. *The metric $(\tilde{Y}, \tilde{d})$ is ℓ_1 with large diameter unit representation.*

PROOF. $(\tilde{Y}, \tilde{d})$ has large diameter because all antipodes are present. Now, the vectors $\mathbf{u}_i$ have unit ℓ_2^2 norm, and so do the vectors $T_P(\mathbf{u}_i)$, $i = 1, \ldots, n$. Notice that in $\tilde{Y}$ we have excluded the point that corresponds to $\mathbf{z}_0$ in the SDP. Now, applying the tensor operation on a ± 1 vector results in a, say, M-dimensional, ± 1 vector, and hence applying a polynomial on such a vector yields a vector which assumes one of two values in each of the coordinates, and further, one of the values, say x_i, $i = 1, \ldots, M$, is the negation of the other. The same holds by including all their antipodes. In other words, all points $\pm T_P(\mathbf{u}_i)$ are vertices of a box centered at the origin. It is easy to see that the ℓ_2^2-metric associated with such a box is ℓ_1 embeddable: any vector $T_P(\mathbf{u}_i)$ (or its antipode) has the form $\mathbf{u}_i' = (s_1^{(i)}x_1, \ldots, s_M^{(i)}x_M)$ where $s_t^{(i)} \in \{\pm 1\}$ and it can be mapped by f to $(2s_1^{(i)}x_1^2, \ldots, 2s_M^{(i)}x_M^2)$. Hence for any two $i, j \in V$

$$\|\mathbf{u}_i' - \mathbf{u}_j'\|_2^2 = \sum_{t=1}^{M}(s_t^{(i)}x_t - s_t^{(j)}x_t)^2 = \sum_{t=1}^{M}|2s_t^{(i)}x_t^2 - 2s_t^{(j)}x_t^2| = \|f(\mathbf{u}_i') - f(\mathbf{u}_j')\|_1.$$

Therefore, for the metric (Y, d) we can apply Corollary 7 with $q = \sum_{i=1}^{n} b_i = 1 - b_0$ and conclude that the left hand side of expression (4) is upper-bounded by

$$b_0(1-b_0) + (1+18\gamma)2\lfloor (1-b_0)^2/4 \rfloor \leq \begin{cases} 0 & \text{if } b_0 \geq 0, \\ \frac{1}{2}(1-b_0)((1-18\gamma)b_0 + (1+18\gamma)) < 0 & \text{if } b_0 \leq -2. \end{cases}$$

Therefore, we have shown that all hypermetrics are satisfied except perhaps those for which $b_0 = -1$ (like the triangle inequality, pentagonal inequality, etc.).

In order to deal with the case $b_0 = -1$ we look deeper into the structure of $T_P(\mathbf{u}_i)$. To start, we simplify our notation by abbreviating $(\mathbf{u}_i \cdot \mathbf{u}_j + 1)(\mathbf{u}_i \cdot \mathbf{u}_j)^{2m/\gamma}$, $(\mathbf{u}_i \cdot \mathbf{u}_j)^{1/\gamma}$, and $\mathbf{u}_i \cdot \mathbf{u}_j$ by H_{ij}, M_{ij} and L_{ij}, respectively, the "high", "medium" and "low" order terms. Then, $P(\mathbf{u}_i \cdot \mathbf{u}_j) = c_2 H_{ij} + c_1 M_{ij} + (1 - (c_1 + 2c_2))L_{ij}$. Note that for distinct $\mathbf{u}_i, \mathbf{u}_j$, we have $|\mathbf{u}_i \cdot \mathbf{u}_j| \leq 1 - 1/m$, and hence H_{ij} is negligible. We therefore omit it in what follows. As $b_0 = -1$, it follows that $\sum_{i=1}^{n} B_{0i} = b_0(1-b_0) = -2$ and hence that the left hand side of (4) is

$$\begin{aligned} &-2 + (1+18\gamma) \sum_{1 \leq i < j \leq n} B_{ij}(1 - P(\mathbf{u}_i \cdot \mathbf{u}_j)) \\ \approx &-2 + (1+18\gamma) \sum_{1 \leq i < j \leq n} B_{ij}(1 - c_1 M_{ij} - (1 - (c_1 + 2c_2))L_{ij}) \ . \end{aligned} \tag{5}$$

Now we make some simple observations. We have $(\sum_{i=1}^{n} b_i)^2 = \sum_{i=1}^{n} b_i^2 + 2\sum_{1 \leq i < j \leq n} b_i b_j$, and since $\sum_{i=1}^{n} b_i = 1 - b_0 = 2$ we get

$$\sum_{1 \leq i < j \leq n} B_{ij} = \frac{1}{2}(4 - \sum_{i=1}^{n} b_i^2). \tag{6}$$

Now, note that for unit vectors $\mathbf{u}, \mathbf{v}$ we have $\|\mathbf{u} - \mathbf{v}\|_2^2 = 2(1 - \mathbf{u} \cdot \mathbf{v})$. Hence the values $2(1 - M_{ij})$, $1 \leq i < j \leq n$, are the ℓ_2^2 distances of unit vectors that have undergone the polynomial tensoring transformation using some monomial (similarly for the values $2(1 - L_{ij})$). Arguing exactly as in Claim 9, the vectors form an ℓ_1-metric that has a large diameter unit representation, and so by Corollary 7 we have

$$\sum_{1 \leq i < j \leq n} B_{ij}(1 - M_{ij}) \leq 2, \text{ and } \sum_{1 \leq i < j \leq n} B_{ij}(1 - L_{ij}) \leq 2. \tag{7}$$

We now use (6), (7), to conclude that

$$\begin{aligned} &\sum_{1 \leq i < j \leq n} B_{ij}(1 - c_1 M_{ij} - (1 - c_1 - 2c_2)L_{ij}) \\ &= 2c_2 \sum_{1 \leq i < j \leq n} B_{ij} + c_1 \sum_{1 \leq i < j \leq n} B_{ij}(1 - M_{ij}) + (1 - c_1 - 2c_2) \sum_{1 \leq i < j \leq n} B_{ij}(1 - L_{ij}) \\ &\leq c_2 \left(4 - \sum_{i=1}^{n} b_i^2\right) + 2c_1 + 2(1 - c_1 - 2c_2) = -c_2 \sum_{i=1}^{n} b_i^2 + 2. \end{aligned}$$

Recall here that by Lemma 8, $c_2 > 9\gamma$, and the SDP vectors $\{\mathbf{w}_i\}$ satisfy the triangle inequality. Therefore when $b_0 = -1$ we may assume that $\sum_{i=1}^n b_i^2 \geq 4$. Theorem 1 now follows since (5) is upper-bounded by

$$-2 + (1 + 18\gamma)\left(-c_2 \sum_{i=1}^{n} b_i^2 + 2\right) \leq -2 + (1 + 18\gamma)\left(-36\gamma + 2\right) = -648\gamma^2. \tag{8}$$

3.4 Discussion and a strengthened version of Theorem 1

In this section we look a bit more carefully at how our result differs from previous work and use the resulting observations to obtain a strengthened version of Theorem 1.

As mentioned in Section 2.2 all previous works [10, 4, 12, 9] studying integrality gaps for VERTEX COVER SDPs use Frankl-Rödl graphs G_m^γ on $n = 2^m$ vertices. Moreover, they all employ tensoring polynomials of some sort to construct their vector solutions. Perhaps the most useful parameter differentiating the vector solutions amongst these papers (including the current paper) is each solution's minimal distance $\Delta = \min_{i \neq j} \|\mathbf{w}_i - \mathbf{w}_j\|_2^2$. In [4, 12] Δ behaves like $1/m$. To a large degree, what allowed the improvement of [9] was a modification of the tensoring polynomials thereby increasing the minimal distance Δ to a constant (an arbitrary small one). The analysis of [9] then showed that the resulting solution satisfies all hypermetrics of support $O(\Delta/\gamma)$ with an integrality gap of $2 - \Theta(\Delta)$ (in particular, taking the smallest possible γ, namely $\gamma = \Theta(\sqrt{m/\log m})$, the analysis in [9] shows that the solution satisfies all hypermetrics of support $O(\sqrt{\log\log n/\log n})$).

In the present work we use similar vectors as the one used in [9] but get more mileage by more carefully analyzing the structure of the ℓ_1-metric that emerges from the solution. In particular, while both [12] and [9] use the fact that removing $\mathbf{w}_0$ from the vector solution gives an ℓ_1-metric, in the current paper we crucially use the fact that our vectors arise by applying tensoring polynomials to "sign" vectors. More precisely, we exploit the fact that the ℓ_1-metric corresponding to the vectors $\{\mathbf{w}_i\}_{i \geq 1}$ has a unit representation with large diameter. The bottom line is that our new analysis allows us to show that *any* hypermetric (not just those with support $O(\Delta/\gamma)$) is satisfied as long as Δ/γ is a sufficiently large constant (our argument does not work for the vector construction of [4] but also does not rule out that same vector construction satisfying SDP (1) strengthened by (2)). But note now that if we take $\gamma = \Theta(\sqrt{\log m/m})$ when defining our Frankl-Rödl instances, then for Δ/γ to be constant it suffices to use a tensoring construction where the minimum distance Δ is of order up to $O(\sqrt{\log m/m})$. In particular, the integrality gap obtained by our analysis is $2 - O(\gamma)$; so taking $\gamma = \Theta(\sqrt{\log m/m})$ gives the following strengthened version of Theorem 1:

THEOREM 10. *The integrality gap of the standard SDP relaxation for* VERTEX COVER *on instances of n vertices tightened with* all *hypermetric inequalities is* $2 - O(\sqrt{\log\log n/\log n})$.

Interestingly, the lower bound in Theorem 10 almost matches the upper bound given by Karakostas [13] who gives an SDP for VERTEX COVER tightened with the triangle inequality and which has integrality gap $2 - \Omega(\sqrt{1/\log n})$.

4 ℓ_1 Embeddability Implies Integrality

This section is devoted to proving Theorem 2 which is based on the following simple observation. Let the metric induced by SDP (1) be ℓ_1, realized by some $\{\lambda_S\}_{S\subseteq X}$. Since every subset S induces a cut, we may restrict ourselves only to subsets S that contain the element 0, corresponding to $\mathbf{z}_0$. Now let $\Lambda = \sum_S \lambda_S$ and consider an orthonormal basis $\{\mathbf{e}_Y\}_{Y\subseteq X}$ ($\mathbf{e}_Y$ is indexed by all subsets of X, and is 1 in the Y-th coordinate and 0 elsewhere). For every $A \subseteq X$ we define $\mathbf{u}_A = \sum_{S:A\subseteq S} \sqrt{\lambda_S}\mathbf{e}_S$. Associate also the singleton $\{0\}$ with the vector $\mathbf{u}_\varnothing$ corresponding to the empty set $\varnothing$. The key observation is that the mapping $\mathbf{z}_i \mapsto \mathbf{u}_{\{i\}}$ is an isometry. This is because $\mathbf{u}_{\{i\}} - \mathbf{u}_{\{j\}}\|_2^2$ equals

$$\begin{aligned}\sum_{S:i\in S}\lambda_S + \sum_{S:j\in S}\lambda_S - 2\sum_{S:\{i,j\}\subseteq S}\lambda_S &= \sum_{S:i\in S}\lambda_S - \sum_{S:\{i,j\}\subseteq S}\lambda_S + \sum_{S:j\in S}\lambda_S - \sum_{S:\{i,j\}\subseteq S}\lambda_S \\ &= \sum_{S:j\notin S\&i\in S}\lambda_S + \sum_{S:i\notin S\&j\in S}\lambda_S = \sum_S \lambda_S\delta_S(i,j)\end{aligned}$$

The last expression is exactly $\|\mathbf{z}_i - \mathbf{z}_j\|_2^2$. Theorem 2 now follows from Lemma 11 below.

LEMMA 11. *Let $G = (V,E)$ be a graph for which the metric induced by the solution of the standard SDP* (1) *is an ℓ_1-metric with unit representation $\{\mathbf{z}_i\}$ that has large diameter. Then the vector solution is a convex combination of vertex covers.*

PROOF. For every $S \subseteq \{1,\ldots,n\}$ consider the characteristic vector $\mathbf{y}^S \in \{0,1\}^n$ with $y_i^S = 1$ if and only if $i \in S$. We prove (A) If $\lambda_S > 0$ then S is a vertex cover; and (B) $\frac{1}{4}\left(\|\mathbf{z}_1+\mathbf{z}_0\|^2,\ldots,\|\mathbf{z}_n+\mathbf{z}_0\|^2\right) = \sum_S \frac{1}{\Lambda}\lambda_S\mathbf{y}^S$.

For (A) note that the SDP edge constraints simply require that the triangle inequality $\|\mathbf{z}_0-\mathbf{z}_i\|^2 + \|\mathbf{z}_0-\mathbf{z}_j\|^2 - \|\mathbf{z}_i-\mathbf{z}_j\|^2 \geq 0$ is tight. The same is true for the vectors $\mathbf{u}_{\{i\}}, \mathbf{u}_{\{j\}}$, since the mapping $\mathbf{z}_i \mapsto \mathbf{u}_{\{i\}}$ is an isometry. It follows that for every edge $ij \in E$ we have

$$\begin{aligned}(\mathbf{u}_\varnothing - \mathbf{u}_{\{i\}})(\mathbf{u}_\varnothing - \mathbf{u}_{\{j\}}) &= \mathbf{u}_\varnothing^2 - \mathbf{u}_\varnothing\cdot\mathbf{u}_{\{i\}} - \mathbf{u}_\varnothing\cdot\mathbf{u}_{\{i\}} + \mathbf{u}_{\{i\}}\cdot\mathbf{u}_{\{j\}} \\ &= \left(\sum_S \lambda_S - \sum_{S\ni i}\lambda_S\right) - \left(\sum_{S\ni j}\lambda_S - \sum_{S\supseteq\{i,j\}}\lambda_S\right) \\ &= \sum_{S\not\ni i}\lambda_S - \sum_{j\in S\not\ni i}\lambda_S = \sum_{j\notin S\not\ni i}\lambda_S,\end{aligned}$$

and the last expression equals 0. Since cut coefficients are non-negative, claim 1 follows.

For (B) it suffices to show that for every $i \in V$, $\frac{1}{4}\|\mathbf{z}_i+\mathbf{z}_0\|^2 = \frac{1}{\Lambda}\sum_{S:i\in S}\lambda_S$. To that end, recall that $\|\mathbf{z}_0-\mathbf{z}_i\|^2 = \sum_S \delta_S(0,i)$ and $0 \in S$. Hence, $2\mathbf{z}_i\mathbf{z}_0 = 2 - \sum_{S\not\ni i}\lambda_S$, and $\frac{1}{4}\|\mathbf{z}_0+\mathbf{z}_i\|^2 = \frac{1}{4}\left(4 - \sum_{S\not\ni i}\lambda_S\right)$. The latter equals $\frac{1}{\Lambda}\sum_{S:i\in S}\lambda_S$ iff $\sum_S \lambda_S = 4$. This is guaranteed by Lemma 6, since the ℓ_1-metric induced by SDP (1) has large diameter.

5 Discussion - Open Problems

Our work raises two natural questions. Theorem 1 implies that the most interesting ℓ_1 inequalities are those that are not hypermetric. Given that hypermetrics are the most natural

inequalities to consider, can we identify another family of interesting yet natural inequalities that could potentially strengthen the standard SDP for VERTEX COVER? Since such inequalities are produced by the Lasserre system, it seems we must better characterize the constraints derived by that system. Second, it is interesting to investigate to what extent our arguments apply to general ℓ_1 inequalities. A positive answer could potentially give a first step towards showing tight integrality gaps for VERTEX COVER in the Lasserre system.

References

[1] A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. $O(\sqrt{\log n})$ approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 573–581, New York, NY, USA, 2005. ACM Press.

[2] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, STOC'2004*, pages 222–231, New York, 2004. ACM Press.

[3] D. Avis and J. Umemoto. Stronger linear programming relaxations of max-cut. *Mathematical Programming*, 97(3):451–469, 2003.

[4] M. Charikar. On semidefinite programming relaxations for graph coloring and vertex cover. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 616–620. Society for Industrial and Applied Mathematics, 2002.

[5] M. Deza and M. Laurent. *Geometry of cuts and metrics*, volume 15 of *Algorithms and Combinatorics*. Springer-Verlag, 1997.

[6] I. Dinur and S. Safra. On the hardness of approximating minimum vertex-cover. *Annals of Mathematics*, 162(1):439–486, 2005.

[7] P. Frankl and V. Rödl. Forbidden intersections. *Trans. Amer. Math. Soc.*, 300(1):259–286, 1987.

[8] K. Georgiou, A. Magen, T. Pitassi, and I. Tourlakis. Integrality gaps of $2 - o(1)$ for vertex cover SDPs in the Lovász-Schrijver hierarchy. In *Proceedings of the Forty-Eighth Annual IEEE Symposium on Foundations of Computer Science*, pages 702–712. IEEE, 2007.

[9] K. Georgiou, A. Magen, and I. Tourlakis. Vertex cover resists SDPs tightened by local hypermetric inequalities. In *Integer Programming and Combinatorial Optimization, 13th International Conference, IPCO 2008*, volume 5035 of *Lecture Notes in Computer Science*, pages 140–153. Springer, 2008.

[10] M. X. Goemans and J. Kleinberg. The Lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM J. Discrete Math.*, 11(2):196–204 (electronic), 1998.

[11] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[12] H. Hatami, A. Magen, and E. Markakis. Integrality gaps of semidefinite programs for vertex cover and relations to ℓ_1 embeddability of negative type metrics. In *10th International Workshop, APPROX 2007*, volume 4627 of *Lecture Notes in Computer Science*, pages 164–179. Springer, 2007.

[13] G. Karakostas. A better approximation ratio for the Vertex Cover problem. In *Proceedings of the Thirty-Second International Colloquium on Automata, Languages and Programming*, pages 1043–1050, 2005.

[14] H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, FOCS'97*, pages 406–415, 1997.

[15] S. Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing, STOC'2002*, pages 767–775, New York, 2002. ACM Press.

[16] S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. In *Proceedings of the 18th IEEE Conference on Computational Complexity*, pages 379–386, 2003.

[17] J. B. Lasserre. An explicit exact SDP relaxation for nonlinear 0-1 programs. In *Integer programming and combinatorial optimization (Utrecht, 2001)*, volume 2081 of *Lecture Notes in Comput. Sci.*, pages 293–303. Springer, Berlin, 2001.

[18] L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1(2):166–190, May 1991.

[19] G. Schoenebeck. Linear level lasserre lower bounds for certain k-CSPs. In *FOCS*, pages 593–602. IEEE Computer Society, 2008.

Appendix

PROOF. [Lemma 8 - sketch] Let $G_m^\gamma = (V, E)$. The standard SDP (1) strengthened with the triangle inequality requires for our vectors $\mathbf{w}_i$ that

$$\|\mathbf{w}_i - \mathbf{w}_0\|^2 + \|\mathbf{w}_j - \mathbf{w}_0\|^2 = \|\mathbf{w}_i - \mathbf{w}_j\|^2 \quad , \quad \forall ij \in E \text{ (the edge constraints)} \tag{9}$$

$$\|\mathbf{w}_i - \mathbf{w}_0\|^2 + \|\mathbf{w}_j - \mathbf{w}_0\|^2 \geq \|\mathbf{w}_i - \mathbf{w}_j\|^2 \quad , \quad \forall i, j \in V \text{ (the triangle inequality)} \tag{10}$$

and that all vectors have unit norm. For an edge $ij \in E$ we have $\mathbf{u}_i \cdot \mathbf{u}_j = -1 + 2\gamma$. Recalling that $\mathbf{w}_i = (18\gamma, \sqrt{1-(18\gamma)^2} T_P(\mathbf{u}_i))$ where P is our "tensoring" polynomial (see section 3.2), it is easy to see that for the above constraints to hold it suffices to have

$$-\frac{1-18\gamma}{1+18\gamma} = P(-1+2\gamma) \leq P(x), \ \forall x \in [-1,1], \tag{11}$$

where the left equality takes care of the edge constraints and the right inequality implies the triangle inequality. Set $c_1 = \eta_1\gamma$ and $c_2 = \eta_2\gamma$.

For any distinct points of the hypercube, the high order term of P is negligible so we can disregard it. Recall that $1/\gamma$ is even. For the edge constraint, i.e. the right inequality in (11), we require $P'(-1+2\gamma) = 0$, and $P''(-1+2\gamma) > 0$. The former requires that $\eta_1(1-2\gamma)^{1/\gamma-1} = 1-(\eta_1+2\eta_2)\gamma$. The left constraint of condition (11) requires that $-\frac{1-18\gamma}{1+18\gamma} = (1-(\eta_1+2\eta_2)\gamma)(-1+2\gamma) + \eta_1\gamma(1-2\gamma)^{1/\gamma}$. Solving the system of inequalities with respect to η_1, η_2 and taking the limit $\gamma \to 0$ (or equivalently $m \to \infty$) we get that $\eta_1 = e^2$ and $\eta_2 = (36-3-e^2)/2 > 9$. Finally it is easy to check that the second derivative is positive as required.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Kolmogorov Complexity in Randomness Extraction

John M. Hitchcock[1]*, A. Pavan[2]†, N. V. Vinodchandran[3]‡

[1]Department of Computer Science, University of Wyoming
jhitchco@cs.uwyo.edu

[2]Department of Computer Science, Iowa State University
pavan@cs.iastate.edu

[3]Department of Computer Science and Engineering, University of Nebraska-Lincoln
vinod@cse.unl.edu

ABSTRACT.
We clarify the role of Kolmogorov complexity in the area of randomness extraction. We show that a computable function is an almost randomness extractor if and only if it is a Kolmogorov complexity extractor, thus establishing a fundamental equivalence between two forms of extraction studied in the literature: Kolmogorov extraction and randomness extraction. We present a distribution $\mathcal{M}_k$ based on Kolmogorov complexity that is complete for randomness extraction in the sense that a computable function is an almost randomness extractor if and only if it extracts randomness from $\mathcal{M}_k$.

1 Introduction

The problem of extracting pure randomness from weak random sources has received intense attention in the last two decades producing several exciting results. The main goal in this topic is to give explicit constructions of functions that are known as *randomness extractors*; functions that output almost pure random bits given samples from a weak source of randomness which may be correlated and biased. Randomness extractors have found applications in several areas of theoretical computer science including complexity theory and cryptography. The body of work on randomness extractors is vast and we do not attempt to list them here. Instead, we refer the readers to survey articles by Nisan and Ta-Shma [10] and Shaltiel [13], and Rao's thesis [11] for an extensive exposition on the topic (with the caveat that some of the recent advances are not reported in these articles).

We will focus on a type of randomness extractors known as *multi-source* extractors. These are multi-input functions with the property that if the inputs come from independent distributions with certain guaranteed randomness, typically measured by their *minentropy*, then the output distribution will be close to the uniform distribution. A distribution over n-bit strings is said to have minentropy k, if any element in the support of the distribution

*This research was supported in part by NSF grants 0515313 and 0652601. Part of this research work done while the author was on sabbatical at CWI.

†This research was supported in part by NSF grants 0830479 and 0916797.

‡This research was supported in part by NSF grants 0830730 and 0916525.

© Hitchcock, Pavan, Vinodchandran; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 215–226
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2320

has a probability $\leq 2^{-k}$. A function $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^m$ is a 2-source extractor for minentropy k if for any two independent distributions X and Y on $\{0,1\}^n$ with minentropy k, the output $f(X,Y)$ is statistically close to the uniform distribution. It is known that such extractors exist for all minentropy levels with optimal parameters [3, 4], but explicitly constructing 2-source extractors for sources with low minentropy is a very active research question.

While minentropy characterizes the amount of randomness present in a probability distribution, Kolmogorov complexity characterizes the amount of randomness present in *individual strings*. The Kolmogorov complexity of a string x, denoted by $K(x)$, is the the length of the shortest program that outputs x. If $K(x) = m$, then x can be viewed as containing m bits of randomness. A string x is *Kolmogorov random* if its Kolmogorov complexity is close to the length of x. A natural notion that arises is that of *Kolmogorov extractors*: explicit functions that extract Kolmogorov complexity from strings that need not be Kolmogorov random. More formally, a 2-string Kolmogorov extractor for complexity k is a function $f : \Sigma^n \times \Sigma^n \rightarrow \Sigma^m$ such that $K(f(x,y))$ is close to m whenever $K(x), K(y) \geq k$ and x and y are Kolmogorov independent ($K(xy) \simeq K(x) + K(y)$). Kolmogorov extractors have recently been of interest to researchers [1, 5, 14, 15]. One of the main observations that emerged from this research is that a randomness extractor is also a Kolmogorov extractor. In particular, in [5], the authors show that the construction due to Barak, Impagliazzo and Wigderson [2] of a multisource extractor is also a Kolmogorov extractor. Zimand takes this approach further and gives constructions of Kolmogorov extractors in other settings [14, 15]. Thus, this line of research uses randomness extractors as a tool in Kolmogorov complexity research. However, the role of Komogorov complexity in the area of randomness extraction has not yet been explored by researchers. We take a step in this direction.

We ask the following question. Is it true that a Kolmogorov extractor is also a randomness extractor? While randomness extractors concern information-theoretic randomness, Kolmogorov extractors concern computational randomness. Thus intuitively it appears that Kolmogorov extractors are weaker objects than randomness extractors. Moreover, if we use the strict definition of extraction, it is easy to come up with a counterexample to this converse. Let f be a Kolmogorov extractor, then $f \circ 1$ (output of f concatenated with bit 1) is also a Kolmogorov extractor. But $f \circ 1$ is not a randomness extractor for any function f because it never outputs 50% of the strings - strings that end with 0. The reason for this counterexample is that any Kolmogorov complexity measure is precise only up to a small additive term. Consequently, a string x of length n is considered Kolmogorov random even if its Kolmogorov complexity is only $n - a(n)$ for a slow growing function $a(n)$ such as a constant multiple of $\log n$ [5]. Thus a more fruitful question is to ask whether a Kolmogorov extractor is also an *almost* randomness extractor. An almost randomness extractor is like a traditional randomness extractor except that we only require the output of an almost extractor to be close to a distribution with minentropy $m - O(\log n)$. For a traditional extractor, the output has to be close to the uniform distribution - the only distribution with minentropy m. Such almost extractors have been considered in the literature (see for example [12]).

Our first contribution is to show an equivalence between Kolmogorov extraction and the above-mentioned slightly relaxed notion of randomness extraction. The following statement is very informal and Section 3 is devoted to giving a precise statement with a proof.

RESULT 1. *A computable function f is a Kolmogorov extractor if and only if f is an almost randomness extractor.*

A randomness extractor is a universal object in the sense that it should extract randomness from *all* distributions with certain minentropy. Can this universality be shifted to a distribution? That is, is there a distribution D so that a computable function f is an extractor if and only if f extracts randomness from D? We call such a distribution a *complete* distribution for randomness extraction. Kolmogorov complexity has proved to be useful in the discovery of distributions with a similar universality property in other areas of computer science including average-case analysis [8] and learning theory [7].

Our second contribution is to present a complete distribution, based on Kolmogorov complexity, for randomness extraction. Fix an input length n. For a number k consider the distribution $\mathcal{M}_k$ that puts uniform weight on all strings of length n with Kolmogorov complexity $\leq k$. Motivated by the proof of our first result we show that the distribution $\mathcal{M}_k$ is a complete distribution for almost extractors. The following statement is informal and the full details are in Section 4.

RESULT 2. *For any k, there is a $k' = k + O(\log n)$ so that $\mathcal{M}_{k'}$ is complete for almost extractors with minentropy parameter k.*

2 Preliminaries, Definitions, and Basic Results

Kolmogorov Extractors

We only review the essentials of Kolmogorov complexity and refer to the textbook by Li and Vitányi [9] for a thorough treatment of the subject. For a string $x \in \{0,1\}^*$, $l(x)$ denotes the length of x. We use the following standard encoding function where a pair $\langle x, y\rangle$ is encoded as $1^{l(l(x))}0l(x)xy$. By viewing $\langle x, y, z\rangle$ as $\langle x, \langle y, z\rangle\rangle$, this encoding can be extended to 3-tuples (and similarly for any k-tuple).

Let U be a universal Turing machine. Then for any string $x \in \{0,1\}^*$, the Kolmogorov complexity of x is defined as

$$K(x) = \min\{l(p) \mid U(p) = x\},$$

that is, the length of a shortest program p that causes U to print x and halt. If we restrict the set of programs to be prefix-free, then the corresponding measure is known as prefix-free Kolmogorov complexity. These two complexity measures only differ by an additive logarithmic factor. We will work with the above-defined standard measure. Since we are flexible about additive logarithmic factors in this paper, our results will hold with the prefix-free version also.

Kolmogorov extractors are computable functions which convert strings that have a guaranteed amount of Kolmogorov complexity into a Kolmogorov random string. We give a general definition of Kolmogorov extractors involving a parameter for *dependency* between the input strings. Consequently, instead of aiming for maximum complexity in the output string, we will consider extractors which lose an additive factor equal to the dependency in the inputs. The following notion of dependency we use is equivalent to the well-studied

notion of *mutual information* in the Kolmogorov complexity literature up to an additive log factor. However, we prefer to use the term dependency in this paper.

DEFINITION 1.[Dependency] *For two strings x and y of the same length, the* dependency between x and y *is*

$$\mathrm{dep}(xy) = K(x) + K(y) - K(xy).$$

DEFINITION 2.[Kolmogorov Extractor] *An $(n, m(n), k(n), \alpha(n))$* Kolmogorov extractor *is a uniformly computable family $\{f_n\}_n$ of functions $f_n : \Sigma^n \times \Sigma^n \to \Sigma^{m(n)}$ where there is a constant c such that for all n, for all $x, y \in \Sigma^n$ with $K(x) \geq k(n)$, $K(y) \geq k(n)$, and $\mathrm{dep}(xy) \leq \alpha(n)$, we have*

$$K(f_n(x, y)) \geq m(n) - \mathrm{dep}(xy) - c \log n.$$

The computability restriction is required to make the definition nontrivial. Otherwise it is easy to come up with Kolmogorov extractors: for any pair of inputs at length n, just output a fixed string of length $m(n)$ that has maximal Kolmogorov complexity.

Randomness Extractors

Randomness extractors are functions which convert weak random sources to a distribution that is statistically close to the uniform distribution. A weak random source is characterized by its minentropy which is defined as follows.

DEFINITION 3. *For a probability distribution X over a universe S, the* minentropy *of X is*

$$-\log\left(\max_{s \in S} X(s)\right) = \min_{s \in S}\left(\log \frac{1}{X(s)}\right).$$

Here we are writing $X(s)$ for the probability that distribution X assigns to outcome s. For an event $T \subseteq S$, $X(T) = \sum_{s \in T} X(s)$ is the probability of T under X.

DEFINITION 4. *For any two distributions X and Y on a universe S, their* statistical distance $|X - Y|$ *is*

$$|X - Y| = \max_{T \subseteq S} |X(T) - Y(T)| = \frac{1}{2} \sum_{s \in S} |X(s) - Y(s)|$$

. If $|X - Y| \leq \epsilon$, we say X and Y are ϵ-close to each other.

DEFINITION 5.[Almost Randomness Extractor] *An $(n, m(n), k(n), \epsilon(n))$* almost randomness extractor *is a family $\{f_n\}_n$ of functions $f_n : \Sigma^n \times \Sigma^n \to \Sigma^{m(n)}$ where there is a constant c such that for all n, for every pair of independent distributions X and Y over Σ^n with minentropy at least $k(n)$, the distribution $f_n(X, Y)$ is $\epsilon(n)$-close to a distribution with minentropy at least $m(n) - c \log n$. Moreover, f is uniformly computable.*

A distribution X over Σ^n is called a *flat distribution* if it is uniform over some subset of Σ^n. For a flat distribution X, we will use X also to denote the support of the distribution X. The following useful theorem due to Chor and Goldreich [3] states that every function that extracts randomness from flat distributions is a randomness extractor.

THEOREM 6. *[3] Let f be a function from $\Sigma^n \times \Sigma^n$ to Σ^m. Suppose for every pair of independent flat distributions X and Y with minentropy k, $f(X,Y)$ is ϵ-close to having minentropy $m - c\log n$. Then f is a (n, m, k, ϵ) almost randomness extractor.*

Let D be a distribution over Σ^m induced by a distribution over $\Sigma^n \times \Sigma^n$. We call D a *nice distribution* if for all $z \in \Sigma^m$, $D(z)$ is a rational number of the form p/q with $q \leq 2^{2n}$. This restriction allows us to effectively cycle through all nice distributions. For any distribution D with minentropy k, there is a nice distribution D' with the same minentropy so that the statistical distance between D and D' is at most $1/2^n$. Because of this we will assume that distribution are nice whenever necessary.

The following lemma due to Guruswami, Umans, and Vadhan [6] is useful to obtain a bound on the minentropy of a distribution. We will state it for nice distributions although the original statement and the proof do not have such a restriction. Their proof can be easily modified to prove this case also.

LEMMA 7. *[6] Let D be a nice distribution and s be an integer. Suppose that for every set S of size s, $D(S) \leq \epsilon$. Then D is ϵ-close to a nice distribution with minentropy at least* $\log(s/\epsilon)$.

Remarks and Clarifications

Although it is typical requirement for the extractors to be *efficiently* computable, the only requirement we need in our proofs is that the extractors are computable. Hence, we will not mention any resource restrictions here. Here we only focus on extractors with 2 inputs. The connection we prove here also holds for extractors with k inputs for any constant $k \geq 2$ with identical proofs. Although the parameters in the definition of the extractors depend on the input length n, we will omit it in the rest of the paper. For instance, a $(n, m(n), k(n), \alpha(n))$ Kolmogorov extractor will be denoted as an (n, m, k, α) extractor. In addition, we also assume that the parameters that depend on input length n are computable functions of n. Finally, henceforth by a randomness extractor we mean an almost randomness extractor unless otherwise mentioned.

Why is there a dependency parameter in the definition of Kolmogorov extractor? Our aim is to establish a tight connection between randomness extractors and Kolmogorov extractors. Randomness extractors typically have four parameters; input length n, output length m, minentropy bound k, and the error parameter ϵ. Except for the error parameter, there is an obvious mapping of parameters between Kolmogorov and randomness extractors. But there appears to be no natural notion of "error" in Kolmogorov extraction. What is a choice for the parameter in the definition of Kolmogorov extractor analogous to the error parameter? Our theorems indicate that the dependency is a good choice.

3 The Equivalence

3.1 Kolmogorov Extractor is a Randomness Extractor

In this subsection we show that for appropriate settings of parameters, a Kolmogorov extractor is also a randomness extractor. First we will give a simple argument for the special

case when the dependency parameter is $O(\log n)$. In this case we get a inverse polynomial error for the randomness extractor. We will only give a sketch of the proof since the subsequent theorem for the general case subsumes this case.

A Special Case

The proof of this special case is a simple application of the following well known coding theorem.

THEOREM 8. *[Coding Theorem] Let D be a probability distribution over $\{0,1\}^*$ that is computable by a program P, there is a constant c such that*

$$\frac{1}{2^{K(x)}} \geq \frac{c}{2^{|P|}} D(x).$$

THEOREM 9. *Let f be a (n, m, k, α) Kolmogorov extractor with $\alpha = O(\log n)$. Then f is a (n, m, k', ϵ) almost randomness extractor where $k' = k + O(\log n)$ and $\epsilon = 1/\mathtt{poly}(n)$.*

PROOF. We provide a proof sketch. Let c be the constant associated with the Kolmogorov extractor f. That is, $K(f(x,y)) \geq m - c\log n - \mathrm{dep}(xy)$ provided $K(x) \geq k$, $K(y) \geq k$, and $\mathrm{dep}(xy) \leq \alpha$.

We will show that for every pair of flat distributions X and Y with minentropy k', $f(X,Y)$ is ϵ-close to a nice distribution with minentropy at least $m - (c+6)\log n$. Then by Theorem 6, it will follow that f is an almost randomness extractor for minentropy k'. For the purpose of contradiction, suppose there are flat distributions X and Y with minentropy k' so that $f(X,Y)$ is ϵ far from all nice distributions with minentropy at least $m - (c+6)\log n$. Let X and Y be the first such distributions (in some fixed ordering of distributions).

The number of flat distributions with minentropy k' is finite, and the number of nice distributions over Σ^m with minentropy at least $m - (c+6)\log n$ is also finite. Thus there is a program p which given n as input, produces the distributions X and Y. Thus the size of p is at most $2\log n$ for large enough n. Let D denote the distribution $f(X,Y)$.

The idea of the rest proof is as follows. Consider the following set S.

$$S = \{\langle x, y\rangle \in X \times Y \mid K(x) \geq k, K(y) \geq k, \text{ and } \mathrm{dep}(xy) \leq 10\log n\}.$$

First using a simple counting argument it is easy to show that S is a large set and hence probability of the complement of S with respect to $X \times Y$ is small. Since f is a Kolmogorov extractor, for all elements $(x,y) \in S$, $K(z)$ is close to m where $z = f(x,y)$. Since D is computable, by the coding theorem, it follows that $D(z) \leq 1/2^{m-O(\log n)}$. Thus, except for a small fraction of strings in $f(\overline{S})$, the strings in the range of f satisfies the minentropy condition. Hence D must be close to a distribution with minentropy $m - c\log n$. ∎

The General Case

We now state and prove the theorem for a general setting of parameters. The proof follows the line of argument of the proof of the special case. But we will use Lemma 7 instead of the coding theorem.

THEOREM 10. *Let f be a (n, m, k, α) Kolmogorov extractor. Then f is a (n, m, k', ϵ) almost randomness extractor where*

(a) if $k' - k > \alpha - 4\log n + 1$, then $\epsilon \leq \frac{1}{2^{\alpha - 4\log n - 1}}$.

(b) if $k' - k \leq \alpha - 4\log n + 1$, then $\epsilon \leq \frac{1}{2^{k' - k - 2}}$.

PROOF. Let c be the constant associated with the Kolmogorov extractor f. That is, $K(f(x, y)) \geq m - c\log n - \mathrm{dep}(xy)$ provided $K(x) \geq k$, $K(y) \geq k$, and $\mathrm{dep}(xy) \leq \alpha$.

We will show that for every pair of flat distributions X and Y with minentropy k', $f(X, Y)$ is ϵ-close to a nice distribution with minentropy at least $m - (c + 9)\log n$ where ϵ is as given in the statement of the theorem. Then by Theorem 6, it will follow that f is an almost randomness extractor for minentropy k'. For the purpose of contradiction, suppose there are flat distributions X and Y with minentropy k' so that $f(X, Y)$ is ϵ far from all nice distribution with minentropy at least $m - (c + 9)\log n$. Let X and Y be the first such distributions (in some fixed ordering of distributions). For simplicity, we will denote the supports of distributions X and Y also by X and Y, respectively. Let D denote the distribution $f(X, Y)$. D is a nice distribution.

The number of flat distributions with minentropy k' is finite, and the number of nice distributions over Σ^m with minentropy at least $m - (c + 9)\log n$ is also finite. Thus there is a program p which given n, c and a code for f as input, produces the flat distributions X and Y by brute-force search method. The size of p is at most $2\log n$ for large enough n. We will split the rest of the proof into two cases.

Case (a). $k' - k > \alpha - 4\log n + 1$.

Define the "good set" S as

$$S = \{\langle x, y\rangle \in X \times Y \mid K(x) \geq k, K(y) \geq k, \text{ and } \mathrm{dep}(xy) \leq \alpha\}.$$

Let S' be the compliment of S within $X \times Y$. That is $S' = X \times Y \setminus S$. We will bound the size of S'. Observe that S' is a subset of the union of following sets:

$$S_1 = \{\langle x, y\rangle \in X \times Y \mid K(x) < k\},$$

$$S_2 = \{\langle x, y\rangle \in X \times Y \mid K(y) < k\},$$

$$S_3 = \{\langle x, y\rangle \in X \times Y \mid \mathrm{dep}(xy) > \alpha\}.$$

Clearly, sizes of S_1 and S_2 are bounded by $2^{k+k'}$. We will bound $|S_3|$. Since the program p produces X and Y and $|X| = |Y| = 2^{k'}$, every string in $X \cup Y$ has Kolmogorov complexity at most $k' + 2\log n$. Thus for any $\langle x, y\rangle \in S_3$ we have that $K(xy) = K(x) + K(y) - \mathrm{dep}(xy) \leq 2k' + 4\log n - \alpha$. So $|S_3| \leq 2^{2k' + 4\log n - \alpha}$. Hence $|S'| \leq |S_1 \cup S_2 \cup S_3| \leq |S_1| + |S_2| + |S_3| \leq 2^{k+k'+1} + 2^{2k' + 4\log n - \alpha}$. Since $k' - k > \alpha - 4\log n + 1$, this sum is $\leq 2^{2k' + 4\log n - \alpha + 1}$. Thus we have the following bound on the probability of S'.

CLAIM 11. *If $k' - k > \alpha - 4\log n + 1$ then $\Pr_{X \times Y}(S') \leq \frac{1}{2^{\alpha - 4\log n - 1}}$*

We assumed that f is not an almost randomness extractor. That is the distribution is ϵ-far from any nice distribution with minentropy $m - (c + 9)\log n$. By Lemma 7, there is a set $U \subseteq \Sigma^m$ of size $2^{m - \alpha - (c+4)\log n}$ such that $D(U) > 1/2^{\alpha - 5\log n}$. Since a program of size

$2\log n$ produces distributions X and Y and f is computable, there is a program of size at most $3\log n$ that produces the set U. Thus for all $u \in U$, $K(u) < m - \alpha - c\log n$.

Since $\Pr_{X\times Y}(S') \leq \frac{1}{2^{\alpha-4\log n-1}} \leq \frac{1}{2^{\alpha-5\log n}}$ and $D(U) > \frac{1}{2^{\alpha-5\log n}}$, there must exist a tuple $\langle x,y\rangle \in S$ so that $f(x,y) \in U$ and for this tuple we have $K(f(x,y)) < m - \alpha - c\log n$. This is a contradiction since f is a Kolmogorov extractor and for all elements $\langle x,y\rangle \in S$, $K(f(x,y)) \geq m - \mathrm{dep}(xy) - c\log n \geq m - \alpha - c\log n$.

Case (b). $k' - k \leq \alpha - 4\log n + 1$.

The proof is very similar. Define the "good set" S as

$$S = \{\langle x,y\rangle \in X\times Y \mid K(x) \geq k, K(y) \geq k, \text{ and } \mathrm{dep}(xy) \leq k' - k + 4\log n\}.$$

In this case, we can bound the size of S' (the compliment of S within $X \times Y$) by considering the following sets.

$$S_1 = \{\langle x,y\rangle \in X\times Y \mid K(x) < k\},$$

$$S_2 = \{\langle x,y\rangle \in X\times Y \mid K(y) < k\},$$

$$S_3 = \{\langle x,y\rangle \in X\times Y \mid \mathrm{dep}(xy) > k' - k + 4\log n\}.$$

Sizes of S_1 and S_2 are bounded by $2^{k+k'}$. We will bound $|S_3|$. Since the program p produces X and Y and $|X| = |Y| = 2^{k'}$, every string in $X \cup Y$ has Kolmogorov complexity at most $k' + 2\log n$. Thus for any $\langle x,y\rangle \in S_3$ we have that $K(xy) = K(x) + K(y) - \mathrm{dep}(xy) \leq 2k' + 4\log n - (k' - k + 4\log n) = k' + k$. So $|S_3| \leq 2^{k'+k}$. Hence $|S'| \leq |S_1| + |S_2| + |S_3| \leq 2^{k+k'+1} + 2^{k'+k} \leq 2^{k+k'+2}$. Thus in this case we have the following bound on the probability of S'.

CLAIM 12. *If* $k' - k \leq \alpha - 4\log n + 1$ *then* $\Pr_{X\times Y}(S') \leq \frac{1}{2^{k'-k-2}}$

We assumed that distribution D is ϵ-far from any nice distribution with minentropy $m - (c+9)\log n$. By Lemma 7, there is a set $U \subseteq \Sigma^m$ of size $2^{m-(k'-k+4\log n)-(c+4)\log n}$ such that $D(U) > 1/2^{k'-k-\log n}$. Since a program of size $2\log n$ produces distributions X and Y and f is computable, there is a program of size at most $3\log n$ that produces the set U. Thus for all $u \in U$, $K(u) < m - (k' - k + 4\log n) - c\log n$. But since $\Pr_{X\times Y}(S') \leq \frac{1}{2^{k'-k-2}} \leq \frac{1}{2^{k'-k-\log n}}$ and $D(U) > \frac{1}{2^{k'-k-\log n}}$, there must exist a tuple $\langle x,y\rangle \in S$ so that $f(x,y) \in U$. This contradicts the fact that f is a Kolmogorov extractor with the prescribed parameters. ∎

3.2 Randomness Extractor is a Kolmogorov Extractor

In this subsection we show that an almost randomness extractor is also a Kolmogorov extractor. We follow the line of proof presented in [5] where it is shown that the construction of a multisource extractor in [2] is also a Kolmogorov extractor. Here we note that in fact the argument works even for almost randomness extractors.

THEOREM 13. *An (n, m, k, ϵ) almost extractor is also a (n, m, k', α) Kolmogorov extractor for $\alpha < \log \frac{1}{\epsilon} - 6 \log n$ and $k' = k + 3 \log n$.*

PROOF. Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ be an (n, m, k, ϵ) almost extractor. Let c be the the associated constant. That is, the minentropy guarantee of the output of f is $m - c \log n$.

Let x_1 and x_2 be two strings with $K(x_1) = k_1 \geq k'$, $K(x_2) = k_2 \geq k'$ and $\mathrm{dep}(x_1 x_2) \leq \alpha$. Let X_1 and X_2 be subsets of $\{0,1\}^n$ with Kolmogorov complexity at most k_1 and k_2 respectively. That is, $X_1 = \{x \in \{0,1\}^n | K(x) \leq k_1\}$ and $X_2 = \{x \in \{0,1\}^n | K(x) \leq k_2\}$. We will also use X_1 and X_2 to denote the flat distributions that put uniform weight on sets X_1 and X_2 respectively (in the next section, we give specific notation for these distributions).

For $t = m - \mathrm{dep}(x_1 x_2) - (c + 6) \log n$, let $T \subseteq \{0,1\}^m$ be the set of strings with Kolmogorov complexity at most t. That is, $T = \{z \mid K(z) < t\}$. We will show that for all u, v so that $f(u, v) \in T$, $K(uv) < k_1 + k_2 - \mathrm{dep}(x_1 x_2)$. This will show the theorem as this will imply $f(x_1, x_2) \notin T$ and hence $K(f(x_1, x_2)) > m - \mathrm{dep}(x_1 x_2) - (c + 6) \log n$.

CLAIM 14. *For all $u \in X_1$ and $v \in X_2$ so that $f(u, v) \in T$, $K(uv) < k_1 + k_2 - \mathrm{dep}(x_1 x_2)$.*

PROOF. It is clear that $|X_i| \leq 2^{k_i}$. Since each string in the set $0^{(n-k)}\{0,1\}^k$ has Kolmogorov complexity $\leq k + 2 \log n + O(\log \log n) \leq k_i$ (for large enough n), we also have that $|X_i| \geq 2^k$. Thus $\Pr_{X_i}(x) \leq \frac{1}{2^k}$ for any $x \in X_i$, X_i has minentropy at least k and f works for $X_1 \times X_2$.

Consider the output distribution $f(X_1, X_2)$ on $\{0,1\}^m$. Let us call this distribution D. Since f is an almost extractor the distribution D is ϵ-close to a distribution with minentropy $m - c \log n$.

Since $|T| \leq 2^t = 2^{m - \mathrm{dep}(x_1 x_2) - (c+6) \log n}$ and D is ϵ-close to a distribution with minentropy $m - c \log n$, we have the following.

$$\begin{aligned} \Pr_D(T) &\leq \frac{|T|}{2^m} \times n^c + \epsilon \\ &\leq 2^{-\mathrm{dep}(x_1 x_2) - 6 \log n} + 2^{-\alpha - 6 \log n} \\ &\leq 2^{-\mathrm{dep}(x_1 x_2) - 6 \log n + 1} \end{aligned}$$

The last two inequalities follow because $\alpha \leq \log(\frac{1}{\epsilon}) - 6 \log n$ and $\mathrm{dep}(x_1 x_2) \leq \alpha$.

Consider the set $S = f^{-1}(T) \cap X_1 \times X_2 \subseteq \{0,1\}^n \times \{0,1\}^n$. We will first bound $|S|$. Every tuple from S gets a weight of $\geq 1/2^{k_1 + k_2}$ according to the joint distribution $X_1 \times X_2$. Thus we have

$$\begin{aligned} \frac{|S|}{2^{k_1 + k_2}} &\leq \Pr_{(X_1, X_2)}(S) \\ &= \Pr_D(T) \\ &\leq (2^{-\mathrm{dep}(x_1 x_2) - 6 \log n + 1}) \end{aligned}$$

Hence $|S| \leq 2^{k_1 + k_2 - \mathrm{dep}(x_1 x_2) - 6 \log n + 1}$.

The sets X_1, X_2, and T are recursively-enumerable and f is computable. Hence there is a program that given $n, k_1, k_2, \mathrm{dep}(x_1 x_2)$, a code for f, and c, enumerates the elements of S. Hence for any $\langle u, v \rangle \in S$, $K(uv) \leq \log |S| + 4 \log n + O(\log \log n) \leq \log |S| + 5 \log n$ for

large enough n. Since $|S| \leq 2^{k_1+k_2-\text{dep}(x_1x_2)-6\log n+1}$, $K(uv) < k_1 + k_2 - \text{dep}(x_1x_2)$ and the claim follows. ∎

3.3 The Error Parameter vs the Dependency Parameter

Theorem 13 suggests that there is a nice logarithmic relation between error of an almost extractor and the dependency parameter of the corresponding Kolmogorov extractor. In particular, in Theorem 13, we show that an (n, m, k, ϵ) almost randomness extractor is a (n, m, k', α) Kolmogorov extractor for $\alpha = \log(1/\epsilon) - O(\log n)$ for k' slightly larger than k ($k' = k + O(\log n)$). On the other hand, the parameters we get in the proof of the converse direction (Kolmogorov extractor ⇒ randomness extractor) are not fully satisfactory. Ideally we would like to prove that every (n, m, k, α) Kolmogorov extractor is a (n, m, k', ϵ) almost randomness extractor with $k' = k + O(\log n)$ and $\epsilon = 1/2^{\alpha - O(\log n)}$ which will be a true converse to Theorem 13. We note that this is not possible in general. In particular, we show that for a (n, m, k, α) Kolmogorov extractor to be a (n, m, k', ϵ) almost randomness extractor with $\epsilon = 2^{\alpha - O(\log n)}$, k' has to be greater than $k + \alpha$ (upto a log factor).

THEOREM 15. *Let f be a (n, m, k, α) Kolmogorov extractor. Then there exists a computable function g which is also a (n, m, k, α) Kolmogorov extractor but g is not a (n, m, k', ϵ) almost randomness extractor for $\epsilon < \frac{1}{2^{k'-k+4\log n}}$ for any k' where $k' < m + k - c\log n$ for all constants c.*

PROOF. Let f be a (n, m, k, α) Kolmogorov extractor. Consider the set $U \subseteq \{0,1\}^n$ defined as $U = \{0,1\}^{k-3\log n}0^{n-k+3\log n}$. For any string $x \in U$, $K(x) < k$. Define the function g as follows: $g(x, y) = 0^m$ if $x \in U$ and $g(x, y) = f(x, y)$ otherwise.

Since membership in the set U is easy to decide and f is computable, g is computable. Also, by definition of g, for all pair of strings x, y so that $K(x) \geq k$, $K(y) \geq k$ and $\text{dep}(x, y) \leq \alpha$, $g(x, y) = f(x, y)$. Hence g is a (n, m, k, α) Kolmogorov extractor.

Now consider two flat distributions X and Y of size $2^{k'}$ such that $U \subseteq X$. Let D denotes the distribution $g(X \times Y)$. Notice that $\Pr_D(0^m) \geq \Pr_X(x \in U) \geq \frac{1}{2^{k'-k+3\log n}}$. Now an easy calculation (omitted because of space constraints) proves the theorem. ∎

4 A Complete Distribution for Randomness Extraction

For integers k and n, let $\mathcal{M}^n_{k'}$ denote the distribution that places uniform weight on the set $\{x \in \{0,1\}^n \mid K(x) \leq k\}$. That is $\mathcal{M}^n_k$ is uniform over all the strings with Kolmogorov complexity $\leq k$. As n will be clear from the context, we will omit n from the notation and call it $\mathcal{M}_k$. We show that $\mathcal{M}_k$ is a complete distribution for randomness extraction in the sense that a computable function f is an almost randomness extractor if and only if it extracts randomness from two independent copies of $\mathcal{M}_k$.

This result is motivated by the proof of the equivalence theorem. Notice that in the proof that a randomness extractor f is also a Kolmogorov extractor, we essentially show that if f extracts randomness from the class of distributions $\{\mathcal{M}_l\}_{l \geq k}$, then it is a Kolmogorov extractor. The other implication shows that if f is a Kolmogorov extractor then it is also a

randomness extractor. Thus intuitively we get that the class $\{\mathcal{M}_l\}_{l\geq k}$ is complete. Below we give a simple argument for completeness.

THEOREM 16. *A computable function f is a (n, m, k, ϵ) almost extractor if and only if there is a constant c so that $f(\mathcal{M}_{k'} \times \mathcal{M}_{k'})$ is ϵ' close to a distribution with minentropy $m - c \log n$ where $k' = k + 2\log n$ and $\epsilon' = \epsilon/n^4$.*

PROOF. The set $0^{(n-k)}\{0,1\}^k$ is a subset of $\mathcal{M}_k$ since every strings in this set has Kolmogorov complexity $\leq k + \log n + O(\log\log n) < k'$. Hence $\mathcal{M}_{k'}$ has minentropy $\geq k$ and since f is an almost extractor for minentropy k it should also extract randomness from $\mathcal{M}_{k'} \times \mathcal{M}_{k'}$.

For the other direction, let f be a function that extracts from $\mathcal{M}_{k'} \times \mathcal{M}_{k'}$. Hence there is a constant c so that $f(\mathcal{M}_{k'} \times \mathcal{M}_{k'})$ is ϵ' close to a distribution with minentropy $m - c\log n$.

For the sake of contradiction suppose f is not an almost extractor for minentropy k. Let X and Y be first two flat distributions over $\{0,1\}^n$ for which the distribution $D = f(X, Y)$ is ϵ-far from all nice distributions with minentropy $m - (c+4)\log n$. Observe that there is a program p which given n, c, and a code for f produces the distributions X and Y. Thus for any $x \in X$, we have $K(x) \leq k + \log n + O(\log\log n) \leq k'$. Similarly for $y \in Y$. Hence we have the following claim.

CLAIM 17. *For all $x \in X$, $K(x) \leq k'$. Similarly for all $y \in Y$, $K(y) \leq k'$. Hence $X \subseteq \mathcal{M}_{k'}$ and $Y \subseteq \mathcal{M}_{k'}$.*

We will show that for all $T \subseteq \{0,1\}^m$, $\Pr_D(T) \leq \frac{|T|}{2^m} \times n^{c+4} + \epsilon$. This suffices to show that D is ϵ-close to a distribution with minentropy $m - (c+4)\log n$.

$$\begin{aligned}
\Pr_D(T) &= \Pr_{X\times Y}(f^{-1}(T) \cap X \times Y) \\
&= \frac{|f^{-1}(T) \cap X \times Y|}{2^{2k}} \\
&\leq \Pr_{f(\mathcal{M}_k \times \mathcal{M}_k)}(T) \times n^4 \\
&\leq (\frac{|T|}{2^m} n^c + \epsilon') \times n^4 \\
&= \frac{|T|}{2^m} n^{c+4} + \epsilon
\end{aligned}$$

The inequality second from the last is because of the assumption that $f(\mathcal{M}_k \times \mathcal{M}_k)$ is ϵ' close to a distribution with minentropy $m - c\log n$. ∎

5 Acknowledgments

We thank Elvira Mayrdomo for useful discussions during the initial phase of this research. We thank the staff of Iowa Western Community College, Atlantic, Iowa for providing us meeting facilities throughout the course of this research.

References

[1] H. Buhrman, L. Fortnow, I. Newman, and N. Vereshchagin. Increasing Kolmogorov complexity. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *LNCS*, pages 412–421, 2005.

[2] B. Barak, R. Impagliazzo, and A. Wigderson. Extracting randomness using few independent sources. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 384–393. IEEE Computer Society, 2004.

[3] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988.

[4] Y. Dodis and R. Oliveira. On extracting randomness over a public channel. In *Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, volume 2764 of *LNCS*, pages 252–263, 2003.

[5] L. Fortnow, J. M. Hitchcock, A. Pavan, N. V. Vinodchandran, and F. Wang. Extracting Kolmogorov complexity with applications to dimension zero-one laws. In *Proceedings of the 33rd International Colloquium on Automata, Languages, and Programming*, number 4051 in LNCS, pages 335–345, 2006.

[6] V. Guruswami, C. Umans, and S. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. In *IEEE Conference on Computational Complexity*, pages 96–108, 2007.

[7] M. Li and P. Vitányi. Learning simple concept under simple distributions. *SIAM Journal on Computing*, 20(5):911–935, 1991.

[8] M. Li and P. Vitányi. Average case complexity under the universal distribution equals worst-case complexity. *Information Processing Letters*, 42(3):145–149, 1992.

[9] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, 1997.

[10] N. Nisan and A. Ta-Shma. Extracting randomness: A survey and new constructions. *Journal of Computer and System Sciences*, 42(2):149–167, 1999.

[11] A. Rao. *Randomness Extractors for Independent Sources and Applications*. PhD thesis, University of Texas, Austin, 2006.

[12] A. Rao. A 2-source almost-extractor for linear entropy. In *APPROX-RANDOM*, pages 549–556, 2008.

[13] A. Shaltiel. *Current trends in theoretical computer science. Vol 1 Algorithms and Complexity*, chapter Recent Developments in extractors. World Scientific, 2004.

[14] M. Zimand. Two sources are better than one for increasing the Kolmogorov complexity of infinite sequences. In *Computer Science Symposium in Russia*, pages 326–338, 2008.

[15] M. Zimand. Extracting the Kolmogorov complexity of strings and sequences from sources with limited independence. In *Symposium on Theoretical Aspects of Computer Science*, pages 607–708, 2009.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Donation Center Location Problem*

Chien-Chung Huang[1†], **Zoya Svitkina**[2]

[1]Max-Planck-Institut für Informatik, Saarbrücken, Germany

[2]Department of Computing Science, University of Alberta, Edmonton, Canada

ABSTRACT. We introduce and study the *donation center location* problem, which has an additional application in network testing and may also be of independent interest as a general graph-theoretic problem. Given a set of agents and a set of centers, where agents have preferences over centers and centers have capacities, the goal is to open a subset of centers and to assign a maximum-sized subset of agents to their most-preferred open centers, while respecting the capacity constraints.

We prove that in general, the problem is hard to approximate within $n^{1/2-\epsilon}$ for any $\epsilon > 0$. In view of this, we investigate two special cases. In one, every agent has a bounded number of centers on her preference list, and in the other, all preferences are induced by a line-metric. We present constant-factor approximation algorithms for the former and exact polynomial-time algorithms for the latter. Of particular interest among our techniques are an analysis of the greedy algorithm for a variant of the maximum coverage problem called *frugal coverage*, the use of maximum matching subroutine with subsequent modification, analyzed using a counting argument, and a reduction to the independent set problem on *terminal intersection graphs*, which we show to be a subclass of trapezoid graphs.

1 Introduction

Suppose that a charitable organization wishes to open a number of locations where people can make donations (e.g. donate blood). There is no cost for opening these centers, but they do have capacities for the number of donors that they can accommodate. We model the potential donors, whom we call agents, as each having a list of locations where she would be willing to go to make a donation. Once some of the centers are opened, each agent goes to the most convenient open one from her list. However, if that center is full (i.e. has exceeded its capacity), then the agent gives up and decides not to donate at all. Our goal is to choose a set of centers to open to maximize the number of collected donations.

Formally, we define the DONATION CENTER LOCATION (DCL) problem as follows. Let $G = (A \cup L, E)$ be a directed bipartite graph, with edges directed from the set A of agents to the set L of donation centers. Every center $l \in L$ has a capacity $c_l \in \mathbb{Z}^+$, and every vertex $a \in A$ has a strictly-ordered preference ranking of its neighbors in L (or, equivalently, of its outgoing edges). These preferences model either distance or some other measure of convenience for the agents over the locations. We have to choose a subset $L' \subseteq L$ of centers to open, and to assign a subset $A' \subseteq A$ of agents to centers in L', in such a way that the number of agents assigned to any center $l \in L'$ is at most c_l, and each $a \in A'$ is assigned to its highest-ranked neighbor in L'. The goal is to maximize $|A'|$, the number of

*This work was supported in part by NSF grant CCF-0728869. Most of the work was done while the authors were at Dartmouth College.

†Research supported by an Alexander von Humboldt fellowship.

© Huang, Svitkina; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 227–238
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2321

assigned agents. Note that once a set L' of locations is selected, it is very easy to find an optimal assignment of agents: if some open center $l \in L'$ is the first choice of more than c_l agents, then an arbitrary subset of c_l of them can be assigned to it (and others will remain unassigned). Thus, our problem statement requiring an explicit assignment from A' to L' is equivalent to one motivated above, which just asks to find L' and assumes that each center $l \in L$ will accommodate an unspecified subset of at most c_l agents who prefer it.

We use notation $l \succ_a l'$ to indicate that agent $a \in A$ prefers center l to center l', where both (a, l) and (a, l') are edges in E. If a solution assigns agent $a \in A'$ to center $l \in L'$, then we write $\mu(a) = l$. We also define $\mu^{-1}(l) = \{a \in A' : \mu(a) = l\}$ to be the set of agents assigned to l. If an assignment μ satisfies the constraints of the DCL problem, then we call it *valid*. Formally, a valid assignment $\mu : A' \to L'$ meets the following conditions:

1. if $a \in A'$, then $(a, \mu(a)) \in E$
2. if $a \in A'$, then there is no $l \in L'$ such that $(a, l) \in E$ and $l \succ_a \mu(a)$
3. if $l \in L'$, then $|\mu^{-1}(l)| \leq c_l$

One special case of DCL that we focus on is the *unit-capacity* case, where $c_l = 1$ for all centers. In that case the assignment $\mu : A' \to L'$ is a matching. This special case establishes a connection between DCL and various matching problems under preferences that have been extensively studied in both computer science and economics literature. It also has an application in network testing [6, 19], which is as follows. In a wireless network consisting of transmitters and receivers, the transmitter nodes have to be tested. For one round of testing, a maximum-cardinality set A' of transmitters has to be matched to a set L' of receivers. The power setting of a transmitter is adjusted based on the distance to its intended receiver, and the signal reaches this receiver as well as all receivers that are closer to the transmitter than it is. The preference lists of transmitters over receivers are complete and are induced by the distance, with closer ones ranked higher. Then Condition 2 for a valid matching requires that a matched receiver not simultaneously be in the range of two active (matched) transmitters, thus preventing interference.

1.1 Related work

Matching entities with preferences is an extensively studied topic in the literature. The most representative is the stable matching (also known as stable marriage) problem [9], where both sides have preferences and a matching is considered stable if there are no two elements that both prefer each other to their assigned matches. Recently, the matching problems in the context of one-sided preferences have also been studied. Examples include popular matching [2, 13, 15], rank-maximal matching [12, 16], and pareto-optimal matching [1]. A major distinction in our model is that an unopened center does not influence the feasibility of a given solution, even if some agent prefers this center to his assigned open one. However, for instance, in the stable marriage problem, a bachelor and a married woman can disturb the stability of a matching.

Our model also resembles the well-studied facility location problems [5, 18] and their capacitated versions [17, 21]. However, in most facility location problems, the algorithm is allowed to assign clients to arbitrary opened facilities, whereas in our case, each client has to go to its nearest one. Also, DCL is a maximization problem and does not have a requirement

of assigning all clients, whereas facility location is usually formulated as assigning all clients while minimizing cost. Thus, there is no direct way to apply known algorithmic techniques for it to our setting.

Network testing is a possible application of DCL. Maximum-cardinality matching between transmitters and receivers has been studied in [6, 19], where the interference between transmitters is modeled in a more crude way: just the presence of an edge between a transmitter and a receiver in the connection graph represents a possible source of interference. In contrast, we use the notion of preferences (relative physical distance) to give a more fine-grained model of interference.

1.2 Our results and techniques

We study the general DCL problem as well as several special cases of it. Most of the versions that we consider here are NP-complete, in which case we study their approximability, either by finding good approximation algorithms, or by proving hardness of approximation results. However, some of the special cases are solvable in polynomial time, and for these we present exact algorithms. Our results are summarized in Table 1. Some of the proofs are omitted here and appear in the full version of this paper [11].

	unit capacity	general capacity
complete preferences	$n^{1/2-\varepsilon}$-hard to approximate (§2)	
bounded degree	APX-hard (§2)	
out-degree bound d	$1/d$ (§3.1)	$1/2d$ (§3.3), $1/\phi d$ ([11])
out-degree bound $d = 2$	$\frac{e}{e+1}$ (§3.2)	$1/2$ ([11])
line metric	polynomial-time (§4)	

Table 1: Summary of results

For the general case of DCL, we show that it is hard to approximate to a factor of $n^{1/2-\varepsilon}$, for any $\varepsilon > 0$. This result also holds for the special case of complete preferences (when G is a complete bipartite graph). In view of this, we focus on two types of special cases, one of which is the bounded degree case. Here the degree in G of any vertex $a \in A$ is upper-bounded by a constant d. We show that the problem remains APX-hard, even in the unit-capacity case with degree bound of 2. For any degree bound d, we give a $1/d$-approximation algorithm for the unit-capacity case. For the special case of degree bound $d = 2$, we improve this ratio to $\frac{e}{e+1} \approx 0.731$. To do this, we introduce a new variant of the maximum coverage problem, called frugal coverage, and analyze the performance of the greedy algorithm on it. For the problem with general capacities and degree bound d, we present a $1/2d$ approximation algorithm that makes use of a maximum matching subroutine. In [11], we improve the analysis to give a $1/\phi d$ approximation, for $\phi \approx 1.618$, and also improve the ratio to $1/2$ for the special case of $d = 2$.

The second special case that we consider is one in which the preferences are induced by a line metric. In particular, all nodes of $A \cup L$ are located on a single line, and each agent ranks the centers in the order of proximity. For this case, we give an exact linear-time

algorithm for the unit-capacity setting. Then we extend it to obtain an exact polynomial-time algorithm for general capacities. To design these algorithms, we reduce the problems to maximum independent set on a special kind of graphs that we call terminal intersection graphs. We then show that these graphs form a subclass of trapezoid graphs [3, 4], for which there are known polynomial-time algorithms that solve maximum independent set [8, 14].

2 Hardness results

We prove that DCL is hard to approximate to a factor of $n^{1/2-\epsilon}$, and that it remains APX-hard even in the bounded-degree case. The proof of the first result uses a non-trivial reduction from the maximum independent set problem, which increases the size of the instance while approximately preserving the value of the optimal solution. The proof of APX-hardness uses a different reduction from independent set on 3-regular graphs. Both proofs appear in [11].

THEOREM 1. *DCL problem is hard to approximate within $O\left(|A \cup L|^{1/2-\epsilon}\right)$ for any $\epsilon > 0$, unless NP=ZPP. This is true even in the case of unit capacities and complete preferences.*

THEOREM 2. *DCL problem is APX-hard, even with unit capacities, out-degree bound of 2 on A and in-degree bound of 3 on L.*

We also show that the special case of DCL in which preferences are induced by a metric is no easier than the general problem with complete preferences[‡]. In fact, arbitrary complete preferences of A over L can be represented by embedding all points of $A \cup L$ into a metric space. To do this, we use the ℓ_∞ metric over an $|L|$-dimensional space. Each element $l_i \in L$ (for $i = 1$ to $|L|$) is mapped to a location x^i, with coordinates $x^i_j = 0$ for $j \neq i$, and $x^i_i = 1$. Each element $a \in A$ is mapped to a location x^a, with $x^a_i = \frac{1}{2} - \frac{rank(a,l_i)}{2|L|}$. Here $rank(a, l_i)$ is the rank that agent a assigns to center l_i, ranging from 1 for the most-preferred center up to $|L|$. With this embedding, the ℓ_∞ distance from a to l_i, for each $1 \leq i \leq |L|$, becomes $\frac{1}{2} + \frac{rank(a,l_i)}{2|L|}$ (with $|x^i_i - x^a_i|$ being the largest coordinate difference). This ensures that for each $a \in A$, the ordering of elements of L by distance is the same as it is by preference.

3 Algorithms for bounded-degree DCL

In view of the hardness results for the general problem, in this section we focus on special cases in which the lengths of the agents' preference lists are bounded by a constant d.

3.1 A linear-time $1/d$ approximation for the unit-capacity case

We partition L into d subsets $L_1, L_2, \cdots, L_d$. A center l is in group L_k if, among all edges from agents to l, the highest rank of these edges is k. We now consider each set L_k separately, and let μ_k denote an arbitrary matching in which each center in L_k is matched to an agent that ranks it k. Note that at least one such agent for each center must exist by definition of L_k, and no agent will be matched twice as it can't have the same rank for two different centers. We

[‡]This reduction was suggested to us by Uri Feige

claim that μ_k is a valid matching in the original problem. If not, suppose that both (a, l) and (a', l') are part of μ_k and $l' \succ_a l$. Then a ranks l' higher than k, contradicting the assumption that $l' \in L_k$. We output the largest μ_k, which satisfies $|\mu_k| = |L_k| \geq \frac{1}{d}|L| \geq \frac{1}{d}\text{OPT}$, and note that the algorithm can be implemented in linear time.

3.2 A $e/(e+1)$ approximation for unit-capacity DCL with degree bound $d = 2$

Here we consider the unit-capacity case in which the out-degree of each agent is at most 2. Our algorithm in the preceding section gives a ratio of $1/2$ in this case, but here we improve it to $\frac{e}{e+1} \geq 0.731$. We first give an approximation-preserving reduction to a problem that we call frugal coverage, and then give a $\frac{e}{e+1}$-factor approximation for frugal coverage. The input to this problem is the same as for set cover, but the objective function is different. We wish to maximize the number of elements covered by the chosen sets *plus* the number of sets that are not chosen.

DEFINITION 3. *In the* frugal coverage problem*, the input is a universe U of elements and a collection $\mathcal{C}$ of subsets of U. The goal is to select a subset $\mathcal{C}' \subseteq \mathcal{C}$ that maximizes $|\bigcup_{S \in \mathcal{C}'} S| + |\mathcal{C} \setminus \mathcal{C}'|$.*

LEMMA 4. *If there is an α-approximation for the frugal coverage problem, then there is an α-approximation for unit-capacity DCL with degree bound 2.*

PROOF. To obtain a reduction, we first do one step of pre-processing on the given DCL instance. If any center $l \in L$ has incoming edges of both rank 1 and rank 2, then we remove all its incoming edges of rank 2. We claim that the value of the optimum is maintained, because any feasible solution that uses the edges that were removed can be transformed into one of the same size which does not use these edges. Suppose $a_1 \in A$ ranks l first, and $a_2 \in A$ ranks l second. Now, if a_2 is matched to l in the optimal solution, then a_1 is unmatched, as otherwise it would prefer l to its match. So we can replace the matched pair (a_2, l) with (a_1, l), preserving the size and feasibility of the solution.

Now we give a reduction from the DCL instance to frugal coverage, assuming that no node $l \in L$ has both rank-1 and rank-2 incoming edges. We also assume without loss of generality that there are no nodes in $A \cup L$ with degree zero. We partition the set L into subsets X and Y, where X contains all the nodes with incoming rank-1 edges, and Y contains all the nodes with incoming rank-2 edges. By our assumptions, these sets are disjoint and cover L. For each $l_1 \in X$, we create a set $S(l_1) \in \mathcal{C}$. For each $l_2 \in Y$, we create an element $e(l_2) \in U$. For each agent $a \in A$ whose preference list is of length two, with $l_1 \succ_a l_2$, we include the element $e(l_2)$ into the set $S(l_1)$.

Given a valid matching μ for the DCL instance, we create a solution to the derived frugal coverage instance with value at least $|\mu|$. In particular, this solution $\mathcal{C}'$ consists of all sets $S(l)$ that correspond to unmatched nodes $l \in X$. If we let $|\mu| = x + y$, where x is the number of matches on rank-1 edges, and y is the number of matches on rank-2 edges, then $|\mathcal{C} \setminus \mathcal{C}'| = x$ and $|\bigcup_{S \in \mathcal{C}'} S| \geq y$. The equality follows because $\mathcal{C}$ corresponds to all nodes of X, and $\mathcal{C}'$ corresponds to the unmatched ones. For the inequality, suppose that a rank-2 match (a, l_2) is part of μ, and consider the center l_1 such that $l_1 \succ_a l_2$. Then l_1 is unmatched, as otherwise the feasibility of μ is violated, and therefore $S(l_1) \in \mathcal{C}'$. Also, by construction,

$e(l_2) \in S(l_1)$. So we have that for each $l_2 \in Y$ matched on rank-2 edge, there exists $l_1 \in X$ such that $e(l_2) \in S(l_1) \in \mathcal{C}'$, and therefore $|\bigcup_{S \in \mathcal{C}'} S| \geq y$.

Conversely, given a solution $\mathcal{C}'$ to the constructed frugal coverage instance, a feasible solution μ to the original DCL instance, with at least as big a value, can be produced. For each $l \in X$ whose corresponding set is not chosen ($S(l) \notin \mathcal{C}'$), choose an arbitrary node $a \in A$ such that (a, l) is an edge, and include (a, l) in μ. For each $l_2 \in Y$ whose corresponding element is covered by the frugal coverage solution ($e(l_2) \in \bigcup_{S \in \mathcal{C}'} S$), find a node $l_1 \in X$ whose corresponding set covers $e(l_2)$ (i.e. with $e(l_2) \in S(l_1)$ and $S(l_1) \in \mathcal{C}'$), choose a node $a \in A$ such that $l_1 \succ_a l_2$ (which enabled us to include $e(l_2)$ in $S(l_1)$ when constructing the instance), and match a to l_2. To ensure that no $a \in A$ is matched twice, and that μ is a valid matching, suppose that there is a node $a \in A$ with $\mu(a) = l_2$, $l_1 \succ_a l_2$, and l_1 also matched. But this is a contradiction because we only matched (a, l_2) if $S(l_1) \in \mathcal{C}'$, and only matched l_1 if $S(l_1) \notin \mathcal{C}'$. Since for each covered element and for each unchosen set we have included one pair into the matching, the size of μ is at least the objective function value of the frugal coverage solution.

To obtain an α-approximation for DCL, perform the above construction, producing an instance of frugal coverage whose optimum is at least $|\mu^*|$, where μ^* is an optimal valid matching. Find an α-approximation to frugal coverage of value at least $\alpha \cdot |\mu^*|$, and transform it back to a DCL solution with at least as big a value. ∎

Algorithm for frugal coverage

We analyze the performance of the greedy algorithm for the frugal coverage problem. This is the same algorithm as is used for set cover [20]: while there is a set that covers at least one new element, choose the one that covers maximum number of new elements and include it in the solution. We note that our approximation guarantee for frugal coverage is better than the best possible factor of $\frac{e-1}{e} \approx 0.632$ for the maximum coverage problem [7].

LEMMA 5. *The greedy algorithm is a $\frac{e}{e+1}$ approximation for the frugal coverage problem.*

PROOF. Let $m = |\mathcal{C}|$ be the number of sets in the instance, $n = |U|$ be the total number of elements, and $n' = |\bigcup_{S \in \mathcal{C}} S|$ be the number of elements that are contained in at least one set. Suppose that the greedy algorithm completes after taking l sets. Then its objective function value is equal to $\text{ALG} = n' + (m - l)$. Let $\mathcal{C}_k$ denote the intermediate solution obtained by the greedy algorithm after including $0 \leq k \leq l$ sets. We observe that the solution $\mathcal{C}_l$ is at least as good as any $\mathcal{C}_k$, because with each step of the algorithm, the number of unused sets $|\mathcal{C} \setminus \mathcal{C}_k|$ decreases by one, and the number of covered elements $|\bigcup_{S \in \mathcal{C}_k} S|$ increases by at least one. By the same reasoning, we know that there is an optimal solution $\mathcal{C}^* \subseteq \mathcal{C}$ to the frugal coverage problem that covers all elements that are contained in at least one set. Let $k^* = |\mathcal{C}^*|$ be the number of sets chosen by this optimal solution. Then its objective function value is $\text{OPT} = n' + (m - k^*)$.

We first give an easy proof to show that the greedy algorithm is at least a $\frac{e-1}{e}$ approximation, and then improve the guarantee. Consider the intermediate greedy solution $\mathcal{C}_{k^*}$ (note that $l \geq k^*$, as k^* is the minimum number of sets that can cover all n' elements). By

the guarantee of the greedy algorithm for the maximum coverage problem [7], $\mathcal{C}_{k^*}$ covers at least $\frac{e-1}{e} \cdot n'$ elements. So the value of the solution is $\text{ALG} \geq \frac{e-1}{e} \cdot n' + (m - k^*) \geq \frac{e-1}{e} \cdot \text{OPT}$.

To improve the guarantee, we observe that $l \leq m$, and therefore $\text{ALG} \geq n'$. Combining with the previous result, we get $\text{ALG} \geq \max(\frac{e-1}{e}n' + m - k^*, n')$. We now consider two cases. The first case is that $n' \geq \frac{e-1}{e}n' + (m - k^*)$, and therefore $n' \geq e(m - k^*)$. Then

$$\text{ALG} \geq n' = \frac{en'}{e+1} + \frac{n'}{e+1} \geq \frac{en'}{e+1} + \frac{e(m-k^*)}{e+1} = \frac{e}{e+1} \cdot \text{OPT}.$$

In the second case, $n' < \frac{e-1}{e}n' + (m - k^*)$, and therefore $m - k^* > n'/e$. Then

$$\begin{aligned} \text{ALG} &\geq \frac{e-1}{e}n' + m - k^* = \frac{e-1}{e}n' + \frac{m-k^*}{e+1} + \frac{e(m-k^*)}{e+1} \\ &> \frac{e-1}{e}n' + \frac{n'}{e(e+1)} + \frac{e(m-k^*)}{e+1} = \frac{en'}{e+1} + \frac{e(m-k^*)}{e+1} = \frac{e}{e+1} \cdot \text{OPT}, \end{aligned}$$

so in either case we get the desired approximation. ∎

Combining Lemmas 4 and 5, we arrive at the following result.

THEOREM 6. *There is an $\frac{e}{e+1} \geq 0.731$ approximation for unit-capacity DCL with degree bound 2.*

We make two remarks before we close this section. First, by the APX-hardness result of Theorem 2, the reduction in Lemma 4, and the constant approximation in Lemma 5, it follows that the frugal coverage problem is APX-complete. Second, the following special case of DCL is solvable in polynomial time: every agent in A has out-degree at most 2 and every center in L has at most two incoming rank-1 edges. To see this, observe that in this setting, under the reduction of Lemma 4, we derive a frugal coverage instance with every set in $\mathcal{C}$ of size at most 2. By the same reasoning as in Lemma 5, there is an optimal solution that covers all elements in $\bigcup_{S \in \mathcal{C}} S$. Thus, the problem is equivalent to finding an optimal set cover where every set is of size at most 2 and can be easily shown to be equivalent to the edge cover problem, which is known to be in P [10].

3.3 A $1/2d$ approximation for DCL with general capacities

As the hardness results of Section 2 still apply to the problem with general capacities, we consider the special case in which each agent has at most d outgoing edges in G. Our algorithm consists of the following steps.

1. Using flow techniques, find a maximum-size assignment μ (not necessarily valid) between A and L on the edges of G, where each agent is assigned to at most one center, and each center l gets at most c_l agents. This assignment disregards the preferences of the agents, and serves as an upper bound on the optimum.
2. Create a directed graph on the set of centers $H = (L, F)$ based on μ. An arc $(l, l') \in F$ is drawn if there is some agent that is assigned to center l by μ, but prefers l' to l. If H contains a directed cycle, then update μ by transferring one agent along each arc of this cycle so as to improve the transferred agents' assignments. Update H, and repeat until H is acyclic. Note that this process terminates in polynomial time.

3. Discard all unassigned agents and unused centers from the graph G to produce a subgraph G'. Furthermore, remove from G' edges from each agent a to centers which a ranks lower than $\mu(a)$. Also remove unused centers from H.
4. Define a topological order over H so that all directed arcs of H go "from left to right".
5. Consider each center node l in H, scanning from left to right, and delete it from G' if the degree of l in G' is greater than $\xi \cdot c_l$, where $\xi > 1$ is a parameter to be optimized later. To delete l, update G' by removing l and the agents assigned to it by μ, along with the incident edges.
6. Return the set U of centers that are still part of G'.

Note that the final solution is not μ, as μ is not necessarily a valid assignment. Instead, it is the set $U \subseteq L$ of open centers, with the best valid assignment of agents to them, which is easy to find as mentioned in the introduction. The possible loss in value of this solution compared to the size of μ is analyzed below.

THEOREM 7. *The above algorithm is a* $1/2d$*-approximation for DCL with degree bound* d*.*

PROOF. As mentioned, the number of agents assigned by μ serves as an upper bound on the optimum. Moreover, step 2 does not alter the size of μ. There are two ways in which the algorithm can lose agents that are matched in step 1. The first is the deletion of centers in step 5, as agents assigned to them may not have any edges to the remaining centers, and thus be lost to the solution. The second reason is that even from centers in U, the contribution to the objective function may be smaller than the number of agents assigned to them by μ. This is because the agents 'switch' from their assigned centers to their best centers in U. As a simple example, consider an instance with two agents and two centers, where both agents prefer l_1 to l_2, and $c_{l_1} = c_{l_2} = 1$. Then μ assigns one agent to each center, and has size two. But opening both centers produces a solution with objective function of 1.

We let $|\mu| = n_u + n_r$, where n_u is the number of agents that are assigned by μ (after step 2) to centers in U, and n_r is the number of agents assigned by μ to other centers, i.e. ones removed by the algorithm in step 5. We first lower-bound n_u, and then lower-bound the size of the solution in terms of n_u. Observe that for every center l deleted in step 5, its degree in G' (at the time of deletion) is greater than ξc_l. At most c_l of these incoming edges come from agents assigned to it by μ, and the rest come from agents that are assigned elsewhere by μ, but prefer l to their current centers (this is because in step 3, we removed edges from each agent a to centers that rank lower than $\mu(a)$). Let us say that one such agent, a, is assigned to a center l' but prefers l to l'. In this case the graph H would contain an edge from l' to l, which means that l' occurs before l in the topological ordering. Furthermore, when l' was considered by step 5 of the algorithm (which happened before l was considered), it was not deleted, since otherwise we would have also deleted all its agents, including a. So any such center l' must be part of U. Now, each agent has at most $d-1$ edges in G' to centers other than its assigned one, so the number of agents assigned to U by μ that contribute the extra $\xi c_l - c_l$ edges to centers $l \notin U$ can be bounded as

$$n_u \;\geq\; \frac{\sum_{l \notin U}(\xi c_l - c_l)}{d-1} \;=\; \frac{\xi - 1}{d-1} \cdot \sum_{l \notin U} c_l \;\geq\; \frac{\xi - 1}{d-1} \cdot n_r. \tag{1}$$

The value of the final solution that assigns agents from A to centers in U in an optimal way can only be higher than if we restrict the assignment to only use agents from some subset $\tilde{A} \subseteq A$. In particular, let $\tilde{A}$ be the set of n_u agents that are assigned to U by μ. For a center $l \in U$, consider the set of agents $\tilde{A}_l \subseteq \tilde{A}$ that rank l highest among centers in U. For any such agent $a \in \tilde{A}_l$, there is an edge in G' from a to l. But since the degree of l in G' is at most ξc_l, the size of $\tilde{A}_l$ is at most ξc_l. Thus, at least a $1/\xi$ fraction of agents in $\tilde{A}_l$ can be assigned to l by a valid assignment. As the sets $\tilde{A}_l$ partition $\tilde{A}$, overall $\text{ALG} \geq n_u/\xi$. Using (1), the approximation ratio becomes

$$\frac{\text{ALG}}{\text{OPT}} \geq \frac{n_u/\xi}{|\mu|} = \frac{n_u/\xi}{n_u + n_r} \geq \frac{n_u/\xi}{n_u + n_u\frac{d-1}{\xi-1}} = \frac{1/\xi}{1 + \frac{d-1}{\xi-1}} \equiv f_d(\xi).$$

Calculus shows that $f_d(\xi)$ is maximized at $\xi = 1 + \sqrt{d-1}$, and the approximation guarantee becomes $1/(d + 2\sqrt{d-1}) \geq 1/2d$. In fact, for large d, it approaches $1/d$. ∎

With a more detailed analysis (see [11]), the above algorithm can be shown to deliver a $1/\phi d$ approximation, for $\phi \approx 1.618$. In addition, for the special case of $d = 2$, another algorithm with an improved guarantee of $1/2$ appears in [11].

4 DCL on a line

In this section we show how to find optimal solutions to unit-capacity and general DCL, in the case that preferences are complete and defined according to distances on a line (with closer points ranked higher). Our algorithms work through a reduction to the independent set problem on a special class of graphs, which we call terminal intersection graphs. As we show, terminal intersection graphs are a subclass of trapezoid graphs [3, 4], for which polynomial-time algorithms for independent set are known. We assume that no two nodes are co-located on the line, and no two distances are equal. Distance between two points on the line is denoted by $d(x, y)$.

DEFINITION 8. *A graph $H = (W, F)$ is a* terminal intersection graph *if there exists a set of intervals $\mathcal{I} = \{I_w = [a_w, b_w] : w \in W\}$ on a line, each with a terminal $c_w \in I_w$, such that there is an edge $(w, w') \in F$ if and only if either $c_w \in I_{w'}$ or $c_{w'} \in I_w$.*

DEFINITION 9. *A graph $H = (W, F)$ is a* trapezoid graph *if there exist two parallel lines such that each vertex $w \in W$ corresponds to a trapezoid T_w defined by the convex hull of two points on the top line and two points on the bottom line, and $(w, w') \in F$ if and only if T_w and $T_{w'}$ intersect.*

LEMMA 10. *Every terminal intersection graph is a trapezoid graph, and the trapezoid model can be found in linear time when a terminal intersection model is given.*

Proof of Lemma 10 appears in [11]. We now give the main results of the section.

THEOREM 11. *Unit-capacity DCL on a line can be solved to optimality in linear time.*

PROOF. We reduce to the independent set problem on terminal intersection graphs, which can be solved in linear time using Lemma 10 and the algorithm for independent set on trapezoid graphs ([14] and the fact that trapezoid graphs are a subclass of co-comparability graphs [8]). Our reduction is also linear-time, so this gives an overall $O(n)$-time algorithm.

We start with a useful observation about the structure of an optimal valid matching on a line. We claim that for any instance, there exists an optimal solution in which every matched center is matched either to its closest agent to the right of it on the line, or to its closest agent to the left. To see this, consider a center l, its closest agent to the right a_1, and an agent a_2 farther to the right. Suppose that l and a_2 are matched. Then there is no other matched center between l and a_2, and there is no matched center within distance $d(l, a_2)$ to the right of a_2 (otherwise a_2 would prefer those centers to l). This implies that a_1 is not matched, as otherwise it would prefer l to its match. So if, instead of (a_2, l), we match (a_1, l), this would also be a feasible solution, since l would be the closest matched center to a_1.

Given the above observation, it suffices to only consider two possible matches for each center l: (a_l, l) and (a_r, l), where a_l and a_r are the nearest agents to the left and to the right, respectively. So there are at most $2|L|$ possible matches to consider, and we have to choose the maximum subset of them which matches each node at most once and fulfills the condition of a valid matching. We do this by creating a terminal intersection graph $H = (W, F)$ in which the nodes correspond to possible matches, and edges correspond to pairs of matches that interfere with each other. Then the maximum independent set in H corresponds to the maximum valid matching in our instance.

For each center $l \in L$ and its potential match a_r on the right we create a vertex w in H specified by the interval $I_w = [l, l + 2 \cdot d(l, a_r)]$ and the terminal $c_w = l$ (we identify the nodes in $A \cup L$ with their coordinates on the line, and hence treat them as numbers). Similarly, for l and a_l we create a node with interval $[l - 2 \cdot d(a_l, l), l]$ and terminal l. This is an interval that is twice as long as the distance between the agent and the center, centered at the agent, and with a terminal at the endpoint corresponding to the center.

We now verify that two vertices have an edge in H if and only if both of their corresponding pairs cannot be included in the matching. Suppose that H contains an edge (w, w'). Then either $c_w \in I_{w'}$ or $c_{w'} \in I_w$, so assume without loss of generality that $c_w \in I_{w'}$. Let (a_w, l_w) be the potential match corresponding to the vertex w, and $(a_{w'}, l_{w'})$ be one corresponding to w'. Then, by geometry, $d(a_{w'}, l_w) \leq d(a_{w'}, l_{w'})$. So either $l_w = l_{w'}$, or $a_{w'}$ is closer to l_w than to $l_{w'}$, and both pairs cannot be matched simultaneously. Conversely, suppose that two pairs (a_w, l_w) and $(a_{w'}, l_{w'})$ cannot both participate in the matching. This could be because $l_w = l_{w'}$, or $a_w = a_{w'}$, or because they violate the preference condition. In the first case, immediately $c_w \in I_{w'}$, so the edge (w, w') is in H; in the second case, assume l_w is closer than $l_{w'}$ to a_w, but then $c_w = l_w \in I_{w'}$; in the last case, assume that $d(a_{w'}, l_w) < d(a_{w'}, l_{w'})$. But since $I_{w'}$ is an interval of length $2d(a_{w'}, l_{w'})$ centered at $a_{w'}$, it includes $c_w = l_w$, so again (w, w') is an edge in H. ∎

THEOREM 12. *DCL on a line can be solved to optimality in polynomial time.*

We sketch the ideas for extending the unit-capacity algorithm to general capacities. The running time is no longer linear, but it remains polynomial. We again construct a terminal

intersection graph H, but this time we reduce to the maximum *weighted* IS problem on it, which is still solvable in polynomial time for trapezoid graphs [8]. Consider a center u with $c_u \leq n$. Any feasible solution assigns k_l agents to it that are to the left of u on the line, and k_r agents that are on the right, for some k_l and k_r with $k_l + k_r \leq c_u$. Analogously to the proof of Theorem 11, we can assume that these agents are the closest k_l ones on the left, and the closest k_r ones on the right. So for each center u, and for each possible k_l and k_r, we create an interval I_u with a terminal located at u. To specify the endpoints of I_u, we let a_l be the k_l-th farthest agent to the left of u, and a_r be the k_r-th agent to the right of u. Then $I_u = [u - 2 \cdot d(a_l, u),\ u + 2 \cdot d(u, a_r)]$. Finally, we set the weight of the corresponding vertex in H to $k_l + k_r$. As before, it can be verified that a set of vertices in H is independent if and only if the corresponding assignment in the original problem is valid. Moreover, the weight of this set is equal to the number of agents assigned in the corresponding solution.

5 Conclusion

We have introduced a new combinatorial problem with a number of applications and made significant progress toward characterizing its complexity and approximability. In doing so, we used a variety of techniques, including a non-trivial hardness proof, an analysis of the greedy algorithm for a new variant of set cover, a counting argument for establishing the approximation ratio in the general capacity case, and a reduction to geometric graphs. Our definitions of the frugal coverage problem and the terminal intersection graphs, as well as our algorithms, may be of more general interest and find applications in other contexts.

One extension of the DCL problem is the non-bipartite version, where G is a general directed graph, and all vertices have preferences over their outgoing edges. A solution consists of sets A', L', and a valid assignment from A' to L' as before, but now A' and L' can be arbitrary disjoint subsets of vertices of G. In the network testing application, the bipartite problem corresponds to the case that transmitters and receivers are two different types of devices, whereas the non-bipartite version models a more general setting in which some or all of devices are capable of performing either function. Our hardness of approximation results extend to the non-bipartite version, and in the unit-capacity setting it admits a $1/d$ approximation for the bounded-degree case as well as a polynomial-time exact algorithm on a line metric. Full description of these results can be found in [11].

Our results highlight a number of questions and related problems that remain open. For example, the weighted version of the problem is a possible extension. For network testing, a natural problem is to minimize the number of rounds required to test all transmitters. Also, an extension of our algorithm for DCL on a line to the case of Euclidean plane may be relevant for this setting. Given our hardness results for DCL, it may be worthwhile to consider alternative formulations, such as minimizing the number of unassigned nodes, instead of maximizing the number of assigned ones. Finally, we leave for future work the investigation of similar problems with two-sided preferences.

Acknowledgements

We thank Peter Winkler and Kurt Mehlhorn for useful discussions, and Uri Feige for suggesting the reduction to the metric case.

References

[1] D. Abraham, K. Cechlárová, D. Manlove, and K. Mehlhorn. Pareto optimality in house allocation problems. *Proc. of the 15th ISAAC*, pages 3–15, 2004.

[2] D. Abraham, R. Irving, K. Mehlhorn, and K. Telikepalli. Popular matchings. *Proc. 16th ACM Symp. on Discrete Algorithms*, pages 424–432, 2005.

[3] D.G. Corneil and P.A. Kamula. Extensions of permutation and interval graphs. *Proc. 18th Southeast. Conf. on Combinatorics, Graph Theory, and Computing*, 1987.

[4] I. Dagan, M. C. Golumbic, and R. Y. Pinter. Trapezoid graphs and their coloring. *Discrete Appl. Math.*, 21(1):35–46, 1988.

[5] Z. Drezner. *Facility Location: A Survey of Applications and Methods*. Springer, 1995.

[6] S. Even, O. Goldreich, S. Moran, and P. Tong. On the NP-completeness of certain network testing problems. *Networks*, 14, 1984.

[7] U. Feige. Threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[8] S. Felsner, R. Muller, and L. Wernisch. Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Appl. Math.*, 74(1):13–32, 1997.

[9] D. Gale and L. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[11] C.-C. Huang and Z. Svitkina. Donation center location problem. Submitted for publication. Manuscript available from the authors' websites.

[12] R. Irving, T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch. Rank-maximal matchings. *Proc. 15th ACM Symp. on Discrete Algorithms*, pages 68–75, 2004.

[13] D. Manlove and C. Sng. Popular matchings in the capacitated house allocation problem. *Proc. 14th European Symposium on Algorithms*, pages 492–503, 2006.

[14] R. McConnell and J. Spinrad. Modular decomposition and transitive orientation. *Discrete Math.*, 201(1-3):189–241, 1999.

[15] J. Mestre. Weighted popular matchings. *Proc. of the 33rd ICALP*, pages 715–726, 2006.

[16] D. Michael. Reducing rank-maximal to maximum weight matching. *Theoretical Computer Science*, 389(1-2):125–132, 2007.

[17] M. Pal, E. Tardos, and T. Wexler. Facility location with nonuniform hard capacities. *Proc. 42nd IEEE Symp. on Foundations of Computer Science*, page 329, 2001.

[18] D. Shmoys. Approximation algorithms for facility location problems. In K. Jansen and S. Khuller, editors, *Approximation Algorithms for Combinatorial Optimization*, pages 27–33. Springer, Berlin, 2000.

[19] L. Stockmeyer and V. Vazirani. NP-completeness of some generalizations of the maximum matching problem. *Information Processing Letters*, 15(1), 1982.

[20] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., 2001.

[21] J. Zhang, B. Chen, and Y. Ye. A multiexchange local search algorithm for the capacitated facility location problem. *Math. Oper. Res.*, 30(2):389–403, 2005.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Non-Local Box Complexity and Secure Function Evaluation

M. Kaplan*, I. Kerenidis*, S. Laplante*, and J. Roland†

ABSTRACT. A non-local box is an abstract device into which Alice and Bob input bits x and y respectively and receive outputs a and b respectively, where a, b are uniformly distributed and $a \oplus b = x \wedge y$. Such boxes have been central to the study of quantum or generalized non-locality as well as the simulation of non-signaling distributions. In this paper, we start by studying how many non-local boxes Alice and Bob need in order to compute a Boolean function f. We provide tight upper and lower bounds in terms of the communication complexity of the function both in the deterministic and randomized case. We show that non-local box complexity has interesting applications to classical cryptography, in particular to secure function evaluation, and study the question posed by Beimel and Malkin [4] of how many Oblivious Transfer calls Alice and Bob need in order to securely compute a function f. We show that this question is related to the non-local box complexity of the function and conclude by greatly improving their bounds. Finally, another consequence of our results is that traceless two-outcome measurements on maximally entangled states can be simulated with 3 non-local boxes, while no finite bound was previously known.

1 Introduction

Communication complexity. Communication complexity is a central model of computation, which was first defined by Yao in 1979 [35] and has since found numerous applications. In this model Alice and Bob receive inputs x and y respectively and are allowed to communicate in order to compute a function $f(x, y)$. The goal is to find the minimum amount of communication needed for this task. In different variants of the model, we allow Alice and Bob to err with some probability, and to share common resources in an attempt to enable them to solve their task more efficiently.

One such resource is shared randomness. When Alice and Bob are not allowed any errors, shared randomness does not reduce the communication complexity. On the other hand, when they are allowed to err, a common random string can reduce the amount of communication needed. However, Newman's result tells us that shared randomness can be replaced by private randomness at an additional cost logarithmic in the input size.

Another very powerful shared resource is entanglement. Using teleportation, Alice and Bob can transmit quantum messages by using their entanglement and only classical communication. This model has been proven to be very powerful, in some cases exponentially more efficient than the classical one. Another way to understand the power of entanglement is by looking at the CHSH game [13], where Alice and Bob receive uniformly random bits x and y respectively and their goal is to output bits a and b resp. such that $a \oplus b = x \wedge y$ without communicating. It is not hard to conclude that even if Alice and Bob share randomness, their optimal strategy will be successful with probability 0.75 over the inputs. However, if

*LRI - Université Paris-Sud
†NEC Laboratories America

© Kaplan, Kerenidis, Laplante, Roland; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 239–250
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2322

they share entanglement, then there is a strategy that succeeds with probability 0.85. This game proves that quantum entanglement can enable two parties to create correlations that are impossible to create with classical means.

Non-local boxes. As we said, entanglement enables Alice and Bob to succeed in the CHSH game with probability 0.85. But what if they shared some resource that would enable them to win the game with probability 1? Starting from such considerations, Popescu and Rohrlich [30] defined the notion of a non-local box. A non-local box is an abstract device shared by Alice and Bob. By one use of a non-local box, we mean that Alice inputs x, Bob inputs y, Alice gets as output a and Bob gets b where a, b are uniformly distributed and more importantly $a \oplus b = x \wedge y$. The name non-local box is due to the property that one use of a non-local box creates correlations between two bits that are maximally non-local (allowing to win the CHSH game with probability one), but still does not allow to communicate, since taken separately, each bit is just an unbiased random coin. As such, a non-local box may be considered as a unit of non-locality. We note here an important property of a non-local box, namely that, similar to entanglement, one player can enter an input and receive an output even before the second player has entered an input.

The importance of the notion of a non-local box has become increasingly evident in the last years. Non-local boxes were first introduced to study (quantum or generalized) non-locality. In particular, it was shown than one of the most studied versions of the EPR experiment, where Alice and Bob perform projective measurements on a maximally entangled qubit pair, may be simulated using only one use of a non-local box [10]. More generally, it was shown that any non-signaling distribution over Boolean outputs may be exactly simulated with some finite number of non-local boxes (for finite input size) [1, 19]. This was later generalized to any non-signaling distribution, except that the simulation may not always be performed exactly for non-Boolean outputs [16]. These results rely on the fact that the set of non-signaling distributions is a polytope, so it suffices to simulate the extremal vertices to be able simulate the whole set. In the context of non-locality, another application of non-local boxes is the study of pseudo-telepathy games [7].

It is easy to see that one use of a non-local box can be simulated with one bit of communication and shared randomness: Alice outputs a uniform bit r and sends x to Bob, who outputs $r \oplus x \cdot y$. However, the converse cannot possibly hold, since a non-local box cannot be used for communication.

The first question is what happens if we use non-local boxes as shared resource in the communication complexity model. Van Dam showed that for any Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, Alice and Bob can use 2^n non-local boxes and no communication at all and at the end output bits a and b such that $a \oplus b = f(x, y)$ [33]. In other words, if non-local boxes were physically implementable, then all functions would have trivial communication complexity. His results were strengthened by Brassard *et al.* who showed that even if a non-ideal non-local box existed, one that solves the CHSH game with probability 0.91, then still all functions would have trivial communication complexity [6]. Note that in these results, the number of non-local boxes needed may be exponential in the input size and do not take into account any properties of the function and more precisely its communication complexity without non-local boxes. It also follows from the work of [1, 6] that for

any Boolean function f, if f has a circuit with fan-in 2 of size s, then there is a deterministic non-local box protocol of complexity $O(s)$, where the bits of the input of f are split arbitrarily among the players. This implies that exhibiting an explicit function for which the deterministic non-local box complexity is superlinear would be a real breakthrough, since it would translate into a superlinear circuit lower bound for this function.

Secure function evaluation. Non-local boxes have also been studied in relation to cryptographic primitives such as Oblivious Transfer or Bit Commitment. Wolf and Wullschleger [34] showed that Oblivious Transfer is equivalent to a *timed* version of a non-local box (up to a factor of 2). To maintain the non-signaling property of the non-local box, one can define timed non-local box as having a predefined time limit, and if any of the players have not entered an input by this time, then some fixed input, say 0, is used instead. Subsequently, Buhrman *et al.* [8] showed how to construct Bit Commitment and Oblivious Transfer by using non-local boxes that do not need to be timed but have to be trusted.

In this paper, we are interested in secure function evaluation, which is one of the most fundamental cryptographic tasks. In this model, Alice and Bob want to evaluate some function of their inputs in a way that does not leak any more information than what follows from the output of the function. It is known that even though not all functions can be evaluated securely in the information-theoretic setting ([5, 11, 12, 26]), all functions can be computed securely in the information theoretic setting, if we have access to a black box that performs Oblivious Transfer or some other complete function, e.g. the AND function ([17, 20]).

There has been a lot of work trying to identify, in various settings, which functions can be easily evaluated in a secure way, *i.e.*, without any invocation of the black box, and which are hard to evaluate securely, *i.e.*, require at least one invocation of the black box ([12, 26, 3, 21, 23, 3, 22]). Moreover, Beaver [2] showed that there exists a hierarchy of different degrees of hardness for the information-theoretic reduction setting, in other words that for all k, there are functions that can be securely evaluated with k invocations of the AND box but cannot be computed with $k-1$ uses of the black box.

Beimel and Malkin [4] proposed a quantitative approach to secure function evaluation by studying how many calls to an Oblivious Transfer or other complete black box one needs in order to securely compute a given function f in the honest-but-curious model. For a Boolean function $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ and deterministic protocols, they provide a combinatorial characterization of the minimal number of AND calls required, which however does not lead to an efficient algorithm to determine how many ANDs are actually required. They also show that at most $2^{|\mathcal{X}|}$ ANDs are needed for any function. In the randomized case, they provide lower bounds depending on the truth-table of the function which can be at most of the order of n. They also state that "it would be very interesting to try and explore tighter connections with the communication complexity of the functions".

Finally, Naor and Nissim [29] have given some connections between the communication complexity of a function f and the communication complexity for securely computing f. These results, translated into the Beimel-Malkin and our model, only show that the number of ANDs is at most exponential in the communication complexity.

Our results. In this paper, we provide more evidence on the importance of non-local boxes by showing how they relate to different models of communication complexity as well as how they can be used as a tool to quantitatively study secure function evaluation. Our results show that non-local boxes, introduced for the study of quantum or more general non-locality, can provide a novel way of looking at questions about classical communication complexity, secure function evaluation and complexity theory.

2 Preliminaries

2.1 Communication Complexity

Let $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ be a bipartite Boolean function. Alice gets an input $x \in \mathcal{X}$ and Bob gets an input $y \in \mathcal{Y}$. We say that Alice and Bob compute $f(x,y)$ in parity if after executing a protocol, Alice outputs a bit a and Bob outputs a bit b such that $a \oplus b = f(x,y)$, where we use $\oplus$ to denote both the logical XOR and addition mod 2. This model differs from the standard model, where one of the players outputs the value of the function, by at most 1 bit.

We use the following notions of communication complexity. In probabilistic models, we assume that the players have a common source of randomness. $D(f)$ and $R_\varepsilon(f)$ denote deterministic and ε-bounded error communication complexity of $f(x,y)$ in parity. $D^{\rightarrow}(f)$ and $R_\varepsilon^{\rightarrow}(f)$ are the one-way deterministic and bounded-error communication complexity of $f(x,y)$ in parity. Finally, $D^{\parallel}(f)$ and $R_\varepsilon^{\parallel}(f)$ are the deterministic and bounded-error communication complexities in the model of simultaneous messages, where Alice and Bob each send a message to the referee and the referee outputs the value of the function $f(x,y)$.

For the model of simultaneous messages, we also consider some natural restrictions on how the referee computes the output from the messages he receives from the players. We assume the messages sent are of the same length. Suppose the referee receives bits $\mathbf{a} = (a_1, \ldots, a_t)$ from Alice, and $\mathbf{b} = (b_1, \ldots, b_t)$ from Bob. If the referee always computes a predefined function $g(\mathbf{a}, \mathbf{b})$, then we write $D^{\parallel,g}(f)$ or $R_\varepsilon^{\parallel,g}(f)$ to be the length of the message sent by the players (not the sum of these lengths, as is done in the standard model). In this paper, we will consider two functions, the inner product modulo 2, $IP_2(\mathbf{a}, \mathbf{b}) = \bigoplus_i (a_i \cdot b_i)$ (where $\cdot$ denotes the multiplication over $\mathbb{GF}_2$, which corresponds to the logical AND) and the majority function, $MAJ(\mathbf{a}, \mathbf{b}) = MAJ(a_1 \oplus b_1, \ldots, a_t \oplus b_t)$.

2.2 Non-local box Complexity

Definition 1 (Non-local box) *A non-local box is a device shared by two parties, which on one side takes Boolean input x and immediately produces Boolean output a, and on the other side takes Boolean input y and immediately produces Boolean output b, according to the following distribution:*

$$\mathbf{p}_{NL}(a,b|x,y) = \begin{cases} \frac{1}{2} & \text{if } a \oplus b = x \cdot y \\ 0 & \text{otherwise.} \end{cases}$$

We study a model akin to communication complexity, where Alice and Bob use non-local boxes instead of communication. In a non-local box protocol, Alice and Bob wish to compute some function $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ in the following way. Alice gets an input $x \in \mathcal{X}$, Bob gets an input $y \in \mathcal{Y}$, and at the end of the protocol, Alice outputs $a \in \{0,1\}$, Bob outputs $b \in \{0,1\}$, such that $a \oplus b = f(x,y)$. For a protocol P, we will write $P(x,y) = (a,b)$.

In the course of the protocol, Alice and Bob are allowed shared randomness and may use non-local boxes, but they may not communicate. Bob is not allowed to see Alice's inputs to the non-local boxes, nor does he see the outcome on Alice's side, and likewise for Alice.

Definition 2 *For any function $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, $NL(f)$ is the smallest t such that there is a protocol that computes f exactly, using t non-local boxes.*

We will label the non-local boxes with labels from 1 to t. (Recall that in general, Alice and Bob are not required to use the t non-local boxes in the same order.) We relax the exactness condition and allow the protocol's outcome to be incorrect with constant probability ε.

Definition 3 *For any function $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, $NL_\varepsilon(f)$ is the smallest t such that there is a protocol P using t non-local boxes, with $\Pr[P(x,y) = (a,b)$ with $a \oplus b = f(x,y)] \geq 1 - \varepsilon$.*

We will also study two variants of the general model, where the non-local boxes are used in a restricted manner. First, we assume that the non-local boxes are used in parallel, that is, the input to any non-local box does not depend on the outcome of any other. The complexity in this model is denoted $NL^{||}$ in the exact case, and $NL_\varepsilon^{||}$ in the ε error case.

Second, we define the model where both players use the non-local boxes in the same order, that is, the non-local boxes are labeled from 1 to t and Alice's input to the non-local box with label i does not depend on the outputs from the non-local boxes labeled from $i+1$ to t (similarly for Bob). The complexity in this model is denoted NL^{ord} in the exact case, and $NL_\varepsilon^{\text{ord}}$ in the ε error case. It is clear that this model is more powerful than the parallel model but less powerful than the general non-local box complexity. In fact, we will only use this last variant when we talk about secure function evaluation. Note also that in all these models, the non-local boxes are still non-signaling and Alice and Bob receive the outputs of the non-local boxes immediately after they enter their inputs.

Finally, we consider a restriction where the players always output the same predefined function g of the outputs of the non-local boxes. Let $(a_1, b_1), \ldots, (a_t, b_t)$ be the outcomes of the t non-local boxes in some particular run of a protocol. Of particular interest are protocols where Alice outputs $a = a_1 \oplus \cdots \oplus a_t$ and Bob outputs $b = b_1 \oplus \cdots \oplus b_t$. The function g is used in a superscript to denote the complexity of a function f in this model, NL^g in the determinstic case, and NL_ε^g in the ε error case, and in particular, $NL^{||,\oplus}$ and $NL_\varepsilon^{||,\oplus}$ when the non-local boxes are in parallel and $g = \oplus$.

2.3 Secure Function Evaluation

We will consider the following cryptographic primitives.

Definition 4 (Oblivious transfer) *A 2-1 Oblivious Transfer (OT) is a device which on input bits p_0, p_1 for Alice and q for Bob, outputs bit b to Bob, such that $b = p_q$.*

Definition 5 (Secure AND) *A secure AND is a device which on input bits p for Alice and q for Bob, outputs bit a to Alice, such that $a = p \cdot q$.*

While at first view, these definitions seem similar to the definition of the non-local box, note that the timing properties are different: for the cryptographic primitives, the outputs are produced only after all the inputs have been entered into the device. It is precisely this subtlety that has led to confusion when trying to use non-local boxes to implement cryptographic primitives, in particular for bit commitment, when timing is particularly important, since a cheating Alice could wait until the reveal phase before committing her bit into the

non-local box, without Bob ever realizing it [8]. However, we will see that this is not an issue for our results on secure computation.

Let $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ be a bipartite Boolean function. We study the number of cryptographic primitives necessary to compute f. In all the models we consider, we require perfect privacy. In the honest-but-curious model, perfect privacy means that when a player follows the protocol, he should not learn *more than required* about the other player's input. In the malicious model, this condition must still hold even if the player does not follow the protocol. *Not more than required* means, for models where the function must be computed in parity, that the players should learn nothing about the other's input, while for models where one of the player should output the function, it means that this player should learn nothing more than what he can infer from his input and the value of the function, while the other player should learn nothing.

Let us note that AND may not be used as a primitive in the malicious model, so we will consider the OT primitive instead. Moreover, in this model, it is known that randomness is necessary to achieve perfect privacy [15], so in this setting we do not consider the deterministic model. Our bounds in the randomized malicious model also hold for the weaker honest-but-curious model. We define $AND(f)$ to be the number of secure AND gates required to securely compute $f(x,y)$ (*not* in parity) in the deterministic, honest-but-curious model. We note that we can allow free two-way communication without in fact changing the complexity [4]. Similarly, $OT_\varepsilon(f)$ is the number of 2-1 Oblivious Transfer calls required to compute $f(x,y)$ in parity with perfect privacy and ε error over the players' private coins, assisted with (free) two-way communication, in the malicious model.

2.4 Complexity Measures

We will compare non-local box complexity to traditional models of communication complexity and prove upper and lower bounds for this new model. Some of these bounds are in terms of the factorization norms of the communication matrix [28] and related measures.

DEFINITION 1. *Let M be a real matrix. The γ_2 norm of M is $\gamma_2(M) = \min_{X^TY=M} col(X)col(Y)$, where $col(N)$ is the largest Euclidian norm of a column of N.*

It is known that $2\log(\gamma_2(M))$ gives a lower bound on deterministic communication complexity of M, where M is a sign matrix of the Boolean function to be computed [28]. In order to lower bound the randomized and quantum communication complexity, we have to consider a "smoothed" version of this measure.

DEFINITION 2. *Let M be a sign matrix and $\alpha \geq 1$. $\gamma_2^\alpha(M) = \min\{\gamma_2(N) : \forall i,j\, 1 \leq M_{i,j}N_{i,j} \leq \alpha\}$. In particular, $\gamma_2^\infty(M)$ is the minimum γ_2 norm over all matrices N such that $1 \leq M_{i,j}N_{i,j}$.*

The measures γ_2^α and γ_2^∞ give upper and lower bounds for bounded-error communication complexity [28]: $2\log(\gamma_2^\alpha(f)/\alpha) \leq R_\varepsilon(f)$ and $R_\varepsilon^{||,MAJ}(f) \leq O((\gamma_2^\infty(f))^2)$ (implicit in [28]), where $\alpha = \frac{1}{1-2\varepsilon}$. The discrepancy of a sign matrix M over inputs $X \times Y$ with respect to distribution μ over the inputs is $Disc_\mu(M) = \max_R \sum_{(x,y)\in R} \mu(x,y)M(x,y)$, where R is taken from all possible rectangles. It is known that $\gamma_2^\infty(f) = \Theta(\frac{1}{Disc(f)})$, and for any α, $\gamma_2^\infty(f) \leq \gamma_2^\alpha(f)$ [28]. Finally, for a Boolean function, the L_1 norm is defined as the sum

of the absolute values of its Fourier coefficients. We can think of the $2n$ bits of input of the function as equally split between Alice and Bob. Grolmusz uses this notion to upper bound the randomized communication complexity by proving that $R_\epsilon(f) \leq O(L_1^2(f))$ [18].

3 Deterministic non-local box complexity

We start by studying a restricted model of non-local box complexity, where the non-local boxes are used in parallel and at the end of the protocol, Alice and Bob output the parity of the outputs of their non-local boxes respectively. We will show that the complexity of f in this model is equal to the rank of the communication matrix of f over $\mathbb{GF}_2$. Note that this rank is equal to the minimum m, such that $f(x,y)$ can be written as $f(x,y) = \bigoplus_{i=1}^{m} a_i(x) \cdot b_i(y)$ (see also [9]). This restricted variant of non-local box complexity is exactly the one that appears in van Dam's work [33], where he shows that any Boolean function f can be computed by such a protocol of complexity 2^n. Moreover, we prove that the restriction that the players output the XOR of the outcomes of the non-local boxes is without loss of generality.

THEOREM 3. $NL^{||,\oplus}(f) = \text{rank}_{\mathbb{GF}_2}(M_f) = D^{\|,IP_2}(f)$.

PROOF. We start by showing that $NL^{||,\oplus}(f) \leq \text{rank}_{\mathbb{GF}_2}(M_f)$. Let $\text{rank}_{\mathbb{GF}_2}(M_f) = t$, *i.e.*, $f(x,y) = \bigoplus_{i\in[t]} p_i(x) \cdot q_i(y)$. Then we construct a protocol that uses t non-local boxes in parallel, where Alice and Bob output the parity of the outcomes of the non-local boxes and for every input (x,y) the output of the protocol is equal to $f(x,y)$. The inputs of Alice and Bob to the i-th non-local box are the bits $p_i(x)$ and $q_i(y), i \in [t]$ respectively and let a_i, b_i the outputs of the non-local box such that $a_i \oplus b_i = p_i(x) \cdot q_i(y)$. Alice and Bob output at the end of the protocol the value $(\bigoplus_{i\in[t]} a_i) \oplus (\bigoplus_{i\in[t]} b_i) = \bigoplus_{i\in[t]} p_i(x) \cdot q_i(y) = f(x,y)$.

Conversely, if there exists a protocol where Alice and Bob use t non-local boxes in parallel with inputs $p_i(x), q_i(y)$ and outputs a_i, b_i, their final output is $(\bigoplus_{i\in[t]} a_i) \oplus (\bigoplus_{i\in[t]} b_i)$ and it always equals $f(x,y)$, then we have $f(x,y) = (\bigoplus_{i\in[t]} a_i) \oplus (\bigoplus_{i\in[t]} b_i) = \bigoplus_{i\in[t]} p_i(x) \cdot q_i(y)$ and hence $\text{rank}_{\mathbb{GF}_2}(M_f) \leq t$. From this last argument, we get $D^{\|,IP_2}(f) \leq NL^{||,\oplus}(f)$ since the players can send p_i and q_i to the referee who computes the inner product. For the converse, if the referee receives m_A, m_B from each player and computes their inner product mod 2, the players can instead input each bit of the message into a non-local box and output the parity of the outputs to obtain the same result.

For the next corollary, we use the fact that $\log(2\text{rank}_{\mathbb{F}}(M_f) - 1) \leq D(f) + 1$ for any field $\mathbb{F}$ (see [27]).

COROLLARY 4. $NL^{||,\oplus}(f) \leq 2^{D(f)}$.

On the other hand, it is easy to see that the one-way communication complexity is a lower bound on the non-local box complexity. Alice can send all her inputs to Bob, and since the non-local box protocol is always correct, they can simulate it, assuming that Alice received only zeros from all non-local boxes.

LEMMA 5. $D^{\rightarrow}(f) \leq NL(f)$.

Moreover, we show that both in the general and in the parallel model of deterministic non-local box complexity, we can assume without loss of generality that the players output the XOR of the outcomes of the non-local boxes. Unlike the general case, showing that

in the parallel case we can assume that the players output the XOR of the outputs of the non-local boxes is not a trivial statement.

THEOREM 6. *$NL(f) \leq NL^{\oplus}(f) \leq NL(f)+2$, and $NL^{||}(f) \leq NL^{||,\oplus}(f) \leq NL^{||}(f)+2$.*

Last, our bounds for deterministic non-local box complexity are tight as can be shown by looking at the Inner Product and Disjointness functions. Indeed, for Inner Product we have $D(IP) = NL^{||}(IP) = n$, while for Disjointness, $NL^{||,\oplus}(DISJ) = 2^{D(DISJ)} = 2^n$. For Disjointness, the circuit size upper bound also implies that $NL(DISJ) = O(n)$, so there is an exponential separation between NL and $NL^{||,\oplus}$.

4 Randomized non-local box complexity

In this section, we consider protocols that use shared randomness and have success probability at least $2/3$. We start by comparing the parallel non-local box complexity to communication complexity. In the full paper, we also exactly characterize $NL_{\varepsilon}^{||,\oplus}$ in terms of the approximate rank (over $\mathbb{GF}_2$) of the communication matrix.

THEOREM 7. *$R_{\epsilon}^{\rightarrow}(f) \leq NL_{\varepsilon}^{||}(f) \leq NL_{\varepsilon}^{||,\oplus}(f) \leq 2^{R_{\epsilon}(f)}$.*

Next, we relate the general non-local box complexity to the following model of communication: Alice and Bob send to a referee one message each and the referee outputs 1 if for the majority of indices, the two messages are equal. We denote the communication complexity in this model by $R_{\epsilon}^{||,MAJ}(f)$. This is a natural model of communication complexity that has appeared repeatedly in the simulation of quantum protocols by classical ones, as well as various upper bounds on simultaneous messages [25, 18, 32, 28].

THEOREM 8. *$R_{\varepsilon}^{\rightarrow}(f) \leq NL_{\varepsilon}(f) \leq O(R_{\epsilon}^{||,MAJ}(f))$.*

PROOF. The lower bound proof follows directly from the deterministic case. For the upper bound, fix a t-bit simultaneous protocol for f, where the referee receives two messages $\mathbf{a}$ and $\mathbf{b}$ of size t from Alice and Bob and outputs $MAJ(a_1 \oplus b_1, \ldots, a_t \oplus b_t)$. It is well-known, by using an addition circuit, that the majority of t bits can be computed by a circuit of size $O(t)$ with AND, NOT gates. Moreover, the distributed AND of two bits can be computed using two non-local boxes [6]. We conclude that the non-local box complexity of the distributed Majority is $O(t)$ and hence the theorem follows.

COROLLARY 9. *$2\log(\gamma_2^{\alpha}(f)/\alpha) \leq NL_{\varepsilon}(f) \leq O((\gamma_2^{\infty}(f))^2)$, where $\alpha = \frac{1}{1-2\varepsilon}$.*

It is known that $\gamma_2^{\infty}(f) = \Theta(\frac{1}{Disc(f)})$, and also that for any α, $\gamma_2^{\infty}(f) \leq \gamma_2^{\alpha}(f)$ [28]. Hence, since discrepancy gives a lower bound on the quantum communication complexity with entanglement $Q_{\varepsilon}^{*}(f)$ [24], we get the following corollary.

COROLLARY 10. *$NL_{\epsilon}(f) \leq O(2^{2Q_{\varepsilon}^{*}(f)})$.*

Finally, we can relate the non-local box complexity of a function f, to the L_1 norm of the Fourier coefficients of f by using a result by Grolmusz. Grolmusz showed that for any Boolean function f, there exists a randomized public coin protocol that solves f with complexity $O(L_1^2(f))$. This protocol can be easily transformed into a simultaneous messages protocol where the referee outputs the distributed majority of the message bits. Hence,

COROLLARY 11. $NL_\epsilon(f) \leq O(L_1^2(f))$.

Let us make here a last remark about the proof of Theorem 8. We started from a Simultaneous Messages protocol where the referee outputs a Majority function and we constructed a non-local box protocol with complexity equal to the communication complexity. If we look at this protocol, we can see that Alice and Bob can use their non-local boxes in the same order. This will be useful when we relate non-local boxes to secure function evaluation.

COROLLARY 12. $R_\varepsilon^{\rightarrow}(f) \leq NL_\varepsilon(f) \leq NL_\varepsilon^{\text{ord}}(f) \leq O(R_\varepsilon^{||,MAJ}(f))$.

Our bounds are almost tight for the general case, but the case of parallel non-local box complexity is more interesting. We can give a simple $O(n)$ parallel protocol for Disjointness, showing that the exponential separation does not hold anymore. It is open whether parallel and general randomized non-local box complexity are polynomially related.

5 Non-local boxes and measurement simulation

In this section we present another application of our results on non-local boxes. Using the recent breakthrough of Regev and Toner [31], who give a two-bit one-way protocol for simulating two-outcome measurements on entangled states for arbitrary dimensions, we show that this can be done with 3 non-local boxes. Previously, no finite upper bound was known for this problem. In the full paper, we prove the following, more general, theorem.

THEOREM 13. *For any non-signaling distribution over binary outputs with uniform marginals, any t-bit communication protocol can be simulated with $2^t - 1$ non-local boxes in parallel.*

The proof builds on an idea presented in [14] to replace communication by non-local boxes, which is here used recursively, and is given in the full paper.

6 Secure Function Evaluation

6.1 Honest-but-curious model

As a starting point, we consider the most basic model, namely deterministic secure computation with ANDs in the honest-but-curious model. Beimel and Malkin [4] have shown that $AND(f) \leq 2^{|\mathcal{X}|}$. We show that it is characterized by the one-way communication complexity of f. (The proof is given in the full paper.)

THEOREM 14. $AND(f) = 2^{D^{\rightarrow}(f)}$.

One can say that this shows that for most functions, randomization is necessary in order to construct efficient protocols even in the honest-but-curious model.

6.2 Malicious model

Due to their non-signaling property, protocols using non-local boxes only and no communication, such as those presented in the previous sections, are trivially secure even against malicious players. Indeed, the non-signaling property implies that the view of the protocol by a possibly dishonest player is always independent from the actions of the other player.

We show that certain such protocols may be transformed into protocols using OTs, namely the protocols where Alice and Bob use their non-local boxes in the same order. At this point, we don't know if this type of protocols are strictly weaker than general non-local box protocols. Nevertheless, our upper bounds in terms of communication complexity hold for such protocols as well (Corollary 12) and hence they translate into upper bounds on $OT_\varepsilon(f)$.

THEOREM 15. *For any* $\varepsilon \geq 0$, $OT_\varepsilon(f) \leq NL_\varepsilon^{\text{ord}}(f)$.

The proof, which will be given in the full version of the paper, consists in first showing how to simulate the non-local box protocol using OTs, following a construction due to Wolf and Wullschleger [34]. The security of the OT protocol then follows from the non-signaling property of the non-local boxes. From the above theorem we can conclude that all the upper bounds that we had for the $NL_\varepsilon^{\text{ord}}$ complexity (see Corollaries 9-12) translate into upper bounds for $OT_\varepsilon(f)$.

We now turn our attention to lower bounds. For this we need to restrict ourselves to what we call 'optimal' secure protocols. An 'optimal' secure protocol is one where the function is computed securely in the usual sense, but we also require that for all the OT calls, there is always an input that remains perfectly secure throughout the protocol. Intuitively, since we try to minimize the number of OTs that we use, it should be the case that these OT calls are really necessary, in the sense that one of the two inputs should always remain secure. If for example both inputs are revealed at some point during the protocol, then one may not use this OT at all, resulting into a more efficient protocol. Even though intuitively our definition seems natural, at this point, we do not know whether this assumption can be done without loss of generality. We denote by $\widehat{OT}_\varepsilon(f)$ the minimum number of OT calls of an 'optimal' secure protocol. In the full paper we provide the formal definition and prove the following

THEOREM 16. $\widehat{OT}_\varepsilon(f) = \Omega(R_\varepsilon(f))$.

7 Conclusion and open questions

We have shown various upper and lower bounds on non-local box complexity, and shown how the upper bounds could be translated into bounds for secure function evaluation. We have also shown how to simulate quantum correlations arising from binary measurements on bipartite entangled states using 3 non-local boxes.

During our investigations, we have come across a series of interesting open questions. 1) While the disjointness function provides an example of exponential gap between parallel and general deterministic non-local box complexity, the gap disappears in the randomized model. Are parallel and general randomized non-local box complexities polynomially related? 2) Are there functions for which $NL_\varepsilon^{\text{ord}}(f) > NL_\varepsilon(f)$? 3) We proved that the communication complexity is a lower bound on OT complexity only under some optimality assumption. Can this assumption be made without loss of generality? 4) Can we prove an analogue of Theorem 16 for non-local boxes? Ideally, we would like to prove that for secure computation with non-local boxes, communication does not help. Indeed, due to the

reduction from non-local boxes to OT boxes and vice versa, this would imply that $NL_\varepsilon^{\text{ord}}(f)$ is exactly $OT_\varepsilon(f)$, and not just an upper bound.

Acknowledgements

We would like to thank Troy Lee and Falk Unger for pointing out the deterministic protocol for Disjointness. We thank Ronald de Wolf for suggesting a randomized protocol for disjointness with bias $1/\log(n)$, using Valiant-Vazirani.

References

[1] Jonathan Barrett and Stefano Pironio. Popescu-Rohrlich correlations as a unit of nonlocality. *Phys. Rev. Lett.*, 95:140401, 2005.

[2] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proc. 28th STOC*, pages 479–488, 1996.

[3] A. Beimel, T. Malkin, and S. Micali. The all-or-nothing nature of two-party secure computation. In *Proc. CRYPTO '99*, pages 80–97, 1999.

[4] Amos Beimel and Tal Malkin. A quantitative approach to reductions in secure computation. In *Proc. TCC'04*, volume 2951, pages 238–257, 2004.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proc. 20th STOC*, pages 1–10, 1988.

[6] Gilles Brassard, Harry Buhrman, Noah Linden, Andre Allan Méthot, Alain Tapp, and Falk Unger. Limit on nonlocality in any world in which communication complexity is not trivial. *Phys. Rev. Lett.*, 96(25):250401, 2006.

[7] Anne Broadbent and André Allan Méthot. On the power of non-local boxes. *Theoretical Computer Science*, 358(1):3–14, 2005.

[8] Harry Buhrman, Matthias Christandl, Falk Unger, Stephanie Wehner, and Andreas Winter. Implications of superstrong nonlocality for cryptography. *Proc. Roy. Soc. A*, 462:1919–1932, 2007.

[9] Harry Buhrman and Ronald de Wolf. Communication complexity lower bounds by polynomials. In *Proc. 16th CCC*, pages 120–130, 2001.

[10] Nicolas J. Cerf, Nicolas Gisin, Serge Massar, and Sandu Popescu. Simulating Maximal Quantum Entanglement without Communication. *Phys. Rev. Lett.*, 94(22):220403, 2005.

[11] D. Chaum, C. Crepeau, and I. Damgard. Multiparty unconditionally secure protocols. In *Proc. 20th STOC*, pages 11–19, 1988.

[12] B. Chor and E. Kushilevitz. A zero-one law for boolean privacy. *SIAM J. Discrete Mathematics*, 4(1):36–47, 1991.

[13] John F. Clauser, Michael A. Horne, Abner Shimony, and Richard A. Holt. Proposed Experiment to Test Local Hidden-Variable Theories. *Phys. Rev. Lett.*, 23:880–884, 1969.

[14] Julien Degorre, Sophie Laplante, and Jérémie Roland. Classical simulation of traceless binary observables on any bipartite quantum state. *Phys. Rev. A*, 75(012309), 2007.

[15] Yevgeniy Dodis and Silvio Micali. Lower bounds for oblivious transfer reductions. In *Proc. EUROCRYPT*, pages 42–55, 1999.

[16] Manuel Forster and Stefan Wolf. The universality of non-local boxes. In *Proc. 9th QCMC*, 2008. To appear.

[17] O. Goldreich and R. Vainish. How to solve any protocol problem - an efficiency improvement. In *Proc. CRYPTO '87*, pages 73–86, 1988.

[18] Vince Grolmusz. On the power of circuits with gates of low l_1 norms. *Theoretical Computer Science A*, 188:117–127, 1997.

[19] Nick S. Jones and Lluis Masanes. Interconversion of nonlocal correlations. *Phys. Rev. A*, 72(5):052312, 2005.

[20] J. Kilian. Basing cryptography on oblivious transfer. In *Proc. 20th STOC*, pages 20–31, 1988.

[21] J. Kilian. A general completeness theorem for two-party games. In *Proc. 23th STOC*, pages 553–560, 1991.

[22] J. Kilian. More general completeness theorems for two-party games. In *Proc. STOC*, pages 316–324, 2000.

[23] J. Kilian, E. Kushilevitz, S. Micali, and R. Ostrovsky. Reducibility and completeness in private computations. *SIAM J. Comput.*, 28(4):1189–1208, 2000.

[24] Ilan Kremer. Quantum communication. Master's thesis, The Hebrew University of Jerusalem, 1995.

[25] Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-way communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

[26] E. Kushilevitz. Privacy and communication complexity. *SIAM J. Discrete Mathematics*, 5(2):273–284, 1992.

[27] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, New York, 1997.

[28] Nati Linial and Adi Shraibman. Lower bounds in communication complexity based on factorization norms. *Random Structures and Algorithms*, 2008.

[29] Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *Proc. 33rd STOC*, 2001.

[30] Sandu Popescu and Daniel Rohrlich. Causality and nonlocality as axioms for quantum mechanics. *Foundations of Physics*, pages 379–385, 1994.

[31] Oded Regev and Benjamin Toner. Simulating quantum correlations with finite communication. In *Proc. 48th FOCS*, pages 384–394, 2007.

[32] Yaoyun Shi and Yufan Zhu. Tensor norms and the classical communication complexity of bipartite quantum measurements. *SIAM J. Comput.*, 38(3):753–766, 2008.

[33] Wim van Dam. Implausible Consequences of Superstrong Nonlocality. Technical Report quant-ph/0501159, arXiv e-Print archive, 2005.

[34] Stefan Wolf and Jürg Wullschleger. Oblivious transfer and quantum non-locality. In *Proc. ISIT*, pages 1745–1748, 2005.

[35] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing. In *Proc. 11th STOC*, pages 209–213, 1979.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Verification and Refutation of Probabilistic Specifications via Games*

Mark Kattenbelt[1], **Michael Huth**[2]

[1] Oxford University Computing Laboratory
Wolfson Building, Parks Rd, Oxford, UK

[2] Department of Computing, Imperial College London
Huxley Building, 180 Queen's Gate, London, UK

ABSTRACT. We develop an abstraction-based framework to check probabilistic specifications of Markov Decision Processes (MDPs) using the stochastic two-player game abstractions (i.e. "games") developed by Kwiatkowska et al. as a foundation. We define an abstraction preorder for these game abstractions which enables us to identify many new game abstractions for each MDP — ranging from compact and imprecise to complex and precise. This added ability to trade precision for efficiency is crucial for scalable software model checking, as precise abstractions are expensive to construct in practice. Furthermore, we develop a four-valued probabilistic computation tree logic (PCTL) semantics for game abstractions. Together, the preorder and PCTL semantics comprise a powerful verification and refutation framework for arbitrary PCTL properties of MDPs.

1 Introduction

Model checking [5, 28] is a methodology for reasoning about the formal correctness of systems. The task of a model checker is to decide whether a *model* M satisfies a *property* ϕ. We write this as a *judgment* $M \models \phi$ and say a model checker *verifies* or *refutes* such judgments.

It is often intractable to verify or refute judgments involving large or infinite-state models directly. A recognised solution is to apply *abstraction*. That is, we can reason about the validity of $M \models \phi$ by model checking judgments involving abstractions A of M. Usually abstraction is used within an *abstraction-refinement loop* [7]. Starting with a very imprecise abstraction A this loop, if necessary, incrementally *refines* A until it is precise enough to either verify or refute $M \models \phi$. When model checking *software* one would typically use *existential abstractions* [8, 1], with which it is possible to verify a certain class of properties. However, to refute these properties, one has to concretise *abstract counter-examples* [7].

In this paper we focus on *probabilistic* models and properties. Unfortunately, counter-examples of probabilistic models are usually complex infinite structures [14]. Hence, refutation by concretising abstract counter-examples akin to existential abstractions is a lot more involved for probabilistic models [16]. This motivates us to consider abstraction schemes with which we can directly refute properties (c.f. *modal* or *mixed abstractions* [26, 10]). Such abstractions also potentially demonstrate the validity or falsity of $M \models \phi$ more compactly and more intuitively than probabilistic witnesses or counter-examples. That is, one may use abstractions as *diagnostic tools* or *certificates* demonstrating validity or falsity of $M \models \phi$ [27].

*This work was, in part, supported by UK EPSRC grants EP/D07956X/2 and EP/E028985/1.

© Mark Kattenbelt and Michael Huth; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 251–262
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2323

The models we consider are Markov decision processes (MDPs), which naturally model a wide range of probabilistic systems due to their ability to capture both *probabilistic* and *non-deterministic* choice. Our abstraction scheme is based on the *two-player stochastic game* (i.e. "game") abstractions suggested in [24]. In [24], these abstractions are used to over and under-approximate reachability probabilities of MDPs — as opposed to the verification and refutation of more complex properties.

Unfortunately, the definition of abstraction in [24] has a shortcoming. Given a particular partition of MDP states it only considers the optimal game over this partition. That is, other than the expected loss of precision that occurs due to joining MDP states, there is no mechanism that enables one to lose more precision — in fact, it is unclear what less precise abstractions over this partition would look like. Optimal abstractions over a partition are usually inefficient to represent and computationally expensive to construct. Moreover, the same job can often be accomplished with less precise abstractions. This is evident in most abstraction-based software model checkers which, for a fixed partition,[†] first consider a coarse abstraction over this partition and consider more precise abstractions over this partition only if this is necessary [1, 6]. Due to the shortcoming of [24] it is not possible to take such an approach with game abstractions and, as constructing optimal game abstractions is expensive, this significantly affects the scalability of, e.g., the method in [22].

Hence, the first issue we address in this paper is the development of an *abstraction preorder* for games which alleviates the shortcomings of [24]. Compared to the work in [24], this preorder identifies many additional game abstractions of varying precisions — even when restricted to a fixed partition. This opens up the possibility of adapting the method in [22] and other tools using game abstractions [25, 21, 20] to reason more efficiently via non-optimal games. Furthermore, instead of over and under-approximating reachability properties of MDPs, we develop a *four-valued probabilistic computation tree logic (PCTL) [15] semantics* for games and show our abstraction preorder preserves this semantics. Our abstraction preorder and PCTL semantics together comprise a powerful abstraction framework with which we can verify and refute arbitrary PCTL specifications of MDPs.

2 Background

Let AP be a fixed set of atomic propositions. Let $\mathbb{B}$ be the Boolean domain. Let $\mathbb{P}X$ be the powerset of a set X, excluding $\emptyset$. A probability distribution over X is a function $\lambda : X \to [0,1]$ such that $\sum_{x \in X} \lambda(x) = 1$ and the set $\{x \in X \mid \lambda(x) > 0\}$ is countable. Let $\mathbb{D}X$ be the set of all distributions over X. For $x \in X$ let $[x] \in \mathbb{D}X$ be the point distribution on x, i.e. $[x](x) = 1$. Every distribution over a countable set can be written as a countable sum of point distributions $\sum_i w_i \cdot [x_i]$.

We model four-valued logic [2] with a *must* (!) and *may* (?) modality of truth. Intuitively, ?-true corresponds to *possible truth* and !-true indicates *certain truth*. We represent *true* (resp. *false*) by being both !-true and ?-true (resp. !-false and ?-false). We represent *uncertainty* by being !-false and ?-true and *inconsistency* by being !-true and ?-false.

Given an arbitrary non-empty sequence $\pi = \omega_0; \omega_1; \omega_2, \ldots$ let $|\pi|$ be the number elements of π minus one. We let $\pi(i)$ be ω_i and, if $|\pi|$ is finite, let $\text{LAST}(\pi)$ be $\pi(|\pi|)$. Finally,

[†]That is to say, a fixed set of predicates when using predicate abstraction [1].

let π^i be the prefix of π such that $|\pi^i| = i$. We write $\pi;\pi'$ for the concatenation of sequences.

Markov decision processes We now introduce Markov decision processes (MDPs) which naturally model systems with both non-deterministic and probabilistic behaviours:

DEFINITION 1. *A* Markov decision process *(MDP) M is a tuple $\langle S, s_i, T, L\rangle$, where:*

- *S is a countable set of states;*
- *$s_i \in S$ is an* initial *state;*
- *$T \in S \to \mathbb{P}\mathbb{D}S$ is a* transition function*;*
- *$L \in S \times \text{AP} \to \mathbb{B}$ is a* labelling function.

The definition of $\mathbb{P}$ ensures totality*: i.e. $\forall s \in S : |T(s)| > 0$ (this totality requirement is not essential and is made for presentational reasons, only). We let $\mathcal{M}$ be the class of all MDPs.*

From a state $s \in S$, a non-deterministic choice *picks a distribution $\lambda \in T(s)$. Then, the next state $s' \in S$ is picked* probabilistically *according to λ. A* path *of M is a sequence over $S \cup \mathbb{D}S$ that strictly alternates between states and distributions as described. Let Π_M and Π_M^∞ be the set of all* finite *and* infinite *paths, respectively. For $\Omega \subseteq S \cup \mathbb{D}S$ we write, e.g., $\Pi_M(\Omega)$ to restrict to paths starting with an element in Ω.*

A strategy *of M is a partial function $\sigma\colon \Pi_M \to \mathbb{D}\mathbb{D}S$, with domain of definition all π with $\text{LAST}(\pi) \in S$, such that $\sigma(\pi) \in \mathbb{D}(T(\text{LAST}(\pi)))$. As is evident from the definition, we consider* randomised *strategies (i.e. strategies that resolve non-determinism with a probabilistic choice). Let Σ_M be all strategies of M. A path π is* consistent *with $\sigma \in \Sigma_M$ iff for all $i \leq |\pi| - 1$ with $\pi(i) \in S$ the probability $\sigma(\pi^i)(\pi(i+1))$ is positive. We write, e.g., $\Pi_{M,\sigma}$ to restrict to paths consistent with σ. For every $s \in S$ and $\sigma \in \Sigma_M$ we construct a probability measure $\mathbf{Pr}^s_{M,\sigma}$ over infinite paths $\Pi^\infty_{M,\sigma}(\{s\})$ with standard techniques [23].*

Probabilistic CTL We define an adequate PCTL fragment with unrestricted negation [15]:

DEFINITION 2. *A PCTL formula is defined with the following BNF-style syntax rules where $a \in \text{AP}$, $k \in \mathbb{N} \cup \{\infty\}$, $p \in [0,1]$ and $\bowtie \in \{\leq, <, \geq, >\}$:*

$$\phi ::= a \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{P}_{\bowtie p}\langle\psi\rangle \qquad\qquad \psi ::= \mathsf{X}\phi \mid \phi_1 \mathsf{U}^{\leq k} \phi_2 \,.$$

We let Φ and Ψ be the set of all PCTL formulae of the form ϕ and ψ respectively.

PCTL semantics of MDPs Finally, we define standard PCTL semantics of MDPs via a satisfaction relation $\models \subseteq \mathcal{M} \times \Phi$ [3]:

DEFINITION 3. *Let $M = \langle S, s_i, T, L\rangle$ be an MDP, and let $\phi \in \Phi$ be a PCTL formula. We define satisfaction relations for states $\models \subseteq S \times \Phi$ and paths $\models \subseteq \Pi^\infty_M(S) \times \Psi$ as follows:*

$$\begin{array}{lcl}
\pi \models \mathsf{X}\phi & \iff & s_1 \models \phi \\
\pi \models \phi_1 \mathsf{U}^{\leq k} \phi_2 & \iff & \exists i \leq k : \big(s_i \models \phi_2 \wedge (\forall j < i : s_j \models \phi_1)\big) \\
s \models a & \iff & L(s,a) \\
s \models \neg\phi & \iff & s \not\models \phi \\
s \models \phi_1 \vee \phi_2 & \iff & (s \models \phi_1 \ \text{ or } \ s \models \phi_2) \\
s \models \mathsf{P}_{\triangleright p}\langle\psi\rangle & \iff & \inf_\sigma \ \mathbf{Pr}^s_{M,\sigma}\{\pi \in \Pi^\infty_{M,\sigma}(s) \mid \pi \models \psi\} \triangleright p \\
s \models \mathsf{P}_{\triangleleft p}\langle\psi\rangle & \iff & \sup_\sigma \mathbf{Pr}^s_{M,\sigma}\{\pi \in \Pi^\infty_{M,\sigma}(s) \mid \pi \models \psi\} \triangleleft p
\end{array}$$

with $\pi = s_0;\lambda_0;s_1;\lambda_1\ldots$, $\triangleright \in \{>, \geq\}$, $\triangleleft \in \{<, \leq\}$ and $\sigma \in \Sigma_M$. Moreover, $M \models \phi$ iff $s_i \models \phi$.

For any PCTL path formula, the set of paths satisfying it is measurable [30]. For MDPs, properties like $\mathsf{P}_{\bowtie p}\langle\psi\rangle$ are *universal* as threshold $\bowtie p$ must be met under *all* strategies. Conversely, properties like $\neg\mathsf{P}_{\bowtie p}\langle\psi\rangle$ hold iff there *exists* a strategy that violates the threshold.

We use standard methods to lift any relation to a relation over distributions [18]. For any relation $R \subseteq X \times Y$ we let $R_{\mathbb{D}} \subseteq \mathbb{D}X \times \mathbb{D}Y$ be the relation such that $\lambda_X \, R_{\mathbb{D}} \, \lambda_Y$ iff there is a weight function $\delta \in X \times Y \to [0,1]$ with (for all $x \in X$, $y \in Y$):

$$\lambda_X(x) = \textstyle\sum_{y' \in Y} \delta(x,y') \qquad \lambda_Y(y) = \textstyle\sum_{x' \in X} \delta(x',y) \qquad \delta(x,y) > 0 \Rightarrow x \, R \, y \tag{1}$$

3 Game-based abstraction framework

We now introduce the components of our abstraction framework. We start by formally defining a class of games ($\mathcal{G}$) and an *embedding function* ($emb : \mathcal{M} \to \mathcal{G}$), which casts MDPs into $\mathcal{G}$. We then define an *abstraction preorder* ($\sqsubseteq_p \subseteq \mathcal{G} \times \mathcal{G}$) as a relation over games. Our last component is a *four-valued PCTL semantics* ($\models^!, \models^? \subseteq \mathcal{G} \times \Phi$) over games. Finally, we show our components satisfy some necessary *soundness properties*. With these components and properties we then show how to verify and refute arbitrary PCTL properties of MDPs.

Stochastic two-player games ($\mathcal{G}$) In comparison to MDPs, games are equipped with an additional level of choice (i.e. their transition function yields *sets of sets* of distributions). Games also have four-valued propositional labelling (through a *must* and *may* labelling):

DEFINITION 4. *A* stochastic two-player game *G is a tuple $\langle S, s_i, T, L^!, L^? \rangle$, where:*

- *S is a countable set of states;*
- *$s_i \in S$ is an* initial *state;*
- *$T \in S \to \mathbb{P}\mathbb{P}\mathbb{D}S$ is a* transition function*;*
- *$L^!, L^? \in S \times \mathrm{AP} \to \mathbb{B}$ are* labelling functions.

By definition of $\mathbb{P}$ we ensure totality *for T: i.e. we have $|T(s)| > 0$ for all $s \in S$ and $|\Lambda| > 0$ for all $\Lambda \in T(s)$. Let $\mathcal{G}$ be the class of all games. We define* player 1 states *as elements of S,* player 2 states *as sets of distributions over player 1 states ($\mathbb{P}\mathbb{D}S$) and* probabilistic states *as distributions over player 1 states ($\mathbb{D}S$). From a player 1 state $s \in S$ player 1 can transition to a player 2 state $\Lambda \in \mathbb{P}\mathbb{D}S$ iff $\Lambda \in T(s)$ (written $s \to_1 \Lambda$). Analogously, from a player 2 state $\Lambda \in \mathbb{P}\mathbb{D}S$ player 2 can transition to a probabilistic state $\lambda \in \mathbb{D}S$ iff $\lambda \in \Lambda$ (written $\Lambda \to_2 \lambda$). Finally, from a probabilistic state $\lambda \in \mathbb{D}S$ the game transitions to a player 1 state $s' \in S$ with probability $\lambda(s')$ (written $\lambda \to_p s'$).*

A play *in G is a sequence of transitions[‡] and hence necessarily strictly alternates between player 1 states, player 2 states and probabilistic states. Let Π_G and Π_G^∞ be the set of all finite and infinite plays of G, respectively. For $\Omega \subseteq S \cup \mathbb{P}\mathbb{D}S \cup \mathbb{D}S$ we write, e.g., $\Pi_G(\Omega)$ to restrict to plays starting with an element in Ω.*

A player 1 strategy *is a partial function $\sigma_1 \in \Pi_G \to \mathbb{D}\mathbb{P}\mathbb{D}S$, with domain of definition all π with $\mathrm{LAST}(\pi) \in S$, such that $\sigma_1(\pi) \in \mathbb{D}(T(\mathrm{LAST}(\pi)))$. Analogously, a* player 2 strategy *is a partial function $\sigma_2 \in \Pi_G \to \mathbb{D}\mathbb{D}S$ with domain of definition all π with $\mathrm{LAST}(\pi) \in \mathbb{P}\mathbb{D}S$, such that $\sigma_2(\pi) \in \mathbb{D}(\mathrm{LAST}(\pi))$. We write Σ_G^1 and Σ_G^2 for the set of all player 1 and player*

[‡]We will manipulate plays as if they are sequences over $S \cup \mathbb{P}\mathbb{D}S \cup \mathbb{D}S$.

2 strategies, respectively. A play π of G is consistent *with $\sigma_1 \in \Sigma_G^1$ if for every $i \leq |\pi| - 1$ with $\pi(i) \in S$ the probability $\sigma_1(\pi^i)(\pi(i+1))$ is positive. Similarly, π is* consistent *with $\sigma_2 \in \Sigma_G^2$ if $\sigma_2(\pi^i)(\pi(i+1))$ is positive whenever $\pi(i) \in \mathbb{PD}S$. For $\sigma_1 \in \Sigma_G^1$ and $\sigma_2 \in \Sigma_G^2$ we write, e.g., $\Pi_{G,\sigma_1,\sigma_2}$ to restrict to plays consistent with σ_1 and σ_2. For $\sigma_1 \in \Sigma_G^1$ and $\sigma_2 \in \Sigma_G^2$ and $\Pi \subseteq \Pi_{G,\sigma_1,\sigma_2}$, we denote with $\Pi^{\uparrow_{\sigma_1,\sigma_2}}$ the infinite plays of G that are consistent with both σ_1 and σ_2 and have a prefix in Π.*

Given strategies $\sigma_1 \in \Sigma_G^1$ and $\sigma_2 \in \Sigma_G^2$ the behaviour of G is purely probabilistic. Hence, for each $\omega \in S \cup \mathbb{PD}S \cup \mathbb{D}S$, using standard techniques [23], we construct a probability space over infinite plays $\Pi^{\infty}_{G,\sigma_1,\sigma_2}(\{\omega\})$ with probability measure $\mathbf{Pr}^{\omega}_{G,\sigma_1,\sigma_2}$ such that $\mathbf{Pr}^{\omega}_{G,\sigma_1,\sigma_2}(\{\omega\}^{\uparrow_{\sigma_1,\sigma_2}}) = 1$ and, for every finite play of non-zero length $\pi \in \Pi_{G,\sigma_1,\sigma_2}(\{\omega\})$:

$$\mathbf{Pr}^{\omega}_{G,\sigma_1,\sigma_2}(\{\pi\}^{\uparrow_{\sigma_1,\sigma_2}}) = \begin{cases} \mathbf{Pr}^{\omega}_{G,\sigma_1,\sigma_2}(\{\pi'\}^{\uparrow_{\sigma_1,\sigma_2}}) \cdot \sigma_1(\pi')(\Lambda') & \text{if } (\pi = \pi' \to_1 \Lambda') \\ \mathbf{Pr}^{\omega}_{G,\sigma_1,\sigma_2}(\{\pi'\}^{\uparrow_{\sigma_1,\sigma_2}}) \cdot \sigma_2(\pi')(\lambda') & \text{if } (\pi = \pi' \to_2 \lambda') \\ \mathbf{Pr}^{\omega}_{G,\sigma_1,\sigma_2}(\{\pi'\}^{\uparrow_{\sigma_1,\sigma_2}}) \cdot \mathrm{LAST}(\pi')(s') & \text{if } (\pi = \pi' \to_p s') \end{cases}$$

REMARK 5. *In figures we depict player 1 states with big open circles, player 2 states with small filled squares and probabilistic states with filled black circles. Labels depict the probability of transitions (omitted for point distributions). We write $a!$, $a?$ and $a!?$ next to s iff $L^!(s,a) \wedge \neg L^?(s,a)$, $\neg L^!(s,a) \wedge L^?(s,a)$ or $L^!(s,a) \wedge L^?(s,a)$ resp., and nothing otherwise.*

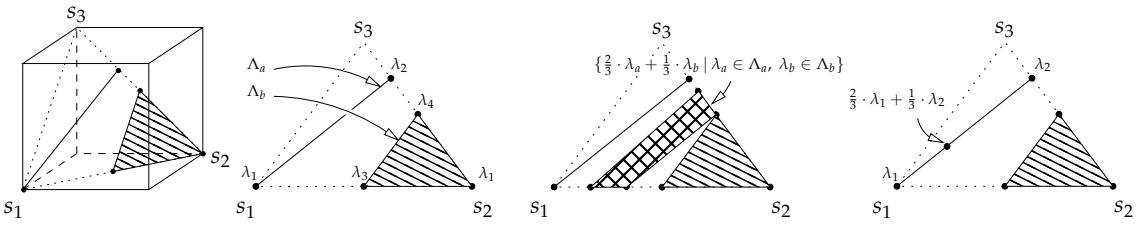
The roles of player 1 & 2 Before we define the components of our abstraction framework we first give an informal account of how a game $\hat{G}$ (over states $\hat{S}$) abstracts an MDP M (over states S). Intuitively, to be sound for PCTL — or any unrestricted branching-time logic — $\hat{G}$ must both under and over-approximate the strategies that are feasible in M. Observe that a strategy of M is simply a particular resolution of non-determinism in M and hence, to under and over-approximate feasible strategies of M, $\hat{G}$ must under and over-approximate the non-deterministic choice $T(s) \in \mathbb{PD}S$ in each state of M.[§] We use player 1 states of $\hat{G}$ to represent sets of states of M and player 2 states of $\hat{G}$ to approximate the non-deterministic choices of these states (i.e. player 2 resolves the non-determinism of M). Informally, in $\hat{s} \in \hat{S}$ player 1 can choose from $\hat{T}(\hat{s}) \in \mathbb{PPD}\hat{S}$ at least one player 2 state that under-approximates $T(s)$ and one player 2 state that over-approximates $T(s)$ for every concrete state $s \in S$ of M that $\hat{s}$ abstracts. As $\hat{s}$ may abstract many states of M, and for each such state we have both under and over-approximating player 2 choices, there may be many player 1 choices in $\hat{T}(\hat{s})$. Hence, player 1 resolves non-determinism introduced by abstraction.

The embedding function (*emb*) The first component of our framework is an *embedding function* $emb : \mathcal{M} \to \mathcal{G}$, which yields an exact representation $emb(M)$ in $\mathcal{G}$ for each MDP M. The embedding function allows us to treat MDPs as a special kind of game.

DEFINITION 6. *Let $emb \in \mathcal{M} \to \mathcal{G}$ be the function which for every MDP $M = \langle S, s_i, T, L \rangle$ yields a game $G = \langle S, s_i, \hat{T}, L, L \rangle$ such that $\hat{T}(s) = \{T(s)\}$ for every $s \in S$.*

Embedded MDPs are exact representations of MDPs in $\mathcal{G}$. Intuitively, we ascribe all of M's non-determinism to player 2. That is, player 2 strategies Σ_G^2 have a one-to-one correspondence with Σ_M. Moreover, player 1 has no power (i.e. $|\Sigma_G^1| = 1$).

[§]Below, we formalise this under/over-approximation in the definition of the preorder $\sqsubseteq_p$.



(a) Player 1 choices in $T(s_1) = \{\hat{\Lambda}_a, \hat{\Lambda}_b\}$. (b) Combined player 1 (left) and 2 (right) transitions.

Figure 1: A geometrical interpretation of games.

EXAMPLE 7. *Consider a program with two 8-bit unsigned integers x, y which first initialises y non-deterministically and then assigns to x, uniformly at random, a number between 0 and 255. We model this program with an MDP M and depict $emb(M)$ in Fig. 3(a).*

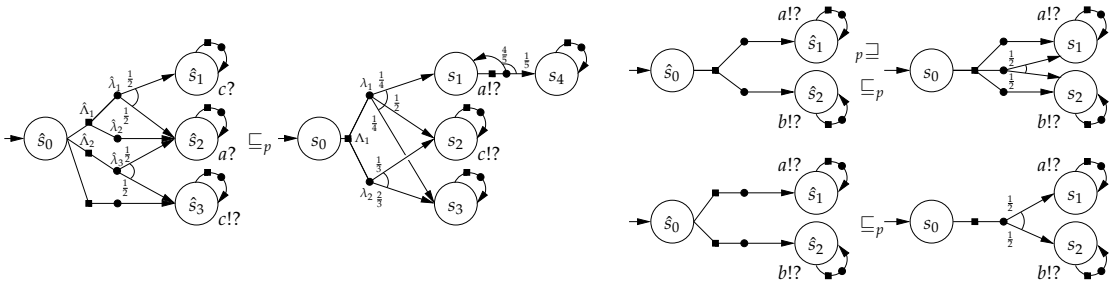
Combined player 1 & 2 transitions Prior to introducing our next component — the abstraction preorder $\sqsubseteq_p$ — we need to introduce *combined transitions*. Combined transitions will enable the preorder take into account that players can make *probabilistic* choices. Combined transitions are well understood for MDPs but are less well-known in games. To explain combined transitions observe that we can interpret probability distributions $\lambda \in \mathbb{D}S$ geometrically as points on a plane in $|S|$-dimensional Euclidean space [27]. Hence, we can interpret the choices available to player 2 and player 1 as a *set of points* and a *set of set of points*, respectively. We illustrate in Fig. 1(a) the choice $T(s_1) = \{\Lambda_a, \Lambda_b\} = \{\{\lambda_1, \lambda_2\}, \{\lambda_3, \lambda_4, \lambda_5\}\}$ available to player 1 in a state s_1 (over a state space $\{s_1,s_2,s_3\}$).

In the player 2 state $\Lambda_a \in \mathbb{P}\mathbb{D}S$ player 2 can make a *probabilistic* choice over probabilistic states $\{\lambda_1, \lambda_2\}$. Hence, it is appropriate to think of Λ_a as defining the hull of a convex shape from which player 2 can draw any probabilistic state (see Fig. 1(b)). To formalise this, akin to [29], we introduce *combined player 2 transitions*. A combined player 2 transition is a move from a player 2 state $\Lambda \in \mathbb{P}\mathbb{D}S$ to a probabilistic state $\lambda \in \mathbb{D}S$, denoted $\Lambda \rightarrow_1^C \lambda$, iff for some $\sum_i w_i \cdot [\lambda_i] \in \mathbb{D}\Lambda$ we have $\lambda = \sum_i w_i \cdot \lambda_i$.

Because player 2 choices are interpreted as convex shapes, the choice available to player 1 in s_1 is a set of convex shapes $T(s_1) = \{\Lambda_a, \Lambda_b\}$. Player 1 can also take any weighted combination of these convex shapes (see Fig. 1(b)). We extend the existing notion of combined transitions over sets of distributions from [29] to combined transitions over sets of sets of distributions as follows: a *combined player 1 transition* is a move from a player 1 state $s \in S$ to a player 2 state $\Lambda \in \mathbb{P}\mathbb{D}S$, denoted $s \rightarrow_1^C \Lambda$, if and only if for some $\sum_i w_i \cdot [\Lambda_i] \in \mathbb{D}(T(s))$ we have $\Lambda = \{\sum_i w_i \cdot \lambda_i \mid \lambda_i \in \Lambda_i \text{ for all } i\}$.

The abstraction preorder ($\sqsubseteq_p$) We can now define the *abstraction preorder* — a relation $\sqsubseteq_p \subseteq \mathcal{G} \times \mathcal{G}$ over games. Intuitively, this preorder defines a notion of precision in $\mathcal{G}$; that is, $\hat{G} \sqsubseteq_p G$ has the meaning that $\hat{G}$ is *less precise* (i.e. *an abstraction of*) G. We can therefore employ the embedding function to define when a game $\hat{G}$ abstracts an MDP M (i.e. when $\hat{G} \sqsubseteq_p emb(M)$).

We define $\sqsubseteq_p$ through a new notion of simulation over games. We consider simulations over disjoint unions $\hat{G} \oplus G$ of games (defined in the obvious way):



(a) Game $\hat{G}$ and embedding $emb(M)$ s.t. $\hat{G} \sqsubseteq_p emb(M)$. (b) Pairs of games $\hat{G}$ and G in $\sqsubseteq_p$.

Figure 2: Games illustrating various points in the paper.

DEFINITION 8. *Let $G = \langle S, s_i, T, L^!, L^? \rangle$ be a game and let $R \subseteq S \times S$ be a relation on S. We call R a* strong probabilistic game-simulation *iff for all s' R s the following conditions hold:*

(i) $L^!(s', a) \Rightarrow L^!(s, a)$ *for all* $a \in \text{AP}$
(ii) $L^?(s', a) \Leftarrow L^?(s, a)$ *for all* $a \in \text{AP}$
(iii) $\forall s \rightarrow_1 \Lambda : \exists s' \rightarrow_1^C \Lambda' : \forall \Lambda' \rightarrow_2 \lambda' : \exists \Lambda \rightarrow_2^C \lambda : \lambda' \, R_{\mathbb{D}} \, \lambda$
(iv) $\forall s \rightarrow_1 \Lambda : \exists s' \rightarrow_1^C \Lambda' : \forall \Lambda \rightarrow_2 \lambda : \exists \Lambda' \rightarrow_2^C \lambda' : \lambda' \, R_{\mathbb{D}} \, \lambda$

Moreover, for games $\hat{G} = \langle \hat{S}, \hat{s}_i, \hat{T}, \hat{L}^!, \hat{L}^? \rangle$ and $G = \langle S, s_i, T, L^!, L^? \rangle$ we let $\hat{G} \sqsubseteq_p G$ iff the largest[¶] *strong probabilistic game-simulation R on $\hat{G} \oplus G$ includes $\hat{s}_i \, R \, s_i$.*

Intuitively, the meaning of $\hat{s} \, R \, s$ is that $\hat{s}$ abstracts s. Conditions (i) and (ii) ensure that the labelling in $\hat{s}$ soundly approximates that of s. The innermost quantifier pair in (iii) formally defines under-approximation of player 2 states: i.e. $\hat{\Lambda} \in \mathbb{PD}\hat{S}$ under-approximates $\Lambda \in \mathbb{PD}S$ iff all transitions $\hat{\Lambda} \rightarrow_2 \hat{\lambda}$ can be simulated by a combined player 2 transition $\Lambda \rightarrow_2^C \lambda$. That is, for these player 2 states, player 2 in G is more powerful than player 2 in $\hat{G}$. The innermost quantifier pair of (iv) defines over-approximation analogously.

Recall that player 1 transitions $s \rightarrow_1 \Lambda$ are such that Λ represents an under or over-approximation of non-determinism in an MDP state that s abstracts. As $\hat{s}$ abstracts all MDP states that s abstracts, player 1 in $\hat{s}$ must both under and over-approximate all player 1 transitions $s \rightarrow_1 \Lambda$ with some combined player 1 move $\hat{s} \rightarrow_1^C \hat{\Lambda}$. This under/over-approximation is realised by the outermost quantifier pair of (iii) and (iv), respectively.

EXAMPLE 9. *Consider games $\hat{G}$ and $emb(M)$ in Fig. 2(a). The largest strong prob. game-simulation R over $\hat{G} \oplus emb(M)$ trivially includes $\langle \hat{s}_1, s_2 \rangle$, $\langle \hat{s}_1, s_3 \rangle$, $\langle \hat{s}_2, s_1 \rangle$, $\langle \hat{s}_2, s_3 \rangle$, $\langle \hat{s}_2, s_4 \rangle$ and $\langle \hat{s}_3, s_2 \rangle$. To see $\hat{s}_0 \, R \, s_0$, i.e. $\hat{G} \sqsubseteq_p emb(M)$, observe that (iii) for $\Lambda_1 \in T(s_0)$ is satisfied by $\hat{\Lambda}_2 \in \hat{T}(\hat{s}_0)$ as $\hat{\lambda}_3 \, R_{\mathbb{D}} \, \lambda_1$ and (iv) is satisfied by $\hat{\Lambda}_1 \in \hat{T}(\hat{s}_0)$ as $\hat{\lambda}_1 \, R_{\mathbb{D}} \, \lambda_1$ and $\frac{2}{3} \cdot \hat{\lambda}_1 + \frac{1}{3} \cdot \hat{\lambda}_2 \, R_{\mathbb{D}} \, \lambda_2$.*

Generality of $\sqsubseteq_p$ Intuitively, in [24], the non-determinism in each MDP state that $\hat{s}$ abstracts is exactly approximated (i.e. both under and over-approximated) by a normal player 1 transition $\hat{s} \rightarrow_1 \hat{\Lambda}$. Our main ability to lose precision arises from the ability to under/over-approximate $T(s')$ with separate player 1 transitions (in combination with the use of *combined* transitions). We illustrate the use of combined transitions with two examples.

[¶]The *largest* strong game-simulation in a game is the union of all its strong game-simulations.

Firstly, the use of combined transitions allows us to abstract probabilistic choice with player 1 non-determinism (see Fig. 2(b) (bottom)). As it is expensive to abstract probabilistic choice it may be advantageous to initially abstract probabilistic behaviour in this way.

Secondly, observe the equivalence classes of $\sqsubseteq_p$ define a notion of equivalence in $\mathcal{G}$ (i.e. G and G' are equivalent iff $G \sqsubseteq_p G'$ and $G' \sqsubseteq_p G$). Fig. 2(b) (top) illustrates that through this equivalence we can consider more compact representations without losing any precision.

Abstract PCTL semantics ($\models^!, \models^?$) The final component of our abstraction framework is a four-valued abstract PCTL semantics $\models^!, \models^? \subseteq \mathcal{G} \times \Phi$ for games. Informally, $G \models^! \phi$ (resp. $G \models^? \phi$) holds only if all MDPs that G abstracts *must* (resp. *may*) satisfy ϕ.

DEFINITION 10. *Let $G = \langle S, s_i, T, L^!, L^? \rangle$ be a game and let $\phi \in \Phi$ be a PCTL formula. We define must/may relations for states $\models^!, \models^? \subseteq S \times \Phi$ and plays $\models^!, \models^? \subseteq \Pi^\infty_G(S) \times \Psi$ as follows (letting $* \in \{!, ?\}$, $\neg! = ?$ and $\neg? = !$):*

$$
\begin{array}{lcl}
\pi \models^* \mathsf{X}\phi & \iff & s_1 \models^* \phi \\
\pi \models^* \phi_1 \mathsf{U}^{\trianglelefteq k} \phi_2 & \iff & \exists i \trianglelefteq k : \ (s_i \models^* \phi_2 \wedge (\forall j < i : s_j \models^* \phi_1) \\
s \models^* a & \iff & L^*(s, a) \\
s \models^* \neg\phi & \iff & s \not\models^{\neg *} \phi \\
s \models^* \phi_1 \vee \phi_2 & \iff & (s \models^* \phi_1 \ \ or \ \ s \models^* \phi_2) \\
s \models^! \mathsf{P}_{\triangleright p}\langle\psi\rangle & \iff & \inf_{\sigma_1} \ \inf_{\sigma_2} \ \mathbf{Pr}^s_{G,\sigma_1,\sigma_2}\{\pi \in \Pi^\infty_{G,\sigma_1,\sigma_2}(s) \mid \pi \models^! \psi\} \triangleright p \\
s \models^? \mathsf{P}_{\triangleright p}\langle\psi\rangle & \iff & \sup_{\sigma_1} \inf_{\sigma_2} \ \mathbf{Pr}^s_{G,\sigma_1,\sigma_2}\{\pi \in \Pi^\infty_{G,\sigma_1,\sigma_2}(s) \mid \pi \models^? \psi\} \triangleright p \\
s \models^! \mathsf{P}_{\triangleleft p}\langle\psi\rangle & \iff & \sup_{\sigma_1} \sup_{\sigma_2} \ \mathbf{Pr}^s_{G,\sigma_1,\sigma_2}\{\pi \in \Pi^\infty_{G,\sigma_1,\sigma_2}(s) \mid \pi \models^? \psi\} \triangleleft p \\
s \models^? \mathsf{P}_{\triangleleft p}\langle\psi\rangle & \iff & \inf_{\sigma_1} \ \sup_{\sigma_2} \ \mathbf{Pr}^s_{G,\sigma_1,\sigma_2}\{\pi \in \Pi^\infty_{G,\sigma_1,\sigma_2}(s) \mid \pi \models^! \psi\} \triangleleft p
\end{array}
$$

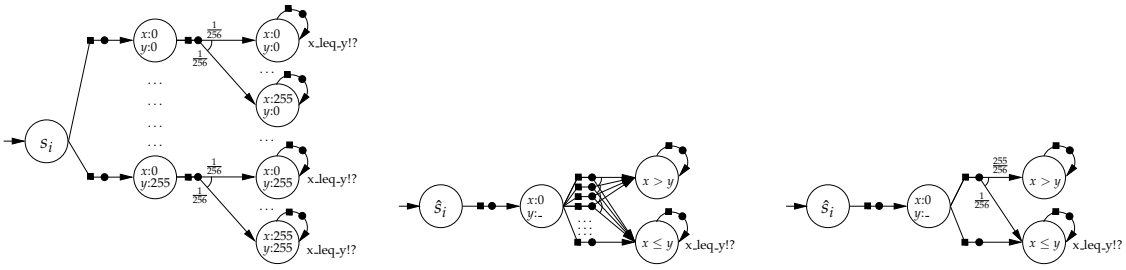
with $\pi = s_0; \Lambda_0; \lambda_0; s_1 \ldots, \triangleright \in \{>, \geq\}$, $\triangleleft \in \{<, \leq\}$, $\sigma_i \in \Sigma^i_G$. Moreover, $G \models^* \phi$ iff $s_i \models^* \phi$.

The four-valued semantics of propositional and temporal operators is standard. The only non-standard semantics is that of the probabilistic operator $\mathsf{P}_{\bowtie p}\langle\psi\rangle$. Recall $\mathsf{P}_{\bowtie p}\langle\psi\rangle$ holds for an MDP if *all* MDP-schedulings meet the threshold $\bowtie p$. As player 2 represents MDP non-determinism, for a lower threshold $\triangleright p$ (upper threshold $\triangleleft p$) we take the infimum (supremum) over player 2 strategies, regardless of whether we are evaluating in the must or may modality. In contrast, whether we take the infimum or supremum over player 1 strategies depends only on the modality. That is, if we are evaluating in the must modality we quantify pessimistically over player 1 strategies (inf. for $\triangleright p$, sup. for $\triangleleft p$) and in the may modality we quantify optimistically over player 1 strategies (sup. for $\triangleright p$, inf. for $\triangleleft p$). For lower thresholds, the modality in which we evaluate path properties corresponds to the modality in which we are evaluating — this has to be inverted for upper thresholds $\triangleleft p$.

Soundness properties Before we can verify and refute judgments over MDPs via games, we need to show our components satisfy the following properties:

LEMMA 11. *For all MDPs M and $\phi \in \Phi$ we have $M \models \phi \Leftrightarrow emb(M) \models^! \phi \Leftrightarrow emb(M) \models^? \phi$.*

PROOF. Follows directly from the fact that embedded MDPs have two-valued propositional labelling, i.e. $L^! = L^?$, and one trivial player 1 strategy, i.e. $|\Sigma^1_G| = 1$. ∎



(a) (b) (c)

Figure 3: The MDP (a) and the two game abstractions (b) and (c) from Example 14.

THEOREM 12. *For all games $\hat{G}, G$ such that $\hat{G} \sqsubseteq_p G$ and all PCTL properties $\phi \in \Phi$ we have $(\hat{G} \models^! \phi \Rightarrow G \models^! \phi)$ and $(\hat{G} \not\models^? \phi \Rightarrow G \not\models^? \phi)$.*

PROOF. We only sketch the structure of the proof here. Let $\hat{s}$ be any state of $\hat{G}$ and let s be any state of G, respectively. The proof shows that $\hat{s} \models^! \phi \Rightarrow s \models^! \phi$ and, dually, that $s \models^? \phi \Rightarrow \hat{s} \models^? \phi$. The main complexity of the proof is due to properties $\mathsf{P}_{\bowtie p}\langle\psi\rangle$.

Intuitively, due to (iii) of Def. 8, for any player 1 strategy in G, player 1 in $\hat{G}$ can choose a strategy under which it knows player 2 must be less powerful in $\hat{G}$ than in G, which ensures $s \models^? \mathsf{P}_{\bowtie p}\langle\psi\rangle \Rightarrow \hat{s} \models^? \mathsf{P}_{\bowtie p}\langle\psi\rangle$. Dually, (iv) of Def. 8 guarantees that for every player 1 strategy in G, player 1 in $\hat{G}$ can choose a strategy under which it knows player 2 is more powerful in $\hat{G}$ than it is in G, which ensures $\hat{s} \models^! \mathsf{P}_{\bowtie p}\langle\psi\rangle \Rightarrow s \models^! \mathsf{P}_{\bowtie p}\langle\psi\rangle$. ∎

Lem. 11 ensures consistency across the two representations of MDPs whereas Th. 12 ensures that any property that is $\models^!$-satisfied (not $\models^?$-satisfied) by a game $\hat{G}$ is also $\models^!$-satisfied (not $\models^?$-satisfied) by any game that is less abstract than $\hat{G}$ — including MDP embeddings.

Verification & refutation via games We can now *verify* the judgment $M \models \phi$ by constructing a game G that abstracts M (i.e. $G \sqsubseteq_p \mathit{emb}(M)$) and that !-satisfies ϕ (i.e. $G \models^! \phi$). It is easy to see this: by Th. 12 $\mathit{emb}(M) \models^! \phi$ and by Lem. 11 this is equivalent to $M \models \phi$. Analogously, we can *refute* $\mathcal{M} \models \phi$ by finding a game G such that $G \sqsubseteq_p \mathit{emb}(M)$ and $G \not\models^? \phi$ (i.e. by Th. 12 $\mathit{emb}(M) \not\models^? \phi$; by Lem. 11 $M \not\models \phi$).

For some games G it may be that both $G \not\models^! \phi$ and $G \models^? \phi$. In this case we can neither verify nor refute $M \models \phi$ and we need to consider *refining* G.

EXAMPLE 13. *For $\hat{G}$ and $\mathit{emb}(M)$ of Fig. 2(a), as $\hat{G} \sqsubseteq_p \mathit{emb}(M)$ and $\hat{G} \models^! \neg\mathsf{P}_{>0.5}\langle\mathsf{X}a\rangle$, using Lem. 11 and Th. 12, we have verified $M \models \neg\mathsf{P}_{>0.5}\langle\mathsf{X}a\rangle$: some scheduling of M satisfies $\mathsf{X}a$ with probability of at most 0.5. However, as $\hat{G} \not\models^! \mathsf{P}_{\leq 0.5}\langle\mathsf{X}a\rangle$ and $\hat{G} \models^? \mathsf{P}_{\leq 0.5}\langle\mathsf{X}a\rangle$ we can neither verify nor refute via $\hat{G}$ whether this threshold holds for* all *schedulings of M.*

Finally, once we find a game G via which we, say, verify $M \models \phi$, we can claim this judgment holds and use G as a certificate to our claim. To confirm our claim one would have to perform the checks $G \sqsubseteq_p \mathit{emb}(M)$ and $G \models^! \phi$. Analogously, we can use games as refutation certificates. We demonstrate our framework with a final motivating example:

EXAMPLE 14. *Reconsider the program and embedded MDP $emb(M)$ of Example 7. Suppose we wish to check the program stops with $x \leq y$ with a probability greater than* 0.003 *(i.e. $M \models \mathsf{P}_{>0.003}\langle true\, \mathsf{U}^{\leq\infty}\, x\_leq\_y\rangle$). We consider a partition which joins all states before the probabilistic assignment and which, after the assignment, divides states according to the predicate $x \leq y$. Fig. 3(b) depicts the (optimal) game G constructed with the techniques in [24]. We can verify our judgment with this game; however, G has many transitions because the probability of $x \leq y$ is different for each initial value of y. With the framework presented in this paper we can verify the judgment with the game $\hat{G}$ depicted in Fig. 3(c) — a much more compact game defined over the same partition.*

4 Discussion and conclusions

We motivated the need for an abstraction-based framework for the verification and refutation of PCTL specifications of MDPs. We constructed such a framework by taking the game abstractions from [24], developing an abstraction preorder and abstract PCTL semantics for these games, and proving these components meet certain soundness properties.

This preorder enables us to lose precision — even for a fixed partitioning of MDP states. This allows us to verify and refute properties of MDPs with more compact games. In many cases losing precision is essential. For example, when abstracting program statements under predicates that contain non-linear arithmetic, computing the optimal abstraction for a set of predicates is very inefficient. Through our abstraction preorder, by losing precision, we may be able to obtain abstractions more efficiently in such cases — for example by considering incrementally precise abstractions over a fixed partition, akin to software model checkers. However, to automate this we need to augment our framework with a refinement procedure. Although procedures exists to refine optimal partition abstractions of games [21, 22], we have not yet adapted these procedures to deal with the additional causes of imprecision that occur in our framework.

The game abstractions considered in practice (in, e.g., [22]) are known to be abstractions by construction — there is no need to check the conditions of $\sqsubseteq_p$. Nevertheless, the computational complexity of deciding $\sqsubseteq_p$ and $\models^!, \models^?$ are still of interest. In a preliminary unpublished version of this paper [19] we show $\sqsubseteq_p$ without combined player 1 transitions is decidable in P and $\models^!, \models^?$ are decidable in NP $\cap$ co-NP.

There is potential to improve precision of our framework as follows: one could equip games with separate must/may transition functions to distinguish under-approximating from over-approximating player 2 states (c.f. [26, 10]). That is, in (iii) of Def. 8 and in the must evaluations of Def. 10 one would use the must transitions and in (iv) of Def. 8 and in the may evaluations of Def. 10 one would use the may transitions. This change would not increase precision for partition abstractions.

Related work In recent papers, many orthogonal challenges related to game abstractions have been addressed: in [21, 22] it is outlined how optimal game abstractions can be constructed from language-level descriptions via SAT; in [20, 22] it is explained how good partition abstractions can be found using automated abstraction refinement.

For probabilistic systems, abstraction frameworks include probabilistic extensions of existential abstraction [11, 16], where MDP are abstracted by MDPs again through the strong

(probabilistic) simulation preorder of [18, 29]. Other frameworks abstract probabilities with intervals (e.g. [18, 17, 13, 4]). When considering these frameworks as an abstraction framework for MDPs we observe the abstract models are unable to distinguish non-determinism of the concrete MDP from the non-determinism that arises through abstraction. As a result, refuting a property $\mathsf{P}_{\leq p}\langle\psi\rangle$, i.e. showing there exists a strategy that exceeds p, can only be achieved by establishing *all* strategies exceed p. As this may not be true for some MDPs the property may not be refutable with these abstractions. By separating the two kinds of non-determinism, our framework does not suffer from the same problem. Note that our argument relies on the presence of non-determinism and does not occur when considering the abstraction schemes in, e.g., [29, 17, 13] as abstraction frameworks for Markov chains (MCs). In fact, our preorder is essentially the strong probabilistic simulation of [29] when restricted to MCs. Finally, we mention [4] in which more efficiently checkable abstractions are obtained by eliminating non-determinism (i.e. MDPs are abstracted by MCs).

For non-probabilistic systems, sound verification and refutation of temporal logics have mostly been developed in a (sometimes implicit) three-valued setting [26, 10]. Our results in the probabilistic setting are, notably, informed by work on modal/mixed transitions system [26, 10] and three-valued abstraction of games [12].

References

[1] T. Ball, T. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. *SIGPLAN Notices*, 36(5):203–213, 2001.

[2] N. D. Belnap. A useful four-valued logic. In *Modern Uses of Multiple-valued Logics*, pp. 8–37, 1977.

[3] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. of FSTTC '95*, vol. 1026 of *LNCS*, pp. 499–513, 1995.

[4] R. Chadha, M. Viswanathan, and R. Viswanathan. Least upper bounds for probability measures and their applications to abstractions. In *Proc. of CONCUR '08*, vol. 5201 of *LNCS*, pp. 264–278, 2008.

[5] E. Clarke and E. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop*, vol. 131 of *LNCS*, 1981.

[6] E. Clarke, D. Kroening, N. Sharygina, and K. Yorav. SATABS: SAT-based predicate abstraction for ANSI-C. In *Proc. of TACAS '05*, vol. 3440 of *LNCS*, pp. 570–574, 2005.

[7] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. of CAV '00*, vol. 1855 of *LNCS*, pp. 154–169, 2000.

[8] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM TOPLAS*, 16(5):1512–1542, 1994.

[9] A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.

[10] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM TOPLAS*, 19(2):253–291, 1997.

[11] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen. Reduction and refinement strategies for probabilistic systems. In *Proc. of PAPM-PROBMIV '02*, vol. 2399 of *LNCS*, pp. 57–76, 2002.

[12] L. de Alfaro, P. Godefroid, and R. Jagadeesan. Three-valued abstractions of games: Uncertainty, but with precision. In *Proc. of LICS '04*, pp. 170–179, 2004.
[13] H. Fecher, M. Leucker, and V. Wolf. Don't know in probabilistic systems. In *Proc. of SPIN '06*, vol. 3925 of *LNCS*, pp. 71–88, 2006.
[14] T. Han and J. Katoen. Counterexamples in probabilistic model checking. In *Proc. of TACAS '07*, vol. 4424 of *LNCS*, pp. 72–86, 2007.
[15] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.
[16] H. Hermanns, B. Wachter, and L. Zhang. Probabilistic CEGAR. In *Proc. of CAV '08*, vol. 5123 of *LNCS*, pp. 162–175, 2008.
[17] M. Huth. On finite-state approximants for probabilistic computation tree logic. *TCS*, 346(1):113–134, 2005.
[18] B. Jonsson and K. G. Larsen. Specification and refinement of probabilistic processes. In *Proc. of LICS '91*, pp. 266–277, 1991.
[19] M. Kattenbelt and M. Huth. Abstraction framework for Markov decision processes and PCTL via games. TR RR-09-01, OUCL, 2009.
[20] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. A game-based abstraction-refinement framework for Markov decision processes. TR RR-08-06, OUCL, 2008.
[21] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. Game-based probabilistic predicate abstraction in PRISM. In *Proc. of QAPL '08*, vol. 220 of *ENTCS*, pp. 5–21, 2008.
[22] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. Abstraction refinement for probabilistic programs. In *Proc. of VMCAI '09*, vol. 5403 of *LNCS*, pp. 182–197, 2009.
[23] J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Denumerable Markov Chains*. 2 edition, 1976.
[24] M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for Markov decision processes. In *Proc. of QEST '06*, pp. 157–166, 2006.
[25] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic games for verification of probabilistic timed automata. In *Proc. of FORMATS '09*, pp. 212–227, 2009.
[26] K. G. Larsen and B. Thomsen. A modal process logic. In *Proc. of LICS '88*, pp. 203–210, 1988.
[27] A. K. McIver, C. C. Morgan, and C. Gonzalia. Proofs and refutations for probabilistic refinement. In *Proc. of FM '08*, vol. 5014 of *LNCS*, pp. 100–115, 2008.
[28] J. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. of Symposium on Programming '05*, vol. 137 of *LNCS*, pp. 337–351, 1982.
[29] R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. In *Proc. of CONCUR '94*, LNCS, pp. 481–496, 1994.
[30] M. Vardi. Verification of probabilistic concurrent finite-state programs. In *Proc. of FOCS '85)*, pp. 327–338, 1985.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Approximating Fault-Tolerant Group-Steiner Problems

Rohit Khandekar[1], Guy Kortsarz[2], and Zeev Nutov[3]

[1]IBM T.J.Watson Research Center. rohitk@us.ibm.com

[2]Rutgers University, Camden. guyk@camden.rutgers.edu

[3]The Open University of Israel. nutov@openu.ac.il

ABSTRACT. In this paper, we initiate the study of designing approximation algorithms for Fault-Tolerant Group-Steiner (FTGS) problems. The motivation is to protect the well-studied group-Steiner networks from edge or vertex failures. In Fault-Tolerant Group-Steiner problems, we are given a graph with edge- (or vertex-) costs, a root vertex, and a collection of subsets of vertices called groups. The objective is to find a minimum-cost subgraph that has two edge- (or vertex-) disjoint paths from each group to the root. We present approximation algorithms and hardness results for several variants of this basic problem, e.g., edge-costs vs. vertex-costs, edge-connectivity vs. vertex-connectivity, and 2-connecting from each group a single vertex vs. many vertices. Main contributions of our paper include the introduction of very general structural lemmas on connectivity and a charging scheme that may find more applications in the future. Our algorithmic results are supplemented by inapproximability results, which are tight in some cases.
Our algorithms employ a variety of techniques. For the edge-connectivity variant, we use a primal-dual based algorithm for covering an *uncrossable* set-family, while for the vertex-connectivity version, we prove a new graph-theoretic lemma that shows equivalence between obtaining two vertex-disjoint paths from two vertices and 2-connecting a carefully chosen single vertex. To handle large group-sizes, we use a p-Steiner tree algorithm to identify the "correct" pair of terminals from each group to be connected to the root. We also use a non-trivial charging scheme to improve the approximation ratio for the most general problem we consider.

1 Introduction

The fault-tolerant network design problems are well-studied in the theory of combinatorial optimization and approximation algorithms. The basic goal in these problems is to design a minimum-cost network that satisfies some prescribed connectivity requirements. Higher connectivity requirements are usually enforced for fault-tolerance — in order to protect connectivity in the solution against edge or vertex failures. A well-studied fault-tolerant network design problem is the Steiner Network problem. In this problem, we are given edge- (or vertex-) connectivity requirement r_{ij} between every pair $\{i, j\}$ of vertices, and the goal is to design a network with at least r_{ij} edge- or vertex-disjoint paths between i and j for each pair $\{i, j\}$.

In this paper, we study fault-tolerant versions of the Group-Steiner problem. In this problem, we are given a (undirected or directed) graph $G = (V, E)$ with edge- or vertex-costs, a root vertex $r \in V$, and a collection of of subsets (groups) $\mathcal{S} = \{S_1, \ldots, S_q\}$ of $V \setminus \{r\}$.

© Khandekar, Kortsarz, and Nutov; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 263–274
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2324

The objective is to find a minimum cost subgraph H of G that contains two edge- or vertex-disjoint paths from each group S_i to the root. This problem models the flexibility, often arising in problems in VLSI design [25], in connecting any vertices from a given group as well as the fault-tolerance which requires the solution to be robust under edge- or vertex-failures.

1.1 Previous work on fault-tolerance problems

Fault tolerance problems have been extensively studied in Combinatorial Optimization and Approximation Algorithms. Consider for example the well known Steiner Network problem. Given an undirected graph $G = (V, E)$ with edge-costs $\{c_e \mid e \in E\}$, and requirement r_{ij} for every pair of vertices $i, j \in V$, the goal is to find a minimum cost subgraph H of G that contains at least r_{ij} edge-disjoint paths between i and j, for all i, j. The network H is fault tolerant in the sense that a pair i, j can sustain $r_{ij} - 1$ link failures and still be connected. The best ratio known for this problem is 2 due to Jain [14].

The internally-disjoint path version of the Steiner Network problem is very hard to approximate [16, 21, 9, 18]. The currently best known ratio for this problem is $O(k^3 \log n)$ for edge-costs due to Chuzhoy and Khanna [8] and $O(k^4 \log^2 n)$ for vertex-costs [20]. The rooted version, where a root s is given and $r_{ij} = 0$ for all pairs i, j so that $i \neq s$ and $j \neq s$, has gotten a significant attention recently on the high level [3, 5, 9, 23, 8, 20]. This problem is at least as hard to approximate as Directed Steiner Tree [18]. The best approximation ratio known for this problem for general rooted demands is $O(k^2)$ for edge-costs and $O(k^2 \log n)$ for vertex-costs [20], where $k = \max r_{ij}$. We mention that prior to the work of [20], a randomized approximation algorithm with ratio $k^{O(k^2)} \log^4 n$ was developed by Chakraborty, Chuzhoy, & Khanna [3], then improved to $k^{O(k)} \log n$ by Chekuri & Korula [5], and then improved to $O(k^2 \log n)$ by Chuzhoy & Khanna [9, 8] and [23]. Note that k can be as large as $\Omega(n)$.

A particular case of fault tolerance problems with 2 disjoint paths has also received an attention on the high level [1, 19, 5]. Lau et al. [19] presented an $O(\log^2 n)$-approximation algorithm for the problem of finding a minimum-cost 2-edge connected subgraph with *at least* k vertices. This problem can be seen as a fault-tolerant generalization of the k-MST problem which requires to find a minimum spanning tree on k vertices. The best known approximation ratio for the k-MST problem is 2 [11]. Chekuri and Korula [6] presented an $O(\log^2 n)$-approximation for the problem of finding a minimum-cost 2-vertex connected subgraph with at least k terminals. In [1, 5], the fault-tolerance version of the Buy-at-Bulk problem was studied, where two edge-disjoint paths are required to be included from every terminal to the root.

In the same spirit, in this paper we consider a generalization of the Group-Steiner Tree problem. In the usual Group-Steiner Tree problem, we are given a graph $G = (V, E)$, edge costs $\{c_e : e \in E\}$, a root $r \in V$, and a collection of subsets (groups) $\mathcal{S} = \{S_1, \ldots, S_q\}$ of $V \setminus \{r\}$. The objective is to find a minimum cost subtree T of G that contains r and at least one vertex from each group $S_i \in \mathcal{S}$. The best known approximation ratio for this problem is $O(\log^3 n)$ [12]. The Fault-Tolerant Group-Steiner Tree problem, on the other hand, requires obtaining two (edge- or vertex-) disjoint paths between each group to the root. We are not aware of any previous work on Fault-Tolerant Group-Steiner problems.

1.2 Problem variants studied in this paper

One can define several variants of the Fault-Tolerant Group-Steiner problem, based on whether we desire 2-edge- or 2-vertex-connectivity, whether we have edge- or vertex-costs, whether we wish to 2-connect to the root a single vertex from each group or two distinct vertices from each group, etc. Below we formally define all the variants studied in this paper. Two paths are said to be *internally-disjoint* if they are vertex-disjoint except for their end-points. Each of the following problems takes a graph $G = (V, E)$ on n vertices with edge-costs $\{c_e \mid e \in E\}$ (or with node-costs $\{c_v \mid v \in V\}$), a root $r \in V$, and groups $\mathcal{S} = \{S_1, \ldots, S_q\}$ as input, and is required to compute a min-cost subgraph H of G with at least two edge/vertex-disjoint paths from each group S_i to the root, so that the endnodes of these paths are distinct. Unless stated otherwise, we consider the edge-cost version and assume that G is undirected. We also assume that the groups $S_1, \ldots, S_q$ are pairwise disjoint.* The vertices in the groups S_i are called *terminals*. We add the prefix EC- for edge-connectivity and the prefix VC- for vertex-connectivity. We add the suffix "-k" after the name of the problem if the instances are restricted to satisfy $|S_i| \leq k$ for all $i = 1, \ldots, q$.

- EC-FTGS: Here for every $i = 1, \ldots, q$, H should contain at least two edge-disjoint $S_i r$-paths; the end-points in S_i of these paths should be distinct.
- VC-FTGS: The same as EC-FTGS, except that the paths should be internally-disjoint.
- EC-FTGS-k and VC-FTGS-k: These are EC-FTGS and VC-FTGS, respectively, restricted to instances with $|S_i| \leq k$ for all $i = 1, \ldots, q$.

In the edge-connectivity case, for both edge and vertex-costs, the version when the end-points in S_i of the two $S_i r$-paths may or may not be distinct, is easily reduced to EC-FTGS as follows. For every terminal s, add a new vertex s' of cost 0 connected to s with an edge of cost 0, and add s' to every group $S \in \mathcal{S}$ that contains s. After this transformation, we can assume, without loss of generality, that the two edge-disjoint paths from each group start from *distinct* terminals in that group. This may double the number of vertices, and cause a constant loss in approximation ratios that depend on n.

We also consider the version when we insist that a single vertex from each group must be 2-edge-connected to the root. Namely, for every $i = 1, \ldots, q$ there should exist a vertex $v_i \in S_i$ such that H contains 2 edge-disjoint rv_i-paths. We call this version Single EC-FTGS.

1.3 The difficulty in approximating Fault-Tolerant problems

When two disjoint paths from every group to the root are required, we cannot use the standard transformation to Bartal trees [2, 10] as done in the approximation of the Group Steiner problem [12]. This is so because disjoint paths from a group to the root in a Bartal tree do not necessarily correspond to disjoint paths in the original graph.

We now give a strong evidence that an approximation that is polylogarithmic in n for either EC-FTGS-k and VC-FTGS-k may be very hard to obtain as it implies solving a long standing open problem. Note that we can reduce the usual Group Steiner problem to EC-FTGS-k or VC-FTGS-k problems by adding a new vertex, which is connected to the root

*This can be typically assumed by making multiple copies of the vertices in multiple groups and adding zero-cost edges connecting the different copies. This reduction, however, increases the number of vertices in the graph, thus possibly affecting the approximation ratio.

Problem	Edge-Connectivity (EC)	Vertex-Connectivity (VC)
FTGS-2	3.55, Vertex-Cover-hard $O(\log n)$, $\Omega(\log n)$-hard (vertex-costs)	$O(\log^2 n)$
FTGS-k	$O(k \log^2 n)$	$O(k \log^2 n)$
FTGS	$O(\sqrt{n} \log n)$ Group Steiner Tree-hard	$O(\sqrt{n} \log n)$ Group Steiner Tree-hard
Single FTGS	Label Cover-hard (directed)	Label Cover-hard (directed)

Table 1: Approximation ratios and hardness results for FTGS variants. The extra assumptions, if any, are given in the parentheses.

by a zero-cost edge, to each group. Since we get one path for "free", any solution to EC-FTGS-k and VC-FTGS-k corresponds to a solution of the usual Group Steiner problem and vice-versa.

Now since an algorithm for EC-FTGS-k or VC-FTGS-k cannot use Bartal trees, it must solve the Group Steiner problem as well **without** using Bartal trees. The question if Group Steiner problem can be approximated within a polylogarithmic ratio without first reducing the graph into trees is a long standing open question and seemingly a very hard one.

To the best of our knowledge, the best known ratio for Group Steiner problem without using Bartal trees is $O(n^\epsilon)$ for any constant $\epsilon > 0$, with running time $n^{f(1/\epsilon)}$. The *recursive greedy* technique [27, 17, 4], used in this algorithm, is a complex greedy approach with quite delicate analysis that seems completely inappropriate for the requirement of two disjoint paths. Thus even an n^ϵ approximation for every universal constant ϵ seems to be a quite significant challenge for our problems in the current state of knowledge and techniques.

In our opinion, this is a strong evidence that it is quite a challenge to get polylogarithmic ratios for either EC-FTGS-k or VC-FTGS-k in polynomial time.

Remark: The above simple reduction shows that the $\Omega(\log^{2-\epsilon} n)$ hardness for the Group Steiner problem applies also for EC-FTGS-k and VC-FTGS-k. However, from the above evidence, EC-FTGS-k and VC-FTGS-k may in fact be much harder to approximate than $\Omega(\log^{2-\epsilon} n)$, and it may be that a polynomial ratio is the best we can hope for.

1.4 Our results

We start with a definition. For two optimization problems $\mathcal{P}_1$ and $\mathcal{P}_2$, we say that $\mathcal{P}_1$ *is* $\mathcal{P}_2$-*hard* if existence of a ρ-approximation algorithm for $\mathcal{P}_1$ implies existence of a ρ-approximation for $\mathcal{P}_2$. Similarly, $\mathcal{P}_1$ is $\Omega(f(n))$-hard if there exists a constant $\epsilon > 0$ so that $\mathcal{P}_1$ admits no $\epsilon f(n)$-approximation algorithm, unless P=NP.

Our main results are summarized in Theorem 1 and in Table 1.

THEOREM 1.

(i) EC-FTGS-2 *admits the following approximation ratios:*
$(2 + \gamma)$ *for edge costs, where* $\gamma < 1.55$ *is the best ratio for the* Steiner Tree *problem,*

and $O(\log n)$ for vertex costs. Moreover, the edge-cost version is Vertex-Cover*-hard and the vertex-cost version is $\Omega(\log n)$-hard.*

(ii) EC-FTGS-k *and* VC-FTGS-k *admit an $O(k \log^2 n)$-approximation algorithm.* EC-FTGS *and* VC-FTGS *admit an $O(\sqrt{n} \log n)$-approximation algorithm.* EC-FTGS *and* VC-FTGS *are* Group Steiner Tree*-hard and thus are $O(\log^{2-\epsilon} n)$-hard for any constant $\epsilon > 0$.*

(iii) *The directed version of* Single EC-FTGS *problem admits no $2^{\log^{1-\epsilon} n}$-approximation for any constant $\epsilon > 0$, unless* NP $\subseteq$ DTIME$(n^{\text{polylog}(n)})$.

The results (i) and (ii) are proved in Sections 2 and 3 respectively. The proof of (iii) is omitted due to lack of space.

2 Proof of Part (i)

We start with some definitions. An edge e is said to *cover* a subset $X \subset V$ of vertices if exactly one end-point of e lies in X. Let $\mathcal{F}$ be a collection of subsets of V. We say that an edge-set E' covers $\mathcal{F}$ if for each $X \in \mathcal{F}$, there is an edge $e \in E'$ that covers X. The Set-Family Edge-Cover problem with edge-cost is to find a minimum-cost collection of edges E' that covers $\mathcal{F}$. In the vertex-cost version, we wish to minimize the total cost of vertices incident to E' that covers $\mathcal{F}$. The family $\mathcal{F}$ may not be given explicitly, but we require that certain queries related to $\mathcal{F}$ can be answered in polynomial time. Specifically, we assume that, in the edge-cost version, for any edge-set E', the inclusion minimal members of $\mathcal{F}$ that are *not* covered by E' can be computed in polynomial time; while, in the vertex-cost version, for any $s, t \in V$, one can compute in polynomial time a min-cost cover of all members of $\mathcal{F}$ that separate s and t.

We call a family $\mathcal{F}$ of sets *uncrossable* if $X \cap Y, X \cup Y \in \mathcal{F}$ or $X \setminus Y, Y \setminus X \in \mathcal{F}$ for any $X, Y \in \mathcal{F}$. Our algorithms for EC-FTGS-2 problem with edge-costs or vertex-costs use the following results, respectively.

THEOREM 2. (Goemans et al. [13]) *The* Set-Family Edge-Cover *problem with edge-costs and with uncrossable set-family $\mathcal{F}$ admits a 2-approximation algorithm.*

THEOREM 3.([22]) *The* Set-Family Edge-Cover *problem with vertex-costs and with uncrossable set-family $\mathcal{F}$ admits an $O(\log n)$-approximation algorithm.*

2.1 Algorithmic results

Since the problem insists that the two edge-disjoint paths from each group must start at distinct terminals in the group, the optimum solution contains a Steiner tree containing all the terminals and the root. Our algorithm first finds a Steiner tree T that connects all the terminals to the root. If we use an α-approximation algorithm for this step ($\alpha < 1.55$ for the edge-costs [26] and $\alpha = O(\log n)$ for the vertex-costs [15]), we get $\text{COST}(T) \leq \alpha \cdot \text{OPT}$ where OPT denotes the cost of the optimum solution.

For $X \subset V$, let $\texttt{deg}_T(X)$ denote the number of edges in T from X to $V \setminus X$. Define an instance of Set-Family Edge-Cover by setting

$$\mathcal{F} = \{X \subseteq V \setminus \{r\} \mid \texttt{deg}_T(X) = 1, S \subseteq X \text{ for some } S \in \mathcal{S}\}.$$

We now present two important observations.

LEMMA 4. *For $I \subseteq E \setminus E(T)$, the set $T \cup I$ is a feasible solution to* EC-FTGS-2 *if, and only if, I covers $\mathcal{F}$.*

PROOF. Note that T has a path from each terminal to the root. Thus by Menger's Theorem, $H = T \cup I$ is a feasible solution to EC-FTGS-2 if, and only if, $\deg_H(X) \geq 2$ for every set $X \subseteq V \setminus \{r\}$ that contains some group $S \in \mathcal{S}$. As $\deg_T(X) \geq 1$ for any $X \subseteq V \setminus \{r\}$ that contains at least one vertex from some group, we obtain that the latter condition is equivalent to $\deg_I(X) \geq 1$ for every $X \in \mathcal{F}$. ∎

LEMMA 5. *The set family $\mathcal{F}$ is uncrossable.*

PROOF. Note that by the definition of $\mathcal{F}$, $X \in \mathcal{F}$ if, and only if, X is a union of a rooted proper subtree of T that contains a group $S \in \mathcal{S}$ and any subset of vertices not in T. Let $X, Y \in \mathcal{F}$. Then $X \cap T, Y \cap T$ are disjoint, or one of them contains the other. In the former case, we have $X \setminus Y, Y \setminus X \in \mathcal{F}$; e.g., $X \setminus Y \in \mathcal{F}$ since $(X \setminus Y) \cap T = X \cap T$, hence $X \setminus Y$ is a union of the subtree contained in X and the vertex subset $X \setminus (T \cup Y)$ disjoint to T. In the latter case, $X \cap Y, X \cup Y \in \mathcal{F}$; e.g., if $X \subseteq Y$, then $X \cap Y$ is a union of the subtree contained in X and some vertices not in T, while $X \cup Y$ is the union of the subtree contained in Y and some vertices not in T. ∎

It is easy to check that for any edge set $I \subset E \setminus E(T)$, the minimal members of the family $\mathcal{F}$ not covered by I can be computed in polynomial time. Moreover, for any $s, t \in V$, a min-cost cover of all members in $\mathcal{F}$ that separate s and t can also be computed in polynomial time. Thus, we can use the algorithms in Theorems 2 and 3 respectively to complete the solutions for the edge- and vertex-cost versions.

2.2 Hardness of approximation results

We now show that EC-EFTGS-2 is Vertex-Cover hard in the case of edge-costs, and that it is Hitting-Set-hard, i.e., $\Omega(\log n)$-hard, in the case of vertex-costs.

Let $J = (V_J, E_J)$ be an instance of Vertex-Cover. Define an instance $\{G = (V, E), \{c_e : e \in E\}, r, \mathcal{S}\}$ of 2-EC-FTGS-2 as follows. Set $V = V_J \cup \{a, r\}$, connect every vertex in V_J to a with an edge of cost 0 and connect a to r with an edge of cost 0. Let T denote the set of these zero-cost edges. Connect each vertex in V_J to r with an edge of cost 1 each. The set $\mathcal{S}$ of pairs is defined by edges of E_J, namely, $\mathcal{S} = \{\{u, v\} \mid (u, v) \in E_J\}$. Note that the optimum solution, without loss of generality, picks all the edges in T. It is now easy to see that $T + H$ is a feasible solution to the obtained instance of EC-FTGS-2 if and only if the set of end-points of the edges in H is a vertex-cover in J.

In the case of vertex-costs, EC-FTGS-2 is easily reduced to the Steiner Tree problem with vertex-costs which is known to be Hitting-Set-hard [15]. Given an instance $\{J = (V_J, E_J), r, S\}$ of Steiner Tree with vertex-costs, for every $s \in S$ add a copy s' of cost 0 and connect s' to r. The set of pairs is $\mathcal{S} = \{\{s, s'\} \mid s \in S\}$. It is easy to see that T is a feasible solution to Steiner Tree with vertex-costs if and only if $T \cup \{(r, s') \mid s \in S\}$ is a feasible solution to the constructed 2-EC-FTGS-2 instance.

The proof of Part (i) of Theorem 1 is thus complete.

3 Proof of Part (ii)

3.1 Algorithm for VC-FTGS-2

In this section, we introduce our main technical ideas. We present an $O(\log^2 n)$-approximation algorithm for VC-FTGS-2.

As in the edge-connectivity case, we first compute a Steiner tree T of cost $\text{COST}(T) \leq \alpha \cdot \text{OPT}$ connecting all terminals to the root. We have $\alpha < 1.55$ for edge-costs and $\alpha = O(\log n)$ for vertex-costs. Our algorithm uses set-cover like approach in which we iteratively add partial solutions with low *density*, i.e., low cost to profit ratio, till we complete the solution. We get one logarithmic factor in the approximation from the set-cover analysis (and since the number of groups is $O(n)$) and another logarithmic factor from the fact that we can only compute $O(\log n)$ approximation to the minimum-density subproblem.

Given a partial solution $I \subset E \setminus E(T)$, let the *deficiency* of I be the number of groups in $\mathcal{S}$ that are not 2-vertex-connected to r in $T \cup I$. Let the *density* of an edge set $F \subset E \setminus (E(T) \cup I)$ be $\text{COST}(F)$ divided by the decrease in the deficiency caused by adding F to $T \cup I$. The following two lemmas captures the essence of our algorithm for VC-FTGS-2.

LEMMA 6. *Given a partial solution $T \cup I$, the problem of computing a minimum density augmenting edge set $F \subset E \setminus (E(T) \cup I)$ for* VC-FTGS-2 *admits an $O(\log n)$-approximation algorithm.*

LEMMA 7. *The algorithm in Lemma 6 can be used to obtain $O(\log^2 n)$ approximation for* VC-FTGS-2.

As mentioned, the proof of Lemma 7 follows from the standard set-cover like analysis and is omitted. In the rest of this section we prove Lemma 6. We ignore the groups in $\mathcal{S}$ that are already connected in $T \cup I$ to the root via 2 vertex-disjoint paths starting from distinct vertices.

We start by recalling some definitions. A vertex $v \in V$ is a cut-vertex of a graph H if $H \setminus \{v\}$ has more connected components than H. A cut-vertex v of H is said to *separate* vertex r and set $S \subset V \setminus \{r, v\}$ if r and S belong to the same connected component of H but $H \setminus \{v\}$ does not contain a path from r to any vertex in S. Consider a group $\{s_1, s_2\} \in \mathcal{S}$. By Menger's Theorem we have:

PROPOSITION 8. *A subgraph that contains an rs_1-path and an rs_2-path, contains such paths that are internally vertex disjoint if and only if it has no cut-vertex that separates r and $\{s_1, s_2\}$.*

Now think of the tree T as being rooted at r. For any two vertices $s_1, s_2 \in T$, we define $\text{LCA}(s_1, s_2)$ to be the least common ancestor of s_1 and s_2 in T. Consider $S = \{s_1, s_2\} \in \mathcal{S}$ with $u = \text{LCA}(s_1, s_2)$ in T. Note that $u \neq r$ since S is not 2-vertex connected to r via paths starting from s_1 and s_2. Let $\mathcal{U}(S) = \{u_1, u_2\}$ be two (possibly identical) vertices defined as follows.

- If $u \notin \{s_1, s_2\}$, then let u_1 (resp. u_2) be the child of u that lies on the rs_1- (resp. rs_2-) path in T.
- If $u \in \{s_1, s_2\}$, then let $u_1, u_2 = u$.

Define a family $\mathcal{U} = \mathcal{U}(\mathcal{S}, T)$ of pairs (groups) as $\mathcal{U} = \{\mathcal{U}(S) \mid S \in \mathcal{S}\}$. Note that the pairs in $\mathcal{U}$ may not be disjoint, and that some "pairs" in $\mathcal{U}$ are in fact singletons. The following lemma captures the "equivalence" between covering pairs in $\mathcal{S}$ and pairs in $\mathcal{U}$.

LEMMA 9. *For any edge set $F \subset E \setminus (E(T) \cup I)$, the solution $H = T \cup I \cup F$ contains 2-vertex disjoint paths (with distinct starting points) between r and S if, and only if, it contains 2-vertex disjoint paths (with distinct starting points) between r and $\mathcal{U}(S)$.*

PROOF. Let $u = \text{LCA}(s_1, s_2)$ where $S = \{s_1, s_2\}$. By Proposition 8, S is 2-connected to r in $H = T \cup I \cup F$ if and only if H has no cut-vertex separating S from r; namely, no vertex on the ur-path in T is a cut-vertex of H. By the definition of $\mathcal{U}(S) = \{u_1, u_2\}$ and Proposition 8, this is equivalent to the property that $\mathcal{U}(S)$ is 2-connected to r in H. ∎

The above lemma implies that the densities of F w.r.t. $\mathcal{S}$ and w.r.t. $\mathcal{U}$ are equal. Furthermore, $T \cup I \cup F$ is a feasible solution w.r.t. $\mathcal{S}$ if and only if it is w.r.t. $\mathcal{U}$. Note that the groups $\mathcal{U}$ satisfy a special property, which is crucial in rest of the analysis.

`Property P`: For all groups $\{u_1, u_2\} \in \mathcal{U}$, either $u_1 = u_2$ or u_1, u_2 have the same parent in T.

LEMMA 10. *Let $U = \{u_1, u_2\} \in \mathcal{U}$. For $F \subset E \setminus (E(T) \cup I)$, the graph $H = T \cup I \cup F$ contains a u_1r-path and a u_2r-path that are internally vertex-disjoint if and only if H contains 2 internally-disjoint paths to r from either u_1 or u_2.*

PROOF. The proof uses Menger's Theorem and `Property P` of the groups in $\mathcal{U}$.

Suppose that H contains 2 internally vertex-disjoint paths from u_1 to r. Then H has no cut-vertex separating r and u_1, by Menger's Theorem. In particular, there is no cut-vertex separating r and $\{u_1, u_2\}$. Thus H contains a u_1r-path and a u_2r-path that are internally vertex-disjoint, by Proposition 8.

Suppose now that H contains a u_1r-path and a u_2r-path that are internally vertex-disjoint. Now we use `Property P`. If $u_1 = u_2$, the proof is complete. Assume therefore that $u_1 \neq u_2$ and that they have a common parent u in T and assume to the contrary that H has no pair of internally vertex-disjoint u_ir-paths for $i = 1, 2$. Then by Menger's Theorem, there are cut-vertices v_1, v_2 in H, where v_i separates u_i from r. As u_1, u_2 have a common parent $u \neq r$, any vertex separating r from one of u_1, u_2 must lie on the ur-path in T. If $v_1 = v_2 = v$, then v separates both u_1, u_2 from r contradicting the assumption (by Proposition 8). Thus $v_1 \neq v_2$, so one of v_1, v_2 is distinct from u, say $v_1 \neq u$. The graph $H \setminus \{v_1\}$ contains a u_2r-path. As $T \setminus \{v_1\}$ contains a u_1u_2-path, this implies that $H \setminus \{v_1\}$ contains a u_1r-path. This contradicts that v_1 separates u_1 and r. ∎

Thus the original density problem can be reduced to the following problem. Given a collection of groups $\{u_1^i, u_2^i\}$ for $i = 1, 2, \ldots$, find a subset $F \subset E \setminus (E(T) \cup I)$ such that the ratio of $\text{COST}(F)$ to the number of groups i such that at least one of u_1^i or u_2^i has 2-vertex-disjoint paths to r in $E(T) \cup I \cup F$.

The problem of finding a subgraph that minimizes the ratio of its cost over the total profit of vertices that are 2-vertex-connected to the root was studied by Chekuri and Korula [6], who gave an $O(\log n)$-approximation for the problem. We use their algorithm to compute an $O(\log n)$-approximation for the density version of our problem, as follows. The input to the algorithm of Chekuri and Korula is the original graph with root r and with *profits* on vertices defined as follows. Let the *profit* $p(u)$ of a vertex u be the number of groups

1. Initialize subgraph $\mathcal{H} \leftarrow \emptyset$ and $q' \leftarrow q$ to be the number of uncovered groups.
2. If $q' > 0$, begin a phase:
   (a) Assign a cost of zero to all vertices in $\mathcal{H}$.
   (b) `Find Twin-pairs`: Compute a subgraph H of cost $O(\text{OPT} \cdot k \log n)$ that contains twin pairs (Definition 11) from at least $q'/2$ uncovered groups.
   (c) `Cover`: Compute a subgraph I of cost $O(\text{OPT} \cdot k \log n)$ that covers at least $q'/2$ uncovered groups.
   (d) Update $\mathcal{H} \leftarrow \mathcal{H} \cup H \cup I$ and update q' to be the number of uncovered groups in $\mathcal{H}$.
3. Output $\mathcal{H}$.

Figure 1: An outline of our algorithm for VC-FTGS-k with vertex-costs.

i such that $u \in \{u_1^i, u_2^i\}$. Thus $p(u)$ denotes the number of new groups that would get connected to the root via 2 vertex-disjoint paths provided u gets connected to the root via 2 vertex-disjoint paths. Note that since both u_1^i or u_2^i may claim profit for covering group i, we may overestimate the profit of 2-vertex connecting a subset of vertices to r by a factor of 2. This introduces another factor 2 in the approximation. Using Chekuri-Korula algorithm, we compute a subgraph which yields a $O(\log n)$ approximation for minimizing the ratio of its cost over the total profit of its vertices that are 2-vertex-connected to the root. This subgraph yields $O(\log n)$ approximation to the problem defined in Lemma 6.

Thus the proof is complete.

3.2 Algorithm for EC-FTGS-k and VC-FTGS-k with edge/vertex costs

We present an algorithm for VC-FTGS-k with vertex-costs, which is more general than the case of edge-costs. Adaptation of this algorithm to EC-FTGS-k is easy.

Fix an optimum solution OPT with cost also denoted by OPT. To simplify the presentation, the algorithm given below is assumed to know the value of OPT. In reality, the algorithm tries all possible guesses for the power of 2 that is closest to OPT and picks the cheapest solution among those computed for each of these guesses.

Our algorithm has logarithmic number of phases. In each phase, it covers at least half of the remaining groups. A high-level pseudo-code of the algorithm is given in Figure 1. At the beginning of each phase, we make the cost of the vertices that are already picked in the solution $\mathcal{H}$ *zero*. In what follows, we analyze a single phase which begins with q' uncovered groups overall. The groups in $\mathcal{S}$ that are already covered are ignored. In what follows, we explain how to implement steps `Find Twin-pairs` and `Cover` respectively.

How to implement step `Find Twin-pairs`.

DEFINITION 11. *For a group $S \in \mathcal{S}$, we say that the terminals $s_1, s_2 \in S$ form a* twin pair *if* OPT *contains two internally-disjoint paths from s_1 and s_2 to r. If there are more than one such pairs for a group, we designate exactly one of these pairs as* twin pair *arbitrarily.*

We do not know which terminals form twin pairs a-priori. Nevertheless, we can compute a low-cost tree that connects the twin pairs from at least half of the remaining groups to the root, as shown below. We iteratively use p-Steiner tree algorithm for $p = q'$. Recall

that the p-Steiner Tree problem is to compute a minimum-cost tree that connects at least p terminals to the root. Let H denote the union of the p-Steiner trees computed so far. Assume that the number of uncovered groups is at least $q'/2$.

LEMMA 12. *Assign a cost of zero vertices in H. Now apply $(c \log n)$-approximation algorithm [7] for the p-Steiner tree problem (where c is a constant) for the instance given by H, root r, $p = q'$ and terminals as the vertices in the the union of uncovered groups in $\mathcal{S}$ but not in H: $\{v \in S \mid S \in \mathcal{S}$ is uncovered, $v \notin H\}$. If the cost of the computed Steiner tree is more than $(c \log n) \cdot \text{OPT}$, then H contains the twin pairs for at least $q'/2$ uncovered groups.*

PROOF. Assume on the contrary that H does not contain twin pairs for at least $q'/2$ uncovered groups. Thus the OPT solution connects at least $2 \cdot q'/2 = q'$ terminals in this p-Steiner tree instance to the root. Since we use a $(c \log n)$-approximation, the cost of the computed Steiner tree is at most $(c \log n) \cdot \text{OPT}$. This is a contradiction, and the lemma follows. ∎

We run the p-Steiner tree algorithm iteratively while the cost of the new tree computed is at most $c \log n \cdot \text{OPT}$. Since H contains at least $p = q'$ new terminals in each iteration, the total number of invocations of such p-Steiner tree algorithm is at most $q' \cdot k/q'$. This holds since the size of each group is at most k. Thus the total cost of the step `Find Twin-pairs` is $\text{COST}(H) \leq O(\text{OPT} \cdot k \log n)$.

How to implement step `Cover`.

Even if H contains the twin pairs of at least $q'/2$ uncovered groups, we still do not know which of the terminals in H form twin pairs. We therefore need one more definition.

DEFINITION 13. *Let T be a spanning tree of H. We say that a vertex u_1* can help *an uncovered group S if there exist distinct vertices $s_1, s_2 \in S \cap T$ and another vertex u_2 so that u_1, u_2 have the same parent $u = \text{LCA}(s_1, s_2)$ in the tree T. The* profit *$p(u)$ of vertex u is defined as the number of uncovered groups in $\mathcal{S}$ that u can help.*

The intuition of the above definition comes from our algorithm for VC-FTGS-2, and in particular, from Lemma 10. Note that the profit of a vertex u is the number of uncovered groups that will get covered if u gets connected to the root via 2 internally-disjoint paths.

Since more than one vertex can claim a profit for covering a single group, it is important to understand how many vertices can help a particular group. Since there are at most k terminals in any group $S \in \mathcal{S}$, it is obvious that there can be at most $O(k^2)$ vertices that can help S. This crude upper bound comes from the fact that there are $O(k^2)$ pairs $s_1, s_2 \in S$ that can give rise to such vertices. However the following lemma presents a careful counting of such vertices.

LEMMA 14. *There are $O(k)$ vertices that can help any single group $S \in \mathcal{S}$.*

PROOF. Consider tree T with terminals in group S marked as s (see Figure 2). Further consider a subtree T' restricted to terminals in S. The vertices $u \in T'$ that can play a role of $\text{LCA}(s_1, s_2)$ for $s_1, s_2 \in S$ (these are shown as red squares in Figure 2) have a degree of at least 3 in T', i.e., $\deg_{T'}(u) \geq 3$. Thus if a vertex v can help group S, it must be a child of a vertex u with $\deg_{T'}(u) \geq 3$. The number of children of a vertex u is $\deg_{T'}(u) - 1$. Therefore, the number of vertices v that can help S is at most $\sum_{u:\deg_{T'}(u)\geq 3}(\deg_{T'}(u) - 1)$.

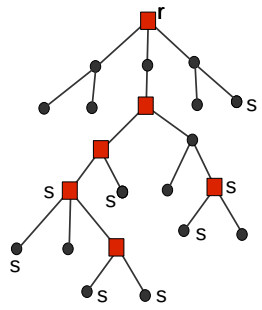


Figure 2: The vertices marked with s are terminals in a group S. The vertices marked with red squares can take a role of $\text{LCA}(s_1, s_2)$ for a pair $\{s_1, s_2\}$ in group S. The children of red squares that lie on a path from some $s \in S$ to r can help S. There are $O(k)$ such vertices.

Since T' is a tree induced on at most k terminals of S, the tree has at most k leaves. By a simple counting argument, it therefore follows that the desired sum $\sum_{u:\texttt{deg}_{T'}(u)\geq 3}(\texttt{deg}_{T'}(u) - 1)$ is $O(k)$. The lemma thus holds. ∎

Now we have all the ingredients to present the step `Cover`. We again use the $O(\log n)$-approximation algorithm of Chekuri and Korula [6] that was also used in Section 3.1, for the problem of finding a subgraph that minimizes the ratio of its cost over the total profit of vertices that are 2-vertex-connected to the root. Recall that the subgraph induced by OPT has cost at most OPT and profit at least $q'/2$ (since T contains twin pairs from at least $q'/2$ uncovered groups). Thus the cost-to-profit ratio of the solution computed by our algorithm is at most $O(\log n \cdot \text{OPT}/q')$. From Lemma 14, we get that the ratio of the cost of this subgraph to the number of uncovered groups covered is at most $O(k \log n \cdot \text{OPT}/q')$.

We apply such algorithm iteratively till end of the phase, i.e., at least $q'/2$ previously uncovered groups are covered in this phase, and call this subgraph I. From the above analysis, it is clear that the total cost of step `Cover` is $\text{COST}(I) = q' \cdot O(k \log n \cdot \text{OPT}/q') = O(k \log n \cdot \text{OPT})$. Thus the total cost of a phase is $\text{COST}(H) + \text{COST}(I) = O(\text{OPT} \cdot k \log n)$. Since there are $O(\log q) = O(\log n)$ phases overall, the total cost of the algorithm is $O(\text{OPT} \cdot k \log^2 n)$ as desired.

As was mentioned in the Introduction, an $O(\sqrt{n} \log n)$-approximation algorithm for VC-FTGS immediately follows from the $O(k \log^2 n)$-approximation for VC-FTGS-k, and thus the proof of Part (ii) of Theorem 1 is now complete.

References

[1] S. Antonakopoulos, C. Chekuri, F. B. Shepherd, and L. Zhang. Buy-at-bulk network design with protection. *FOCS* 2007.

[2] Y. Bartal. On approximating arbitrary metrices by tree metrics. *STOC* 1998.

[3] T. Chakraborty, J. Chuzhoy, and S. Khanna. Network design for vertex connectivity. *STOC* 2008.

[4] M. Charikar, C. Chekuri, T.-Y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33(1):73–91, 1999.

[5] C. Chekuri and N. Korula. Single-sink network design with vertex connectivity requirements. *FSTTCS* 2008.

[6] C. Chekuri and N. Korula. Pruning 2-Connected Graphs. *FSTTCS* 2008.

[7] F. Chudak, T. Roughgarden, and D. Williamson. Approximate k-MSTs and k-Steiner Trees via the Primal-Dual Method and Lagrangean Relaxation. *IPCO* 2001.

[8] J. Chuzhoy and S. Khanna. An $O(k^3 \log n)$-approximation algorithm for vertex-connectivity survivable network design. *FOCS* 2009.

[9] J. Chuzhoy and S. Khanna. Algorithms for single-source vertex-connectivity. *FOCS* 2008.

[10] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

[11] N. Garg. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. *STOC* 2005.

[12] N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000.

[13] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, E. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. *SODA* 1994.

[14] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[15] P. N. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *J. Algorithms*, 19(1):104–115, 1995.

[16] G. Kortsarz, R. Krauthgamer, and J. R. Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM J. Computing*, 33(3):704–720, 2004.

[17] G. Kortsarz and D. Peleg. Approximating the weight of shallow steiner trees. *Discrete Applied Mathematics*, 93(2-3):265–285, 1999.

[18] Y. Lando and Z. Nutov. Inapproximability of survivable networks. *Theor. Comput. Sci.*, 410(21-23):2122-2125, 2009.

[19] L. C. Lau, J. Naor, M. R. Salavatipour, and M. Singh. Survivable network design with degree or order constraints. *STOC* 2007.

[20] Z. Nutov. Approximating minimum cost connectivity problems via uncrossable bifamilies and spider-cover decompositions. *FOCS* 2009.

[21] Z. Nutov. Approximating connectivity augmentation problems. *SODA* 2005.

[22] Z. Nutov. Approximating Steiner Networks with Node Weights. *LATIN* 2008.

[23] Z. Nutov. An almost $O(\log k)$-approximation for k-connected subgraphs. *SODA* 2009.

[24] R. Raz. A parallel repitition theorem. *SIAM J. Computing*, 27(3):763–803, 1998.

[25] G. Reich and P. Widmayer. Beyond steiner's problem: a VLSI oriented generalization. In *Proceedings of the workshop on Graph-theoretic concepts in computer science*, 1990.

[26] G. Robins and A. Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM J. Discrete Math.*, 1(19):122–134, 2005.

[27] A. Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, 1997.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Bounded Size Graph Clustering with Applications to Stream Processing

Rohit Khandekar[1], Kirsten Hildrum[1], Sujay Parekh[1], Deepak Rajan[1], Jay Sethuraman[2], and Joel Wolf[1]

[1]IBM T.J.Watson Research Center
{rohitk,hildrum,sujay,drajan,jlwolf}@us.ibm.com

[2]Columbia University
js1353@columbia.edu

ABSTRACT. We introduce a graph clustering problem motivated by a stream processing application. Input to our problem is an undirected graph with vertex and edge weights. A cluster is a subset of the vertices. The *size* of a cluster is defined as the total vertex weight in the subset plus the total edge weight at the boundary of the cluster. The bounded size graph clustering problem (BSGC) is to partition the vertices into clusters of size at most a given budget and minimize the total edge-weight across the clusters. In the *multiway cut* version of the problem, we are also given a subset of vertices called *terminals*. No cluster is allowed to contain more than one terminal. Our problem differs from most of the previously studied clustering problems in that the number of clusters is not specified. We first show that the feasibility version of the multiway cut BSGC problem, i.e., determining if there exists a clustering with bounded-size clusters satisfying the multiway cut constraint, can be solved in polynomial time. Our algorithm is based on the min-cut subroutine and an uncrossing argument. This result is in contrast with the NP-hardness of the min-max multiway cut problem, considered by Svitkina and Tardos (2004), in which the number of clusters must equal the number of terminals. Our results for the feasibility version also generalize to any symmetric submodular function. We next show that the optimization version of BSGC is NP-hard by showing an approximation-preserving reduction from the $\frac{1}{3}$-balanced cut problem. Our main result is an $O(\log^2 n)$-approximation to the optimization version of the multiway cut BSGC problem violating the budget by an $O(\log n)$ factor, where n denotes the number of vertices. Our algorithm is based on a set-cover-like greedy approach which iteratively computes bounded-size clusters to maximize the number of new vertices covered.

1 Introduction

Graph partitioning and clustering are fundamental optimization problems with applications to a variety of areas like VLSI design, divide-and-conquer algorithms, computer vision, data analysis, discovering communities in social networks, and learning. In this paper we introduce a graph clustering problem motivated by *System S*, a stream computing system [1] being developed at IBM research. Consider a system that takes, as input, a high-throughput data stream such as live option trading or stock feeds in financial services, physical link statistics in networking and telecommunications, sensor readings in environmental monitoring and emergency response, or live experimental data in scientific applications. This system is required to generate responses derived from on-line processing of the data

© Khandekar et al.; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 275–286
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2325

in real-time. An application in this system can be modeled as a graph in which the vertices represent domain-specific *operators* consuming and producing streams of data, and the edges represent the *streams* themselves.

It is very convenient to compose these applications out of type-generic, built-in stream processing operators [7]. But these operators are usually small: they typically spend more effort sending and receiving data than they do in actual processing. Taken individually, such operators can therefore become performance bottlenecks in the system. The good news is that it is actually possible to "fuse" operators at compile time. If two adjacent operators are fused, the downstream operator is invoked by means of a function call from the upstream operator. As a result, there is effectively no cost to sending data between fused pairs of adjacent operators.

Thus, to efficiently deploy such a CPU-intensive application in a distributed computing environment, one has to decide how to partition the operators into clusters, for example one for each computing host. The total CPU requirement for a cluster of operators comes from two sources. The first is the computational needs of the operators in the cluster. This can be modeled by associating a non-negative weight with each operator u. The total computational needs of a cluster is then the sum of its operators weights. The second is the communication overhead, incurred at the *boundary* of the cluster, for receiving and sending streams to operators outside the cluster. This can be modeled by associating a non-negative weight with each edge $e = (u, v)$. The total communication needs of a cluster is then the cut-weight with respect to these edge-weights. As noted, an edge contained within a cluster is converted into a function-call, incurring negligible overhead and as such not accounted for in the computational needs of the cluster. The consideration of the CPU requirement imposes a natural constraint on the clustering: the total CPU requirement of each cluster must be at most the capacity of a host.

We frequently encounter additional resource constraints that cannot be captured as CPU requirements. For example, some operators make extensive use of specific hardware (such as a network card). Clustering two such operators together would cause poor performance. To handle such situations, we include in the problem a set of terminals T and insist that each cluster contain at most one terminal from T.

A high-throughput application, if not carefully deployed, may overload the network capacity. Therefore, a natural goal when computing a bounded-size clustering is to minimize the total edge-weights across the clusters.

Formal problem definition. With the above motivation, we introduce the `Bounded-Size Graph Clustering` (`BSGC`) problem, defined formally as follows. Consider an undirected graph $G = (V, E)$ on n vertices with vertex-weights $w_v \in \mathbb{Q}_+$ and edge-weights $w_e \in \mathbb{Q}_+$. Here $\mathbb{Q}_+$ denotes the set of non-negative rational numbers. For a subset $S \subset V$, let $\delta(S)$ denote the set of edges with exactly one end-point in S, $w(S) = \sum_{v \in S} w_v$, $w(\delta(S)) = \sum_{e \in \delta(S)} w_e$, and $\texttt{size}(S) = w(S) + w(\delta(S))$. We are also given a set of *terminals* $T \subseteq V$ and a *budget* $B \in \mathbb{Q}_+$. The `BSGC` problem is to find a partitioning of the vertex set into clusters $S_1, \ldots, S_k$ such that

- the size of each cluster is bounded: $\texttt{size}(S_i) \leq B$ for all i;
- each cluster contains *at most* one terminal, i.e., $|S_i \cap T| \leq 1$ for all i; and

- the total edge-weight across the clusters, $\frac{1}{2}\sum_{i=1}^{k} w(\delta(S_i))$, is minimized.

We call a clustering that satisfies the first two properties *feasible*. From the second condition, it is clear that the number of clusters k is at least the number of terminals $|T|$. However, k is *not* given as input, and it may be *larger* than the number of terminals.

Our results. Our main results are summarized below:

1. First, in Section 2, we consider the *feasibility* version of BSGC, i.e., to compute a feasible clustering without considering the total cut-weight. We show that we can compute a feasible clustering, if it exists, in polynomial time. Our algorithm uses a minimum cut subroutine and an uncrossing argument to make the clusters disjoint. This result generalizes to any symmetric submodular function [6]. See Section 2.1 for more details.
2. In Section 3.1 we show that the BSGC problem is NP-hard by an approximation preserving reduction from the $\frac{1}{3}$-balanced cut problem. Recall that the best-known approximation for the $\frac{1}{3}$-balanced cut problem that does not violate* the balance constraint is $O(\log n)$ [10].
3. Finally, in Section 3.2, we present a pseudo-approximation for the *optimization* version of the problem. More precisely, we present a deterministic polynomial-time algorithm that computes a clustering $\{S_1, \ldots, S_k\}$ such that $|S_i \cap T| \leq 1$ and $w(\delta(S_i)) \leq O(\log n) \cdot (B - w(S_i))$ for all i, and the total cut-weight is $O(\log^2 n)$ times the optimum cut-weight. Note that the above condition implies that $(c \log n) \cdot w(S_i) + w(\delta(S_i)) \leq (c \log n) \cdot B$ for some absolute constant $c > 0$, i.e., the budget is violated by an $O(\log n)$ factor.

Related work. A problem that is closely related to the feasibility version of BSGC was studied by Svitkina and Tardos [11]. In that problem, called *min-max multiway cut*, an edge-weighted undirected graph with terminals $T \subseteq V$ is given. The goal is to partition vertices into $|T|$ clusters such that each cluster contains *exactly* one terminal and the maximum cut value of a cluster is minimized. A crucial difference is that BSGC does not require the number of clusters to be exactly $|T|$. Svitkina and Tardos show that the min-max multiway cut problem is NP-hard and present an $O(\log^2 n)$-approximation† for it. They do so using, as a subroutine, the maximum-size bounded capacity cut problem (MaxSBCC), defined as follows: Given an undirected graph $G = (V, E)$ with vertex and edge weights, two vertices $s, t \in V$, and a budget $B > 0$, find an s-t cut $(S, V \setminus S)$ such that $w(\delta(S)) \leq B$ and $w(S)$ is maximized. Svitkina and Tardos iteratively solve MaxSBCC with varying vertex weights and combine those cuts to compute their final clustering.

Several cut problems with budget constraints were also considered by Engelberg et al. [5]. In particular, they consider budgeted multiway cut problems in which there is a budget on the total cut-value and the objective is either to maximize the number of terminal-pairs separated, to maximize the number of terminals that are completely separated from

*We can obtain an $O(\sqrt{\log n})$ approximation if we violate the budget by a constant factor [2].

†In fact, they present an $O(\log^3 n)$-approximation using a subroutine for finding minimum cuts with the specified number of vertices. Using an improved $O(\log n)$-approximation for the subroutine [10], their algorithm can be shown to yield an $O(\log^2 n)$-approximation.

other terminals, or simply to maximize the number of connected components. They use Räcke's tree decomposition [10] and Gomory-Hu trees [8] to design their approximation algorithms.

Most graph clustering problems in the literature specify the number of clusters required as part of the input. One important exception is *correlation clustering*, introduced by Bansal et al. [3]. In this problem, the edges of an undirected graph are labelled with either "$+$" or "$-$". Given a clustering of the vertices, let the number of "agreements" be the number of $+$ edges inside the clusters plus the number of $-$ edges across the clusters. Similarly the number of "disagreements" is the total number of edges minus the number of agreements. Bansal et al. and several subsequent papers design approximation algorithms for maximizing the number of agreements or minimizing the number of disagreements.

Finally, Khandekar et al. [9] consider a variant of our graph clustering problem (with significantly more elaborate practical constraints). Their (partially heuristic) solution is, in fact, implemented as a key component in *System S* [1].

Our techniques. In contrast to min-max multiway cut, relaxing the constraint on the number of clusters makes it possible to find a polynomial-time algorithm for the feasibility version of BSGC. For each vertex v, our algorithm computes a cluster, of size at most B, containing v and at most one terminal. To this end, we first augment the graph by adding a new vertex with edges to all the old vertices and "translate" the vertex weights into weights on the new edges. Later we show that the problem of computing a desired cluster can be reduced to minimum cut computations in the augmented graph. The clusters thus computed may not be disjoint, however. The algorithm then systematically *uncrosses* the clusters to make them disjoint, using an argument similar to [11], while satisfying the budget and the multiway cut constraints. This result applies more generally: if the size of a cluster S is defined as $\sum_{v \in S} w_v + f(S)$, where f is a symmetric submodular set function, we can determine in polynomial time if there exists a clustering such that each cluster contains at most one terminal and has size at most B.

The optimization version of BSGC is different from the traditional graph partitioning into clusters of bounded-size, because the size of a cluster includes its cut-cost. Therefore the hierarchical partitioning approach – iteratively splitting clusters into two until the size constraints are satisfied – does not work for the BSGC problem. For example, after splitting the given graph into two, there may not exist a feasible clustering respecting this split, even if the original graph has a feasible clustering.

Our approach for the optimization problem resembles that of Svitkina and Tardos [11]. We think of our problem as an instance of the set-cover problem where the sets are the subsets $S \subseteq V$ such that $|S \cap T| \leq 1$ and $\texttt{size}(S) \leq B$. Let the cost of such a set be $w(\delta(S))$. The problem is then to find a minimum-cost collection of sets that covers all the vertices. Now in order to use the greedy algorithm, we need the following *oracle*: given a subset of vertices not yet covered, find a set S that minimizes the ratio of $w(\delta(S))$ and the number of vertices in S that are not yet covered. Unfortunately the oracle itself is NP-hard. We then use a *hierarchical tree-decomposition* of graphs by Räcke [10] to get a $O(\log n)$-approximation

to the oracle. More precisely, we find $S \subseteq V$ such that

$$(c \log n) \cdot w(S) + w(\delta(S)) \leq (c \log n) \cdot B$$

for an absolute constant $c > 0$ that also minimizes the desired ratio to within an $O(\log n)$ factor. This, combined with a standard set-cover analysis, yields our final result. Once again we use an uncrossing argument to make the clusters disjoint.

2 The feasibility version

Since the definition of the size of a cluster involves both the vertex and edge weights, it is not clear a-priori if the feasibility version, i.e., to determine if there *exists* a feasible clustering, is tractable. For example, the clustering obtained by putting each vertex into a separate cluster may not be feasible. Assuming that the problem is feasible, we now present a polynomial-time algorithm for finding a clustering $\{S_1, \ldots, S_k\}$ such that $\texttt{size}(S_i) \leq B$ and $|S_i \cap T| \leq 1$ for all i.

Our idea is to construct a new graph in which the vertex weights are converted into edge weights on artificial edges. This means our algorithm can work just with cuts. We construct this graph $G' = (V', E')$ as follows. (See Figure 1.) Let $V' = V \cup \{s\}$ for a new vertex s and $E' = E \cup \{(s, v) \mid v \in V\}$. Each edge $e \in E' \cup E$ inherits its weight w_e, and $e = (s, v) \in E'$ gets a weight of $w_e = w_v$ for all $v \in V$. Note that for a cluster $S \subseteq V$, we have that $\texttt{size}(S)$ equals the capacity of the cut $(S, V' \setminus S)$ in G'.

In a problem instance *without* terminals, we note that Gomory-Hu trees [8] allow us to determine feasibility. Consider the Gomory-Hu tree $\mathcal{T}$ of G'. In a feasible instance, the minimum cut in G' between s and any other vertex u is at most B, and the edges in $\mathcal{T}$ that are incident to s have weight at most B each. The removal these edges from T gives a partitioning of vertices into clusters, say $S_1, \ldots, S_k$. It is easy to see that this is a feasible clustering for our problem. If there are edges in $\mathcal{T}$ that are incident to s and have weight greater than B, the problem instance is not feasible.

To approach the problem *with* terminals, we start by stating a useful lemma that will simplify the presentation of our algorithm. The following lemma states that it is enough to compute possibly *overlapping* clusters that satisfy the given constraints. The basic technique used in this lemma is systematic *uncrossing*.

LEMMA 1. *Given clusters $\{S_1, \ldots, S_k\}$ such that $\cup_i S_i = V$ and $|S_i \cap T| \leq 1$ for all i, we can compute in polynomial time clusters $\{U_1, \ldots, U_k\}$ such that $\cup_i U_i = V$, $|U_i \cap T| \leq 1$, $w(U_i) \leq w(S_i)$, $w(\delta(U_i)) \leq w(\delta(S_i))$ for all i, and moreover $U_i \cap U_j = \emptyset$ for $i \neq j$*

PROOF. For two disjoint subsets $A, B \subset V$, let $w(A, B) = \{w_e \mid e = (u, v), u \in A, v \in B\}$ be the total edge-weight between A and B. We define an uncrossing operation for two intersecting sets A and B as follows. If $w(A \cap B, A \setminus B) < w(A \cap B, B \setminus A)$, we let $A' \leftarrow A \setminus B$ and $B' \leftarrow B$, else we let $A' \leftarrow A$ and $B' \leftarrow B \setminus A$. Note that we have: $w(A') \leq w(A), w(B') \leq w(B), w(\delta(A')) \leq w(\delta(A)), w(\delta(B')) \leq w(\delta(B))$, and $A' \cap B' = \emptyset$.

We apply the above uncrossing operation to $S_1, \ldots, S_k$ systematically, obtaining $U_1, \ldots, U_k$ as follows. We first let $U_1 = S_1$ and make it disjoint from $S_2, \ldots, S_k$ in that order. Then we let $U_2 = S_2$ and repeat. In the end, we have sets U_i with the desired properties. ∎

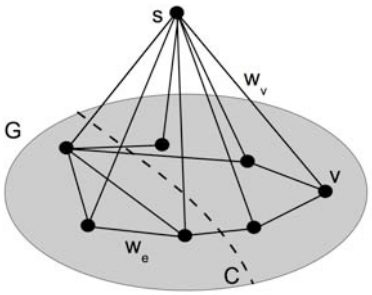


Figure 1: Construction of graph G'. The capacity of the cut $(C, (V \setminus C) \cup \{s\})$ in G' is $\texttt{size}(C)$.

LEMMA 2. *For any $v \in V$, in polynomial-time, we can find a cluster $S_v \subset V$ such that $v \in S_v$, $|S_v \cap T| \leq 1$, and $\texttt{size}(S_v) \leq B$.*

PROOF. Since BSGC is feasible, for any $v \in V$, the cluster S_v^* in a feasible clustering satisfies the above conditions.

If v is a terminal, we find a minimum cut in G' that separates v from $(T \setminus \{v\}) \cup \{s\}$ by doing a single min-cut computation.‡ Let S_v denote the vertices on the v-side of this cut. From the minimality of the cut, we have $\texttt{size}(S_v) \leq \texttt{size}(S_v^*) \leq B$.

If v is not a terminal, we try all possible values of $S_v^* \cap T$. It can either be empty or a singleton set containing a terminal. If $S_v^* \cap T = \emptyset$, we can find a minimum cut in G' that separates v from $T \cup \{s\}$. On the other hand, if $S_v^* \cap T = \{t\}$, we can find a minimum cut in G' that separates $\{v, t\}$ from $(T \setminus \{t\}) \cup \{s\}$. In either case, we can find S_v satisfying the desired properties. ∎

We can now find a feasible clustering in polynomial-time as follows.

1. Compute clusters S_v for all v satisfying the conditions in Lemma 2.
2. Systematically uncross clusters S_v to make them disjoint using Lemma 1.

2.1 Generalizations to symmetric submodular functions

A function $f : 2^V \to \Re_+$ is called *submodular* if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ holds for all $A, B \subseteq V$, and it is called *symmetric* if $f(A) = f(V \setminus A)$ holds for all $A \subseteq V$. For an undirected graph $G = (V, E)$ with edges weights w_e, the function $f(S) = w(\delta(S))$ for $S \subseteq V$ is symmetric and submodular. We can extend the results for the feasibility version of the problem to general symmetric submodular functions. The feasibility version of the bounded size clustering problem for symmetric submodular function is defined as follows. Given a symmetric submodular function $f : 2^V \to \Re_+$, a weight function $w : V \to \Re_+$, a set of terminals $T \subseteq V$, and a budget B, find a partitioning of V into clusters such that for each cluster $S \subseteq V$ we have $|S \cap T| \leq 1$ and $\texttt{size}(S) = \sum_{v \in S} w_v + f(S) \leq B$.

We now briefly outline how Lemmas 1 and 2, and hence our algorithm for the feasibility version, can be generalized to symmetric submodular functions. The generalization of the

‡This can be done by shrinking $(T \setminus \{v\}) \cup \{s\}$ into a super-vertex s' (or alternately adding very high weight edges between vertices in $(T \setminus \{v\}) \cup \{s\}$) and finding a min-cut separating v and s'.

proof of Lemma 1 follows from the observation that for any two sets $A, B \subseteq V$, we have $f(A \setminus B) + f(B \setminus A) = f(A \cap \overline{B}) + f(B \cap \overline{A}) = f(A \cap \overline{B}) + f(\overline{B} \cup A) \leq f(A) + f(\overline{B}) = f(A) + f(B)$. Thus either $f(A \setminus B) \leq f(A)$ or $f(B \setminus A) \leq f(B)$ holds.

To generalize the proof of Lemma 2, we introduce a new element s to the ground set V and define a symmetric submodular function $g : V \cup \{s\} \to \Re_+$ as

$$g(A) = \begin{cases} \sum_{v \in A} w_v, & \text{if } s \notin A, \\ \sum_{v \notin A} w_v, & \text{if } s \in A. \end{cases}$$

The function g corresponds to adding edges of weight w_v between s and $v \in V$. We also lift f from V to $V \cup \{s\}$ by defining $f(A) = f(V \cap A)$ for $A \subseteq V \cup \{s\}$. It is easy to see that for any $A \subseteq V$, we have $\texttt{size}(A) = f(A) + g(A)$. Now note that a set separating two subsets $A_1, A_2 \subset V$ of elements that minimizes the symmetric submodular function $f + g$ can be computed by "merging" the elements A_1 (respectively, A_2) into a super-element a_1 (respectively, a_2) and using standard algorithms for symmetric submodular function minimization [6] to separate elements a_1 and a_2. The proof of Lemma 2 thus holds for symmetric submodular functions as well.

3 The optimization version

3.1 NP-hardness

We present an approximation preserving reduction from the $\frac{1}{3}$-balanced cut problem, which is NP-hard, to the BSGC problem with $T = \emptyset$. The $\frac{1}{3}$-balanced cut problem is defined as follows: given undirected graph $G = (V, E)$ on n vertices with vertex weights $w_v \geq 0$ and edge weights $w_e \geq 0$, partition the vertices into two non-empty clusters $S \subset V$ and $V \setminus S$ such that $\min\{w(S), w(V \setminus S)\} \geq \frac{1}{3}w(V)$ and $w(\delta(S))$ is minimized. This problem is NP-hard [4] and the best-known approximation for this problem that does not violate* the balance constraint is $O(\log n)$ [10].

LEMMA 3. *If there is a ρ-approximation for the* BSGC *problem with $T = \emptyset$, there is a ρ-approximation to the $\frac{1}{3}$-balanced cut problem.*

PROOF. Given an instance (G, w) of the $\frac{1}{3}$-balanced cut problem, we create an instance of the BSGC problem as follows. We scale the vertex and edge weights so that $1 = \min_{v \in V} w_v > 2\sum_{e \in E} w_e$ and let $B = \frac{2}{3}w(V) + \frac{1}{2}$ and $T = \emptyset$. We then compute a ρ-approximation for the BSGC problem. We can assume, without loss of generality, that the output consists of *exactly* two clusters, as follows. As long as we have at least three clusters, say S_1, S_2, S_3 with $w(S_1) \leq w(S_2) \leq w(S_3)$, we can merge S_1 and S_2 into a single cluster without violating the budget constraint. This merge does not increase the total edge-weight across the clusters. Since $\min_v w_v = 1$, it is now easy to see that the resulting two clusters, say $\tilde{S}_1$ and $\tilde{S}_2 = V \setminus \tilde{S}_1$, satisfy the balance condition and form a ρ-approximation for the $\frac{1}{3}$-balanced cut problem. ∎

3.2 The algorithm

In this section, we show how to find $\{S_1, \ldots, S_k\}$ such that $|S_i \cap T| \leq 1$ and $w(\delta(S_i)) \leq O(\log n) \cdot (B - w(S_i))$ for all i such that the total cut-weight is $O(\log^2 n)$ times the optimum

1. Initialize $U \leftarrow V$ be to the set of not-yet-covered vertices.
2. Initialize the set of clusters $\mathcal{S} \leftarrow \emptyset$.
3. While $U \neq \emptyset$ do:
   (a) Find an approximately valid set $S \subseteq V$ such that
   $$\frac{w(\delta(S))}{|S \cap U|} \leq (c \log n) \cdot \frac{2\text{OPT}}{|U|}.$$
   (b) Add S to $\mathcal{S}$.
   (c) Let $U \leftarrow U \setminus S$.
4. Uncross the clusters in $\mathcal{S}$.

Figure 2: Algorithm for BSGC

cut-weight. We think of BSGC as a set-cover problem. The elements to be covered are the vertices and the sets are "valid" subsets of V.

DEFINITION 4. *A subset $S \subseteq V$ is called* valid *if $|S \cap T| \leq 1$ and $w(S) + w(\delta(S)) \leq B$. A subset $S \subseteq V$ is called* approximately valid *if $|S \cap T| \leq 1$ and*

$$(c \log n) \cdot w(S) + w(\delta(S)) \leq (c \log n) \cdot B$$

holds, where $c > 0$ is an absolute constant, the value of which will be fixed later.

Let the cost of S be $w(\delta(S))$. Clearly the optimum covers all the elements using only valid subsets. Let OPT denote the cost of this optimum set cover. Note that the number of sets is exponential in general. However the greedy set cover algorithm only needs the following oracle: given a subset $U \subseteq V$ of "yet to be covered" vertices, find a valid set S that minimizes the ratio $\frac{w(\delta(S))}{|S \cap U|}$. Unfortunately, even this oracle is NP-hard, and hence we use an approximation for the oracle. Our algorithm, given in Figure 2, picks approximately valid subsets one by one to cover all the vertices. Then, using Lemma 1, it uncrosses the clusters to make them disjoint.

Finding a minimum ratio approximately valid set

LEMMA 5. *Given a non-empty subset $U \subseteq V$, we can find in polynomial time an approximately valid subset $S \subseteq V$ such that*

$$\frac{w(\delta(S))}{|S \cap U|} \leq (c \log n) \cdot \frac{2\text{OPT}}{|U|}.$$

It is easy to see that this lemma combined with the analysis of the greedy set-cover algorithm yields our result.

Proof of Lemma 5. We argue that there exists a valid subset $S^* \subseteq V$ such that $\frac{w(\delta(S^*))}{|S^* \cap U|} \leq \frac{2\text{OPT}}{|U|}$. Consider the optimum clustering $\{S_i^*\}$. Note that

$$\min_i \frac{w(\delta(S_i^*))}{|S_i^* \cap U|} \leq \frac{\sum_i w(\delta(S_i^*))}{\sum_i |S_i^* \cap U|} = \frac{2\text{OPT}}{|U|}.$$

Thus the cluster S_i^* that minimizes the ratio $\frac{w(\delta(S_i^*))}{|S_i^* \cap U|}$ is a candidate set.

We next use the following tree decomposition result of Räcke [10]. Given an edge-weighted undirected graph $G = (V, E)$, a tree decomposition $\mathcal{T}$ is an edge-weighted rooted tree which has a one-to-one correspondence between the vertices V and the leaves of $\mathcal{T}$.

THEOREM 6. *[Räcke [10]] There exists a probability distribution on polynomially many tree decompositions $\mathcal{T}$ such that for all sets $S \subseteq V$ and all $\mathcal{T}$, we have $w(\delta(S)) \leq w_{\mathcal{T}}(\delta(S))$ and*

$$\mathbb{E}_{\mathcal{T}}[w_{\mathcal{T}}(\delta(S))] \leq (c \log n) \cdot w(\delta(S))$$

for an absolute constant $c > 0$. Here $w_{\mathcal{T}}(\delta(S))$ denotes the minimum cut in $\mathcal{T}$ that separates leaves in S from the other leaves. Moreover such a distribution and tree decompositions can be found in polynomial time.

Let the constant $c > 0$ be as given in Theorem 6. Our algorithm first computes the tree decompositions given in Theorem 6 and assigns a weight of w_v to each leaf corresponding to vertex v in each of these tree decompositions. From Theorem 6 and an averaging argument, there exists a tree decomposition, say $\mathcal{T}^*$, in this collection such that

$$\frac{w_{\mathcal{T}^*}(\delta(S^*))}{|S^* \cap U|} \leq (c \log n) \cdot \frac{\text{OPT}}{|U|} \quad \text{and} \quad w_{\mathcal{T}^*}(\delta(S^*)) \leq (c \log n) \cdot (B - w(S^*)).$$

Of course, we do not know which of the polynomially many tree decompositions $\mathcal{T}^*$ corresponds to a-priori. Therefore our algorithm tries each of these tree decompositions $\mathcal{T}$ and computes the set S, if it exists, such that

$$|S \cap T| \leq 1 \quad \text{and} \quad w_{\mathcal{T}}(\delta(S)) \leq (c \log n) \cdot (B - w(S)) \tag{1}$$

holds and such that

$$\frac{w_{\mathcal{T}}(\delta(S))}{|S \cap U|} \tag{2}$$

is minimized. Finally, it outputs the set computed in this manner with the minimum ratio (2).

Now fix a tree decomposition $\mathcal{T}$. In order to compute a set S satisfying (1) with the minimum ratio (2), the algorithm runs the following dynamic program. For each value of $k \in \{1, \ldots, |U|\}$ and each possible weight $w \leq B$, it computes S, if it exists, such that $w(S) = w$, $|S \cap T| \leq 1$, $|S \cap U| = k$, and $w_{\mathcal{T}}(\delta(S))$ is minimized. If $w_{\mathcal{T}}(\delta(S)) \leq (c \log n) \cdot (B - w)$ holds, it stores the set S as a candidate set. In the end, it outputs the candidate set with minimum ratio (2).

The dynamic program. To this end, using standard scaling techniques we assume that the vertex and edge weights in $\mathcal{T}$ are polynomially bounded in n. More precisely, we can assume without loss of generality that $w_v \leq B$ for all $v \in V$; otherwise no feasible clustering exists. Next we shrink all the edges $e \in E$ with $w_e > B$, since such edges cannot cross clusters in a feasible clustering. Furthermore, for all v such that $w_v \leq B/n$, we set $w_v = 0$, and for all e such that $w_e \leq B/n^2$, we set $w_e = 0$. In doing so, we can only violate the budget by an extra constant factor. By scaling if necessary, we assume that the vertex and edge weights and the budget B are non-negative integers. We also assume for simplicity that $c \log n$ is an integer.

We can assume, without loss of generality, that $\mathcal{T}$ is a *binary* tree. If some internal node v has $l > 2$ children, we can replace v by a binary tree with l leaves and attach the l children to the l leaves one-to-one. We also give a cost of $1 + (c \log n) \cdot B$ to the edges of this new binary tree. Since $w_{\mathcal{T}}(\delta(S^*)) \leq (c \log n) \cdot B$, no edge of such a high cost will be present in the cut $w_{\mathcal{T}}(\delta(S))$ output by the dynamic program. Thus, computing the desired set S in the original tree is equivalent to computing S in the transformed binary tree.

For a node $v \in \mathcal{T}$, let $\mathcal{T}_v$ denote the subtree hanging from node v (including node v). Now for each node $v \in \mathcal{T}$, our dynamic program builds the following table. For each $I \subset T$ with $|I| \leq 1$, integer weights $w \leq B$ and $w_1, w_2 \leq (c \log n) \cdot B$, and an integer $k \leq |U|$, we store a subset $S[v, I, w, w_1, w_2, k]$ of the leaves in $\mathcal{T}_v$, if it exists, such that

1. $S \cap T = I$,
2. $w(S) = w$,
3. the minimum cut in $\mathcal{T}_v$ separating S from the remaining leaves in $\mathcal{T}_v$ has weight w_1,
4. the minimum cut in $\mathcal{T}_v$ separating S from the remaining leaves in $\mathcal{T}_v$ *as well as* v has weight w_2, and
5. $|S \cap U| = k$.

Observe that a cut separating S from the remaining leaves in $\mathcal{T}_v$ may contain node v on either side of the cut. Therefore, $w_1 \leq w_2$. It is easy to see that this table is of polynomial size. The final output of the dynamic program is computed as follows: among all possible sets $S[r, I, w, w_1, w_2, k]$, where r is the root of $\mathcal{T}$, output a set satisfying (1) that minimizes the ratio (2).

We next show how to compute this table in bottom-up fashion in polynomial time. If v is a leaf node, the table has no entries. For internal nodes v that have leaf nodes as its children, it is easy to compute such a table. For all other internal nodes v, let p and q be its children and assume that such tables are already computed for nodes p and q. Let w_{vp} (resp. w_{vq}) denote the weight of edge (v, p) (resp. (v, q)) in $\mathcal{T}$.

Given values of (I, w, w_1, w_2, k), we find disjoint sets

$$S^p = S[p, I^p, w^p, w_1^p, w_2^p, k^p]$$

and

$$S^q = S[q, I^q, w^q, w_1^q, w_2^q, k^q]$$

if they exist, for all possible decompositions $I = I^p \cup I^q$, $w = w^p + w^q$, and $k = k^p + k^q$, and all possible choices of $w_1^p, w_1^q, w_2^p, w_2^q$, such that the following conditions hold:

$$w_1 = \min\{w_1^p + w_1^q + \min\{w_{vp}, w_{vq}\}, w_2^p + w_1^q, w_1^p + w_2^q\}, \tag{3}$$

and

$$w_2 = \min\{w_2^p, w_1^p + w_{vp}\} + \min\{w_2^q, w_1^q + w_{vq}\}. \quad (4)$$

Note that the expression (3) considers all possible ways of realizing a cut with weight w_1 that separates S from the remaining leaves in $\mathcal{T}_v$. In fact, the three terms correspond to whether nodes p and q are on the same side of the cut as sets S^p and S^q, respectively. Similarly, the expression (4) considers all possible ways of realizing a cut with weight w_2 that separates S from the remaining leaves in $\mathcal{T}_v$ as well as v.

If there exist such sets S^p and S^q for any such decomposition, ties broken arbitrarily, we store $S = S^p \cup S^q$ as the entry $S[v, I, w, w_1, w_2, k]$. Otherwise we leave the entry empty. The correctness and the polynomial size of the dynamic program follows easily.

4 Conclusions

A consequence of our work is that the min-max multiway cut problem becomes polynomial-time solvable if there are allowed to be clusters *without* terminals. This raises a question of whether other graph problems become similarly easier if the number of clusters is not specified as part of the input. Our work also introduces many interesting open questions. Since the feasibility version of BSGC is solvable in polynomial time, can one approximate BSGC, say within a poly-logarithmic factor, *without* violating the budget constraint? In stream processing applications, it is often important to find a clustering to minimize the maximum *latency* of a path taken by a data stream, where an edge on a path contributes to the latency only if it goes between two clusters. Studying the approximability of this problem is an important research direction.

Acknowledgments

We would like to thank Nikhil Bansal for useful discussions.

References

[1] System S stream computing system. http://www-01.ibm.com/software/data/infosphere/streams.

[2] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings, Symposium on Theory of Computing (STOC)*, pages 222–231, 2004.

[3] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004.

[4] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Inform. Process. Lett.*, 42:153–159, 1992.

[5] Roee Engelberg, Jochen Könemann, Stefano Leonardi, and Joseph Naor. Cut problems in graphs with a budget constraint. *J. Discrete Algorithms*, 5(2):262–279, 2007.

[6] S. Fujishige. *Submodular Functions and Optimization*. North-Holland, 1991.

[7] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. SPADE: The System S declarative stream processing engine. In *Proceedings, ACM SIGMOD Conference*, 2008.

[8] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *J. Soc. Indust. Appl. Math.*, 9(4):551–570, 1961.

[9] R. Khandekar, K. Hildrum, S. Parekh, D. Rajan, J. Wolf, K.-L. Wu, H. Andrade, and B. Gedik. COLA: Optimizing stream processing applications via graph partitioning. In *Proceedings, Middleware Conference*, 2009.

[10] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings, Symposium on Theory of Computing (STOC)*, pages 255–264, 2008.

[11] Zoya Svitkina and Éva Tardos. Min-max multiway cut. In *Proceedings, Approx-Random*, pages 207–218, 2004.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

A Fine-grained Analysis of a Simple Independent Set Algorithm*

Joachim Kneis, Alexander Langer, Peter Rossmanith
Dept. of Computer Science, RWTH Aachen University, Germany
{kneis,langer,rossmani}@cs.rwth-aachen.de

ABSTRACT. We present a simple exact algorithm for the INDEPENDENT SET problem with a runtime bounded by $O(1.2132^n poly(n))$. This bound is obtained by, firstly, applying a new branching rule and, secondly, by a distinct, computer-aided case analysis. The new branching rule uses the concept of satellites and has previously only been used in an algorithm for sparse graphs. The computer-aided case analysis allows us to capture the behavior of our algorithm in more detail than in a traditional analysis. The main purpose of this paper is to demonstrate how a very simple algorithm can outperform more complicated ones if the right analysis of its running time is performed.

1 Introduction

INDEPENDENT SET is one of the most important graph problems. Although it is one of the classical NP-complete problems, it allows for very fast exact algorithms. Even the very trivial *branching* algorithm that recursively tries whether a node of degree at least two belongs to an independent set or not yields a runtime of $O^*(1.47^n)$†. More sophisticated algorithms improve this bound by a large margin.

In this paper, we present a new algorithm for INDEPENDENT SET with a runtime of $O^*(1.2132^n)$ that improves over the runtime $O^*(1.2201^n)$ of the previously best published algorithm by Fomin, Grandoni, and Kratsch [5]. Our algorithm is based on their algorithm and is rather simple: We only use two simple branching rules and few simplification rules. The improvement is based on (1) the usage of the new *satellites* branching rule, and (2) on a new kind of a computer-generated proof. The latter enables us to estimate the effects of reduction rules beyond the neighborhood of a single vertex.

Of course, there is a long history of computer-aided proofs, e.g., for the four color theorem [1, 2]. Still, computer-aided proofs are often hard to verify and sometimes regarded as unaesthetic. We propose a framework that hopefully allows a better and easier verification of automated proofs. The INDEPENDENT SET problem is well-suited for our framework, since the efficiency of branching algorithms for INDEPENDENT SET depends mostly on the case distinctions in small induced subgraphs. Our approach is to use a computer to generate all of them and to evaluate the algorithm in every individual case. Only when time or space constraints render it impossible to generate all cases with a computer, we switch to a classical analysis.

We only briefly recall previous results for INDEPENDENT SET: The first algorithm that improves over the trivial bounds is due to Tarjan and Trojanowski [19]. This algorithm,

*Supported by the DFG under grant RO 927/7
†The O^* notation suppresses polynomial factors.

© Kneis, Langer, Rossmanith; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 287–298
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2326

which was introduced as early as in 1977, already has a runtime bound of $O^*(1.261^n)$. Further improvements were achieved by Jian [8] and Robson [17] to $O^*(1.235)$ and $O^*(1.228^n)$, respectively. In the same paper, Robson was also able to prove an upper bound of $O^*(1.211^n)$ using exponential space. It is noteworthy that this is based on the Memorization technique, which cannot be used in algorithms that employ so-called folding to remove nodes of degree two.

Fomin, Grandoni, and Kratsch [5] recently employed their Measure & Conquer technique [6] to a new algorithm for INDEPENDENT SET with a runtime bounded by $O^*(1.2201^n)$ that requires only polynomial space. The algorithm itself is extremely simple and the improved runtime is mainly due to an elegant analysis and a new branching rule using mirrors.

Furthermore, it is worthwhile to mention that there is work in progress that might to lead to an even faster, but very complicated algorithm that is partly computer-generated. A preliminary version was published by Robson as a technical report [16, 18].

2 Preliminaries

Let $G = (V, E)$ denote an undirected graph. The size of a maximum independent set in G is denoted by $\alpha(G)$. For any $v \in V$ and any $i \in \mathbf{N}$, the set of nodes of having distance exactly i to v is denoted by $N^i(v)$, i.e., the neighborhood of v is denoted by $N(v) = N^1(v)$. Similarly, $N^i[v]$ denotes the set of nodes having distance at most i to v, such that $N[v] = N^1[v] = N(v) \cup \{v\}$. The degree of a node $v \in V$, i.e., the number of its neighbors in G, is denoted by $\deg(v) = \deg_G(v)$. We assume the reader is familiar with the basic techniques and notation of branching algorithms, in particular with the concept of *measures* (or *potentials*), *branching vectors*, and their corresponding *branching number*.

The concept of *mirrors* was introduced by Fomin, Grandoni, and Kratsch [5]: For some $v \in V$, a node $u \in N^2(v)$ is called *mirror* of v, if $N(v) \setminus N(u)$ is a clique. We denote the set of of a node v mirrors by $M(v)$. Mirrors allow for efficient branching [5]:

LEMMA 1. *Let $G = (V, E)$ be a graph, and $v \in V$. Then $\alpha(G) = \max\{\alpha(G \setminus (M(v) \cup \{v\})), \alpha(G \setminus N[v]) + 1\}$.*

We also apply the concept of *satellites*, which has only been used in algorithms for sparse graphs [9] before. Figure 1 shows some examples of mirrors and satellites, the latter of which are defined as follows:

DEFINITION 2. *Let G be a graph $v \in V$. A node $u \in N^2(v)$ is called* satellite *of v, if there is some $u' \in N(v)$ such that $N[u'] \setminus N[v] = \{u\}$. The set of satellites of a node v is denoted by $S(v)$, and we also use the notation $S[v] := S(v) \cup \{v\}$.*

Note that simple branching algorithms such as the one by Fomin, Grandoni, and Kratsch or our own algorithm typically perform well when they branch on a node v such that $N^2(v)$ is large. If $N^2(v)$ is rather small, there usually is some mirror u of v and branching on v according to Lemma 1 is still efficient. However, there are also some situations where $N^2(v)$ is small, but v has no mirrors, which is the case in four of the five hardest cases in the analysis of Fomin, Grandoni, and Kratsch [5]. Fortunately, satellites allow us to improve these cases, since by the number of edges between $N(v)$ and $N^2(v)$, we can conclude that a satel-

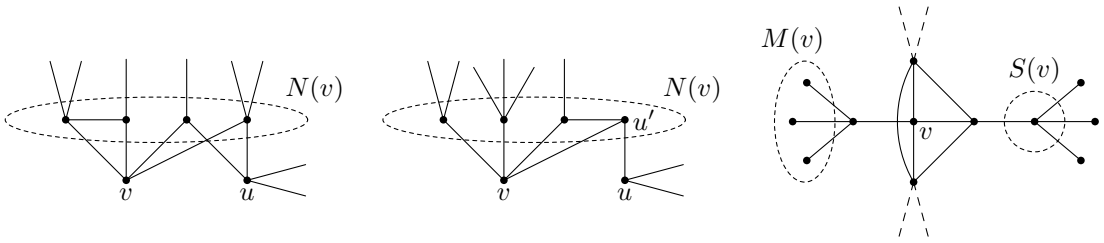


Figure 1: In the graph depicted on the left, u is a mirror of v. In the graph depicted in the middle, u is a satellite of v (through u'). An optimal independent set in the graph on the right contains all nodes in $M(v)$ but no node in $S(v)$. Thus, branching on $G \setminus (\{v\} \cup M(v))$ and $G \setminus N[\{v\} \cup S(v)]$ at the same time does not yield the correct solution.

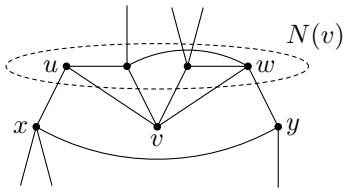


Figure 2: The node v has two adjacent satellites and thus $\alpha(G) = \alpha(G \setminus \{v\})$: If an optimal independent set contains x and v, we can replace v by w. If it contains y and v, we can pick u instead of v.

lite of v must exist, if there is no mirror. The following lemma ([9], Lemma 1) defines the corresponding branching rule.

LEMMA 3. *Let $G = (V, E)$ be a graph, and $v \in V$. Then $\alpha(G) = \max\{\alpha(G \setminus \{v\}), \alpha(G \setminus N[S[v]]) + |S(v)| + 1\}$.*

Note that satellites are particularly useful on graphs with large maximum degree that cannot be analyzed in a computer-aided proof. Unfortunately, we cannot simultaneously branch on mirrors and satellites, as depicted in Figure 1.

Furthermore, our algorithm uses the following well-known reduction rules for independent set that we shortly recall: Firstly, any nodes of degree zero or one can be added to the solution. Similarly, nodes that *dominate*[‡] some other node can be removed from G. Finally, a nodes of degree two not subject to domination can be *folded*, i.e., its neighbors can be merged and the node itself can be removed (see, e.g., [4]). Moreover, Fürer's reduction rule [7] guarantees that each small induced subgraph contains at least three nodes with edges to three distinct nodes in the remaining graph. A precise definition can be found in the appendix.

Finally, satellites can also be used in the following reduction rule (exemplified in Figure 2), which was proven in [9].

LEMMA 4. *Let $G = (V, E)$ be a graph, and $v, u, w \in V$, such that $u, w \in S(v)$ and $\{u, w\} \in E$. Then $\alpha(G) = \alpha(G \setminus \{v\})$.*

[‡]Let $u, v \in V$ be two adjacent nodes. We say u dominates v iff $N[u] \supseteq N[v]$.

We say G is *reduced*, if no further reduction rules can be applied. Moreover, we denote by $R(G)$ the reduced graph obtained from G by applying the reduction rules in some (arbitrarily) fixed order until no more rules can be applied.

Following the Measure & Conquer paradigm [6], we define the following measure on G. This will allow us to prove a better runtime bound than an analysis in $|V|$.

DEFINITION 5. *Let $\varphi_i = 0$ for $i \leq 2$, $\varphi_i = 1$ for $i \geq 7$, $\varphi_3 = 0.474506$, $\varphi_4 = 0.786716$, $\varphi_5 = 0.920901$, and $\varphi_6 = 0.979383$: For a graph $G = (V, E)$ and $v \in V$, we let $\varphi_G(v) := \varphi_{\deg_G(v)}$ and*

$$\varphi(G) = \sum_{v \in V} \varphi_G(v).$$

Obviously, $\varphi(G) \leq |V|$. Any runtime bound in φ therefore immediately implies a runtime bound in $|V|$. Note that the values for φ_i used in this definition are chosen in a way that optimizes the obtained runtime bound. However, these values depend on several thousand recurrences introduced later on, hence they can not be derived easily. We thus used a complex optimization heuristic to compute these values. For $i \geq 5$, φ_i is determined by the runtime on regular graphs of degree i, whereas the values for φ_3 and φ_4 are determined by more complex cases (see [10]).

We write $\varphi(v)$ instead of $\varphi_G(v)$ whenever G can easily be deducted from the context, and let $\Delta_d := \min\{\, \varphi_i - \varphi_{i-1} \mid 4 \leq i \leq d \,\}$ be the minimal measure difference between two nodes in reduced graphs with maximum degree d. Applying the reductions rules does not increase the measure:

LEMMA 6. *Let $G = (V, E)$ be a graph. Then, $\varphi(R(G)) \leq \varphi(G)$.*

PROOF. Removing nodes from the graph respects $\varphi = \varphi(G)$, as some nodes are removed completely and the degree of some adjacent nodes decreases. This does not increase the degree of any node and since $\varphi_i \leq \varphi_{i+1}$ for all $i \in \mathbf{N}$, φ decreases whenever a node is removed.

Whenever a node is folded, its two neighbors u, v are merged. The new node v' can be of higher degree than u and v, but will be at most $\deg(v') \leq \deg(u) + \deg(v) - 2$. Thus, the measure changes by at most $a := \varphi_{\deg(u)+\deg(v)-2} - \varphi_{\deg(u)} - \varphi_{\deg(v)}$. A short computation of all possible combinations shows $a \leq 0$.

Finally, Fürer's reduction rule either removes some nodes of the separator $\{u_1, u_2\}$, adds at most on edge between $\{u_1, u_2\}$ or merges $\{u_1, u_2\}$ into a new node u. Similar to the cases above, removing nodes and merging nodes cannot increase φ. Adding an edge between u_1 and u_2 does not increase φ, because at the same time other edges incident to u_1 and u_2 are removed. ∎

3 A Simple Algorithm for the Independent Set Problem

Combining all the results above, we easily obtain a simple algorithm for the INDEPENDENT SET problem (see Algorithm 1). Its correctness is easy to see, since the reduction rules are valid and the two possible respective branching rules are correct by Lemmas 1 and 3.

Algorithm 1 A fast algorithm for INDEPENDENT SET.

Input: a graph $G = (V, E)$
Output: $\alpha(G)$

```
01: apply reduction rules to G;
02: if G is not connected then compute α for each component independently;
03: if G is cubic then apply algorithm for cubic graphs;
04: select v ∈ V of maximum degree that yields the best branching number;
05: if the mirror branch on v is more efficient than the satellite branch then
06:         return max(α(G \ M[v]), 1 + α(G \ N[v]));
07: else return max(α(G \ {v}), 1 + |S(v)| + α(G \ N[S[v]]));
```

THEOREM 7. *Let $G = (V, E)$. Then, Algorithm 1 correctly returns $\alpha(G)$.*

The remaining part of this paper is devoted to the runtime analysis of Algorithm 1. Basically, this is done by a large case distinction on the effects of the branching and reduction rules when branching on a node v, until we obtain a cubic graph where a faster algorithm exists [9]. If v is of rather high degree, even the trivial algorithm is fast enough. However, with decreasing degree of v, the effects of branching and the subsequent application of reduction rules become more and more important. Down to a degree of five (in general) and for some special cases of degree four, we are still able to give a classical theoretical analysis. However, for the majority of cases having maximum degree of four, even very similar graphs can result in completely different branching vectors. These effects are extremely hard to tackle by an analysis that combines multiple cases. The sharpest runtime bound can be obtained by looking at each possible case individually.

4 A Computer-Aided Case Distinction

Since it is impossible to enumerate each of the millions of possible cases by hand, we use a computer-aided case distinction. Computer-aided proofs are nothing new in the analysis of algorithms, although they still play only a minor role in this field. One example for a computer-aided proof is the algorithm for MAX-2SAT by Kojevnikov and Kulikov [11].

The main problem of computer-generated proofs, and maybe the cause why they are only seldom used, is the complicated verification. While traditional mathematical proofs can be checked rather easily — or at least, we are used to it — this does not hold for computer programs. We therefore propose a framework for computer-aided proofs that allows for an easier verification.

4.1 A General Framework for Computer-Aided Proofs

The first step in any computer-aided proof is to decide which parts of the proof should use the aid of a computer and which parts should be be proven by hand. This step naturally must contain a (traditional) proof of how the computer-aided parts can be incorporated into the traditional proof. The second step is to develop a program that outputs the proof itself and additionally a well-defined *certificate* that lets a reader validate the proof (on a related

note, we refer the reader to the concept of robust and certifying algorithms, see, e.g., [3, 12]). Finally, the proof must be independently validated using the certificate.

We use the framework for the INDEPENDENT SET problem as follows: As outlined above, we want to use a computer-aided proof for certain graphs of maximum degree four, all remaining cases are to be proven by a traditional analysis. Since there are infinitely many graphs of maximum degree four, we only evaluate the branching on a finite number of subgraphs (called *graphlets*, for a formal definition see below). By Theorem 13, this is sufficient.

We developed a computer program that generates all of these graphlets and simulates the two possible branchings (mirrors and satellites) and the subsequent application of the reduction rules. This yields a list of corresponding branching vectors. A complete documentation of this program can be found in [13]. The certificate is given as the complete list of graphlets generated together with their corresponding branching vectors. The certificate and its documentation is publicly available at [10].

In order to verify our proof, one can use the certificate to check (1) whether the certificate is complete, i.e., contains each graphlet or an isomorphic one, (2) whether the corresponding branching vector matches the graphlet, and finally (3) whether the branching vector yields a branching number at most 1.2132.

Finally, an independent team developed programs that validated our certificate, and verified that each of the aforementioned claims actually holds. In the verification team, there was a strong emphasize on clean and simple code so that the verification process can easily be understood by third parties. A full documentation of the verification programs can be found in [15].

We are not aware of any similarly exhaustive approaches to computer-aided proofs that include a formal definition of goals, the proof including a certificate, and particularly an independent verification, with a full documentation of the programs available. An example of an automated proof coming close to our framework are those for MAX-2-SAT by Kojevnikov and Kulikov [11]. Their certificate however does not seem to have been verified independently before publication.

4.2 Generating all Graphlets of Maximum Degree Four

In this section, we give the theoretical foundations for the computer-aided proof. Firstly, we define a notion for reduction rules applied to only a well-defined subgraph of a graph G and show that it this suffices to obtain lower bounds for the real effects of the reduction rules.

DEFINITION 8. *Let $G = (V, E)$ be a graph and let $I \subseteq V$. We define $R_I(G)$ as the graph obtained from G by applying the reduction rules applied to nodes in I only, i.e., (1) remove $u \in I$ if $\deg(u) \leq 1$; (2) remove $u \in I$ if u dominates some $u' \in I$; (3) remove $u \in I$ if u has adjacent satellites $u_1, u_2 \in S(u) \cap I$; (4) if I contains a separator for G of size at most two, apply Fürer's reduction rule; and (5) apply folding to $u \in I$ if $N(u) \subseteq I$.*

From now on, we wlog assume that the reduction rules R on G are always applied in the same order as in R_I.

LEMMA 9. *Let $G = (V, E)$ be a reduced graph of maximum degree d, $I \subseteq V$, $U \subseteq I$. Let $G' = (V', E') = R_I(G \setminus U)$ and let $\Delta_e := \big|\{\, \{u, v\} \in E(G) \mid u \in I, v \notin I \,\}\big| - \big|\{\, \{u, v\} \in E(G') \mid u \in I, v \notin I \,\}\big|$ denote by how much the number of edges between I and $V \setminus I$ changes when applying the reduction rules to I. Then*

$$\begin{aligned} \varphi(R(G \setminus U)) &\leq \varphi(R_I(G \setminus U)) = \varphi(G')) \\ &\leq \varphi(G) - \sum_{v \in I \setminus V'} \varphi_G(v) - \sum_{v \in I \cap V'} \big(\varphi_G(v) - \varphi_{G'}(v)\big) - \Delta_e \min\{\varphi_3/3, \Delta_d\}. \end{aligned}$$

This lemma allows us to evaluate our branching on subgraphs $G[I]$ quite easily: After removing nodes by branching and applying the reduction rules R_I, we can simply count how the degree of all nodes in I changed and add $\min\{\varphi_3/3, \Delta_d\}$ for each removed edge from I to the remaining graph. Note that this is the minimum value each edge contributes to the measure.

DEFINITION 10. *Let $H = (I \cup O, E)$ be graph, such that $I \cap O = \emptyset$, and let $v \in I$ such that $I = N^i[v]$, $O = N^{i+1}(v)$ and $\deg(u) = 1$ for $u \in O$. Moreover, let $\deg(v) \geq deg(u)$ for all $u \in I \cup O$. We call (H, v)* graphlet *of radius i. We call I the* inner nodes *of (H, v) and the set of edges between I and O the* anonymous edges.

Note that the notation of the radius i is motivated by the fact that we are only interested in the number of edges from $N^i(v)$ to $N^{i+1}(v)$ and to which nodes in $N^i(v)$ they are incident. Similarly to Lemma 9, we will restrict our branching and the application of the reduction rules to $I = N^i[v]$.

DEFINITION 11. *Let $G = (V, E)$ be a graph, $v \in V$, and $(H = (I \cup O, E'), v)$ be a graphlet of radius i. We say G contains (H, v) iff (1) $I \subseteq V$, (2) $H[I]$ is an induced subgraph of G, (3) $N_G^{i-1}[v] = I$, and (4) $\deg_G(u) = \deg_H(u)$ for all $u \in I$.*

Note that by these conditions, $|O| = |\{\, \{u, w\} \in E \mid u \in I, w \notin I \,\}|$. While this definition is somewhat technical, the intuition behind it is rather simple: The nodes in I form not only an induced subgraph of G, but I is only connected to $G \setminus I$ via nodes in $N_G^i(v)$. Moreover, the degree of all nodes in I is the same in both graphs and thus the number of edges between $G[I]$ and $G \setminus I$ as well as between $H[I] = G[I]$ and $H \setminus I$ is identical. See Figure 3 for an example.

LEMMA 12. *Let $G = (V, E)$ be a reduced graph that contains a graphlet $(H = (I \cup O, E'), v)$ and $U \subseteq I$. Let $G' = R_I(G \setminus U)$ and $H' = R_I(H \setminus U)$. Then, (1) $G[V \setminus I] = G'[V' \setminus I]$, (2) $\deg_{G'}(v) \leq \deg_{H'}(v)$ for all $v \in I$, and (3) $\big|\{\, \{u, w\} \in E(G') \mid u \in I, w \notin I \,\}\big| \leq \big|\{\, \{u, w\} \in E(H') \mid u \in I, w \notin I \,\}\big|$.*

PROOF. Since we restrict the reduction rules to I, any edge that is not incident to I cannot be affected by the reduction rules. Moreover, only nodes in I can be removed by the restricted reduction rules. Thus, $G[V \setminus I] = G'[V' \setminus I]$.

By induction over the number of applied reduction rules, we easily obtain $G_i[I] = H_i[I]$ and $\deg_{G_i}(u) \leq \deg_{H_i}(u)$ for all $u \in I$, where G_i (and H_i, resp.) denotes the graph G (and H, resp.) after i reduction steps:

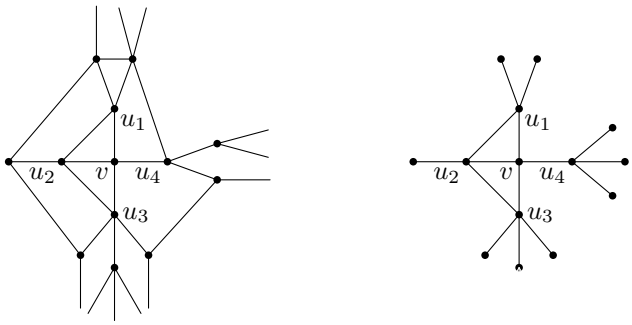


Figure 3: The graph G on the left contains the graphlet (H, v) of orbit 1 on the right. Note that u_1 and u_4 have a common neighbor in $N^2(v)$ in G, but not in (H, v).

1. If I contains a separator of size two in H_i, this is also a separator in G_i and vice versa. Moreover, $G_i[I] = H_i[I]$ implies that Fürer's reduction rule is applied in exactly the same way in both graphs, as the optimal independent sets of these graphs are the same. Hence, $G_{i+1}[I] = H_{i+1}[I]$ and $\deg_{G_{i+1}}(u) \leq \deg_{H_{i+1}}(u)$ for all $u \in I$.

2. Let $u \in I$ be a node that is removed by one of the reduction rules. Removing u in G_i and removing u in H_i removes exactly the same edges within $G_i[I] = H_i[I]$ before the node is removed. Thus, after the removal $\deg_{G_{i+1}}(w) \leq \deg_{H_{i+1}}(w)$ for all $w \in I$ and again $G_{i+1}[I] = H_{i+1}[I]$.

3. Let $u \in I$ be a node that is subject to folding in R_I. By definition of R_I, both neighbors u_1, u_2 of u must belong to I. In G_i as well as in H_i, any edge $\{u_2, w\}$ becomes the new edge $\{u_1, w\}$, Moreover, u and u_2 are removed in both graphs. Therefore, $G_{i+1}[I] = H_{i+1}[I]$ holds after folding u. Since only edges incident to u_2 are changed, we have $\deg_{G_{i+1}}(w) \leq \deg_{H_{i+1}}(w)$ for all $w \in I \setminus \{u_1\}$.

Let $S_H = (N_{H_i}(u_1) \cap N_{H_i}(u_2)) \setminus \{u\}$ and $S_G = (N_{G_i}(u_1) \cap N_{G_i}(u_2)) \setminus \{u\}$. Then $S_H \subseteq S_G$, as the only common neighbors of u_1 and u_2 in H_i must be in I and $G_i[I] = H_i[I]$. But then, $\deg_{G_{i+1}}(u_1) = \deg_{G_i}(u_1) + \deg_{G_i}(u_2) - 2 - |S_G|$ and $\deg_{H_{i+1}}(u_1) = \deg_{H_i}(u_1) + \deg_{H_i}(u_2) - 2 - |S_H|$ imply $\deg_{G_{i+1}}(u_1) \leq \deg_{H_{i+1}}(u_1)$ by induction. We obtain

$$\left|\left\{\, \{u, w\} \in E(G') \mid u \in I, w \notin I \,\right\}\right| \leq \left|\left\{\, \{u, w\} \in E(H') \mid u \in I, w \notin I \,\right\}\right|$$

as a direct consequence of this. ∎

Combining the results above, we can now conclude that is sufficient to evaluate our branching algorithm on graphlets of some fixed radius. After branching and applying the reduction rules to the inner nodes of the graphlet, we only need to analyze how the inner nodes changed and how many anonymous edges are removed to obtain a branching number and (together with the remaining cases) an upper bound for the runtime of Algorithm 1.

THEOREM 13. *Let $G = (V, E)$ be a reduced graph of maximum degree d that contains the graphlet $(H = (I \cup O, E'), v)$. Let $U \subseteq I$ and $H' = R_I(H \setminus U)$. Then*

$$\begin{aligned}\varphi(G) - \varphi(R(G \setminus U)) \;\geq\; & \sum_{u \in I \cap V(H')} \varphi_H(u) - \varphi_{H'}(u) + \sum_{u \in I \setminus V(H')} \varphi_H(u) \\ & + \Delta_{E(H)} \min\{\varphi_3/3, \Delta_d\},\end{aligned}$$

where

$$\Delta_{E(H)} := \left|\{\, \{u,w\} \in E(H) \mid u \in I, w \in O \,\}\right| - \left|\{\, \{u,w\} \in E(H') \mid u \in I, w \in O \,\}\right|$$

denotes the number of anonymous edges that are removed by the reduction rules.

PROOF. Let $G' = R_I(G \setminus U)$. By Lemma 9, we have

$$\begin{aligned}\varphi(G) - \varphi(R(G \setminus U)) \;\geq\; & \sum_{u \in I \cap V(G')} \varphi_G(u) - \varphi_{R_I(G \setminus U)}(u) + \sum_{u \in I \setminus V(G')} \varphi_G(u) \\ & + \Delta_{E(G)} \min\{\varphi_3/3, \Delta_d\},\end{aligned}$$

where

$$\Delta_{E(G)} = \left|\{\, \{u,w\} \in E(G) \mid u \in I, w \notin I \,\}\right| - \left|\{\, \{u,w\} \in E(G') \mid u \in I, w \notin I \,\}\right|.$$

Since G contains the graphlet (H, v), we have

$$\left|\{\, \{u,w\} \in E(G) \mid u \in I, w \notin I \,\}\right| = \left|\{\, \{u,w\} \in E(H) \mid u \in I, w \notin I \,\}\right|.$$

Thus, statement (3) from Lemma 12 yields $\Delta_{E(G)} \geq \Delta_{E(H)}$. By the definition of graphlets, $\varphi_G(u) = \varphi_H(u)$ for all $u \in I$. Moreover, Lemma 12 implies $\deg_{G'}(u) \leq \deg_{H'}(u)$ for all $u \in I$. Hence, $\varphi_{G'}(u) \leq \varphi_{H'}(u)$ for all $u \in I$ and we obtain the claimed estimation. ∎

We can now use use a computer-aided proof for the following theorem:

THEOREM 14. *Let $G = (V, E)$ be a reduced graph of maximum degree four and let $v \in V$ such that $\deg(v) = 4$ and $|N^2(v)| \leq 7$. Then branching on v as described in Algorithm 1 yields a branching with a branching number of at most* 1.2132.

PROOF. Let $\mathcal{H}$ denote the set of all graphlets (H, v) of radius 2 such that $\deg(v) = 4$ and $|N^2(v)| \leq 7$. Then G contains some graphlet $(H', v) \in \mathcal{H}$. By Theorem 13, it is sufficient to simulate the branching on (H', v) and count how the node of the inner nodes changes and how many anonymous edges are removed.

We now have a formal specification of what the computer shall compute as required by the framework outlined in the previous section. Generating all graphlets and computing the branching vectors yields a branching number of at most 1.2132. The certificate is publicly available at [10] and a complete description of the generation and verification programs can be found in [13] and [15], respectively. ∎

5 A Traditional Analysis of the Remaining Cases

Finally, we give an traditional analysis for the remaining cases. Due to the combinatorial explosion, it is impossible to use a computer-aided case distinction for these cases using the methods described in the previous section.

Once the graph is cubic, we apply the algorithm for sparse graphs by Razgon [14], which solves INDEPENDENT SET on cubic graphs in time $O^*(1.0892^n)$. However, we need to be careful because in general, $n \geq \varphi(G)$, but we measure the running time in the latter. Rewriting the statement in terms of our measure, we obtain the following bound.

COROLLARY 15. *Let $G = (V, E)$ be a reduced cubic graph, i.e., $\varphi(G) = n\varphi_3$. Then, Algorithm 1 solves* INDEPENDENT SET *on G in time $O^*(1.0892^n) = O^*(1.0892^{\varphi(G)/\varphi_3}) \leq O^*(1.198^{\varphi(G)})$.*

Please note that we could use a slower but simpler algorithm on cubic graphs, as long as its runtime is at most $O^*(1.096^n)$. For increased readability, we will denote the measure difference between G and $R(G \setminus U)$, for $U \subseteq V$, by $\Delta_\varphi(U) := \varphi(G) - \varphi(R(G \setminus U))$.

For graphs of maximum degree four, we only need to handle the case where $|N^2(v)| \geq 8$. Since a lot of nodes are affected in this case, we easily obtain a good runtime bound.

LEMMA 16. *Let $G = (V, E)$ be a reduced graph of maximum degree four. Let $v \in V$ such that $\deg(v) = 4$ and $|N^2(v)| \geq 8$. Then branching on v as described in Algorithm 1 yields a branching with a branching number of at most 1.201.*

PROOF. In $G \setminus N[v]$, the degree of all nodes in $N^2(v)$ is reduced by at least one. Thus, the measure changes by at least $8 \min\{\varphi_3, \varphi_4 - \varphi 3\}$. Let $d_3 = |\{\, u \in N(v) \mid \deg(u) = 3 \,\}|$. Then $\Delta_\varphi(\{v\}) = \varphi_4 + d_3\varphi_3 + (4 - d_3)(\varphi_4 - \varphi_3)$ and $\Delta_\varphi(N[v]) = \varphi_4 + d_3\varphi_3 + (4 - d_3)\varphi_4$. Computing all five possible branching vectors $\big(\varphi_4 + d_3\varphi_3 + (4 - d_3)(\varphi_4 - \varphi_3), \varphi_4 + d_3\varphi_3 + (4 - d_3)\varphi_4\big)$ yields the desired bound. ∎

Now, only the analysis for graphs of higher degree remains. A complete list of the respective branching vectors and their corresponding branching numbers obtained in the following lemmas can be found at [10].

LEMMA 17. *Let $G = (V, E)$ be a reduced graph of maximum degree $d \geq 5$ and let $v \in V$ such that $\deg(v) = d$. Moreover, let $M(v) = \emptyset$ and $S(v) = \emptyset$. Then branching on v as described in Algorithm 1 yields a branching with a branching number of at most 1.2132.*

LEMMA 18. *Let $G = (V, E)$ be a reduced graph of maximum degree $d \geq 5$ and let $v \in V$ such that $\deg(v) = d$. Moreover, let $u \in M(v)$. Then branching on v as described in Algorithm 1 yields a branching with a branching number of at most 1.2132.*

PROOF. Let $l = \deg(u)$, $S := N(v) \cap N(u)$ and $s := |N(v) \cap N(u)|$. Moreover, let $T := (N(v) \cup N(u)) \setminus S$. Note that the degree of all nodes in S decreases by at least two in $R(G \setminus \{v, u\})$. Therefore, we have

$$\Delta_\varphi(\{v, u\}) \geq \varphi_d + \varphi_l + \sum_{w \in T} (\varphi_{\deg(w)} - \varphi_{\deg(w)-1}) + \sum_{w \in S} (\varphi_{\deg(w)} - \varphi_{\deg(w)-2}) \text{ and}$$

$$\Delta_\varphi(N[v]) \geq \varphi_d + \varphi_l - \varphi_{l-s} + \sum_{w \in N(v)} \varphi_{\deg(w)} + (d - s) \min\{\varphi_3, \Delta_d\},$$

which yields a good enough branching vector $(\Delta_\varphi(\{v, u\}), \Delta_\varphi(N[v]))$ for all cases. ∎

LEMMA 19. *Let $G = (V, E)$ be a reduced graph of maximum degree $d \geq 5$ and let $v \in V$ such that $\deg(v) = d$. Moreover, let $S(v) \neq \emptyset$ and let $M(v) = \emptyset$. Then branching on v as described in Algorithm 1 yields a branching with a branching number of at most 1.2132.*

PROOF. Note that $M(v) = \emptyset$ implies that each node in $N^2(v)$ has at most $d-2$ neighbors in $N(v)$. Assume $|S(v)| \geq 2$ or $S(v) = \{u\}$ and $N(u) \setminus N(v) \neq \emptyset$. Since wlog $V \setminus (N[v] \cup N[S(v)]) \neq \emptyset$ (otherwise the graph is of constant size), we obtain the branching vectors

$$\Big(\varphi_d + \sum_{w \in N(v)} (\varphi_{\deg(w)} - \varphi_{\deg(w)-1}), \varphi_d + \sum_{w \in N(v)} \varphi_{\deg(w)} + 2\varphi_3 + 3\min\{\varphi_3/3, \Delta_d\}\Big),$$

because at least two nodes in $N^2(v)$ of degree at least three are removed and at least three edges connect the corresponding graph to the remaining graph. Otherwise, G contains a separator of size two.

Finally, let $S(v) = \{u\}$ and $N(u) \subseteq N(v)$. Let $\deg(u) = d'$. Since at least $d - d'$ nodes in $N(v)$ have at least two neighbors in $N^2(v)$ (otherwise, $|S(v) > 1|$), we obtain the branching vector

$$\Big(\varphi_d + \sum_{w \in N(v)} (\varphi_{\deg(w)} - \varphi_{\deg(w)-1}), \varphi_d + \varphi_{d'} + \sum_{w \in N(v)} \varphi_{\deg(w)} + 2(d-d')\min\{\varphi_3/3, \Delta_d\}\Big).$$

Again, these branching vectors are good enough except if $\deg(v) = 5$ and all neighbors of v are of degree of five as well. But then we can branch on a neighbor v' of v, such that $N(v')$ contains a node of degree four or less, because the satellite is no mirror and hence of degree 3. ∎

We now easily obtain our main result:

THEOREM 20. *Let $G = (V, E)$ be a graph. Algorithm 1 solves* INDEPENDENT SET *on G in time bounded by $O^*(1.2132^n)$.*

PROOF. First note that for graphs of maximum degree $d > 7$, even the simple branching vector

$$\Big(\varphi_d + \sum_{w \in N(v)} (\varphi_{\deg(w)} - \varphi_{\deg(w)-1}), \varphi_d + \sum_{w \in N(v)} \varphi_{\deg(w)}\Big)$$

is good enough. Also note that $\varphi_i = \varphi_7$ for all $i \geq 8$, and thus increasing the maximum degree to values larger than 8 can never yield a worse branching vector than for a smaller maximum degree, as $N(v)$ contains only more neighbors and it makes no difference whether $N(v)$ contains node of degree 8 or a node of higher degree. A complete list of the respective branching vectors for graphs of maximum degree 8 can be found at [10].

For graphs of maximum degree at most seven, the runtime bound follows from Lemmas 17, 18, and 19, Lemma 16 and Theorem 14, as well as Corollary 15. ∎

6 Conclusion

Although it took some considerable effort to analyze the running time of our algorithm for INDEPENDENT SET, in particular the computer-aided part and its independent verification, we believe the results legitimate this effort. Hopefully, the proposed framework is able to resolve some doubt regarding computer-aided proofs, especially since the certificate can be (and already has been) used to independently validate the proof. We hope that this approach can be used for other problems as well.

References

[1] K. Appel and W. Haken. Solution of the four color map problem. *Scientific American*, 237:108–121, 1977.

[2] K. Appel, W. Haken, and J. Koch. Every planar map is four colorable. *Journal of Mathematics*, 21:439–567, 1977.

[3] M. Blum and S. Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, 1995.

[4] J. Chen, I. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. In *Proc. of 25th WG*, number 1665 in LNCS, pages 313–324. Springer, 1999.

[5] F. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: A simple $O(2^{0.288n})$ independent set algorithm. In *Proc. of 17th SODA*, pages 18–25, 2006.

[6] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: Domination – A case study. In *Proc. of 32nd ICALP*, LNCS, pages 191–203. Springer, 2005.

[7] M. Fürer. A faster algorithm for finding maximum independent sets in sparse graphs. In *Proc. of 7th LATIN*, number 3887 in LNCS, pages 491–501. Springer, 2006.

[8] T. Jian. An $O(2^{0.304n})$ algorithm for solving Maximum Independent Set problem. *IEEE Transactions on Computers*, 35(9):847–851, 1986.

[9] J. Kneis and A. Langer. Satellites and mirrors for solving independent set on sparse graphs. Technical Report AIB-2009-08, RWTH Aachen University, April 2009.

[10] J. Kneis, A. Langer, and P. Rossmanith. Independent set proof homepage, 2009. `http://www.tcs.rwth-aachen.de/independentset/`.

[11] A. Kojevnikov and A. S. Kulikov. A new approach to proving upper bounds for MAX-2-SAT. In *Proc. of 17th SODA*, pages 11–17, 2006.

[12] D. Kratsch, R. M. McConnell, K. Mehlhorn, and J. Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. *SIAM J. Comput.*, 36(2):326–353, 2006.

[13] M. Nett. Generating all relevant cases for the maximum independent set problem. Technical Report AIB-2009-09, RWTH Aachen University, 2009.

[14] I. Razgon. Faster computation of maximum independent set and parameterized vertex cover for graphs with maximum degree 3. *J. Discrete Algorithms*, 7(2):191–212, 2009.

[15] F. Reidl and F. Sánchez Villaamil. Automatic verification of the correctness of the upper bound of a maximum independent set algorithm. Technical Report AIB-2009-10, RWTH Aachen University, 2009.

[16] J. M. Robson. Personal communication.

[17] J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7:425–440, 1986.

[18] J. M. Robson. Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, Université Bordeaux I, LaBRI, 2001.

[19] R. E. Tarjan and A. E. Trojanowski. Finding a Maximum Independent Set. *SIAM Journal on Computing*, 6(3):537–550, 1977.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Using Elimination Theory to construct Rigid Matrices

Abhinav Kumar*, Satyanarayana V. Lokam†, Vijay M. Patankar†, Jayalal Sarma M. N.‡

ABSTRACT. The rigidity of a matrix A for target rank r is the minimum number of entries of A that must be changed to ensure that the rank of the altered matrix is at most r. Since its introduction by Valiant [22], rigidity and similar rank-robustness functions of matrices have found numerous applications in circuit complexity, communication complexity, and learning complexity. Almost all $n \times n$ matrices over an infinite field have a rigidity of $(n-r)^2$. It is a long-standing open question to construct infinite families of *explicit* matrices even with superlinear rigidity when $r = \Omega(n)$.
In this paper, we construct an infinite family of complex matrices with the largest possible, i.e., $(n-r)^2$, rigidity. The entries of an $n \times n$ matrix in this family are distinct primitive roots of unity of orders roughly $\exp(n^4 \log n)$. To the best of our knowledge, this is the first family of concrete (but not entirely explicit) matrices having maximal rigidity and a succinct algebraic description.
Our construction is based on elimination theory of polynomial ideals. In particular, we use results on the existence of polynomials in elimination ideals with effective degree upper bounds (effective Nullstellensatz). Using elementary algebraic geometry, we prove that the dimension of the affine variety of matrices of rigidity at most k is exactly $n^2 - (n-r)^2 + k$. Finally, we use elimination theory to examine whether the rigidity function is semicontinuous.

1 Introduction

Valiant [22] introduced the notion of matrix rigidity. The rigidity function $\mathrm{Rig}(A,r)$ of a matrix A for target rank r is defined to be the smallest number of entries of A that must be changed to ensure that the altered matrix has rank at most r. It is easy to see that for every $n \times n$ matrix A (over any field), $\mathrm{Rig}(A,r) \leqslant (n-r)^2$. Valiant also showed that, over an infinite field, almost all matrices have rigidity exactly $(n-r)^2$. It is a long-standing open question to construct infinite families of *explicit* matrices with superlinear rigidity for $r = \Omega(n)$. Here, by an explicit family, we mean that the $n \times n$ matrix in the family is computable by a deterministic Turing machine in time polynomial in n or by a Boolean circuit of size polynomial in n. Lower bounds on rigidity of explicit matrices are motivated by their numerous applications in complexity theory. In particular, Valiant showed that lower bounds of the form $\mathrm{Rig}(A, \epsilon n) = n^{1+\delta}$ (where ϵ and δ are some positive constants) imply that the linear transformation defined by A cannot be computed by arithmetic circuits of linear size and logarithmic depth consisting of gates that compute linear functions of their inputs. Since then, applications of lower bounds on rigidity and similar rank-robustness functions have been found in circuit complexity, communication complexity, and learning complexity ([7],

*abhinav@math.mit.edu, Department of Mathematics, MIT, USA.
†{satya,vij}@microsoft.com, Microsoft Research India, Bangalore, India.
‡jayalal@tsinghua.edu.cn, Institute for Theoretical Computer Science, Tsinghua University, Beijing, China. This work was supported in part by the National Natural Science Foundation of China Grant 60553001, the National Basic Research Program of China Grant 2007CB807900, 2007CB807901.

© Kumar,Lokam,Patankar,Sarma; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 299–310
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2327

[13], [15], [18], [19]). Two comprehensive surveys on this topic are [4] and [5]. Over finite fields, the best known lower bound for explicit A was first proved by Friedman [8] and is $\mathrm{Rig}(A,r) = \Omega(\frac{n^2}{r}\log\frac{n}{r})$ for parity check matrices of good error-correcting codes. Over infinite fields, the same lower bound was proved by Shokrollahi, Spielman, and Stemann [21] for Cauchy matrices, Discrete Fourier Transform matrices of prime order (see [14]), and other families. Note that this type of lower bound reduces to the trivial $\mathrm{Rig}(A,r) = \Omega(n)$ when $r = \Omega(n)$. In [16], lower bounds of the form $\mathrm{Rig}(A,\epsilon n) = \Omega(n^2)$ were proved when $A = (\sqrt{p_{jk}})$ or when $A = (\exp(2\pi \mathrm{i}/p_{jk}))$, where p_{jk} are the first n^2 primes. These matrices, however, are not explicit in the sense defined above.

In this paper, we construct an infinite family of complex matrices with the highest possible, i.e., $(n-r)^2$ rigidity. The entries of the $n \times n$ matrix in this family are primitive roots of unity of orders roughly $\exp(n^4 \log n)$. We show that the real parts of these matrices are also maximally rigid. Like the matrices in [16], this family of matrices is not explicit in the sense of efficient computability described earlier. However, one of the motivations for studying rigidity comes from algebraic complexity. In the world of algebraic complexity, any element of the ground field (in our case $\mathbb{C}$) is considered a primitive or atomic object. In this sense, the matrices we construct are explicitly described algebraic entities. To the best of our knowledge, this is the first construction giving an infinite family of non-generic/concrete matrices with maximum rigidity. It is still unsatisfactory, though, that the roots of unity in our matrices have orders exponential in n. Earlier constructions in [16] use roots of unity of orders $O(n^2)$ but the bounds on rigidity proved there are weaker: $n(n-cr)$ for some constant $c > 2$.

We pursue a general approach to studying rigidity based on elementary algebraic geometry and elimination theory. To set up the formalism of this approach, we begin by reproving Valiant's result that the set of matrices of rigidity less than $(n-r)^2$ form a Zariski closed set in $\mathbb{C}^{n\times n}$, i.e., such matrices are solutions of a finite system of polynomial equations (hence a generic matrix has rigidity at least $(n-r)^2$). In fact, we prove a more general statement: the set of matrices of rigidity at most k has dimension (as an affine variety) exactly $n^2 - (n-r)^2 + k$. This sheds light on the geometric structure of rigid matrices. Our transversality argument in this context is clearer and cleaner than an earlier attempt in this direction (in the projective setting) by [11]. To look for specific matrices of high rigidity, we consider certain elimination ideals associated to matrices with rigidity at most k. A result in [1] using effective Nullstellensatz bounds [2], [9] shows that an elimination ideal of a polynomial ideal must always contain a nonzero polynomial with an explicit degree upper bound (Theorem 8). We then use simple facts from algebraic number theory to prove that a matrix whose entries are primitive roots of sufficiently high orders cannot satisfy any polynomial with such a degree upper bound. This gives us the claimed family of matrices of maximum rigidity.

Our primary objects of interest in this paper are the varieties of matrices with rigidity at most k. For a fixed k, we have a natural decomposition of this variety based on the patterns of changes. We prove that this natural decomposition is indeed a decomposition into *irreducible* components (Corollary 13). In fact, these components are defined by elimination ideals of determinantal ideals generated by all the $(r+1)\times(r+1)$ minors of an $n \times n$ matrix of indeterminates. Better effective upper bounds on the degree of a nonzero polynomial in

the elimination ideal of determinantal ideals than given by Theorem 8 would lead to similar improvements in the bound on the order of the primitive roots of unity we use to construct our rigid matrices. While determinantal ideals have been well-studied in mathematical literature, their elimination theory does not seem to have been as well-studied. Application to rigidity of these elimination ideals of determinantal ideals might be a natural motivation for studying them.

We next consider the question: given a matrix A, is there a small neighborhood of A within which the rigidity function is nondecreasing, i.e. such that every matrix in this neighborhood has rigidity at least equal to that of A? This is related to the notion of *semicontinuity* of the rigidity function. We give a family of examples to show that the rigidity function is in general not semicontinuous. However, the *specific* matrices we produce above, by their very construction, have neighborhoods within which rigidity is nondecreasing.

1.1 Definitions and Notations

Let F be a field. Then, by $M_n(F)$ we denote the algebra of $n \times n$ matrices over F. At times, when it is clear from the context, we will denote $M_n(F)$ by M_n. Let $X \in M_n(F)$. Then by X_{ij} we will denote the (i,j)-th entry of X. Given $X \in M_n(F)$, the support of X is defined as $Supp(X) := \{(i,j) \mid X_{ij} \neq 0 \in F\}$. Given a non-negative integer k, we define

$$S(k) := \{X \in M_n(F) : |Supp(X)| \leqslant k\}.$$

Thus, $S(k)$ is the set of matrices over F with at most k non-zero entries. A *pattern* π is a subset of the positions of an $n \times n$ matrix. Then, we define:

$$S(\pi) := \{X \in M_n(F) : \ Supp(X) \subseteq \pi\}.$$

Note that $S(k) = \cup_{|\pi|=k} S(\pi)$.

We say that a matrix X is (r,k)-rigid if changing at most k entries of X does not bring down the rank of the matrix to a value $\leqslant r$. More formally,

DEFINITION 1. *A matrix X is (r,k)-rigid if $rank(X+T) > r$ whenever $T \in S(k)$.*

DEFINITION 2. *The rigidity function $\mathrm{Rig}(X,r)$ is the smallest integer k for which the matrix X is not (r,k)-rigid. That is, $\mathrm{Rig}(X,r)$ is the minimum number of entries we need to change in the matrix X so that the rank becomes at most r:*

$$\mathrm{Rig}(X,r) := \min\{Supp(T) \ : \ \mathrm{rank}(X+T) \leqslant r\}.$$

Sometimes, we will allow T to be chosen in $M_n(L)$ for L an extension field of F. In this case we will denote the rigidity by $Rig(X,r,L)$.

Let $\mathsf{RIG}(n,r,k)$ denote the set of $n \times n$ matrices X such that $\mathrm{Rig}(X,r) = k$. Similarly, we define $\mathsf{RIG}(n,r,\geqslant k)$ to be the set of matrices of rigidity at least k and $\mathsf{RIG}(n,r,\leqslant k)$ to be the set of matrices of rigidity at most k. For a pattern π of size k, let $\mathsf{RIG}(n,r,\pi)$ be the set of matrices X such that for some $T_\pi \in S(\pi)$ we have $\mathrm{rank}(X+T_\pi) \leqslant r$. Then we have

$$\mathsf{RIG}(n,r,\leqslant k) = \bigcup_{\pi,|\pi|=k} \mathsf{RIG}(n,r,\pi).$$

1.2 Elimination Theory: Closure Theorem

We refer the reader to a standard text in algebraic geometry [6, 20] for the necessary background. Here we recall a basic result from Elimination Theory which is directly used in the paper. As the name suggests, Elimination Theory deals with elimination of a subset of variables from a given set of polynomial equations and finding the *reduced set* of polynomial equations (not involving the eliminated variables). The main results of Elimination Theory, especially the Closure Theorem, describe a precise relation between the reduced ideal and the given ideal, and its corresponding geometric interpretation.

Given an ideal $I = \langle f_1, \ldots, f_s \rangle \subseteq F[x_1, \ldots, x_n]$, the l-th *elimination ideal* I_l is the ideal of $F[x_{l+1}, \ldots, x_n]$ defined by $I_l := I \cap F[x_{l+1}, \ldots, x_n]$.

THEOREM 3.(Closure Theorem, page 125, Theorem 3 of [6])
Let I be an ideal of $F[x_1, \ldots, x_n, y_1, \ldots, y_m]$ and $I_n := I \bigcap F[y_1, \ldots, y_m]$ be the n-th elimination ideal associated to I. Let $V(I)$ and $V(I_n)$ be the subvarieties of $\mathbb{A}^{n+m}$ and $\mathbb{A}^m$ (the affine spaces over F of dimension $n+m$ and m respectively) defined by I and I_n respectively. Let p be the natural projection map from $\mathbb{A}^{n+m} \to \mathbb{A}^m$ (projection map onto the y-coordinates). Then,

1. *$V(I_n)$ is the smallest (closed) affine variety containing $p(V(I)) \subseteq \mathbb{A}^m$. In other words, $V(I_n)$ is the Zariski closure of $p(V(I))(\bar{F}) \subseteq \bar{F}^m$.*
2. *When $V(I)(\bar{F}) \neq \phi$, there is an affine variety W strictly contained in $V(I_n)$ such that $V(I_n) - W \subseteq p(V(I))$.*

2 Use of Elimination Theory

2.1 Determinantal Ideals and their Elimination Ideals

We would like to investigate the structure of the sets $\mathsf{RIG}(n, r, \leqslant k)$ and $\mathsf{RIG}(n, r, \pi)$ and their Zariski closures

$$\begin{aligned} \mathcal{W}(n, r, \leqslant k) &:= \overline{\mathsf{RIG}(n, r, \leqslant k)} \quad \text{and} \\ \mathcal{W}(n, r, \pi) &:= \overline{\mathsf{RIG}(n, r, \pi)} \end{aligned}$$

in the n^2-dimensional affine space of $n \times n$ matrices. Let X be an $n \times n$ matrix with entries being indeterminates $x_1, \ldots, x_{n^2}$. For a pattern π of k positions, let T_π be the $n \times n$ matrix with indeterminates $t_1, \ldots, t_k$ in the positions given by π. Note that saying $X + T_\pi$ has rank at most r is equivalent to saying that all its $(r+1) \times (r+1)$ minors vanish. Let us consider the ideal generated by these minors:

$$I(n, r, \pi) := \left\langle Minors_{(r+1)\times(r+1)}(X + T_\pi) \right\rangle \subseteq F[x_1, \ldots, x_{n^2}, t_1, \ldots, t_k]. \tag{1}$$

It then follows from the definition of rigidity that $\mathsf{RIG}(n, r, \pi)$ is the projection from $\mathbb{A}^{n^2} \times \mathbb{A}^k$ to $\mathbb{A}^{n^2}$ of the algebraic set $V(I(n, r, \pi))(F)$. Thus, if we define the elimination ideal

$$EI(n, r, \pi) := I(n, r, \pi) \cap F[x_1, \ldots, x_{n^2}] \subseteq F[x_1, \ldots, x_{n^2}],$$

then by the Closure Theorem (Theorem 3), we obtain

$$\mathcal{W}(n,r,\pi) = V(EI(n,r,\pi)). \tag{2}$$

Note that

$$\mathcal{W}(n,r,\leqslant k) = \bigcup_{\pi,|\pi|=k} \mathcal{W}(n,r,\pi).$$

2.2 Valiant's Theorem

The following theorem due to Valiant [22, Theorem 6.4, page 172] says that a generic matrix has rigidity $(n-r)^2$. That is, for $k < (n-r)^2$, the dimension of $\mathcal{W}(n,r,\leqslant k)$ is strictly less than n^2.

A reader familiar with Valiant's proof will realize that our proof is basically a rephrasing of Valiant's proof in the language of algebraic geometry. The point of this proof is to set up the formalism and use it later; in particular, when we compute the exact dimension of the rigidity variety $\mathcal{W}(n,r,\leqslant k)$.

THEOREM 4.(Valiant) *Let $n \geqslant 1, 0 < r < n$ and $0 \leqslant k < (n-r)^2$. Let $\mathcal{W} := \mathcal{W}(n,r,\leqslant k)$ be as above. Then,*

$$\dim(\mathcal{W}) < n^2.$$

PROOF. Let $\pi \subseteq \{(i,j) | 1 \leqslant i,j \leqslant n\}$ be a pattern of size k. Let τ be the index set of a fixed $r \times r$ minor. For a matrix B, let B_τ denote the minor of B indexed by τ. Define $\mathsf{RIG}(n,r,\pi,\tau)$ to be the set of all $n \times n$ matrices A that satisfy the following properties: there exists some $n \times n$ matrix T_π such that

1. $Supp(T_\pi) \subseteq \pi$,
2. $\mathrm{rank}(A + T_\pi) = r$, and
3. $\det((A+T_\pi)_\tau) \neq 0$ where τ denotes the fixed $r \times r$ minor as above.

Recall that $S(\pi)$ is the set of matrices whose support is contained in π. Let us also define

$$\mathsf{RANK}(n,r,\tau) := \{C \in M_n \mid \mathrm{rank}(C) = r \text{ and } \det(C_\tau) \neq 0\}.$$

By definition, every element $A \in \mathsf{RIG}(n,r,\pi,\tau)$ can be written as $C - T_\pi$, with $C \in \mathsf{RANK}(n,r,\tau)$ and $T_\pi \in S(\pi)$.
We state the following lemma without proof. (Details can be found in the full version [10]).

LEMMA 5. $\dim(\mathsf{RANK}(n,r,\tau)) = n^2 - (n-r)^2$.

Consider the following natural map Φ:

$$\mathbb{A}^{n^2-(n-r)^2} \times \mathbb{A}^k \supset \mathsf{RANK}(n,r,\tau) \times S(\pi) \xrightarrow{\Phi} M_n \cong \mathbb{A}^{n^2}, \tag{3}$$

taking (X, T_π) to $X + T_\pi$. The image of Φ is exactly $\mathsf{RIG}(n,r,\pi,\tau)$.

Also, note that $\dim(S(\pi)) = |\pi|$. We note that if there is a surjective morphism from an affine variety X to another affine variety Y, then $\dim Y \leqslant \dim X$ (we defer a formal statement to the full version [10]). Thus for $k \leqslant (n-r)^2 - 1$, we get

$$\dim(\overline{Im(\Phi)}) = \dim(\overline{\mathsf{RIG}(n,r,\pi,\tau)}) \leqslant n^2 - (n-r)^2 + k < n^2.$$

Note that

$$\mathcal{W} = \bigcup_{\tau,\pi} \overline{\mathsf{RIG}(n, r, \pi, \tau)}$$

and that completes the proof of the theorem. ∎

Thus we have proved that the set of matrices of rigidity strictly smaller than $(n-r)^2$ is contained in a proper closed affine variety of $\mathbb{A}^{n^2}$, and thus is of dimension strictly less than n^2. In other words, a *generic matrix*, i.e. a matrix that lies outside a certain proper closed affine subvariety of $\mathbb{A}^{n^2}$, is *maximally rigid*. Therefore, over an infinite field F (for instance, an algebraically closed field), there always exist maximally rigid matrices.

We now refine Valiant's argument and prove the following exact bound on the dimension of $\mathcal{W}$. The main point of the proof is a *lower bound* on $\dim(\mathcal{W})$.

THEOREM 6. *Let* $0 \leqslant r \leqslant n$ *and* $0 \leqslant k \leqslant (n-r)^2$. *Then*

$$\dim(\mathcal{W}) = n^2 - (n-r)^2 + k.$$

PROOF. With the notation of the previous proof, we have the map

$$\Phi : \mathsf{RANK}(n, r, \tau) \times S(\pi) \to M_n.$$

defined above. Let $\mathsf{RANK}(n, \leqslant r), \mathsf{RANK}(n, r)$ be the set of $n \times n$ matrices of rank at most r and exactly r respectively. Let $S(k)$ be the set of matrices of support at most k.

Now note that $GL(n) \times GL(n)$ acts on $\mathsf{RANK}(n, \leqslant r)$ by multiplication on the left and the right, and that the action is transitive on the set of matrices with rank exactly r, which forms a Zariski open subset of $\mathsf{RANK}(n, \leqslant r)$. Therefore $\mathsf{RANK}(n, \leqslant r)$ is an irreducible algebraic variety. It is not difficult to see (for instance, from the computation below of the tangent space) that its singular locus is exactly $\mathsf{RANK}(n, \leqslant r-1)$, the set of matrices with rank less than r.

On the other hand, $S(k)$ splits into components $S(\pi)$ depending on the pattern π and is thus a union of various affine subspaces (each associated to a π of size at most k). The nonsingular elements of $S(k)$ are those which have support of size exactly k.

We can put together the maps Φ arising from various choices of τ and π to write the map

$$\widetilde{\Phi} : \mathsf{RANK}(n, \leqslant r) \times S(k) \to \mathsf{RIG}(n, r, \leqslant k).$$

We can easily see that $\widetilde{\Phi}$ is a surjective morphism of affine varieties. If we can find a nonsingular point of $\mathsf{RANK}(n, \leqslant r) \times S(k)$ for which the map on tangent spaces is injective, then the dimension of the target space $\mathsf{RIG}(n, r, \leqslant k)$ will equal $\dim \mathsf{RANK}(n, \leqslant r) + \dim S(k) = n^2 - (n-r)^2 + k$, proving the theorem. Since the map on tangent spaces is simply addition of matrices, we need to show that the subspaces do not intersect non-trivially and that would complete the proof of the theorem. For any smooth point $x \in \mathsf{RANK}(n, r)$, the smooth locus of $\mathsf{RANK}(n, \leqslant r)$, we will find a pattern π of size k and $y \in S(\pi)$ for which the tangent spaces at x and y intersect transversely.

Assume first that the point x is $\begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix}$. We choose the pattern π to lie completely in the bottom right hand block of size $(n-r) \times (n-r)$, and choose any smooth point y of $S(\pi)$ (i.e. having all k entries nonzero).

The tangent space of x is $\begin{pmatrix} * & * \\ * & 0 \end{pmatrix}$.

That is, it consists of the subspace of M_n consisting of matrices with arbitrary entries except in the lower $(n-r) \times (n-r)$ block, which is constrained to be the zero submatrix. The dimension of the tangent space is $r^2 + 2r(n-r) = n^2 - (n-r)^2$, as expected. The tangent space of y is $\begin{pmatrix} 0 & 0 \\ 0 & *_\pi \end{pmatrix}$ where $*_\pi$ means that the entries in positions of π are arbitrary, and the other entries are zero.

It's clear that these two tangent spaces intersect transversely.

Now, we need to show this for a more general $x \in \mathsf{RANK}(n,r)$. Assume that the top left $r \times r$ minor of x is nonsingular (else we can multiply by permutation matrices on left and right, noting that permutations just shuffle the various $S(\pi)$ for $|\pi| = k$).

The first r columns of x are independent and span the column space of x, so there exists a matrix $g = \begin{pmatrix} I_r & * \\ 0 & I_{n-r} \end{pmatrix}$ such that xg has the form $\begin{pmatrix} * & 0 \\ * & 0 \end{pmatrix}$. Then using that the first r rows of xg are independent and span its row space, we can find an invertible matrix $h = \begin{pmatrix} * & 0 \\ * & I_{n-r} \end{pmatrix}$ such that $hxg = \begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix}$. The tangent space of x is $h^{-1} \begin{pmatrix} * & * \\ * & 0 \end{pmatrix} g^{-1}$. We need to show this does not intersect $S(\pi)$ for some π. That is, $\begin{pmatrix} * & * \\ * & 0 \end{pmatrix}$ does not intersect $h \begin{pmatrix} 0 & 0 \\ 0 & *_\pi \end{pmatrix} g$ except in zero. But this follows from the fact that the latter is a matrix of the same form (in fact, multiplication by h and g leave any element of $S(\pi)$ unchanged). ∎

Remarks: *A similar argument or line of study - though in the projective setting - is also found in [11]. Our formalism and proofs seem clearer and simpler. Our theorem is also very explicit.*

2.3 Rigid Matrices over the field of Complex Numbers

Recall that to say that the rigidity of a matrix A for target rank r is at least k, it suffices to prove that the matrix A is not in $\mathcal{W}(n, r, \leqslant (k-1))$. We use this idea to achieve the maximum possible lower bound for the rigidity of a family of matrices over the field of complex numbers $\mathbb{C}$. As a matter of fact, we obtain matrices with real algebraic entries with rigidity $(n-r)^2$.

THEOREM 7. *Let $\delta(n) = n^{4n^4}$. Let $p_{i,j}$ for $1 \leqslant i, j \leqslant n$ be distinct primes such that $p_{i,j} > \delta(n)$. Let $K = \mathbb{Q}(\zeta_{1,1}, \ldots, \zeta_{n,n})$ where $\zeta_{i,j} = e^{2\pi \mathrm{i}/p_{i,j}}$. Let $A(n) := (\zeta_{i,j}) \in M(n,K)$. Then, for any field L containing K,*

$$\mathrm{Rig}(A(n), r, L) = (n-r)^2.$$

PROOF. For simplicity, we will index the $\zeta_{i,j}$ by ζ_α for $\alpha = 1$ to n^2, and similarly p_α. We prove the theorem by showing that

$$A(n) \notin \mathcal{W}(n, r, \leqslant (n-r)^2 - 1)(L).$$

Thus it is sufficient to prove that

$$A(n) \notin \mathcal{W}(n,r,\pi)(L)$$

for any pattern π with $|\pi| = (n-r)^2 - 1$. Let π be any such pattern. To simplify notation, let us define, $\mathcal{W} := \mathcal{W}(n,r,\pi)(L)$. By Theorem 4 we have:

$$\dim(\mathcal{W}) \leqslant \dim(\mathcal{W}(n,r,\leqslant (n-r)^2-1)) \leqslant (n^2-1) < n^2.$$

Equivalently (by Hilbert's Nullstellensatz),

$$EI(n,r,\pi) \neq (0).$$

Proving that $A(n) \notin \mathcal{W}$ is equivalent to showing the existence of a $g \in EI(n,r,\pi)$ such that $g(A(n)) \neq 0$. We produce such a g using the following theorem:

THEOREM 8.([1], page 6, Theorem 4) *Let $I = \langle f_1, \ldots, f_s \rangle$ be an ideal in the polynomial ring $F[Y]$ over an infinite field F, where $Y = \{y_1, \ldots, y_m\}$. Let d be the maximum* total degree *of the generators f_i. Let $Z = \{y_{i_1}, \ldots, y_{i_\ell}\} \subseteq Y$ be a subset of indeterminates of Y. If $I \cap F[Z] \neq (0)$ then there exists a non-zero polynomial $g \in I \cap F[Z]$ such that, $g = \sum_{i=1}^{s} g_i f_i$, with $g_i \in F[Y]$ and $\deg(g_i f_i) \leqslant (\mu+1)(m+2)(d^\mu+1)^{\mu+2}$, where $\mu = min\{s,m\}$.*

Let us apply Theorem 8 to our case - in the notation of this theorem our data is as follows: $F := \mathbb{Q}, Y := \{x_1, \ldots, x_{n^2}, t_1, \ldots, t_k\}, Z := \{x_1, \ldots, x_{n^2}\}, \Sigma_{r+1} :=$ set of all minors of size $(r+1)$, $f_\tau := \det((X+T_\pi)_\tau)$ for $\tau \in \Sigma_{r+1}$, here by Y_τ we denote the τ-th minor of Y, and $I := I(n,r,\pi) = \langle f_\tau : \tau \in \Sigma_{r+1} \rangle$ as defined in (1).

Furthermore, we have:

$$\begin{aligned}
m &= n^2 + (n-r)^2 - 1 \leqslant 2n^2 - 2 \\
\mu &= \min\left\{ n^2 + (n-r)^2 - 1, \binom{n}{r+1}^2 \right\} \\
&\leqslant n^2 + (n-r)^2 - 1 \leqslant 2n^2 - 2, \\
d &= r+1 \leqslant n, \\
I \cap F[Z] &= EI(n,r,\pi) \neq (0).
\end{aligned}$$

By Theorem 8 there exists a

$$g \neq 0 \in EI(n,r,\pi) \subseteq \mathbb{Q}[x_1, \ldots, x_{n^2}]$$

such that

$$\deg(g) \leqslant (2n^2-1)(2n^2)(n^{2n^2-2}+1)^{2n^2} < n^{4n^4} = \delta(n).$$

We will now apply the following Lemma 9, which we prove later, to this situation.

LEMMA 9. *Let N be a positive integer. Let $\theta_1, \cdots, \theta_m$ be m algebraic numbers such that for any $1 \leqslant i \leqslant m$, the field $\mathbb{Q}(\theta_i)$ is Galois over $\mathbb{Q}$ and such that*

$$[\mathbb{Q}(\theta_i) : \mathbb{Q}] \geqslant N \quad \textit{and}$$

$$\mathbb{Q}(\theta_i) \cap \mathbb{Q}(\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_m) = \mathbb{Q}.$$

Let $g(\underline{x}) \neq 0 \in \mathbb{Q}[x_1, \ldots, x_m]$ such that $\deg(g) < N$. Then,

$$g(\theta_1, \ldots, \theta_m) \neq 0.$$

Let us set $m = n^2, N = \delta(n), l := \deg(g) \leqslant N$ in Lemma 9. It is now easy to check that

$$[\mathbb{Q}(\zeta_\alpha) : \mathbb{Q}] = p_\alpha - 1 \geqslant \delta(n) = N$$

and

$$\mathbb{Q}(\zeta_\alpha) \cap \mathbb{Q}(\zeta_1, \ldots, \zeta_{\alpha-1}, \zeta_{\alpha+1}, \ldots, \zeta_{n^2}) = \mathbb{Q}.$$

The latter follows from the fact that the prime p_α is totally ramified in $\mathbb{Q}(\zeta_\alpha)$ and is unramified in $\mathbb{Q}(\zeta_1, \ldots, \zeta_{\alpha-1}, \zeta_{\alpha+1}, \ldots, \zeta_{n^2})$; see Theorem 4.10 in [17]. Thus Lemma 9 is applicable and we get:

$$g(\zeta_1, \ldots, \zeta_{n^2}) \neq 0.$$

To complete the argument (for Theorem 7), now we prove Lemma 9.

Proof of Lemma 9 : By induction on m. For $m = 1$ this is trivial. Now suppose that the statement is true when the number of variables is strictly less than m. Assuming that the statement is not true for m, we will arrive at a contradiction. This will prove the Lemma.

Let $g \in \mathbb{Q}[\underline{x}]$ with $l := \deg(g) < N$ be such that

$$g(\theta_1, \ldots, \theta_m) = 0,$$

with θ_i, $1 \leqslant i \leqslant m$, satisfying the conditions as in the theorem. Since the statement is true for any $(m-1)$ number of variables, without loss of generality, we can assume that all the variables and hence x_m appears in g. Let us denote x_m by x. Let us write

$$g(x_1, \ldots, x_m) = \sum_{i=0}^{l} f_i(x_1, \ldots, x_{m-1}) x^{l-i}.$$

Note that $l < N$ and $\deg(f_i) < N$ for $0 \leqslant i \leqslant l$. Since $g \neq 0$, for some i, $0 \leqslant i \leqslant l$ the polynomial $f_i \neq 0$. Thus, by the inductive hypothesis,

$$f_i(\theta_1, \ldots, \theta_{m-1}) \neq 0.$$

Thus $g(\theta_1, \ldots, \theta_{m-1})(x) \neq 0 \in \mathbb{Q}(\theta_1, \ldots, \theta_{m-1})[x]$. This implies that θ_m satisfies a non-zero polynomial over $\mathbb{Q}(\theta_1, \ldots, \theta_{m-1})$ of degree $\leqslant l < N$. Thus:

$$[\mathbb{Q}(\theta_1, \ldots, \theta_m) : \mathbb{Q}(\theta_1, \ldots, \theta_{m-1})] \leqslant l < N. \tag{4}$$

On the other hand, since $\mathbb{Q}(\theta_m) \cap \mathbb{Q}(\theta_1, \ldots, \theta_{m-1}) = \mathbb{Q}$ and the fields $\mathbb{Q}(\theta_i)$ are Galois over $\mathbb{Q}$, it can be concluded by the property of such extensions ([12] Theorem 1.12, page 266) that

$$[\mathbb{Q}(\theta_1, \ldots, \theta_{m-1})(\theta_m) : \mathbb{Q}(\theta_1, \ldots, \theta_{m-1})] = [\mathbb{Q}(\theta_m) : \mathbb{Q}] \geqslant N.$$

This contradicts (4) above and that proves the lemma.

This concludes the proof of Theorem 7.■

Note that Theorem 7 is true for any family of matrices $A(n) = [\theta_{i,j}]$ provided the $\theta_{i,j}$ satisfy Lemma 9. Hence, we have

COROLLARY 10. *Let $A(n) := (\zeta_{i,j} + \overline{\zeta_{i,j}})$, where $\zeta_{i,j}$ are primitive roots of unity of order $p_{i,j}$ such that $p_{i,j} - 1 \geqslant 2\delta(n)$ (here $\overline{\zeta_{i,j}}$ denotes the complex conjugate of $\zeta_{i,j}$). Then, $A(n) \in M(n, \mathbb{R})$ has $\mathrm{Rig}(A(n), r) = (n-r)^2$.*

3 Reduction to Determinantal Ideals

In this section, we show that the natural decomposition of the rigidity varieties $\mathcal{W}(n, r, \leqslant k) = \cup_{|\pi|=k} \mathcal{W}(n, r, \pi)$ is indeed a decomposition into *irreducible* affine algebraic varieties. In fact, these components turn out to be varieties defined by elimination ideals of determinantal ideals generated by all the $(r+1) \times (r+1)$ minors.

To show the decomposition, we will continue to use the notation from Section 2. Consider the matrix $X + T_\pi$. Let $x = \{x_1, \ldots, x_{n^2}\} = x_{\bar{\pi}} \cup x_\pi$, where x_π is the set of variables that are indexed by π and $x_{\bar{\pi}}$ is the set of remaining variables.

Let

$$J := I(n, r, \pi) = \left\langle Minors_{(r+1)\times(r+1)}(X + T_\pi) \right\rangle$$

be the ideal of $\mathbb{Q}[x, t] = \mathbb{Q}[x_\pi, x_{\bar{\pi}}, t_\pi]$ generated by the $(r+1) \times (r+1)$ minors of $X + T_\pi$. Let

$$\begin{aligned} J_1 &:= J \cap \mathbb{Q}[x_\pi, x_{\bar{\pi}}] \subseteq \mathbb{Q}[x_1, \ldots, x_{n^2}], \\ J_2 &:= J_1 \cap \mathbb{Q}[x_{\bar{\pi}}], \\ I_{r+1} &:= \left\langle Minors_{(r+1)\times(r+1)}(X) \right\rangle \subseteq \mathbb{Q}[x], \quad \text{and} \\ EI_{r+1} &:= I_{r+1} \cap \mathbb{Q}[x_{\bar{\pi}}] \subseteq \mathbb{Q}[x_{\bar{\pi}}]. \end{aligned}$$

Notice that since J_1 is the elimination ideal of J w.r.t. eliminating variables t_π, a matrix A lies in $\mathcal{W}(n, r, \leqslant k) = \overline{\mathsf{RIG}(n, r, \leqslant k)}$ if and only if its entries lie in the variety defined by the ideal J_1. Also, I_{r+1} is the ideal generated by the $(r+1) \times (r+1)$ minors of X and EI_{r+1} its elimination ideal for the rational ring generated by the variables $x_{\bar{\pi}}$.

PROPOSITION 11. *$J_1 = J_2\mathbb{Q}[x]$ (the ideal generated by J_2 in $\mathbb{Q}[x]$) and $J_2 = EI_{r+1}$. In particular, $EI(n, r, \pi) = EI_{r+1}\mathbb{Q}[x]$ considered as ideals in $\mathbb{Q}[x]$.*

PROOF. First, notice that in the $(r+1) \times (r+1)$ minors of $X + T_\pi$, the variable $t_{i,j}$, for $(i, j) \in \pi$, always occurs in combination with $x_{i,j}$ as $t_{i,j} + x_{i,j}$. Therefore, eliminating the variables t_π will also automatically eliminate the variables x_π, giving the equality of the generators of the ideals J_1 and J_2. Therefore $J_1 = J_2\mathbb{Q}[x]$. More formally, consider the isomorphism between the two coordinate rings $\phi : \mathbb{Q}[x_\pi, x_{\bar{\pi}}, t_\pi]$ and $\mathbb{Q}[x_\pi, x_{\bar{\pi}}, t_\pi]$ defined by letting $\phi(t_{i,j}) = x_{i,j} + t_{i,j}$ for each $(i, j) \in \pi$ and $\phi(x_{i,j}) = x_{i,j}$ for all $(i, j) \notin \pi$. The ideal

$J_1 = J \cap \mathbb{Q}[x_\pi, x_{\bar{\pi}}] \subseteq \mathbb{Q}[x_1, \ldots, x_{n^2}]$ must equal the ideal $\phi(\phi^{-1}(J) \cap \phi^{-1}\mathbb{Q}[x_1, \ldots, x_{n^2}])$, since ϕ is an isomorphism. But $\phi^{-1}(J)$ is generated by matrices only involving the variables of t_π and $x_{\bar{\pi}}$, whereas $\phi^{-1}\mathbb{Q}[x_1, \ldots, x_{n^2}]) = \mathbb{Q}[x_1, \ldots, x_{n^2}]$, so that $\phi^{-1}(J) \cap \phi^{-1}\mathbb{Q}[x_1, \ldots, x_{n^2}]$ is generated by polynomials only involving the variables of $x_{\bar{\pi}}$. Therefore $\phi^{-1}(J_1) = \phi^{-1}(J) \cap \phi^{-1}\mathbb{Q}[x_1, \ldots, x_{n^2}] = J_2\mathbb{Q}[x]$ and taking the image under ϕ, we get $J_1 = J_2\mathbb{Q}[x]$.

The equation $J_2 = EI_{r+1}$ follows from similar considerations, noting that the variables $x_{i,j}$ for $(i,j) \in \pi$ always occur in the combination $x_{i,j} + t_{i,j}$. Therefore eliminating them eliminates $t_{i,j}$ as well. More formally, consider the isomorphism $\psi : \mathbb{Q}[x_\pi, x_{\bar{\pi}}, t_\pi] \to \mathbb{Q}[x_\pi, x_{\bar{\pi}}, t_\pi]$ defined by letting $\psi(x_{i,j}) = x_{i,j} + t_{i,j}$ for each $(i,j) \in \pi$, while $\psi(t_{i,j}) = t_{i,j}$ for $(i,j) \in \pi$ and $\psi(x_{i,j}) = x_{i,j}$. Then again we have $J_2 = J_1 \cap \mathbb{Q}[x_{\bar{\pi}}] = J \cap \mathbb{Q}[x_{\bar{\pi}}] = \psi(\psi^{-1}(J) \cap \psi^{-1}(\mathbb{Q}[x_{\bar{\pi}}])) = \phi(I_{r+1}\mathbb{Q}[x, t_\pi] \cap \mathbb{Q}[x_{\bar{\pi}}]) = \phi(EI_{r+1}) = EI_{r+1} \subset \mathbb{Q}[x_{\bar{\pi}}]$. ∎

The following is a well-known theorem; see [3, Chapter 2].

THEOREM 12. *Let* $\mathsf{RANK}(n, \leqslant r)$ *be the set of all rank* $\leqslant r$ *matrices of* $M_n \cong \mathbb{A}^{n^2}$. *Then*

- $I(\mathsf{RANK}(n, \leqslant r)) = I_{r+1}$ *and* $\mathsf{RANK}(n, \leqslant r) = V(I_{r+1})$.
- I_{r+1} *is a prime ideal of* $\mathbb{Q}[X]$. *In particular,* $\mathsf{RANK}(n, \leqslant r)$ *is an irreducible variety.*

From Theorem 12 and Proposition 11 we get the following corollary (see [10]) for details).

COROLLARY 13. *In the natural decomposition* $\mathcal{W}(n, r, \leqslant k) = \cup_{|\pi|=k}\mathcal{W}(n, r, \pi)$, *the* $\mathcal{W}(n, r, \pi)$ *are irreducible varieties.*

4 Semicontinuity of Rigidity

Intuitively, if a function is (lower) semicontinuous at a given point, then within a small neighborhood of that point the function is nondecreasing. (See the full version [10] of the paper for a formal treatment of the material in this section). The rank function of a matrix, for example, is a lower semicontinuous function on the space of all $n \times n$ complex matrices. It is possible to construct give examples (we defer this to the full version [10]) to show that the rigidity function is not semicontinuous in general. However, it seems to have semicontinuity property at some interesting matrices. In particular, the matrices $A(n)$ from Theorem 7 have an open neighborhood around them within which the rigidity function is constant. This is a direct consequence of their very construction since they are outside the closed sets $\mathcal{W}(n, r, \leqslant (n-r)^2 - 1)$. These examples motivate us to study the properties of the Euclidean closure and Zariski closure of the set $\mathsf{RIG}(n, r, \leqslant k)(\mathbb{C})$. In fact, we are able to argue that these two coincide.

PROPOSITION 14. *The Euclidean Closure of* $\mathsf{RIG}(n, r, \leqslant k)(\mathbb{C})$ *equals its Zariski Closure.*

PROOF. Recall that we can write $\mathsf{RIG}(n, r, \leqslant k) = \bigcup_{\pi,\ |\pi|=k} \mathsf{RIG}(n, r, \pi)$. Thus, to prove the proposition, it is sufficient to prove that for any pattern π, the Euclidean closure of $\mathsf{RIG}(n, r, \pi)$ equals its Zariski Closure. By Closure Theorem, there exists a subvariety V strictly contained in $\mathcal{W} := \overline{\mathsf{RIG}(n, r, \pi)}$ such that $\mathcal{W}(\mathbb{C}) - V(\mathbb{C}) \subseteq \mathsf{RIG}(n, r, \pi)(\mathbb{C}) \subseteq \mathcal{W}(\mathbb{C})$. Since $\mathcal{W}(\mathbb{C})$ is closed in the Euclidean topology, we will done if we prove that the Euclidean closure of $\mathcal{W}(\mathbb{C}) - V(\mathbb{C})$ is $\mathcal{W}(\mathbb{C})$. This is precisely the statement of the following lemma from [20], which we state below for easy reference. Also note that, by Corollary 13, W is an irreducible variety for every pattern π and hence the lemma is applicable.

LEMMA 15. ([20, Lemma 1, page 124]) *If X is an irreducible algebraic variety and Y a proper subvariety of X then the set $X(\mathbb{C}) - Y(\mathbb{C})$ is dense in $X(\mathbb{C})$.*

References

[1] A. Bernasconi, E. W. Mayr, M. Mnuk, and M. Raab. Computing the Dimension of a Polynomial Ideal. http://www14.informatik.tu-muenchen.de/personen/raab/, 2002.

[2] W. D. Brownawell. Bounds on the degrees of Nullstellensatz. *Annals of Mathematics*, 126:577–592, 1987.

[3] W. Bruns and U. Vetter. *Determinantal Rings*, volume 1327 of *Lect. Notes in Math.* 1980.

[4] M. Cheraghchi. On Matrix Rigidity and the Complexity of Linear Forms. *Electronic Colloquium on Computational Complexity (ECCC)*, (070), 2005.

[5] B. Codenotti. Matrix Rigidity. *Linear Algebra and its Applications*, 304(1–3):181–192, 2000.

[6] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties,and Algorithms*. Under Graduate Textbooks in Mathematics. 3rd edition, 2007.

[7] J. Forster. A Linear Lower Bound on the Unbounded Error Probabilistic Communication Complexity. *Journal of Computer and System Sciences*, 65(4):612–625, 2002.

[8] J. Friedman. A Note on Matrix Rigidity. *Combinatorica*, 13(2):235 – 239, 1993.

[9] J. Kollar. Sharp Effective Nullstellensatz. *Jl. of American Math. Soc.*, 1(4):963–975, 1988.

[10] A. Kumar, S. V. Lokam, V. Patankar, and J. M. N. Sarma. Using Elimination Theory to Construct Rigid Matrices. Manuscript, 2009.

[11] J. M. Landsberg, J. Taylor, and N. K. Vishnoi. The Geometry of Matrix Rigidity. Technical Report GIT-CC-03-54, Georgia Institute of Technology, 2003.

[12] S. Lang. *Algebra*. Springer-Verlag, revised third edition, 2004.

[13] N. Linial and A. Shraibman. Learning complexity vs communication complexity. *Combinatorics, Probability and Computing*, 18(1-2):227–245, 2009.

[14] S. V. Lokam. On the Rigidity of Vandermonde matrices. *Theoretical Computer Science*, 237:477–483, 2000.

[15] S. V. Lokam. Spectral Methods for Matrix Rigidity with Applications to Size-Depth Tradeoffs and Communication Complexity. *Jl. of Comp. Syst. Sci.*, 63(3):449–473, 2001.

[16] S. V. Lokam. Quadratic Lowerbounds on Matrix Rigidity. In *Proc. of Internatical Conf. on Theory and Applications of Models of Computation*, volume 3959 of *LNCS*, 2006.

[17] W. Narkiewicz. *Elementary and Analytic Theory of Algebraic Numbers*, volume XI of *Springer Monographs in Mathematics*. Springer, 2004.

[18] R. Paturi and P. Pudlák. Circuit Lower Bounds and Linear Codes. *Teoria slozhnosti vychislenij IX*, 316:188–204, 2004. ECCC Techreport : TR04-04.

[19] A. A. Razborov. On Rigid Matrices. Manuscript, (Russian), 1989.

[20] I. R. Shafarevich. *Varieities in Projective Space*, volume 1 of *Basic Algebraic Geometry*. Springer Verlag, second edition, 1994.

[21] D. A. Spielman, V. Stemann, and M. A. Shokhrollahi. A Remark on Matrix Rigidity. *Information Processing Letters*, 64(6):283 – 285, 1997.

[22] L. G. Valiant. Graph Theoretic Arguments in Low Level Complexity. volume 53 of *Lecture Notes in Computer Science*, pages 162–176. Springer Verlag, 1977.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

On Nondeterministic Unranked Tree Automata with Sibling Constraints

Christof Löding and Karianto Wong*
RWTH Aachen University, Germany

ABSTRACT. We continue the study of bottom-up unranked tree automata with equality and disequality constraints between direct subtrees. In particular, we show that the emptiness problem for the nondeterministic automata is decidable. In addition, we show that the universality problem, in contrast, is undecidable.

1 Introduction

We continue the study of bottom-up unranked tree automata with equality and disequality constraints between direct subtrees, introduced in [12], which extend the corresponding automaton model known from the ranked setting [1]. This extension constitutes a part of the efforts in transferring the results known in the context of automata on ranked trees to the unranked case, which has attracted much attention from the research community as a formal model for XML documents; for references, see, e.g., the surveys [14, 15].

The distinguishing feature of unranked trees is that the number of children of the nodes, as opposed to ranked trees, is not a priori bounded by any fixed rank. In order to cope with this phenomenon, bottom-up automata on unranked trees, usually, incorporate regular languages in their transitions. It then turns out that finite automata on unranked trees enjoy many of the good properties of their counterpart in the ranked case. In many application domains, however, it is often desired to add some expressive power to the basic model without losing too many of the decidability results. A common approach to doing this is to add some constraints to the transitions of the automata. In fact, many models of finite automata with constraints on (ranked as well as unranked) trees have appeared in the literature.

An example of adding constraints in tree automata is counting constraints, which is particularly interesting in the unranked case as the number of successors of a node might be unbounded. For instance, in Presburger automata [17] (cf. also sheaves automata [7]) the application of a bottom-up transition is subject to the satisfaction of certain numerical conditions involving the subtrees (or the states reached at the root of these subtrees), such as "the number of a-rooted subtrees is twice the number of b-rooted subtrees". It turns out that adding these constraints retains many of the good properties of the basic model; in particular, the emptiness problem remains decidable.

Another type of constraints that has been considered in the literature is the equality (and disequality) constraints. Here, the application of a bottom-up transition is subject to whether certain subtrees of the current node are equal. It turns out, however, that these constraints, in the most general form where comparisons between arbitrary subtrees are allowed, are too powerful, in the sense that the emptiness problem becomes undecidable [13].

*supported by DFG research grant *Algorithmische Theorie der Baumautomaten*

© Christof Löding, Karianto Wong; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 311–322
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2328

In order to obtain decidability, thus, some restrictions on how equality constraints are used in ranked tree automata have been suggested. Such a restriction can be found in reduction automata [3, 8]; here, one requires that the number of equality and disequality tests in each path of a run tree must be bounded. Another possible restriction, suggested in [1], is that equality constraints may only be applied to sibling subtrees. For a more thorough overview of these automata models, the reader is referred to [4].

In [12], we extend tree automata with equality constraints between siblings to the unranked setting. In order to be able to address the (possibly) unbounded number of siblings to be compared, while still maintaining a finite representation, we suggest using formulas of monadic second-order logic. For this model, it has been shown that the emptiness problem for the deterministic case is decidable, while leaving open the nondeterministic case. It also turns out that the nondeterministic automata are strictly more expressive than the deterministic ones.

In this paper, we settle the nondeterministic case: we show that the emptiness problem, as in the deterministic case, is decidable. For this, despite the fact that determinization is not possible, we incorporate a kind of subset construction directly into our algorithm for the deterministic case, which then yields an emptiness algorithm for the nondeterministic case. In addition, we show that the universality problem in undecidable, which is achieved via a reduction of the halting problem for two-register machines.

There is a tight connection between our automaton model and data words that we want to point out. Generally speaking, a data word is a finite word to each position of which is attached a data value, i.e., a value from an infinite domain like the natural numbers. There are several automaton models on data words (and data trees) that have been proposed in the literature; for a recent survey, see, e.g., [16]. In order to maintain decidability results, these automaton models, usually, can only compare data values with respect to equality. Now, as trees can be used to represent data values, equality between data values amounts to equality between trees. Thus, with an appropriate encoding of data words as unranked trees, our automaton model can be used to describe languages of data words.

This paper is organized as follows. In Section 2, we fix our notations and recall our automaton model as well as some known results. Section 3 is devoted to our main result, namely that the emptiness problem for our automaton model is decidable. In Section 4 we show that the universality problem, in contrast, is undecidable. Then, in Section 5 we discuss the connection with data languages. Finally, Section 6 concludes with some remarks on the complexity issues and further prospects.

2 Preliminaries

We denote the set of (positive) natural numbers by $\mathbb{N}$ (respectively, $\mathbb{N}_+$).

For every set A, we denote by 2^A the power set of A and by $\mathbb{N}^A$ the set of mappings assigning a natural number to each member of A. We denote the set of all finite (nonempty) words over A by A^* (respectively, A^+). We denote the empty word by ε. For every word $w \in A^*$, we denote its length by $|w|$.

Let A be a finite, nonempty alphabet. A nonempty word w over A defines a logical structure with the set of w's positions as its universe, equipped with the successor predi-

cate $S(x,y)$, the order predicate $x < y$, and the label predicate $a(x)$, for each $a \in A$; these predicates are interpreted over $\{1,\dots,|w|\}$ as usual. The formulas of monadic second-order (MSO) logic over words over A are built up from: first-order variables $x, y, z, \dots$ (ranging over positions); set variables $X, Y, Z, \dots$; atomic formulas $x = y$, $x < y$, $S(x,y)$, $X(x)$, and $a(x)$, for all $a \in A$; Boolean connectives; and first-order as well as set quantifiers. We write $\varphi(x_1,\dots,x_n,X_1,\dots,X_m)$ to indicate that the MSO-formula φ may contain free occurrences of the variables $x_1,\dots,x_n,X_1,\dots,X_m$.

In the sequel, let Σ be a nonempty, finite (tree-labeling) alphabet. A tree domain D is a nonempty, prefix-closed subset of $\mathbb{N}_+^*$ such that, for each $u \in D$ and $i > 0$, if $ui \in D$, then also $uj \in D$, for each $j \in \{1,\dots,i\}$. A finite unranked tree t over Σ (Σ-labeled tree, for short) is a mapping $t\colon \operatorname{dom}_t \to \Sigma$ where dom_t is a finite tree domain. The elements of dom_t are called the nodes of t, and the node ε is called the root of t. A node $u \in \operatorname{dom}_t$ is said to have $k \geq 0$ successors if $uk \in \operatorname{dom}_t$ but $u(k+1) \notin \operatorname{dom}_t$. In this case, we call ui the i-th successor of u, and we say that ui and uj are sibling nodes, for each $i, j \in \{1,\dots,k\}$. A leaf of t is a node without any successor. Given a node u of t, the subtree of t at u is the tree given by $t|_u$ with $\operatorname{dom}_{t|_u} = \{v \in \mathbb{N}_+^* \mid uv \in \operatorname{dom}_t\}$ and $t|_u(v) = t(uv)$, for all $v \in \operatorname{dom}_{t|_u}$. Further, $t|_u$ is called a direct subtree of t if $|u| = 1$. We write t as $a(t_1 \dots t_k)$ to indicate that its root is labeled with a and that it has k successors at which the subtrees $t_1,\dots,t_k$ are rooted. We denote the set of all Σ-labeled trees by $\mathcal{T}_\Sigma$.

Let Q be a finite, nonempty set (of tree automaton states). An *atomic sibling constraint over Q* is given by an MSO-formula $\varphi(x,y)$ over words over Q, with two free first-order variables x and y, and has either of the following forms:

$$(\forall^{=}) \quad \forall x \forall y \,.\, \varphi(x,y) \;\to\; t_x = t_y \qquad (\exists^{=}) \quad \exists x \exists y \,.\, \varphi(x,y) \;\wedge\; t_x = t_y$$
$$(\forall^{\neq}) \quad \forall x \forall y \,.\, \varphi(x,y) \;\to\; t_x \neq t_y \qquad (\exists^{\neq}) \quad \exists x \exists y \,.\, \varphi(x,y) \;\wedge\; t_x \neq t_y$$

Intuitively, an $\exists^{=}$-constraint (respectively, $\exists^{\neq}$) says that "there is a pair of positions that satisfies φ, and the subtrees at these positions are equal (or distinct, respectively)", and a $\forall^{=}$-constraint (respectively, $\forall^{\neq}$) says that "for each pair of positions that satisfies φ the subtrees at these positions must be equal (or distinct, respectively)". A nonempty word w over Q together with a sequence $t_1 \dots t_{|w|}$ of Σ-labeled trees (attached to w's positions) are said to satisfy an atomic sibling constraint if, depending on the constraint type, the following holds:

- $\exists^{=}$- or $\exists^{\neq}$-constraint: There exist some positions x, y in w such that $\varphi(x,y)$ is satisfied and $t_x = t_y$ (respectively, $t_x \neq t_y$).
- $\forall^{=}$- or $\forall^{\neq}$-constraint: For all positions x, y in w, if $\varphi(x,y)$ is satisfied, then $t_x = t_y$ (respectively, $t_x \neq t_y$).

For the sake of simplicity, we will sometimes refer to atomic sibling constraints simply by the underlying MSO-formulas whenever no confusion might arise. A *sibling constraint over Q* is a Boolean combination of atomic constraints. As a remark, $\forall^{=}$-constraints are, with respect to negation, dual to $\exists^{\neq}$-constraints, and, similarly, $\forall^{\neq}$-constraints are dual to $\exists^{=}$-constraints. Thus, without loss of generality, we will consider only positive Boolean combinations (i.e., without negation) of atomic constraints.

A (nondeterministic) *unranked tree automaton with equality and disequality constraints between siblings* (*UTACS*) over Σ is defined as a tuple $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$ where: Q is a finite,

nonempty set of states; $F \subseteq Q$ is the set of accepting or final states; $\Lambda \subseteq \Sigma \times Q$ is the set of leaf transitions; and Δ is the set of inner-node transitions of the form (L, α, a, q), where $L \subseteq Q^+$ is a regular set, α is a sibling constraint over Q, $a \in \Sigma$, and $q \in Q$. Note that we can assume, without loss of generality, that α is a conjunction of atomic constraints; starting from sibling constraints in disjunctive normal form, each transition with a disjunction of sibling constraints can be split into several transitions each of which contains only a conjunction of atomic constraints.

For every Σ-labeled tree t, a run of $\mathfrak{A}$ on t is a Q-labeled tree $\rho\colon\ \mathrm{dom}_t \to Q$ such that: (a) for each leaf node $u \in \mathrm{dom}_t$, we have $(t(u), \rho(u)) \in \Lambda$; (b) for each node $u \in \mathrm{dom}_t$ with $k \geq 1$ successors, there exists a transition $(L, \alpha, t(u), \rho(u))$ in Δ such that the word $\rho(u1) \dots \rho(uk)$ belongs to L and, together with the tree sequence $t|_{u1} \dots t|_{uk}$, satisfies α. In case such a run exists, we write $t \to_{\mathfrak{A}} \rho(\varepsilon)$ (or simply $t \to \rho(\varepsilon)$), and say that t reaches or evaluates to $\rho(\varepsilon)$. Further, $\delta(t)$ denotes the set of states reached by t, i.e., $\delta(t) = \{q \in Q \mid t \to q\}$. Note that $\delta(t)$ can effectively be determined. The tree t is accepted by $\mathfrak{A}$ if $\delta(t) \cap F \neq \emptyset$. The set of trees accepted by $\mathfrak{A}$ is denoted by $T(\mathfrak{A})$. We call $\mathfrak{A}$ *deterministic* if, for each tree t, there exists at most one state q with $t \to q$.

Let us recall some properties of UTACS (cf. [12]). The class of UTACS is closed under union and intersection, and the class of deterministic UTACS are closed under all Boolean operations. Moreover, UTACS, in general, cannot be determinized; that is, there exists some UTACS-definable tree language which cannot be recognized by any deterministic UTACS.

3 The Emptiness Problem

The main result of this section is:

THEOREM 1. *The emptiness problem for nondeterministic UTACS is decidable.*

In [12], we have given an emptiness algorithm for the *deterministic* case. Toward showing Theorem 1, we propose incorporating a kind of subset construction into this algorithm in order to obtain an emptiness algorithm for the nondeterministic case; in this way, we have thus avoided determinization, which, in general, is not possible for nondeterministic UTACS. In order to accomplish this, we will need to refine some notions we have used in the deterministic case and adapt the algorithm appropriately. For the sake of clarity, in the following we will directly present our method for the nondeterministic case, while sometimes making reference to the deterministic case as we see fit.

Throughout this section, let $\mathfrak{A} = (Q, \Sigma, \Lambda, \Delta, F)$ be a nondeterministic UTACS.

The key difference between nondeterministic and deterministic UTACS is that for the former the state reached by a tree is, in general, not unique. Nevertheless, the *set of states* reached by every tree is unique; in other words, we have:

REMARK 2. *For every pair, t and t', of Σ-labeled trees, if $\delta(t) \neq \delta(t')$, then $t \neq t'$.*

Our emptiness algorithm is actually an adaptation of the standard marking algorithm (see, e.g., [4, Chapter 8]). The main idea is to maintain, for each set of states $S \subseteq Q$, a collection T_S containing trees t with $\delta(t) = S$. For this, we iteratively construct new trees $a(t_1 \dots t_m)$, for some $a \in \Sigma$, where the trees $t_1, \dots, t_m$ have been constructed in previous rounds, by checking whether some *transition that reaches S* is *applicable*. The algorithm then

terminates as soon as some tree t with $\delta(t) \cap F \neq \emptyset$ has been constructed, or, otherwise, if we have constructed *enough* trees to conclude that the language recognized by the underlying automaton is empty.

In the sequel, we define the notions needed for our algorithm. First, we introduce transitions that have state sets as target. Second, we consider the applicability of such transitions. Third, we provide a bound on the number of trees we need to collect. Due to space limitations, however, we will omit most of the technical details.

Subset transitions and suitable words. As we are considering sets of states instead of mere states as the target of a transition, we are going to consider a collection of transitions instead of a single transition, which reflects the possibility to reach, with each tree, more than one target state. Such a collection of transitions, intuitively, specifies which 'normal transitions' we can apply in order to reach the states in S.

DEFINITION 3. *Let a be a symbol from Σ, and let S be a nonempty subset of Q. A* subset transition w.r.t. a and S *(or, for short, (a, S)-transition) is a collection of transitions given by a mapping $\theta\colon S \to \Delta$ such that, for each $q \in S$, the transition $\theta(q) \in \Delta$ reads the symbol a and has q as its target state. We denote the set of all (a, S)-transitions by Θ^a_S.*

An application of a subset transition $\theta \in \Theta^a_S$ to a tree $t = a(t_1 \dots t_m)$, actually, consists of applying all the transitions referred to therein to t; that is, for each of these transitions, say (L, α, a, q), there is a sequence of states $w = q_1 \dots q_m$, with $q_i \in \delta(t_i)$, for each $i = 1, \dots, m$, such that, firstly, w belongs to L and, secondly, w and $t_1 \dots t_m$ satisfy the constraint α.

Note that with this definition the result of applying a subset transition $\theta \in \Theta^a_S$ to a tree $t = a(t_1 \dots t_m)$ is, in general, not exactly S; instead, $\delta(t)$ might be a superset of S, as the definition does not forbid other transitions than the ones mentioned in θ to be applied to t.

In order to analyze the conditions under which a subset transition is applicable, we focus on the sequences of state sets that underlie an application of the subset transition (i.e., the state sets occurring at the children of the node under consideration). Let $\theta \in \Theta^a_S$ be a subset transition. A nonempty word over the power set of Q, say $\xi = S_1 \dots S_m \in (2^Q)^+$, is called *suitable* for θ (or θ-suitable, for short) if it can be used in an application of θ under the assumption that a sequence of trees $t_1, \dots, t_m$ with $\delta(t_i) = S_i$, for each $i = 1, \dots, m$, exists (thus resulting in a tree $t = a(t_1 \dots t_m)$ with $S \subseteq \delta(t)$). We denote the set of θ-suitable words by $\mathrm{suit}(\theta)$.

Note that with the notion of suitability we ignore the actual fact whether the trees needed to apply a subset transition exist. Instead, we focus on the sequences of state sets that can possibly be used for applying a subset transition under the assumption that the trees needed for the application exist; if an application is indeed possible, we then just have to arrange these trees appropriately. Thus, not surprisingly, analyzing the suitable words for a subset transition amounts to analyzing whether the equality and disequality constraints of the subset transition under consideration do not contradict one another.

More precisely, $\xi = S_1 \dots S_m \in (2^Q)^+$ is suitable for θ if there exists a family of words $(w^\tau)_{\tau \in \theta(S)}$ such that the following holds:

- For each transition $\tau \in \theta(S)$, say, $\tau = (L^\tau, \alpha^\tau, a, q^\tau)$, we have that $w^\tau \in L^\tau$ and, for each $i = 1, \dots, m$, the state at the i-th position of w^τ, say q^τ_i, belongs to S_i.

- Every two positions that are required to be equal (w.r.t. the evaluation of the constraint α^τ on w^τ, for *all* $\tau \in \theta(S)$), due to Remark 2, are labeled with the same state set.
- The equality and disequality constraints (again, with respect to the evaluation of α^τ on w^τ, for *all* $\tau \in \theta(S)$) do not contradict one another.

In particular, the satisfaction of these conditions allows an assignment of trees (if these exist) to the positions $1, \ldots, m$ in order to apply all the transitions τ under consideration using the corresponding words w^τ.

Later in the emptiness algorithm, we want to look for some subset transition that is applicable using only the trees we have constructed in the previous rounds. To this end, we introduce a further restriction on the notion of suitable words. Let $\mathfrak{R} \subseteq 2^Q$ be a set of state sets, and let $\bar{d}\colon \mathfrak{R} \to \mathbb{N}$ be a mapping assigning to each state set $K \in \mathfrak{R}$ a natural number. A word $\xi = S_1 \ldots S_m \in (2^Q)^+$ is called *suitable for θ with respect to $\mathfrak{R}$ and $\bar{d}$* (or $(\theta, \mathfrak{R}, \bar{d})$-suitable, for short) if it can be used in an application of θ under the assumption that there is a sequence of trees $t_1, \ldots, t_m$ satisfying the following:

- for each $i = 1, \ldots, m$, $\delta(t_i) = S_i$;
- for each $K \in \mathfrak{R}$, the number of distinct trees among $t_1, \ldots, t_m$ that reach K, i.e., the cardinality of the set $\{t_i \mid 1 \leq i \leq m \text{ and } \delta(t_i) = K\}$, does not exceed $\bar{d}(K)$. In other words, $\bar{d}(K)$ gives the number of available distinct trees that evaluate to K.

Note that, for each $K \in 2^Q \setminus \mathfrak{R}$, we do not put any restriction on the number of distinct trees among $t_1, \ldots, t_m$ that reach K. The sets of $(\theta, \mathfrak{R}, \bar{d})$-suitable words is denoted by $\mathrm{suit}(\theta, \mathfrak{R}, \bar{d})$.

As sibling constraints are based on MSO-formulas, it turns out that the suitability conditions introduced above can be translated into MSO-formulas, which justifies the following lemma.

LEMMA 4. *For each subset transition θ, each $\mathfrak{R} \subseteq 2^Q$, and each $\bar{d}\colon \mathfrak{R} \to \mathbb{N}$, the sets* $\mathrm{suit}(\theta)$ *and* $\mathrm{suit}(\theta, \mathfrak{R}, \bar{d})$ *are regular. In particular, it is decidable whether these sets are empty.*

The bound lemma. As in the deterministic case, the next step is to assert the existence of a certain bound on the number of distinct trees needed for each state set in order to apply a subset transition. Such a bound is given in Lemma 5 below. Consequently, our emptiness algorithm needs to collect, for each state set, only as many distinct trees as this bound.

For every θ-suitable word ξ, let $[\![\xi, \theta]\!] \in \mathbb{N}^{(2^Q)}$ be a mapping assigning to each set of states the number of distinct trees evaluating to this state set that are needed in order to apply θ (with respect to a particular application of θ using ξ). Note that $[\![\xi, \theta]\!]$, as has been remarked in [12], does not merely depend on ξ and θ, but also on a certain application of θ using ξ. That is, whenever we pick a θ-suitable word ξ, we always implicitly refer to such a particular application of θ, which then gives a unique value of $[\![\xi, \theta]\!]$. Note also that each value of $[\![\xi, \theta]\!]$, in general, does not need to exceed $|\xi|$.

LEMMA 5. *There exists some $B \in \mathbb{N}$ such that, for each subset transition θ of $\mathfrak{A}$ and each θ-suitable word ξ, there exists a θ-suitable word ξ' satisfying the following properties:*

1. *For each $R \subseteq Q$, $[\![\xi', \theta]\!](R) \leq B$.*
2. *For each $R \subseteq Q$, $[\![\xi', \theta]\!](R) \leq [\![\xi, \theta]\!](R)$.*
3. *For each $R \subseteq Q$, if R occurs in ξ, then it occurs in ξ' as well.*

In essence, the lemma asserts the existence of a bound B such that for each subset transition θ, if we can apply it using ξ, and if this application needs more than B distinct trees for some state set R, then we can as well apply θ using another word ξ', in place of ξ, such that the latter application needs only at most B distinct trees, for each state set. The second condition says, moreover, that the latter application can be carried out using only the trees which have already been available to the former application of θ. The third condition is merely a technical condition asserting that all state sets occurring in ξ also occur in ξ'. Note that, by the definition of subset transitions, the state sets reached by the application of θ using ξ and using ξ' might be different, in contrast to the corresponding bound lemma in the deterministic case (cf. [12, Lemma 3]).

As in the deterministic case, the bound lemma is established by a brute-force algorithm finding the desired bound iteratively. We start with some initial bound on the number of distinct trees needed for each state set in order to apply a subset transition and try all possible scenarios of the actual number of distinct trees for each state set within this bound, which boils down to checking the sets of suitable words (in each iteration with respect to the corresponding value of the bound) for emptiness; by Lemma 4, the emptiness of these sets is indeed decidable. In fact, our algorithm for finding the bound is a straightforward adjustment of the bound algorithm of the deterministic case: we only need to replace 'state' with 'set of states' and 'transition' with 'subset transition'.

The emptiness algorithm. The main idea of the our emptiness algorithm (Algorithm 1 on page 318) is to collect, for each state set $S \subseteq Q$, a certain number of trees that evaluate to S in T_S; let $\bar{d} \in \mathbb{N}^{(2^Q)}$ be such that $\bar{d}(S)$ keeps track of the cardinality of T_S. We collect trees by iteratively constructing new trees out of the trees we have collected in previous rounds by means of some applicable subset transition. In order to check the applicability of subset transitions, we look for subset transitions for which the set of suitable words has not yet been exhausted (cf. the **if**-condition of Line 6–10). Here, the crucial point is to find some appropriate suitable word ξ, which can effectively be done since the emptiness of $\mathrm{suit}(\theta, \bar{d})$ is decidable, and the algorithm, at any point during its execution, stores only a finite number of trees.

In order to guarantee termination, we set a bound on the number of trees we are collecting, i.e., the algorithm terminates as soon as this bound has been reached. Such a bound is provided by Lemma 5, which says that, for each state set S, it suffices to collect up to B trees. We encounter some difficulties, though: as has been noted before, applying a subset transition (say, for a state set S) using a suitable word might lead to a tree that does not evaluate exactly to S but, instead, to some superset S' of S. In order to deal with this, we observe that, as far as the applicability of (subset) transitions is concerned, trees evaluating to S' can be used as a replacement for trees evaluating to S; in this case, we have to keep the trees used for S' and the ones used for S separately in order to maintain the satisfaction of the disequality constraints. Thus, instead of collecting B trees for S and S' each, we can as well collect, for instance, $(2 \cdot B)$ trees for S'.

We exploit this observation in the algorithm by considering, for each state set S, not only T_S, but also the union of all $T_{S'}$ with $S' \supseteq S$, which is denoted by $T_S{\uparrow}$ and which is referred to as a *(tree) collection*. In other words, we put a bound, z, on the cardinality of such tree

Algorithm 1 The emptiness algorithm

```
1: procedure EMPTY(𝔄)
2:     compute the bound B according to Lemma 5
3:     initialize each T_S with {a ∈ Σ | δ(a) = S}
4:     z := (B + 1) · 2^{2|Q|}
5:     repeat
6:         if there exist some subset transition θ ∈ Θ^a_S, some word ξ = S_1 ... S_m ∈ suit(θ, d̄),
7:             and some trees t_1, ..., t_m with t_i ∈ T_{S_i} such that
8:             – T_S↑ is not full, i.e., |T_S↑| < z,
9:             – θ can be applied using ξ and t_1, ..., t_m, and
10:            – a(t_1 ... t_m) has not been constructed before, i.e., a(t_1 ... t_m) ∉ ⋃_{R⊆Q} T_R
11:        then add a(t_1 ... t_m) to T_R, where R = δ(a(t_1 ... t_m)), and update d̄
12:            if T_S↑ has become full (i.e., |T_S↑| ≥ z) then z := z − 2^{|Q|}
13:    until no new tree can be constructed
14:    if T_S ≠ ∅ for some S ⊆ Q with S ∩ F ≠ ∅ then return 'T(𝔄) ≠ ∅'
15:    else return 'T(𝔄) = ∅'
16: end procedure
```

collections; that is, we consider a tree collection $T_S{\uparrow}$ *full* (with respect to z) if $|T_S{\uparrow}| \geq z$. Since there are $2^{|Q|-|S|}$ supersets of S, it suffices, for $T_S{\uparrow}$, to collect $B \cdot 2^{|Q|-|S|}$ trees (i.e., B trees for each superset of S). Furthermore, in order to cope with some technicalities arising from the correctness proof of the algorithm (cf. Lemma 7 below), we initialize z with $(B+1) \cdot 2^{2|Q|}$ and decrease z by $2^{|Q|}$ each time a tree collection turns full.

REMARK 6. *Since the bound z is non-increasing, once a tree collection has been declared full, it stays full until the termination of the algorithm. In particular, z is decreased at most $2^{|Q|}$ times since the decrement only takes place if a tree collection turns full (and there are $2^{|Q|}$ of them). Moreover, upon termination of the algorithm, the value of z is at least $B \cdot 2^{2|Q|}$.*

Therefore, for each tree collection, at most $(B+1) \cdot 2^{2|Q|}$ trees are constructed, and in each iteration of the **repeat**-loop a new tree is constructed. Consequently, this loop is iterated at most $((B+1) \cdot 2^{3|Q|})$-times, so the algorithm eventually terminates.

The algorithm is sound as trees are constructed according to the subset transitions of $\mathfrak{A}$. The completeness of the algorithm follows from Lemma 7 below, which is similar to the completeness lemma of the deterministic case (cf. [12, Lemma 6]).

LEMMA 7. *For each tree $t \in \mathcal{T}_\Sigma$ and each state set $S \subseteq Q$, if $\delta(t) = S$, then $t \in T_S$ (that is, the tree t is eventually constructed by the algorithm), or $T_S{\uparrow}$ has been declared full (for some value of z) upon the termination of the algorithm.*

As with its deterministic-case counterpart, the proof of this lemma goes by an induction on the structure of t. However, it is more involved, in particular for the case $t = a(t_1 \dots t_m)$ with $\delta(t) = S$ but $t \notin T_S$ in the induction step. For this, we have to show that the tree collection $T_S{\uparrow}$ can be declared full by constructing as many trees as necessary. In order to achieve this, we also need to reduce, in the course of the algorithm, the requirement of a tree collection being full, which is done by decreasing the bound z on the collection size.

The complexity of our emptiness algorithm depends on the bound B given by the bound lemma. Unfortunately, we have not yet been able to give an upper bound for B since in the proof of the bound lemma we make use of Dickson's Lemma [9], which guarantees that our algorithm for finding the desired bound indeed terminates, but which does not come with any complexity analysis (recall that the proof of Dickson's Lemma is a non-constructive one). Thus, the complexity of our emptiness algorithm is still an open issue.

4 The Universality Problem

The universality problem for UTACS is the question whether a given UTACS accepts all its input trees. For deterministic UTACS, this problem is decidable since deterministic UTACS are effectively closed under complementation; the decidability of universality then follows from the decidability of emptiness (see [12]). For nondeterministic UTACS, in contrast, it turns out that this problem is undecidable.

THEOREM 8. *The universality problem for nondeterministic UTACS is undecidable.*

In order to show this, we use a reduction from the halting problem for 2-register machines: given a 2-register machine, we construct a UTACS such that the 2-register machine has a halting computation if and only if there exists some unranked tree that is not accepted by the UTACS, which is supposed to be the encoding of the halting computation. Due to space limitations, we will only present a brief sketch of the encoding we use and point out the main difficulties arising in the proof of Theorem 8.

In the core of the reduction is how the computations of a 2-register machine are encoded as unranked trees. As usual, a computation of a 2-register machine is a sequence $\kappa_1 \dots \kappa_m$ where, for $i = 1, \dots, m$, $\kappa_i = (p_i, d_i, e_i)$ is a configuration, which records the current control state p_i as well as the contents $d_i, e_i \in \mathbb{N}$ of the registers. Basically, we want to encode such a computation as a word of the form $p_1 \perp a^{d_1} \perp\!\!\!\perp b^{e_1} \$ \dots \$ p_m \perp a^{d_m} \perp\!\!\!\perp b^{e_m}$, where $a, b, \perp, \perp\!\!\!\perp, \$$ are new symbols. We then consider each symbol of this word as the root of a unary tree (of a certain depth) and connect these trees with a single root, thus obtaining a tree that represents the underlying computation.

This encoding is actually quite similar to the encoding of the solutions of PCP (Post's Correspondence Problem) as data words in [2], which is used to show that the satisfiability problem for the logic $\mathrm{FO}^3(\sim, S)$ over data words is undecidable. In fact, the unary trees mentioned in our encoding above can be seen as data values that are attached to the word encoding of a 2-register-machine computation (see also Section 5 below).

Although the reduction, given the encoding, is fairly standard, we encounter some technical difficulties arising from the restricted quantification patterns in our definition of UTACS constraints (i.e., only $\forall x \forall y$ and $\exists x \exists y$). As an illustration, consider the case of a word encoding of a 2-register machine computation which contains two consecutive configurations, say $\kappa = p \perp a^d \perp\!\!\!\perp b^e$ and $\kappa' = p' \perp a^{d'} \perp\!\!\!\perp b^{e'}$, which do not represent a correct execution of, say, an increment to the first register. This occurs if, for example, $e < e'$ (that is, κ' contains more b's than κ). Intuitively, this can be captured by expressing the following constraints: first, in each of κ and κ', all the unary trees attached to the b-positions are pairwise different; second, for each b-position of κ there exists a b-position in κ' with the same unary tree;

third, there exists a b-position in κ' such that the unary tree attached to it does not occur at the b-positions of κ. Notice the quantification patterns occurring in these constraints: $\forall x \forall y$, $\forall x \exists y$, and $\exists x \forall y$, respectively. The first quantification pattern, $\forall x \forall y$, can easily be handled by UTACS constraints. The third one, $\exists x \forall y$, can be handled by first nondeterministically guessing the position of x and then comparing this position with all positions y using UTACS constraints. The second quantification pattern, $\forall x \exists y$, however, cannot be captured by UTACS constraints, so we have to overcome this difficulty by putting some additional requirements on the unary trees used in the tree encoding of a halting computation.

5 UTACS and Data Languages

In this section, we are interested in a connection between languages of data words and tree automata with equality constraints, which is established by encoding data words as trees of a certain form. With this encoding, then, data equality amounts to equality between trees.

For the ease of exposition, we consider data words over Σ and $\mathbb{N}_+$, where Σ is a finite alphabet. A data word $w = w_1 \dots w_m$ is a finite sequence of pairs of the form $w_i = (a_i, d_i) \in \Sigma \times \mathbb{N}_+$. We encode such a data word as a tree of the form $\top\ (\ a_1\ \underline{d_1}\ a_2\ \underline{d_2}\ \dots\ a_m\ \underline{d_m}\ )$ where $\top$ is a new symbol and $\underline{d_i}$, for each $i = 1, \dots, m$, encodes the data value d_i (i.e., a positive integer) as a unary tree over $\{\bullet\}$ of depth d_i. Consequently, we can use UTACS accepting trees of this form to define languages of data words. For such a language of data words, in particular, the emptiness problem then amounts to the emptiness problem for UTACS, which, by Theorem 1, is decidable.

Following the notations of [16], more specifically, we consider a fragment of the logic $\mathrm{MSO}(\sim, <, S)$ over data words (with $\sim$ being the data-equality predicate; i.e., $x \sim y$ holds if the data value at position x is equal to the one at position y), which contains positive Boolean combinations of formulas of the form $\exists X_1 \dots \exists X_n. \Big(\theta(X_1, \dots, X_n) \wedge \alpha(X_1, \dots, X_n) \Big)$, where θ is an $\mathrm{MSO}(<, S)$-formula (i.e., without $\sim$) over Σ with free occurrences of the set variables $X_1, \dots, X_n$, and α is a positive Boolean combination of formulas of the forms:

$$\forall x \forall y. \left[\varphi(X_1, \dots, X_n, x, y) \rightarrow x \sim y \right] \qquad \exists x \exists y. \left[\varphi(X_1, \dots, X_n, x, y) \wedge x \sim y \right]$$
$$\forall x \forall y. \left[\varphi(X_1, \dots, X_n, x, y) \rightarrow x \not\sim y \right] \qquad \exists x \exists y. \left[\varphi(X_1, \dots, X_n, x, y) \wedge x \not\sim y \right]$$

where φ, in turn, is an $\mathrm{MSO}(<, S)$-formula (i.e., without $\sim$) over Σ with the free variables $X_1, \dots, X_n$ and x, y. Note that these kinds of formulas correspond to the types of constraints we have used in defining the transitions of UTACS. In fact, for every formula of the logic over data words described above, we can construct a UTACS recognizing the set of trees encoding the data words defined by the formula. This allows us to derive from Theorem 1 that the satisfiability problem for this logic is decidable. Furthermore, as remarked in Section 4, in the proof of Theorem 8 we actually encode computations of 2-register machines as data words. Consequently, the validity problem for this logic (i.e., the problem of determining, for a given formula, whether all data words satisfy this formula) is undecidable.

In comparison with the logic $\mathrm{FO}^2(\sim, <, S)$ (first-order logic with two variables) over data words, which is considered in [2], the logic described above seems to be weaker because of the restricted use of data comparisons. In particular, for the languages of data

words defined by our logic, the projection w.r.t. the finite alphabet always yields a regular language; this can be shown by using an analysis related to the notion of suitability used in Section 3. For instance, the language of data words (over the label alphabet $\{a, b\}$) satisfying "every two positions labeled with a carry different data values, and for each position labeled with a there exists a position labeled with b with the same data value" cannot be defined in our logic since the projection of this language w.r.t. $\{a, b\}$ yields a language which is not regular, but it can be defined in $\mathrm{FO}^2(\sim, <, S)$ (see [2]). On the other hand, formulas of our logic may use more than just two variables, while still being decidable. This allows us to define, for instance, the language of data words satisfying "between every two a-positions carrying the same data value there exists a b-position" by saying that every two a-positions with no b-position in between must carry different data values, that is, $\forall x \forall y. \left[x < y \wedge a(x) \wedge a(y) \wedge \neg(\exists z. x < z < y \wedge b(z)) \rightarrow x \not\sim y \right]$. To the best of our knowledge, it is still an open question whether this language can be expressed in $\mathrm{FO}^2(\sim, <, S)$ (see [16]).

6 Conclusions

We have shown that the emptiness problem for nondeterministic unranked tree automata with sibling equality and disequality constraints is decidable by extending the method we have proposed previously for the deterministic case. However, the precise complexity (both lower and upper bound) of the problem is still missing. One possible approach towards an upper bound for our method is to provide a different proof for the bound lemma which avoids the use of Dickson's Lemma.

We believe that the connection between our automaton model and languages of data words deserves further studies. Here, it might be worthwhile to study the precise relation of the logic over data words emerging from our automaton model to the existing formalisms for data languages and also to study its complexity.

We remark that our method of deciding emptiness for UTACS, actually, does not rely on the fact that trees are compared with respect to equality. In fact, our method still works if we consider, for instance, automata with sibling constraints with respect to structural equality (two trees are said to be structurally equal if they share the same set of nodes). Thus, we would like to study (in the unranked setting) automata with constraints regarding more general types of relations between trees than just equality, like, e.g., relations defined by tree transducers. This is actually related to some recent works on (ranked) tree automata with constraints. In the automaton models of [5, 6], constraints are posed not directly on the input subtrees, but, instead, on some output trees, called memories, which are produced during a bottom-up run of the automata. Similarly, in [11], one defines a function assigning to each tree a certain size and poses (numerical) constraints with respect to this size function.

Finally, it would be interesting to compare our automaton model with the automata defined in [10], where it is allowed to compare subtrees which are not necessarily siblings but may be remotely located in the tree.

Acknowledgements. We thank Luc Segoufin for many valuable discussions. We also thank the anonymous referees for their numerous comments and suggestions; due to space

limitations, however, we were not able to include all these suggestions.

References

[1] Bogaert, B., Tison, S.: Equality and disequality constraints on direct subterms in tree automata. In *Proc. STACS 1992*. *LNCS 577*. Springer (1992)

[2] Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In *Proc. LICS 2006*. IEEE Computer Society (2006)

[3] Caron, A.C., Comon, H., Coquidé, J.L., Dauchet, M., Jacquemard, F.: Pumping, cleaning and symbolic constraints solving. In *Proc. ICALP 1994*. *LNCS 820*. Springer (1994)

[4] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: *Tree Automata Techniques and Applications*. Available on `http://www.grappa.univ-lille3.fr/tata` (2007) Released on 12 October 2007.

[5] Comon, H., Cortier, V.: Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science* **331** (2005)

[6] Comon-Lundh, H., Jacquemard, F., Perrin, N.: Visibly tree automata with memory and constraints. *Logical Methods in Computer Science* **4** (2008)

[7] Dal Zilio, S., Lugiez, D.: XML schema, tree logic and sheaves automata. In *Proc. RTA 2003*. *LNCS 2706*. Springer (2003)

[8] Dauchet, M., Caron, A.C., Coquidé, J.L.: Automata for reduction properties solving. *Journal of Symbolic Computation* **20** (1995)

[9] Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics* **35** (1913)

[10] Filiot, E., Talbot, J.M., Tison, S.: Tree automata with global constraints. In *Proc. DLT 2008*. *LNCS 5257*. Springer (2008)

[11] Habermehl, P., Iosif, R., Vojnar, T.: Automata-based verification of programs with tree updates. In *Proc. TACAS 2006*. *LNCS 3920*. Springer (2006)

[12] Karianto, W., Löding, C.: Unranked tree automata with sibling equalities and disequalities. In *Proc. ICALP 2007*. *LNCS 4596*. Springer (2007) Full version appeared as Technical Report AIB-2006-13, RWTH Aachen University.

[13] Mongy-Steen, J.: *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, Université de Lille I (1981)

[14] Neven, F.: Automata, logic, and XML. In *Proc. CSL 2002*. *LNCS 2471*. Springer (2002)

[15] Schwentick, T.: Automata for XML – a survey. *J. Comput. Syst. Sci.* **73** (2007)

[16] Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In *Proc. CSL 2006*. *LNCS 4207*. Springer (2006)

[17] Seidl, H., Schwentick, T., Muscholl, A.: Counting in trees. In *Logic and Automata: History and Perspectives*. Amsterdam University Press (2008)

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Functionally Private Approximations of Negligibly-Biased Estimators

André Madeira[1*], **S.Muthukrishnan**[1,2]

[1] Rutgers University
Piscataway, NJ, USA
{amadeira,muthu}@cs.rutgers.edu

[2] Google Research
New York, NY, USA
muthu@google.com

ABSTRACT. We study functionally private approximations. An approximation function g is *functionally private* with respect to f if, for any input x, $g(x)$ reveals no more information about x than $f(x)$. Our main result states that a function f admits an efficiently-computable functionally private approximation g if there exists an efficiently-computable and negligibly-biased estimator for f. Contrary to previous generic results, our theorem is more general and has a wider application reach.
We provide two distinct applications of the above result to demonstrate its flexibility. In the data stream model, we provide a functionally private approximation to the L_p-norm estimation problem, a quintessential application in streaming, using only polylogarithmic space in the input size. The privacy guarantees rely on the use of pseudo-random *functions* (PRF) (a stronger cryptographic notion than pseudo-random generators) of which can be based on common cryptographic assumptions. The application of PRFs in this context appears to be novel and we expect other results to follow suit. Moreover, this is the first known functionally private streaming result for *any* problem.
Our second application result states that every problem in some subclasses of $\sharp\mathsf{P}$ of hard counting problems admit efficient and functionally private approximation protocols. This result is based on a functionally private approximation for the $\sharp\mathsf{DNF}$ problem (or estimating the number of satisfiable truth assignments to a Boolean formula in disjunctive normal form), which is an application of our main theorem and previously known results.

1 Introduction

Consider a two-party functionality $f(x_1, x_2) = (y_1, y_2)$, where (x_i, y_i) is the private input/output pair of party $i \in \{1, 2\}$. Informally, a *private computation* of f is one that computes f correctly and guarantees that each party i learns only y_i and nothing else.

Interestingly, Feigenbaum et al. [1] observed that the private computation of an approximation function $g(x_1, x_2) = (\tilde{y}_1, \tilde{y}_2)$ of f can potentially leak more information than the computation of f itself. Indeed, consider function $f(x_1, x_2)$ computing the Hamming distance between binary vectors x_1 and x_2. Let g be an approximation of f where the least significant bit of $g(x_1, x_2)$ corresponds to some arbitrary bit of x_1 and all the remaining bits of g equals those of f. Although g is indeed a good approximation, it leaks more information about x_1 than f does. In view of this problem, the authors argued that it is natural to

*Supported in part by NSF grant CCF-0728937.

© Madeira, Muthukrishnan; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 323–334
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2329

require that g be also *functionally private* with respect to f; i.e. roughly speaking, there should be no (or it is computationally infeasible to find an) i such that $\tilde{y}_i$ "leaks" more information than y_i does (we make it precise in Section 2). As approximations are often used in place of exact computations to reduce computing resources, the definition also captures the notion that efficiency and privacy should not be conflicting goals.

We observe that although a series of seminal results [5, 19] claim that *any* efficiently-computable (read polynomial-time) distributed protocol for a functionality f can be "compiled" into a *private protocol*, one cannot claim the same for approximations. Indeed, the functional privacy property is inherent to the *description* of g and not of any protocol computing g. Hence, there is no hope for a "compiler-like" solution for approximations. Consequently, the focus on *functional privacy* has been on designing protocols for a particular set of functions of interest (or classes thereof). Unfortunately, since the definition of functional privacy first appeared in [1] few results have surfaced. Most are either tailored protocols for specific functions of interest [1, 2, 8, 10] or impossibility results [6]. An exception are the more general feasibility results of [1] that claims functionally private approximations for a specific set of conforming Monte-Carlo simulations. Unfortunately, the results are limited in scope and rigid in their requirements as we outline and discuss in Section 3.

Our main result, on the other hand, roughly states that a function f admits an efficient functionally private approximation g if there exists an efficient *negligibly biased* estimator for f. The result is flexible enough under many circumstances. We demonstrate this point by providing two distinct applications of it. The first relates to a quintessential problem in the data stream model of computation [12]: the estimation of the L_p norm of vectors, which in the non-private streaming setting spurred several new results. The second is concerned with feasibility results for $\sharp\mathsf{P}$ problems. Before presenting our contributions, we start with some relevant context.

Private Streaming Computations. Consider two parties Alice and Bob. Alice sees an n-dimensional vector a given as a series of coordinate updates. The j^{th} update is (j, j_i, j_u) where $j_i \in [n]$ refers to the dimension of the vector, and j_u the change to that dimension, i.e., $a[j_i] \rightarrow a[j_i] + j_u$. We visualize a as the stream. Each update has to be processed quickly and there is only limited memory to store a. Formally, we are allowed space polylogarithmic in n and various parameters of interest, as well as similar update and processing time. Similarly, Bob is given input vector b given as a stream. When a function f needs to be computed at time t, Alice and Bob communicate with each other to evaluate $f(a_t, b_t)$ where a_t (b_t) denotes Alice's (resp. Bob's) vector at time t (hereafter, we drop the subscript t whenever the context allows). Total communication is in bits polylogarithmic in n and other parameters. This is the distributed data stream model [12].

Our focus is on achieving *functionally private* protocols in the streaming model. In this setting, as in general private computation, Alice and Bob do not wish to reveal the contents of their streaming data. This stringent requirement is a result of either binding legal reasons or sheer competitiveness. However, in the spirit of cooperation or as required by law, they might be willing to perform a specific data analysis task in a secure way. This is the context for the problems we study. For the purposes of this paper, we will address a common streaming analysis that is already well-studied in the literature [7, 11, 14] (but in a secure way) and not delve deeper into its many applications (which can be found in [12]). Specif-

ically, we consider the following problem: compute the L_p norm of vector $a - b$, denoted $L_p(a-b) = ||a-b||_p$, for $p \in [0,2]$. Recall that $L_p(x) = (\sum_i |x_i|^p)^{1/p}$. Nearly all non-trivial streaming analyses — including the problem above — are in fact approximate (exact computations are impossible without linear space [12]) and hence we focus on *functionally private* approximations.

Private Computation of ♯P-complete Problems. In this setting, Alice and Bob hold finite inputs a and b respectively. Similar in spirit as before, they wish to compute a ♯P-complete function f of their private inputs such that no information other than $f(a,b)$ (and whatever can be inferred from it) "leaks". However, as $f(a,b)$ is an intractable problem, they must settle on computing an efficiently-computable functionally private approximation instead.

Results. Our contributions are as follows:

1. We show that if there exists a *negligibly biased* estimator (NBE) $\mathcal{A}(x,\epsilon',\delta')$ of $f(x)$[†], which $\langle\epsilon',\delta'\rangle$-approximates[‡] f for $\epsilon' = 1/2$ and $\delta' = \mu(\kappa)$ in time $\mathsf{poly}(\kappa, \log|x|)$[§], and a public upper bound τ on $f(x)$, then there exists a *functionally private* $\langle\epsilon,\delta\rangle$-approximation g of f computable in time $\mathsf{poly}(\kappa, \log|x|, \log\tau, 1/\epsilon, \log(1/\delta))$ for a security parameter κ. Thus, if $\tau = \mathsf{poly}(|x|)$ as below, g is $\mathsf{polylog}(|x|)$-computable.
   The proof consists of taking enough samples from Bernoulli random variable (r.v.) with success probability $p = \mathcal{O}(\mathcal{A}(\cdot)/\tau)$ and ensuring $p = \Theta(1/c) \leq 1$ for a tight approximation using $\tilde{\mathcal{O}}(c)$ samples.[¶] The output then depends solely on $\mathsf{E}[\mathcal{A}(\cdot)/\tau]$. Since this is negligibly far from $f(x)/\tau$ we argue that functional privacy is implied.
   This is a *general* result for any function f and is not limited to any format as opposed to the feasibility results in [1]. We believe that it is of general interest and will prove useful to other functionally private protocols such as the following results.
2. We design a *functionally private* $\langle\epsilon,\delta\rangle$-approximation g for the L_p norm, $p \in (0,2]$, of an n-dimensional vector using $\tilde{\mathcal{O}}\left(\kappa^2 \log^2 n\right)$ bits of space on a security parameter κ.
   Our result is based on a slight adaptation of the recent non-private unbiased estimator for L_p [11] applied to our first result. To ensure functional privacy, we use a Pseudo-Random Functions (PRF), a stronger cryptographic notion than a Pseudo-Random Generator (PRG) that suffices for standard non-private streaming computations. Sampling from sketches and the use of PRFs in this context appear to be novel.
   From above, private streaming protocols for the L_p distance of two vectors follows. These are the first known private streaming protocols for any problem.
3. We design a *functionally private* $\langle\epsilon,\delta\rangle$-approximation g for the ♯DNF problem, or estimating the number of satisfiable assignments of a formula in disjunctive normal form, a ♯P-complete problem. In a nutshell, we rely on the result of Karp and Luby [9] to construct an unbiased estimator suitable for application of our first result.
   The result yields functionally private $\langle\epsilon,\delta\rangle$-approximations to all problems within some logic-based subclasses of ♯P. Specifically, we show that ♯DNF is complete under a private and approximation-preserving reduction for the $\sharp\Sigma_1$ and $\sharp\mathsf{R}\Sigma_2$ classes,

[†] informally, X is a NGE if $\mathsf{E}[X]$ is negligibly far from $f(x)$ and has finite variance. See Section 2 for details.
[‡] a function g $\langle\epsilon,\delta\rangle$-approximates f if $\Pr[|g(x) - f(x)| > \epsilon f(x)] \leq \delta$ for all inputs x.
[§] $\mathsf{poly}(n)$ ($\mathsf{polylog}$) means any polynomial in n (in $\log n$ respectively).
[¶] the notation $\tilde{\mathcal{O}}(n)$ should be read as $\mathcal{O}(n \log(1/\delta)/\epsilon^2)$ throughout the paper.

yielding functionally private approximations to all problems therein.

Although our goal is on achieving private protocols, we omit the details about constructing a secure two-party protocol. As Feigenbaum et al. [1] indicated, the challenge typically boils down to proving functional privacy when designing a private approximation protocol. Additionally, most of the construction details of a secure protocol are orthogonal to our main contributions in this paper. We refer the reader to [1, 3] for such details.

2 Preliminaries

Let $[m]$ denote the integer range $1, \ldots, m$. We denote a *negligible* function in a positive integer parameter κ by $\mu(\kappa) \in \kappa^{-w(1)}$. A function f is said to be *overwhelming* if $1 - f$ is negligible. Polynomial time means time polynomial in $n, 1/\epsilon$, and security parameter κ and is denoted by poly. Similarly, by polylog, we mean time polylogarithmic in n, but poly in $1/\epsilon$ and κ. Finally, we say a function is *efficient* if it is poly-time computable.

DEFINITION 1. *[$\langle\epsilon, \delta\rangle$-approximation] A function g is an $\langle\epsilon, \delta\rangle$-approximation of f if, $\forall x$, $\Pr[|g(x) - f(x)| > \epsilon f(x)] \leq \delta$ holds for arbitrary $\epsilon, \delta \in (0, 1)$. The function g depends on both ϵ and δ and the probabilistic guarantees are over the randomness of g.*

Below is the general notion of indistinguishability of distributions in Cryptography.

DEFINITION 2. *[indistinguishability of distributions] Two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ are said to be* computationally *indistinguishable, denoted $\mathcal{D}_1 \stackrel{c}{\equiv} \mathcal{D}_2$, if for every pair of random variables $X_1 \sim \mathcal{D}_1$ and $X_2 \sim \mathcal{D}_2$ and for any family of polynomial-size circuits $\{C_\kappa\}$ we have $|\Pr(C_\kappa(X_1) = 1) - \Pr(C_\kappa(X_2) = 1)| \leq \mu(\kappa))$ for a security parameter κ. Distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ are* statistically *indistinguishable, denoted $\mathcal{D}_1 \stackrel{s}{\equiv} \mathcal{D}_2$, if for any $X_1 \sim \mathcal{D}_1$ and $X_2 \sim \mathcal{D}_2$ the statistical distance $SD(X_1, X_2) = \frac{1}{2}\sum_a |\Pr[X_1 = a] - \Pr[X_2 = a]| \leq \mu(\kappa)$. Note that $\mathcal{D}_1 \stackrel{s}{\equiv} \mathcal{D}_2$ implies $\mathcal{D}_1 \stackrel{c}{\equiv} \mathcal{D}_2$ but not necessarily vice-versa.*

Consider the *functional privacy* definition for general approximations from [1].

DEFINITION 3. *[functional privacy [1]] A function g is* functionally private *with respect to a function f if there exists a probabilistic poly-time algorithm (a.k.a. simulator) $\mathcal{S}$ such that, for any input x, $\{\mathcal{S}(f(x))\} \stackrel{\tau}{\equiv} \{g(x)\}$ where $\stackrel{\tau}{\equiv}$ denotes either $\equiv, \stackrel{c}{\equiv}$, or $\stackrel{s}{\equiv}$.*

This definition captures the notion that the approximation output $g(x)$ does not reveal extra information about x besides what can be inferred from $f(x)$. Moreover, the functional privacy definition is independent of how g is computed or whether f is efficiently computable or not. Indeed, f could be a hard problem and thus $\mathcal{S}$ is modeled as having only access to $f(x)$ and not an oracle access to f.

3 Functional Privacy: current techniques and limitations

The seminal work of [1] presented a feasibility result for the following set of functions. Consider a two-party computation where Alice and Bob hold private inputs a and b of size n respectively and let x represent the input pair (a, b). Let $f(x) = \psi(\Pr[\xi])$, where ξ is an event or Bernoulli trial parameterized by a and b and ψ is an approximation-preserving

function that is efficient to compute and invert. It was shown that f admits an efficient *functionally private approximation* g as long as $\mathsf{Pr}[\xi] \geq 1/\mathsf{poly}$. Essentially, g is constructed by applying ψ to the outcome of a sampling algorithm estimating $\mathsf{Pr}[\xi]$ directly from a and b via poly independent samples. Correctness follows from Chernoff bounds. On the other hand, the functional privacy simulator works as follows: given $f(x)$, apply ψ to poly independent samples of a Bernoulli random variable with success probability equal to an $\Omega(\kappa)$-bit approximation of $\mathsf{Pr}[\xi] = \psi^{-1}(f(x))$. Functional privacy follows from the fact that the simulated distribution is statistically indistinguishable (in a security parameter κ) from the one induced by g —and thus also computationally indistinguishable. Additionally, [1] extended the results to functions of the form $f(x) = \psi(\phi(\xi_1, \xi_2, \ldots, \xi_t))$ for a polynomial-size, constant-depth arithmetic formula $\phi(\cdot)$ of "coin manipulation" gates.[‖]

We outline some problems with the above feasibility results. The main drawback is the stringent structure on $f(x) = \psi(\phi(\cdot))$. It restricts f to be the result of some Monte-Carlo experiment, where coin manipulations suffices in making $\phi(\cdot)$ simulatable from $f(x)$ alone using $\psi^{-1}(\cdot)$. Unfortunately, this structure might not always be easily attainable. Indeed, for the problem we consider in Section 5, an efficient (and *known*) solution is to construct a coin $\phi(\cdot) = f(x)/h(x)$ for a function $h(x)$ not inferred from $f(x)$ alone. It turns out that $h(x)$ depends on the *structure* of x and thus of private inputs a and b. In that case, $\psi^{-1}(f(x))$ cannot yield $f(x)/h(x)$ properly as required without the knowledge of $h(x)$.

A second drawback is the requirement that $\mathsf{Pr}[\xi] \geq 1/\mathsf{poly}$. Essentially, it requires taking poly samples for a tight approximation. This might be prohibitive for very large inputs. In many cases, the only acceptable goal is to take $\mathsf{polylog}$ samples, as the sampling complexity is closely related to the communication complexity of a private distributed protocol [1]. Specifically, when a tighter range for $\mathsf{Pr}[\xi]$ is known, it is reasonable to expect a much better sampling complexity. Indeed, that is the case of the stand-alone private protocol of [8], which reduces the sampling complexity to $\mathsf{poly}(\kappa, \log n)$ by ensuring that $\mathsf{Pr}[\xi] \in \Theta(1/\kappa)$.

We address both concerns simultaneously. Roughly speaking, we show that it suffices to design a *negligibly biased* estimator (NBE) that $\langle \epsilon, \delta \rangle$-approximates f for f to admit a *functionally private approximation* g. Contrary to above, the NBE carries no restriction. For example, the NBE can be constructed out of a Monte-Carlo experiment or in any other way. In other words, it is applicable to *any* function f as long as a suitable NBE is available. Therefore, our result widens and also encompasses the previous feasibility results of [1].

3.1 Randomness in Private Streaming

Although there are a few deterministic streaming results (c.f. [12]), most streaming protocols employ the use of randomization. The amount of randomness required varies and typically ranges between pairwise and full independence. In particular, the streaming problem we consider in this paper requires $\mathcal{O}(n)$ fully independent random variables, where n is the stream size. Unfortunately, truly independence requires $\Omega(n)$ random bits, a prohibitive storage requirement for data streaming applications. In such cases, a common approach is

[‖] The gates result from the observation that given two independent coins with unknown probabilities $p, q \geq 1/\mathsf{poly}$, one can construct (in poly time) coins with probabilities $p \cdot q$, $1 - p$, or any convex combination of p, q.

to use a Pseudo-Random Generator (PRG) suitable for space-bounded computations. Indyk [7] pioneered this approach by using Nisan's PRG [15] construction, which fools space-bounded algorithms. An interesting property of the PRG is that it provides easy access to any bits of the pseudo-random pad. The property is used to ensure that any bit can be accessed efficiently every time it is requested; a critical part for streaming applications.

Unfortunately, space-fooling PRGs are not sufficient for functional privacy. In short, the security convention in Cryptography is to bound the adversary to $\mathsf{poly}(\kappa)$-time as opposed to $\mathcal{O}(\kappa)$-space for a security parameter κ. A typical adversary in the former model can break the randomness security in the latter (c.f. [3]).

In this paper, we consider a different approach. In a nutshell, we employ the use of a Pseudo-Random Function (PRF) [4] as follows. A brief review of PRF is informative. Let I_κ denote the set of all κ-bit strings. Consider H_κ the set of all functions from I_κ into I_κ (note that $|H_\kappa| = 2^{\kappa \cdot 2^\kappa}$). Let $F = \{F_\kappa\}$ be a function ensemble where F_κ assumes values from H_κ. Then, F is a PRF if it has the following properties: (a) indexing: each function in F_κ has a unique κ-bit index associated with it $F_\kappa = \{f_s | s \in I_\kappa\}$; (b) poly-time evaluation: $f_s(x)$ can be computed in $\mathsf{poly}(\kappa)$-time given $s \in I_\kappa$ and $x \in I_\kappa$; and (c) pseudo-randomness: no $\mathsf{poly}(\kappa)$-time probabilistic algorithm can distinguish the functions in F_κ from the ones in H_κ. Intuitively, given a κ-bit truly-random seed string s, a function f_s chosen from F_κ is as good as a random function to any $\mathsf{poly}(\kappa)$ adversary.

Many PRF constructions exist and suffice for our results. Our result in Section 5, however, uses the PRF construction of [13] because, to the best of our knowledge, it is currently the most efficient construction regarding the evaluation of $f_s(x)$.

4 Functional Privacy of Negligibly Biased Estimators

Consider a positive single-output deterministic function f with input size n. Our result is inspired in a technique implicit in the private protocol of [8]. We begin with a new definition.

DEFINITION 4. *[negligibly biased estimator (NBE)] A random variable X is a* negligibly biased *estimator for $f(x)$ in a parameter $\kappa \in \mathbb{N}$ if, for any admissible input x, $\mathsf{E}[X] \in (1 \pm \mu(\kappa))f(x)$ and $\mathrm{Var}[X] < \infty$.*

Observe that securely computing an NBE is not necessarily a functionally private approximation. Indeed, the higher moments of such computation depend on the input x. The following theorem attempts in squashing them and remove non-simulatable information.

THEOREM 5. *Suppose there exists an algorithm $\mathcal{A}(x, \epsilon', \delta')$ that $\langle 1/2, \mu(\kappa)\rangle$-approximates a positive function $f(x)$ with the following conditions. For any input x:*

a) $\mathcal{A}$ is a negligibly biased *estimator for $f(x)$ in a security parameter $\kappa \in \mathbb{N}$;*

b) $\exists$ an upper bound τ of $f(x)$, which is considered public knowledge.

Then, f admits a functionally private $\langle \epsilon, \delta\rangle$-approximation function such that:

1. *it is computable in time $\mathcal{O}\big((\log \tau)(\kappa + \log(\log \tau) + \log(1/\delta)/\epsilon^2) \cdot T_{\mathcal{A}}(|x|, 1/2, \mu(\kappa))\big)$;*
2. *uses $\mathcal{O}\big((\log \tau + \log \kappa + \log\log[(1/2\delta)/\epsilon^2]) + S_{\mathcal{A}}(|x|, 1/2, \mu(\kappa))\big)$ of space,*

where $T_{\mathcal{A}}(n, \epsilon', \delta')$ and $S_{\mathcal{A}}(n, \epsilon', \delta')$ are the running time and space usage of $\mathcal{A}(x, \epsilon, \delta)$ resp..

PROOF. We prove it constructively; i.e. we show how Function 1 achieves the claims. Let $\mathrm{Bernoulli}(q)$ represents a Bernoulli r.v. with success probability q.

Inputs: input x and parameters $\tau \geq f(x)$, $\epsilon \in (0,1)$, $\delta \in (0,1)$, security parameter κ, and access to a NBE $\mathcal{A}(x, \epsilon', \delta')$ for $\epsilon' = 1/2$ and $\delta' = \mu(\kappa)$.

Output: a functionally private $\langle \epsilon, \delta \rangle$-approximation of $f(x)$

1. Let $N = \Theta(\kappa + \log(\log \tau) + \log(2/\delta)/\epsilon^2)$
2. For each iteration $i = 0, \ldots, \lceil \log \tau \rceil$:
   (a) Compute $Z_i = \sum_j^N Z_{i,j}$, where each $Z_{i,j}$ is the outcome of an independent trial of
   $$\text{Bernoulli}\left(\frac{\mathcal{A}(x, 1/2, \mu(\kappa))}{(3/2)(\tau/2^i)}\right) \quad (1)$$
   until iteration ℓ where Z_ℓ exceeds $N/8$.
   (b) Abort if any call to $\mathcal{A}(\cdot) > (3/2)(\tau/2^i)$ and output failure.
3. Output $F = Z_\ell \cdot (3/2)(\tau/2^\ell)/N$

Function 1: Functionally private approximation function given an NBE.

Correctness. For each iteration $i = 0, 1, \ldots, \lceil \log \tau \rceil$, let the collection of r.v.s $\{X_{i,j}\}_{j \in [N]}$ represent the N independent outcomes of calling $\mathcal{A}(x, 1/2, \mu(\kappa))$. Each $X_{i,j}$ is an negligibly biased $\langle 1/2, \mu(\kappa) \rangle$-approximation of $f(x)$; i.e. with overwhelming probability in κ it holds that a sample from $X_{i,j} \in (1 \pm 1/2) f(x)$ and $\mathsf{E}[X_{i,j}] = (1 \pm \mu(\kappa)) f(x)$. As in Function 1, define Bernoulli r.v.s $\{Z_{i,j}\}_{j \in [N]}$ where each $Z_{i,j}$ has success probability $p_{i,j} = X_{i,j}/[(3/2)(\tau/2^i)]$.

Let $Z_i = \sum_j^N Z_{i,j}$. Also, let ℓ be the smallest index such that $Z_\ell > N/8$ as stated in Function 1 and let ℓ' be the index such that $\tau/2^{\ell'+1} \leq f(x) < \tau/2^{\ell'}$ (note that there is always such an index by definition of τ and iteration range of ℓ'). First, note that for any iteration $i = 0, 1, \ldots, \ell'$, $p_{i,j} \leq 1$ because $\tau/2^{\ell'} \geq f(x)$ and the confidence guarantees of $\mathcal{A}(\cdot)$ hold overwhelming in κ; i.e. only with $\mu(\kappa)$ probability, the protocol aborts and we can safely assume this does not happen. Therefore, all sample probabilities are proper in that range. We then show that $\ell \leq \ell'$ always holds; i.e. $Z_{\ell'} \geq N/8$ holds with overwhelming probability in κ. Indeed, the expectation of the Bernoulli trials at iteration ℓ' is

$$\mathsf{E}[Z_{\ell',j}] = \mathsf{E}\left[\frac{\mathcal{A}(x, 1/2, 2^{-\kappa})}{(3/2)(\tau/2^{\ell'})}\right] \geq \frac{\mathsf{E}[\mathcal{A}(x, 1/2, 2^{-\kappa})]}{(3/2)(2f(x))} = \frac{(1 \pm \mu(\kappa)) f(x)}{3f(x)} \geq 1/4.$$

In turn, $\mathsf{E}[Z_{\ell'}] \geq N/4$ by linearity of expectations and thus

$$\mathsf{Pr}\left[Z_{\ell'} < N/8\right] \leq \mathsf{Pr}\left[Z_{\ell'} < (1/2)\mathsf{E}[Z_{\ell'}]\right] \leq \left(\frac{e^{-1/2}}{(1/2)^{(1/2)}}\right)^{\mathsf{E}[Z_{\ell'}]} \leq e^{-N/8} \leq \mu(\kappa),$$

which follows from a Chernoff bound and choice of N. Therefore, a suitable index $\ell \leq \ell'$ can be found in at most $\log(\tau) + 1$ iterations overwhelmingly in κ.

Now, recall that the output is $F = Z_\ell \cdot (3/2)(\tau/2^\ell)/N$. For the possible candidate exit iterations $i \leq \ell$, we have that

$$\mathsf{E}[Z_i] = \mathsf{E}\left[\sum_j^N Z_{i,j}\right] = \sum_j^N \mathsf{E}\left[\frac{\mathcal{A}(x, 1/2, 2^{-\kappa})}{(3/2)(\tau/2^i)}\right] = N\frac{f(x)}{(3/2)(\tau/2^i)} = \Theta(N).$$

Thus, by a Chernoff bound and union bound over the iterations,

$$\begin{aligned}\Pr[F > (1+\epsilon)f(x)] &= \log(\tau)\cdot\Pr\left[Z_i\cdot(3/2)(\tau/2^i)/N > (1+\epsilon)f(x)\right]\\ &= \log(\tau)\cdot\Pr[Z_i > (1+\epsilon)\mathsf{E}[Z_i]]\\ &\le \log(\tau)\cdot e^{-\Theta(N)\frac{\epsilon^2}{3}} \le e^{-(\kappa\epsilon^2+\log(2/\delta))} \le \delta/2.\end{aligned}$$

A similar result holds for $\Pr[F < (1-\epsilon)f(x)] \le \delta/2$. Therefore, we have shown that $\Pr[F \in (1\pm\epsilon)f(x)] \ge 1-\delta$ as desired. The running time follows from at most $\log(\tau)$ iterations of $\tilde{\mathcal{O}}(\kappa)$ independent samples of $T_{\mathcal{A}}(n, 1/2, \mu(\kappa))$. Space follows as one $\log\tau$-bit counter and one $\log N$-bit counter suffice for computing the Z_i's.

Privacy. F is functionally private to $f(x)$ as the Bernoulli trials can be simulated by an algorithm with similar skeleton as Function 1 but with success probabilities

$$p_{i,j} = \frac{f(x)}{(3/2)(\tau/2^i)}$$

instead in (1) (recall that $f(x)$ is given to the simulator, see Definition 3). Now, note that they are statistically indistinguishable from the protocol trials because each $X_{i,j} = \mathcal{A}(x,\epsilon',\delta')$ is a negligibly biased estimator of $f(x)$; i.e. $\mathsf{E}[\mathcal{A}(x,\epsilon',\delta')] = (1\pm\mu(\kappa))f(x)$ overwhelmingly in κ for $\epsilon' = 1/2$ and say $\delta' = 2^{-\Theta(\kappa)}$.** Indeed, the samples gathered until the last iteration ℓ were generated from proper probabilities (≤ 1) as argued earlier. Finally, recall that the higher moments of the Bernoulli random variables depend solely on its expectation —thus effectively squashing any non-simulatable higher moments of $\mathcal{A}(\cdot)$. Since τ is considered public, functional privacy is implied. ∎

Remark. The theorem is most useful when the upper bound τ is at most single-exponential in $f(x)$; as we shall see in the next section.

5 Functionally Private Streaming Approximation for the L_p Norm

The L_p norm, for $p \in (0,2]$, of a vector $a \in \{-M, M\}^n$ is defined as $L_p(a) = ||a||_p = (\sum_i^n |a_i|^p)^{1/p}$. In this section, we prove the following theorem.

THEOREM 6. *There exists a* functionally private $\langle\epsilon,\delta\rangle$*-approximation of* $||a||_p$, $p\in(0,2]$, *in the streaming setting, requiring only* $\mathcal{O}\left(\kappa^2\log^2(nM)(\kappa+\log(1/\delta)/\epsilon^2)\right)$ *bits of space, and* $\mathcal{O}\left(\kappa^2\log(nM)(\kappa+\log(1/\delta)/\epsilon^2)\right)$ *update and* $\mathcal{O}\left(\kappa\log^2(nM)(\kappa+\log(1/\delta)/\epsilon^2)\right)$ *update query time for arbitrary* $\epsilon,\delta\in(0,1)$ *and security parameter* κ.

Before proceeding, it is instructive to recall the estimator of [11].

Geometric Mean Unbiased Estimator for L_p [11]. Let $\mathbf{R}$ be the $\mathbb{R}^{\ell\times n}$ projection matrix with i.i.d. entries $R_{i,j} \sim S(p,1)$, where $S(p,\gamma)$ denotes a discretized symmetric p-stable distribution over $\mathbb{R}$ with scale parameter γ. Let $x = \mathbf{R}a$ be the "sketch" of a as $\ell \ll n$ (ℓ is set later).

**let us not confuse ϵ' and δ' with ϵ and δ. The former parameters are the ones used for invoking the NBE $\mathcal{A}$, while the latter are the error and confidence parameters of the functionally private approximation function.

By the properties of the distribution, each $x_j = \sum_i a_i R_{i,j} \sim ||a||_p X_j$, where $X_j \sim S(p,1)$. Equivalently, we can write $x_j \sim S(p, ||a||_p)$. Such distributions exists for $p \in (0,2]$. Thus, to estimate $||a||_p$, it boils down to approximating the scale parameter γ from ℓ i.i.d. samples. In [7], the author proposed using the estimator median$(|x_1|, |x_2|, \ldots, |x_\ell|)$. However, [11] has shown that not only it is severely biased but also hard to bias-correct it analytically or algorithmically. Therefore, for $p \in (0,2]$, [11] proposed using a bias-corrected version of the *geometric mean* estimator:

$$\hat{L}_{p,\mathrm{gm}} = \prod_{j=1}^{\ell} |x_j|^{1/\ell} \Big/ \left[\frac{2}{\pi}\Gamma\left(\frac{p}{\ell}\right)\Gamma\left(1-\frac{1}{\ell}\right)\sin\left(\frac{\pi}{2}\frac{p}{\ell}\right)\right]^{\ell}, \tag{2}$$

where $\Gamma(z)$ is the Gamma function of a real-valued z. The estimator is strictly unbiased, or $\mathsf{E}[\hat{L}_{p,\mathrm{gm}}] = ||a||_p$. Moreover, it has finite variance and exponential tail bounds, crucial for an $\langle \epsilon, \delta \rangle$-approximation of $||a||_p$ for arbitrary $\epsilon, \delta \in (0,1)$.

The correctness of the construction relies on building the projection matrix $\mathbf{R}$ from truly random samples. Unfortunately, that requires $\Omega(n\ell)$ bits of storage. By using the Pseudo-Random Function construction of [13] instead (see Section 3.1) we only need to store a κ-bit seed s_j per each sample $j \in [\ell]$. This is correct as long as $\kappa = \Omega(\log n)$ because we use the vector coordinate $i \in \{0,1\}^{\log n}$ as input for the PRF given seed s_j.

Proof of Theorem 6. We transform the *unbiased geometric estimator* $\hat{L}_{p,\mathrm{gm}}$ of (2) to an NBE with $\Omega(\kappa)$-bit precision. The theorem then follows by applying Theorem 5.

Specifically, for $p = 1$, the denominator in (2) simplifies to $[2\sin(\pi/2\ell)/\sin(\pi/\ell)]^{\ell} = 1/\cos^{\ell}(\pi/2\ell)$. It is known that it suffices to use $\mathcal{O}(\log 1/\epsilon)$ terms to $(1 \pm \epsilon)$-approximate $\cos^{\ell}(x)$ (by bounding the Taylor polynomial), or in our case $\mathcal{O}(\kappa\ell)$ terms to $(1 \pm \mu(\kappa))$-approximate For $p = 2$, the same denominator simplifies to $[p\Gamma(1/\ell)/\Gamma(1/p\ell)]^{\ell}$. Approximating it negligibly in κ implies getting an $(1 \pm 2^{-\kappa\ell})$ approximation to the Gamma function (note the power ℓ). A result from [17] does so with $\mathcal{O}(\kappa\ell)$ time with relative error $2^{-\kappa\ell}$. A similar argument applies for $p \in (0,2]$. Finally, observe that for agreed-upon values of κ, ϵ, and δ, the correction factor can be pre-computed (the theorem claims assume this fact).

Now, we validate our storage claims. Recall that Theorem 5 makes at most $\mathcal{O}(\log \tau) \cdot N$ invocations to the NBE $\mathcal{A}$, where τ is an upper bound on $f(x)$ and $N = \Theta(\kappa + \log(\log \tau) + \log(1/\delta)/\epsilon^2)$. Since $\tau \leq nM^2$ (for any $p \in (0,2]$)) we have $\mathcal{O}\big(\log(nM)(\kappa + \log(1/\delta)/\epsilon^2)\big)$ invocations of $\mathcal{A}$.[††] On the other hand, each invocation of $\mathcal{A}$ requires taking ℓ samples (or sketches). In [11], it was shown that setting $\ell = \mathcal{O}\big(\log(1/\delta)/\epsilon^2\big)$ suffices for an $\langle \epsilon, \delta \rangle$-approximation of $||a||_p$ using (2). Since $\mathcal{A}$ is called with $\epsilon' = 1/2$ and $\delta' = \mu(\kappa) = 2^{-\Theta(\kappa)}$ in Theorem 5, we have that each invocation requires $\ell = \mathcal{O}\Big(\log(1/2^{-\Theta(\kappa)})/(1/2)^2\Big) = \mathcal{O}(\kappa)$ sketches. Therefore, multiplying the number of invocations by the number of samples we get that the total storage requirement is $\mathcal{O}\big(\kappa \log(nM)(\kappa + \log(1/\delta)/\epsilon^2)\big)$ sketches. Each of these sketches require a counter and a κ-bit seed for the PRF. The former requires $\mathcal{O}(\log(nM))$ bits as the maximum value for $f(x)$ is nM^2 for any $p \in (0,2]$. Thus, the total storage is $\mathcal{O}\Big(\kappa^2 \log^2(nM)(\kappa + \log(1/\delta)/\epsilon^2)\Big)$ bits as desired.

[††] assuming $\kappa = \Omega(\log\log\tau) = \Omega(\log(\log nM))$.

The amount of computation per update per sketch is dominated by κ modulo multiplications and one exponentiation of κ-bit numbers when using the PRF construction of [13]. These can be performed in $\mathcal{O}(\kappa)$ constant-time computations. The update time is thus simply $\mathcal{O}\big(\kappa^2 \log(nM)(\kappa + \log(1/\delta)/\epsilon^2)\big)$ as desired assuming that operations on $\mathcal{O}(\log(nM))$-bit strings are constant. Finally, the query time is $\mathcal{O}\big(\kappa^2 \log(nM)(\kappa + \log(1/\delta)/\epsilon^2)\big)$ as the work of Function 1 is simply linear in the storage size once all sketches are available. ∎

6 Private Approximation of ♯P-complete problems

Consider the following abstract problem. Let U be a finite set whose elements are binary strings of size n. Let the Boolean function $h : U \to \{0,1\}$ partition U. The goal is to estimate the cardinality of $D = \{u | u \in U \wedge h(u) = 1\}$. Most problems in ♯P can be formulated as the problem above. Indeed, ♯P can be seen as the class of function problems counting the number of accepting paths in an NP machine [18]. In this section, we focus on obtaining efficient (read poly-time) functionally private approximations to the above abstract problem as exact solutions are typically not feasible.

Monte-Carlo sampling methods are useful in estimating $\mu = |D|/|U|$. From Chernoff bounds, an $\langle \epsilon, \delta \rangle$-approximation is possible using $\tilde{\mathcal{O}}(1/\mu)$ independent samples of $h(u)$ for an u chosen uniformly at random (u.a.r.) from U. An *efficient* algorithm, however, requires that $\mu \geq 1/\mathsf{poly}$ provided that it is poly-time computable to sample an element u u.a.r. from U and compute $h(u)$. Unfortunately, μ may be exponentially small in n, requiring a prohibitive super-polynomial samples. An alternative approach is the method of Karp and Luby [9]. The crux is on finding a small enough multiset V, containing all elements of D, such that $\mu = |D|/|V|$ is large enough for efficient sampling. The following theorem summarizes their *coverage algorithm*, as it is known, for an abstract Union of Sets problem.

THEOREM 7. *[Karp and Luby [9]] Let U and D be defined as before. Suppose there are sets $\{D_1, \ldots, D_m\} \subseteq D$ s.t. $D = \bigcup_i^m D_i$ and the following conditions hold, $\forall i \in [m]$:*

1. *$|D_i|$ can be computed in $\mathsf{poly}(n, m)$ time;*
2. *any element $s \in D_i$ can be sampled u.a.r. from D_i in $\mathsf{poly}(n, m)$ time;*
3. *given any $s \in D$, it can be decided if $s \in D_i$ in $\mathsf{poly}(n, m)$ time.*

Then, an $\langle \epsilon, \delta \rangle$-approximation for $|D|$ can be computed in $\mathsf{poly}\,(n, m, 1/\epsilon, \log(1/\delta))$ time.

Private Coverage Algorithm. What prevents the *coverage algorithm* from being functionally private to f using current techniques is the fact that $|V|$ depends on x. Indeed, $|V|$ cannot be inferred from $f(x)$ alone and thus the higher moments of the distribution induced by X depends on the structure of x and thus breaks functional privacy (c.f. Section 2).

Let X_j be a Bernoulli r.v. representing the j^{th} sample of a coin with success probability $p = |D|/|V|$ as in the proof of Theorem 7 [9]. Alternatively, one might be tempted to construct an event "$Y_j = 1$" where Y_j is a Bernoulli r.v. with probability $q = |V|/\tau$ and sample from the joint Bernoulli distribution $\mathsf{E}\big[\text{"}X_j = 1\text{"} \text{ and } \text{"}Y_j = 1\text{"}\big] = p \cdot q = \frac{|D|}{|V|} \cdot \frac{|V|}{\tau} = \frac{|D|}{\tau}$ for a publicly known value τ (or one that can be inferred from $f(x)$), where $p = \mathsf{E}\big[X_j\big]$. That way the output distribution depends solely on $|D|$ (and no-harm τ) and functional privacy is implied by the feasibility results of [1] using their formula $f(x) = \psi(\phi(\cdot))$ where

$\phi = p \cdot q$ and $\psi(n) = n \cdot \tau$ (see Theorem 6.4 in [1]). However, we note that in this case τ must be larger than $|V|$ so that the coin is proper. Unfortunately, the only known upper bound on $|V|$ we know of without knowing x is $m2^n$ as every element can be part of each set D_i. In such case, $q = |V|/(m2^n) < 1/\mathsf{poly}$ for small values of $|V|$ and no efficient sampling is possible.

Our approach instead is to squash the higher moments of X to prevent non-simulatable information from leaking. To that end, we use the unbiased coverage algorithm of Theorem 7 as the *negligibly biased* estimator in our main theorem, Theorem 5. The result is below.

THEOREM 8. *Let U, D and V and the set forth conditions on them be as in Theorem 7. Furthermore, suppose there exists a publicly known upper bound τ on $f(x)$. Then there exists a functionally private $\langle \epsilon, \delta \rangle$-approximation for $|D|$ in $\mathsf{poly}(\kappa, n, m, \log \tau, 1/\epsilon, \log(1/\delta))$ time for a security parameter κ.*

PROOF. Let $\mathcal{A}(x, \epsilon, \delta)$ be the coverage algorithm of Theorem 7. The theorem follows from a direct application of Theorem 5 using $\mathcal{A}$ with parameters $\epsilon = 1/2$ and $\delta = \mu(\kappa)$ and upper bound $\tau = 2^n$. ∎

Private ♯DNF. Let $F = \bigvee_i^m C_i$, be a propositional formula in *disjunctive normal form* where each C_i is a conjunction of a subset of literals defined with respect to n Boolean variables $x_1, \ldots, x_n$. The goal is to output the number of satisfiable assignments to F, or $\sharp F$. The problems is ♯P-complete [18]. In [9], Karp and Luby also showed a connection between the abstract Union of Sets problem and ♯DNF. Our result below uses this connection.

COROLLARY 9. *There exists a functionally private $\langle \epsilon, \delta \rangle$-approximation for ♯DNF computable in $\mathsf{poly}(n, m, 1/\epsilon, \log(1/\delta))$ time.*

PROOF. The claims follows directly from Theorem 8. Essentially, we show set $D = \bigcup_i^m D_i$ can be built as required and the conditions put forth in Theorem 7 (and Theorem 8) hold. Let each D_i be the set of assignments satisfying clause C_i. Then, clearly $\sharp F = |D|$. The conditions are met as follows, $\forall i \in [m]$: 1) $|D_i|$ can be computed in $\mathcal{O}(1)$ as $|D_i| = 2^{n-|C_i|}$; 2) sampling an element $s \in D_i$ u.a.r. from D_i requires setting the proper assignments for the literals in C_i and choosing u.a.r. from $\{\mathsf{true}, \mathsf{false}\}$ for the other literals not in C_i; and 3) trivial to evaluate whether or not $s \in D_i$ for any $s \in D$ in $\mathcal{O}(n)$ time. The corollary follows. ∎

Further Applications. In [16], it was shown that ♯$k$logDNF (a special case of ♯DNF restricting the formula to at most $k \log n$ variables per disjunct.) and ♯DNF are complete for classes $\sharp\Sigma_1$ and $\sharp\mathsf{R}\Sigma_2$ respectively. These are logic-based classes of counting problems. The problems are complete under a product reduction, which is a reduction from f to g where $\exists \phi, h \in \mathsf{FP}, h : \mathbb{N} \to \mathbb{N}$ such that $\forall x, f(x) = g(\phi(x)) \cdot h(|x|)$, with FP being the complexity class of polynomial-time computable functions problems.

Observe that the reduction is private and approximation-preserving. Note that $h(|x|)$ not only preserves approximability but also does not leak anything about x. We conclude that a functionally private $\langle \epsilon, \delta \rangle$-approximation to g implies one to f. Consequently, we have that all problems in $\sharp\Sigma_1$ and $\sharp\mathsf{R}\Sigma_2$ can be privately approximated, including problems such as ♯NON-VERTEX-COVERS, ♯NON-CLIQUES, ♯NON-DOMINATING-SETS, and ♯NON-HITTING-SETS to cite a few (c.f. [16]). We defer details to the full version.

References

[1] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J. Strauss, and Rebecca N. Wright. Secure Multiparty Computation of Approximations. *ACM Transactions on Algorithms (TALG)*, 2(3):435–472, 2006.

[2] Michael Freedman, Kobbi Nissim, and Benny Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT'04: Advances in Cryptology*, pages 1–19, 2004.

[3] Oded Goldreich. *Foundations of Cryptography: Volume II, Basic Applications*, volume 2. Cambridge University Press, New York, NY, USA, July 2004.

[4] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions. *Journal of the ACM*, 33(4):792–807, 1986.

[5] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play ANY mental game. In *STOC'87*, pages 218–229, New York, NY, USA, 1987. ACM.

[6] Shai Halevi, Robert Krauthgamer, Eyal Kushilevitz, and Kobbi Nissim. Private Approximation of NP-hard Functions. In *STOC'01*, pages 550–559, 2001.

[7] Piotr Indyk. Stable Distributions, Pseudorandom Generators, Embeddings, and Data Stream Computation. *Journal of the ACM*, 53(3):307–323, 2006.

[8] Piotr Indyk and David P. Woodruff. Polylogarithmic private approximations and efficient matching. In *TCC'06*, volume 3876, pages 245–264, 2006.

[9] Richard M. Karp and Michael Luby. Monte-Carlo Algorithms for Enumeration and Reliability Problems. In *SFCS'83*, pages 56–64. IEEE Computer Society, 1983. An extended abstract appeared in FOCS'97.

[10] Joe Kilian, André Madeira, Martin J. Strauss, and Xuan Zheng. Fast Private Norm Estimation and Heavy Hitters. In *TCC'08: Proceedings of the Fifth Theory of Cryptography Conference*, pages 176–193, New York, NY, USA, 2008. Springer Berlin/Heidelberg.

[11] Ping Li. Estimators and Tail Bounds for Dimension Reduction in $l_\alpha (0 < \alpha \leq 2)$ Using Stable Random Projections. In *SODA'08*, pages 10–19, Philadelphia, PA, USA, 2008.

[12] S. Muthukrishnan. *Data Streams: Algorithms and Applications*, volume 1. Now Publishers, August 2005.

[13] Moni Naor and Omer Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. *Journal of the ACM*, 51(2):231–262, 2004.

[14] J. Nelson and D. P. Woodruff. Revisiting Norm Estimation in Data Streams. *ArXiv e-prints*, nov 2008.

[15] N. Nisan. Pseudorandom Generators for Space-Bounded Computations. In *STOC'90*, pages 204–212. ACM, 1990.

[16] Sanjeev Saluja, K. V. Subrahmanyam, and Madhukar N. Thakur. Descriptive Complexity of $\sharp\mathsf{P}$ Functions. *Journal of Computer and System Sciences*, 50(3):493–505, June 1995.

[17] J. L. Spouge. Computation of the gamma, digamma, and trigamma functions. *SIAM Journal on Numerical Analysis*, pages 931–944, 1994.

[18] Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[19] Andrew C. Yao. Protocols for Secure Computations. In *SFCS'82*, pages 160–164, 1982.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Nash Equilibrium in Generalised Muller Games

Soumya Paul[1] **and Sunil Simon**[2]

[1] The Institute of Mathematical Sciences
C.I.T. Campus, Chennai 600 113, India.
soumya@imsc.res.in

[2] Institute for Logic, Language and Computation
University of Amsterdam.
s.e.simon@uva.nl

ABSTRACT. We suggest that extending Muller games with preference ordering for players is a natural way to reason about unbounded duration games. In this context, we look at the standard solution concept of Nash equilibrium for non-zero sum games. We show that Nash equilibria always exists for such generalised Muller games on finite graphs and present a procedure to compute an equilibrium strategy profile. We also give a procedure to compute a subgame perfect equilibrium when it exists in such games.

1 Introduction

Infinite two player games on graphs have been shown to have various applications in different branches of mathematics and computer science. These are games played on a directed graph where players take turns to move and trace out a path in the graph. The winning condition is given by a set of infinite paths. Such games are well studied in descriptive set theory and topology in the form of Banach-Mazur games. In computer science these are commonly used as models of games in verification and synthesis of open reactive systems. The key question of interest for such games is that of determinacy. That is, whether one of the players always has a winning strategy. It turns out that determinacy depends crucially on the topological properties of the winning set. A celebrated result by Martin [7] showed that all games with Borel winning conditions are determined.

Martin's result however, does not make any assertion as to whether it is possible to determine who the winner is or how "complex" the winning strategy is. These turn out to be the core questions in solving the verification and synthesis problems as well. Winning conditions for games which arise in computer science are typically specified as logical formulas in S1S, first order logic or LTL and are therefore regular conditions. A seminal result due to Büchi and Landweber [1] says that for games played on finite graphs where the wining condition is specified as a Muller set, the winner can be determined and that the winning strategy can be effectively synthesised in finite memory strategies.

A natural generalisation of two player zero sum games is multi-player games where each player has a win-lose objective. Players' objectives are allowed to overlap and therefore these define non-zero sum games. For non-zero sum games, determinacy is usually replaced

© Paul, Simon; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 335–346
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2330

by one of the most important solution concepts in game theory, that of Nash equilibrium: a strategy profile where none of the players have an incentive to deviate unilaterally. It has been shown in [3] that every multiplayer game with Borel winning condition has a Nash equilibrium (see [2] for a detailed exposition). The main idea here is the effective use of threat strategies whereby a player deviating from the equilibrium profile is punished by others to receive the outcome which she can guarantee on her own. For games where the win-lose objectives are regular, an equilibrium profile can be effectively synthesized as well.

For games of infinite duration, it is questionable whether Nash equilibrium defines a satisfactory notion of rational behaviour. A more refined concept is that of subgame perfect equilibrium which takes into account perturbations of players as well. The existence of subgame perfect equilibrium for multiplayer games with win-lose objectives was shown in [9]. [5] unifies both results and shows that the crucial requirement for the existence of equilibrium for such multiplayer games is the determinacy of the underlying two player games.

From a game theoretic perspective, it is quite natural to consider games where players have utilities associated with plays rather than just win-lose conditions. We suggest that in case we restrict our attention to classifying regular plays then this can be captured in terms of a generalised Muller game. These are Muller games where instead of interpreting the Muller table as defining win-lose conditions, we associate utilities over the sets in the Muller table. Such games define non-zero sum objectives for players and we can therefore ask the question whether Nash equilibrium always exists for this class of games. In this context we show the following results:

- Nash equilibrium always exists for generalised Muller games played on finite graphs.
- An equilibrium profile can be effectively synthesized.

One could employ threat strategies to show the existence of equilibrium. However, for infinite games with non-zero sum objectives, even coming up with rationality assumptions which justify the use of such strategies is a challenging task. On the other hand, backward inductive equilibrium profiles are known to be more versatile in the case of finite games. We show that the standard backward induction algorithm [10] can be effectively used to prove the existence of Nash equilibrium and to synthesize an equilibrium profile in generalised Muller games.

Subgame perfect equilibria in general need not exists for such games. However, we show that:

- It is decidable to check whether subgame perfect equilibrium exists in a generalised Muller game.
- It is possible to effectively synthesize a subgame perfect equilibrium profile (when it exists).

2 Preliminaries

We begin with a description of the game arena and the objectives of the players. We look at unbounded duration, turn based games played on finite graphs.

2.1 Game Arena

A game G consists of an arena $\mathcal{A}$ and an objective *Win*. For a directed graph $\mathcal{A} = (V, E)$ and for a node $v \in V$, let $vE = \{v' \mid (v, v') \in E\}$. Let $N = \{1, \ldots, n\}$ be the set of players. A game arena is a finite graph $\mathcal{A} = (V, E)$ where V is the set of game positions and $E \subseteq V \times V$ is the move relation. V is partitioned into sets $V = V_1 \cup \ldots \cup V_n$ where for all $i \in N$, V_i is the set of game positions of player i. For simplicity we assume that for all $v \in V$, the set vE is nonempty. An initialised game is a game G along with a starting vertex $v_0 \in V$. Henceforth when we use the notation (G, v_0), we will generally mean an initialised game with initial vertex v_0.

Given a game (G, v_0), a play in (G, v_0) can be viewed as follows: initially a token is placed at vertex v_0. At any point, if the token is at a vertex $v \in V_i$ (i.e. a player i vertex) then she moves the token to some $v' \in vE$. In this way an infinite path, $\pi = v_0 v_1 \ldots$ where for all $j > 0$ we have $(v_{j-1}, v_j) \in E$, called a play is constructed in the arena.

For a finite sequence $\rho = v_0 v_1 \ldots v_k$ let $\mathit{first}(\rho) = v_0$, $\mathit{last}(\rho) = v_k$ and for an infinite sequence $\pi = v_0 v_1 \ldots$ let $\inf(\pi)$ denote the set of nodes that appear infinitely often in π. For any sequence $\pi = v_0 v_1 \ldots$, let $\pi(i)$ denote the ith element of π, π_i denote the length i prefix of π, $|\pi|_v$ denote the number of v's occuring in π and $Occ(\pi) = \{v \mid |\pi|_v > 0\}$.

2.2 Strategies

A strategy for player i specifies at each game position of i which move to choose. It is a function $\sigma_i : V^* V_i \to V$ from the set of all finite plays (histories) ending in a player i node to the set of game positions which satisfies the condition:

- for all $\pi = v_0 \ldots v_k$, such that $v_k \in V_i$, $\sigma_i(\pi) \in v_k E$.

Let $T_{\mathcal{A}}$ denote the tree unfolding of $\mathcal{A}$. A strategy σ_i can also be thought of as a subtree T_{σ_i} of $T_{\mathcal{A}}$ (called the strategy tree) where for each player i node there is a unique outgoing edge and for any other player node, we include all the edges.

A strategy σ is said to be bounded memory if there exists a finite state machine $\mathcal{M} = (M, g, h, m^I)$ where M is the memory of the strategy, $m^I \in M$ is the initial memory, $g : V \times M \to M$ the memory update function, and $h : V \times M \to V$ is the output function which specifies the choice of the player such that if $v_0 \ldots v_k$ is a play and $m_0 \ldots m_{k+1}$ is a sequence determined by $m_0 = m^I$ and $m_{i+1} = g(v_i, m_i)$ then $\sigma(v_0 \ldots v_k) = h(v_k, m_{k+1})$. The strategy σ is said to be memoryless if M is a singleton.

Let Ω_i denote the set of all strategies for player i. A strategy profile $\bar{\sigma} = (\sigma_1, \ldots, \sigma_n)$ defines a unique play in the game, we use $\pi_{\bar{\sigma}}$ to denote this play. We often use the notation $\bar{\imath}$ to denote the set $N \setminus \{i\}$ and $\bar{\sigma}_{-i}$ to denote the tuple $(\sigma_1, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n)$.

2.3 Objectives

The arena specifies the rules of the game and the moves of the players. To describe a game fully, the objectives of the players have to be specified. The players play in a way that they can 'achieve/avoid' these objectives. The objective of a player is usually a subset ot the set of plays. However, for algorithmic analysis, the objectives need to be finitely presentable. The most widely studied of these presentations are ω-regular objectives, mean-payoff objectives

and so on. These naturally arise in the specifications encountered in the verification and synthesis of reactive systems. In this paper we concentrate on a specific type of ω-regular objective, the Muller objective.

Binary Objectives: In this case the objective of each player i is an omega regular subset *Win* of plays. The game is not antagonistic since objectives of players are allowed to overlap. For instance, for Muller objectives, each player i has a collection $\mathcal{F}_i$ of Muller sets. She wins the game if and only if the game eventually settles down to some subset F of the set of vertices V such that $F \in \mathcal{F}_i$; otherwise she loses. We often call these objectives 'win-lose objectives'.

Generalised Objectives: In this paper we are concerned with games where players have preference orderings on the various Muller sets. Formally, player i has a total order $\sqsubseteq_i$ on the Muller sets. Such an ordering can also be viewed as an utility function $u_i : 2^V \to \mathbb{N}$. Since a strategy profile $\bar{\sigma} \in \Pi_{i=1}^n \Omega_i$ defines a unique Muller set $F = \inf(\pi_{\bar{\sigma}})$, we may also think of u_i to be a function from $u_i : \Pi_{i=1}^n \Omega_i \to \mathbb{N}$. We call such games generalised Muller games.

2.4 Best Response and Equilibrium

The notion of best response and equilibrium is defined as follows:

- A strategy σ_i of player i is said to be a best response for $\bar{\sigma}_{-i}$ if for all $\sigma_i' \in \Omega_i$, $u_i(\pi_{(\bar{\sigma}_{-i},\sigma_i')}) \leq u_i(\pi_{(\bar{\sigma}_{-i},\sigma_i)})$.
- A strategy tuple $\bar{\sigma} = (\sigma_1, \ldots, \sigma_n)$ is a Nash equilibrium if for all $i \in N$, σ_i is the best response for $\bar{\sigma}_{-i}$.
- A subgame perfect equilibrium (SPE) [8] can be defined in our setting as follows. Let ρ be a (finite) path in the arena $\mathcal{A}$. Given a strategy σ_i for player i, the strategy $\sigma_i(\rho)$ is defined to be a function: $\sigma_i(\rho) : \rho V^* V_i \to V$ such that $\sigma_i(\rho)(\rho') = \sigma_i(\rho\rho')$. Let $\bar{\sigma}(\rho)$ denote the tuple $(\sigma_1(\rho), \ldots, \sigma_n(\rho))$. A strategy tuple $\bar{\sigma}$ in the initialised game (G, v_0) is said to be an SPE if for every vertex v in $\mathcal{A}$ and for every path ρ from v_0 to v in $\mathcal{A}$, $\bar{\sigma}(\rho)$ is a Nash equilibrium for the initialised game (G, v).

2.5 Computing Nash Equilibrium

In [3], the authors show that n-player games with win-lose Borel objectives always have a Nash equilibrium. An equilibrium profile is where the players play 'threat' strategies in that, if any player i unilaterally deviates from her prescribed behaviour, all the other players punish her by playing a profile where she can never gain anything more than what she would have had she stuck to her prescribed strategy. The procedure can be appropriately modified to show that Nash equilibrium always exists in a generalised Muller game.

Threat strategies are naturally defined in the case of games with win-lose objectives. However, with general non-zero sum games, it is not clear whether threat strategies constitute 'efficient' solutions profiles. For finite games backward inductive solution profiles are known to preserve nice properties like Pareto efficiency [4]. Here we show that the standard

backward induction procedure can be effectively utilised for computing Nash equilibria in generalised Muller games.

3 Solving Generalised Games

In this section, we develop our procedure for solving generalised Muller games and prove its correctness. The central idea of the procedure is to perform a finite unfolding of the game arena, making use of the 'latest appearance record' (LAR) data structure [6] and apply a backward induction on this unfolding.

3.1 The LAR Tree

Let $\mathcal{A} = (V, E)$ be a finite graph and $\sharp \notin V$. Let $\prec$ be a total order on the nodes of V. We Let

$$L_{\mathcal{A}} = \{l \in (V \cup \{\sharp\})^{|V|+1} \mid |l|_{\sharp} = 1 \wedge \forall v \in V(|l|_v = 1)\}$$

The set $L_{\mathcal{A}}$ is called the LAR memory. Henceforth we shall refer to elements from $L_{\mathcal{A}}$ as $x\sharp y$ where $x, y \in V^*$. We define a function $next : L_{\mathcal{A}} \times V \to L_{\mathcal{A}}$ as

$$next(x\sharp y, v) = \begin{cases} x'\sharp x''yv & \text{iff } x\sharp y = x'vx''\sharp y \\ xy'\sharp y''v & \text{iff } x\sharp y = x\sharp y'vy'' \\ x\sharp y & \text{iff } x\sharp y = x\sharp y'v \end{cases}$$

For a finite play $\rho = v_0v_1 \dots v_k$ in the arena we define LAR(ρ) inductively as:

- LAR(v_0)=$x\sharp v_0$ where x is ordered according to the total order $\prec$.
- LAR($v_0 \dots v_i$)=$next$(LAR($v_0 \dots v_{i-1}$),v_i), $i \geq 1$.

Given an arena $\mathcal{A} = (V, E)$ and an element $x\sharp y \in L_{\mathcal{A}}$ the (finite) LAR tree $T_{fin}(\mathcal{A}, x\sharp y)$ corresponding to $\mathcal{A}$ and $x\sharp y$, or just $T_{fin}(x\sharp y)$ when the arena $\mathcal{A}$ is fixed, is constructed as follows:

- $x\sharp y$ is the root of $T_{fin}(x\sharp y)$.
- For any node $x'\sharp y'v$ of $T_{fin}(x\sharp y)$, and for all $u \in vE$, $x''\sharp y'' = next(x'\sharp y'v, u)$ is a child of $x'\sharp y'v$ iff there is no node $x''\sharp y''$ in the unique path from the root to $x'\sharp y'v$, or $x''\sharp y''$ is the first node to repeat in the path.

That $T_{fin}(x\sharp y)$ is well defined follows from the fact that the function $next$ is well defined. And the fact that $T_{fin}(x\sharp y)$ is finite can be ascertained by noting that along any sequence of the elements of $L_{\mathcal{A}}$ of length $(|V| + 1)! + 1$, at least one element is bound to repeat, by the pigeonhole principle.

3.2 Ensuring a Muller Set

Let $\mathcal{A} = (V, E)$ be an arena, v_0 be an initial vertex and $T_{fin}(x\sharp v_0)$ be the LAR tree of $\mathcal{A}$ corresponding to the LAR $x\sharp v_0$ where x is ordered according to the total order $\prec$. Let $\mathcal{F} \subseteq 2^V$ be a collection Muller sets and $M \subsetneq N, M \neq \emptyset$ be a subset of players. We label the leaf nodes of $T_{fin}(x\sharp v_0)$ with $\mathcal{F}$ or $\bar{\mathcal{F}}$ as follows. For a leaf node $x\sharp y$ of $T_{fin}(x\sharp v_0)$, let ρ be the unique path in $T_{fin}(x\sharp v_0)$ from the root to $x\sharp y$. Let ρ' be the least suffix of ρ such that $first(\rho') = last(\rho') = x\sharp y$ (note that such a suffix always exists by construction of $T_{fin}(x\sharp v_0)$).

Let Let $l_{max} = max\{|y| \mid x\sharp y \in Occ(\rho')\}$ and let $L_\rho = \{x\sharp y \mid |y| = l_{max}\}$. If there exists $x'\sharp y' \in L_\rho$ such that $\{y'\} \in \mathcal{F}$ then we label the leaf $x\sharp y$ with $\mathcal{F}$. Otherwise we label it with $\bar{\mathcal{F}}$.

We now label the entire $T_{fin}(x\sharp v_0)$ with $\mathcal{F}$ or $\bar{\mathcal{F}}$ and construct (memoryless) strategies $\mu_i : L_\mathcal{A} \to L_\mathcal{A}$, $i \in M$ using the following backward induction procedure.

Procedure 1

Suppose all children of node $x\sharp yv$ of $T_{fin}(x\sharp v_0)$ have been labelled. Let $T_{x\sharp yv}$ be the set of children of $x\sharp yv$ and let $T^{\mathcal{F}}_{x\sharp yv} \subseteq T_{x\sharp yv}$ be the nodes among these children that have been labelled with $\mathcal{F}$. Then

- $v \in V_i$ such that $i \in M$:
  - If $T^{\mathcal{F}}_{x\sharp yv} \neq \emptyset$ then let $x'\sharp y'v'$ be such that $v' \prec v''$ for all $x''\sharp y''v'' \in T^{\mathcal{F}}_{x\sharp yv}$. Label $x\sharp yv$ with $\mathcal{F}$ and put $\mu_i(x\sharp yv) = x'\sharp y'v'$.
  - If $T^{\mathcal{F}}_{x\sharp yv} = \emptyset$ then let $x'\sharp y'v' \in T_{x\sharp yv}$ be such that $v' \prec v''$ for all $x''\sharp y''v'' \in T_{x\sharp yv}$. Label $x\sharp yv$ with $\bar{\mathcal{F}}$ and put $\mu_i(x\sharp yv) = x'\sharp y'v'$.
- $v \in V_i$ such that $i \notin M$:
  - If $T^{\mathcal{F}}_{x\sharp yv} = T_{x\sharp yv}$, which means that every child of $x\sharp yv$ is labelled $\mathcal{F}$, then label $x\sharp yv$ with $\mathcal{F}$.
  - If $T^{\mathcal{F}}_{x\sharp yv} \subsetneq T_{x\sharp yv}$ then there exists a child $x'\sharp y'v'$ of $x\sharp yv$ such that $x'\sharp y'v'$ has label $\bar{\mathcal{F}}$. Label $x\sharp yv$ with $\bar{\mathcal{F}}$.

Note that, choosing the least v in the order $\prec$ in the above procedure ensures that the μ_i's constructed are well defined.

Players M are said to be able to ensure the Muller sets $\mathcal{F}$ by strategy μ_i, $i \in M$ on $T_{fin}(x\sharp v_0)$ if the root of $T_{fin}(x\sharp v_0)$ is labelled $\mathcal{F}$ by the above procedure and μ_i are the strategies constructed.

Given a memoryless strategy μ_i for player i on an LAR tree $T_{fin}(x\sharp y)$ we can construct the corresponding bounded memory strategy σ_i for player i on the arena $\mathcal{A}$ as follows:

- The memory M of σ_i is the set $L_\mathcal{A}$ and the initial memory m_I is $x\sharp y$.
- The memory update function $g_i : V \times M \to M$ is defined as $g_i(v, x'\sharp y') = next(x'\sharp y', v)$.
- The output function $h_i : V_i \times M \to V$ is defined as $h(v, x'\sharp y) = \mu_i(x'\sharp y)$.

For a word on notation, we denote memoryless strategies on $T_{fin}(\cdot)$ by μ and we denote the bounded memory strategies on the arena $\mathcal{A}$ by σ.

A strategy σ on the arena $\mathcal{A}$ is said to exist in $T_{fin}(x\sharp y)$ if it corresponds to some strategy μ on $T_{fin}(x\sharp y)$. A strategy μ is said to exist in $T_{fin}(x\sharp y)$ if it is some subtree of $T_{fin}(x\sharp y)$.

LEMMA 1. *If players $M \subsetneq N$, $M \neq \emptyset$ can ensure Muller sets $\mathcal{F}$ in $T_{fin}(x\sharp v_0)$ by strategies μ_i, $i \in M$, then they can ensure $\mathcal{F}$ in (G, v_0) by the bounded memory strategies σ_i corresponding to μ_i.*

PROOF. Suppose not and suppose that players M can ensure $\mathcal{F}$ in $T_{fin}(x\sharp v_0)$ by $\mu_i, i \in M$ but they cannot ensure $\mathcal{F}$ in (G, v_0) by the corresponding strategies σ_i. Then there exists a play π in (G, v_0) conforming to σ_i, $i \in M$ such that it settles down to a Muller set $F' \notin \mathcal{F}$. There are two cases to consider.

The first case is when there exists $v \in F'$ such that $v \notin F$ for any $F \in \mathcal{F}$. Let j be the first index such that $\pi(j) = v$ and $\pi(j-1) \in V_k$, $k \notin M$. Let ρ be the (finite) path in $T_{fin}(x\sharp v_0)$ corresponding to π. j must be greater than $|\rho|$; otherwise ρ couldn't have been labelled $\mathcal{F}$ and hence μ_i's couldn't have ensured $\mathcal{F}$. Let $x'\sharp y'$=LAR(π_{j-1}). By the construction of the LAR tree $T_{fin}(x\sharp v_0)$ there exists a node $x'\sharp y' \in \rho$ itself. But this means that player k had the option of playing v at the node $x'\sharp y'$ forcing $x'\sharp y'$ and hence the root to be labelled $\bar{\mathcal{F}}$. But this would contradict the fact that μ_i's ensure $\mathcal{F}$ in $T_{fin}(x\sharp v_0)$.

The other case is when there exists $v \in F \in \mathcal{F}$ such that $v \notin F'$. Let ρ be the (finite) path in $T_{fin}(x\sharp v_0)$ corresponding to π. Let l be the biggest index l such that $\pi(l) = v$ but $l < |\rho|$. Suppose $\pi(l-1) \in V_i$, $i \in M$. Then for all indices $l_1, l_2, \ldots$ such that $l < l_1 < l_2 < \ldots$ and LAR(π_{l_1})= LAR(π_{l_2})=... = LAR(π_{l-1}), player i has to play v as it is prescribed by the memoryless strategy μ_i, and hence in turn by the corresponding bounded memory strategy σ_i. But this contradicts the fact that the π settles down to F'.

Finally, suppose $\pi(l-1) \in V_k$, $k \notin M$. Then player k has the option of playing v at $\pi(l-1)$ and al all indices $l_1, l_2, \ldots$ such that $l < l_1 < l_2 < \ldots$ and LAR(π_{l_1})= LAR(π_{l_2})=... = LAR(π_{l-1}). Hence μ_i's could not have ensured $\mathcal{F}$ in $T_{fin}(x\sharp v_0)$ as the leaf node of ρ wouldn't have been labelled $\mathcal{F}$ and hence neither the root. ∎

LEMMA 2. *Let $\mathcal{F}$ be a collection of Muller sets. If players $M \subsetneq N$, $M \neq \emptyset$ have strategies σ_i, $i \in M$ to ensure $\mathcal{F}$ in the game (G, v_0), then they have strategies $\mu_i, i \in M$ to ensure $\mathcal{F}$ in $T_{fin}(x\sharp v_0)$.*

PROOF. Suppose players M do not have strategies $\mu_i, i \in M$ to ensure $\mathcal{F}$ in $T_{fin}(x\sharp v_0)$ then $T_{fin}(x\sharp v_0)$ being a finite tree (and hence a finite extensive form game) it follows that players $N \setminus M$ have strategies $\mu_i, i \in N \setminus M$ to ensure $2^V \setminus \mathcal{F}$ in $T_{fin}(x\sharp v_0)$, since finite games are determined. Then by Lemma 1, players $N \setminus M$ have bounded memory strategies $\sigma_i, i \in N \setminus M$ corresponding to the μ_i's to ensure $2^V \setminus \mathcal{F}$ in (G, v_0) as well. But this contradicts the assumption that players M have strategies to ensure $\mathcal{F}$ in (G, v_0). ∎

Combining the above two lemmata, we have the following theorem.

THEOREM 3. *Let (G, v_0) be an n-player game, N being the set of players. Let $M \subsetneq N$, $M \neq \emptyset$ be a subset of players and $\mathcal{F}$ be a collection of a Muller sets consisting of the nodes of the arena of the game. Then players M can ensure $\mathcal{F}$ in (G, v_0) if and only if they can ensure $\mathcal{F}$ in $T_{fin}(x\sharp v_0)$. Also, if players M can ensure $\mathcal{F}$ in $T_{fin}(x\sharp v_0)$ then the bounded memory strategies σ_i, $i \in M$ corresponding to the memoryless strategies μ_i, $i \in M$ computed by Procedure 1, ensures $\mathcal{F}$ in (G, v_0).*

3.3 Equilibrium Computation

Let (G, v_0) be a generalised Muller game with the set of players N and let u_i be the utility function of player i over the Muller sets. We label the leaf nodes of the LAR tree $T_{fin}(x\sharp v_0)$ consistently with tuples $(x_1, \ldots, x_n) \in \mathbb{N}^n$ as follows.

For a leaf node $x\sharp y$ of $T_{fin}(x\sharp v_0)$, let ρ be the unique path in $T_{fin}(x\sharp v_0)$ from the root to $x\sharp y$. Let ρ' be the least suffix of ρ such that $first(\rho') = last(\rho') = x\sharp y$. Let $l_{max} = max\{|y| \mid x\sharp y \in Occ(\rho')\}$ and let $L_\rho = \{x\sharp y \mid |y| = l_{max}\}$. Observe that, by the property

of the LAR construction $y = y'$ for all $x\sharp y,\ x'\sharp y' \in L_\rho$. Let $Y = y'$ such that $x'\sharp y' \in L_\rho$. Label the leaf $x\sharp y$ with $(u_1(Y), \ldots, u_n(Y))$.

We now label the entire tree T_{fin} consistently, with tuples $(x_1, \ldots, x_n) \in \mathbb{N}^n$, and compute a strategy tuple $\bar{\mu} = (\mu_1, \ldots, \mu_n)$ as follows:

Procedure 2

Suppose all children of node $x\sharp yv$ have been labelled and $v \in V_i$. Let
$u_{x\sharp yv} = max\{u_i(x'\sharp y') \mid x'\sharp y' \text{ is a child of } x\sharp yv\}$,
$T_{x\sharp yv} = \{x'\sharp y' \mid x'\sharp y' \text{ is a child of } x\sharp yv \text{ and } u_i(x'\sharp y') = u_{x\sharp yv}\}$.
Put $\mu_i(x\sharp yv) = x'\sharp y'v' \in T_{x\sharp yv}$ such that $v' \prec v''$ for all $x''\sharp y''v'' \in T_{x\sharp yv}$. Label $x\sharp yv$ with $(u_1(x'\sharp y'v'), \ldots, u_n(x'\sharp y'v'))$.

THEOREM 4. *Every generalised Muller game (G, v_0) has a Nash equilibrium.*

The proof shows that the bounded memory strategy tuple $(\sigma_1, \ldots, \sigma_n)$ corresponding to the tuple $(\mu_1, \ldots, \mu_n)$ constructed by Procedure 2, is an equilibrium tuple in the game (G, v_0). The essence of the proof is the same as the one for Theorem 3: player i has an incentive to deviate from σ_i in (G, v_0) if and only if she has an incentive to deviate from μ_i in $T_{fin}(x\sharp yv_0)$. We omit the full proof due to space limitations.

Complexity. Let the number of vertices in the arena $\mathcal{A}$ be m. In Procedure 2, the number of permutations of the m vertices of the arena is equal to m. Thus the size of the LAR memory $L_\mathcal{A}$ may be as big as $(m+1)!$. This means that each path of the LAR tree might be $(m+1)!$ nodes long. As there are $O(m^{m!})$ such paths and the backward induction procedure runs in time linear in the size of the LAR tree, the running time of Procedure 2 is $O(m^{m!})$.

4 Subgame Perfection

Nash equilibrium, as a solution concept, has its limitations. One such limitation is that it does not take into account the sequential nature of the game. In an extensive form game, if a player deviates from equilibrium behaviour even for just one move, Nash equilibrium says nothing about the outcome of the game. One possible refinement to Nash equilibrium is to insist that strategies are optimal after every prefix. This is achieved by subgame perfect equilibrium [8]. Ummels [9] has shown that subgame perfect equilibria always exist for n-player infinite games on graphs for ω-regular win-lose objectives. The question therefore arises whether subgame perfect equilibria exist for n-player infinite games where the objectives are not win-lose but generalised.

For finite extensive form games, the backward induction procedure does indeed yield a subgame perfect equilibrium profile. Since our construction of the equilibrium profile for generalised Muller games employs a backward induction procedure (Procedure 2), it is natural to ask if the profile constructed is subgame perfect. The answer is affirmative for win-lose objectives as we show in the following proposition.

PROPOSITION 5. *Every generalised Muller game with binary objectives has a subgame perfect equilibrium.*

PROOF. We show that the strategy tuple $\bar{\sigma}$ corresponding to the tuple $\bar{\mu}$ constructed by Procedure 2 is a subgame perfect equilibrium of (G, v_0) when the objectives of the players are binary.

Suppose $\bar{\sigma}$ is not an SPE. Then there exists a vertex $v \in V$, $v \in V_i$ say, and a path ρ from v_0 to v such that $\bar{\mu}(\rho)$ is not an equilibrium tuple. Let ρ' be the (finite) path in $T_{fin}(x \sharp v_0)$ corresponding to ρ.

Suppose player j has an incentive to deviate from $\sigma_j(\rho)$. If $|\rho| < |\rho'|$, then player j has an incentive to deviate from μ_j as well. But this contradicts the fact that $\bar{\mu}$ is an equilibrium tuple (Theorem 3).

So assume that $|\rho| \geq |\rho'|$. Then by the property of the LAR tree $T_{fin}(x \sharp v_0)$, there exists ρ'' such that $|\rho''| < |\rho'|$ and LAR(ρ'') = LAR(ρ). Now, since $\bar{\sigma}_j$ corresponds to $\bar{\mu}_j$ which is a memoryless strategy constructed from $T_{fin}(x \sharp v_0)$, it prescribes the same action at ρ and ρ'' (since LAR(ρ'') = LAR(ρ)). Thus if player j has an incentive to deviate from $\sigma_j(\rho)$, she has an incentive to deviate from $\sigma_j(\rho'')$ as well which in turn means she has an incentive to deviate from σ_j in the first place. But this again contradicts fact that $\bar{\sigma}$ is an equilibrium tuple (Theorem 3). ∎

The argument for the above proof breaks down when the objectives of the players are not binary but generalised.

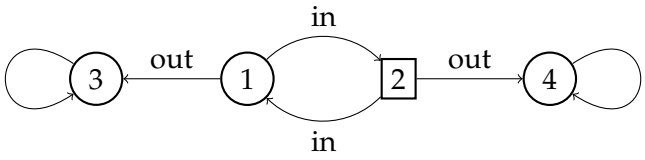


Figure 1: Non existence of subgame perfect equilibrium

Example 1 Consider the game arena shown in Figure 1. Player 1 nodes are denoted by ◯ and player 2 nodes are denoted by □. The game starts at node 1. The utilities of the players for the relevant Muller sets are as follows: $u_1(\{3\}) = 1, u_1(\{1,2\}) = 0, u_1(\{4\}) = 2$ and $u_2(\{3\}) = 0, u_2(\{1,2\}) = 2, u_2(\{4\}) = 1$. Procedure 2 gives the following strategies μ_1 and μ_2 for players 1 and 2 respectively. μ_1 prescribes that player 1 stays 'in' in her first move expecting player 2 to go 'out' and hence give 1 a better payoff. But if she plays 'in' then player 2 stays 'in' as prescribed by μ_2 because that gives her a better payoff. To this 1 assumes that player 2 will stay in forever and hence plays 'out' in her next move as prescribed by μ_1. The profile (μ_1, μ_2) is thus not subgame perfect. One can verify that the above game does not have a subgame perfect equilibrium.

The above example shows that in general subgame perfect equilibria need not exist for generalised Muller games. Thus an obvious question to ask would be: Is it decidable to check whether subgame perfect equilibrium exists in a given generalised Muller game? In this section, we develop a procedure to decide the existence of sub-game perfect equilibrium and to compute the equilibrium profile when it exists.

First, it is important to note that given an arena $\mathcal{A}$ and an initial vertex v, for any bounded memory strategy on $\mathcal{A}$ that uses memory $L_{\mathcal{A}}$ the initial element of $L_{\mathcal{A}}$ does not matter. In other words, no matter what element $x\sharp y$ such that $last(x\sharp y) = v$ of $L_{\mathcal{A}}$ we take as the root of the LAR tree, the backward induction procedure (Procedure 2) gives all the bounded memory strategies that are possible by using memory $L_{\mathcal{A}}$ and updating it as described in Section 3. This is because starting from any vertex v of $\mathcal{A}$, the tree explores all possible cycles reachable from v and a path of the LAR tree is terminated if and only if a cycle is completed.

Define the following property for a strategy tuple $\bar{\mu}$ on $T_{fin}(x\sharp v_0)$

Property 1 *For every $x\sharp y \in \mathcal{T}_{fin}(x\sharp v_0)$, there exists a strategy tuple $\bar{\mu}'$ on $\mathcal{T}_{fin}(x\sharp y)$ such that $\bar{\mu}'$ is derived by backward induction on $\mathcal{T}_{fin}(x\sharp y)$ and $\bar{\mu}'(x\sharp y) = \bar{\mu}(x\sharp y)$.*

Given an game (G, v_0), let $Path(G, v_0)$ be the set of all finite paths starting at v_0 in G. Define $P : L \to 2^{Path(G,v_0)}$ such that $P(x\sharp y) = \{\rho \in Path(G, v_0) \mid \mathrm{LAR}(\rho) = x\sharp y\}$.

Given a strategy tuple $\bar{\sigma}$ on (G, v_0) define $Q_{\bar{\sigma}} : L \to 2^{2^V}$ as $Q_{\bar{\sigma}}(x\sharp y) = \{\inf(\pi_{\bar{\sigma}(\rho)}) \mid \rho \in P(x\sharp y)\}$ where $\pi_{\bar{\sigma}(\rho)}$ is the play conforming to $\bar{\sigma}(\rho)$. Let $C_{\bar{\sigma}}$ be a choice function $C_{\bar{\sigma}} : L \to 2^V$ such that

Property 2 *$x'\sharp y'$ is a child of $x\sharp y$ in $T_{fin}(x\sharp v_0)$ and $C_{\bar{\sigma}}(x\sharp y) \in Q_{\bar{\sigma}}(x'\sharp y')$ implies $C_{\bar{\sigma}}(x'\sharp y') = C_{\bar{\sigma}}(x\sharp y)$.*

It follows that

Property 3 *If $C_{\bar{\sigma}}(x\sharp yv) = F$, $v \in V_i$ then there actually exists a $\rho \in Path(G, v_0)$ such that LAR(ρ)$= x\sharp yv$, $\inf(\pi_{\bar{\sigma}(\rho)}) = F$, $\sigma_i(\rho) = v$ and $\inf(\pi_{\bar{\sigma}(\rho v)}) = F$.*

Assume for the moment that given any strategy tuple $\bar{\sigma}$, we have such a function $C_{\bar{\sigma}}$ satisfying Property 2. Now let $\bar{\sigma}$ be an SPE on (G, v_0). For every $i \in N$, construct σ_i' as follows: $\sigma_i' : V^*V_i \to V$ such that $\sigma_i'(uv) = uvv'$ iff $C_{\bar{\sigma}}(\mathrm{LAR}(uv)) = C_{\bar{\sigma}}(\mathrm{LAR}(uvv'))$, $(v, v') \in E$

LEMMA 6. *$\bar{\sigma'}$ is an SPE on (G, v_0)*

PROOF. Suppose not. Then there exists $\rho \in Path(G, v_0)$ such that $\bar{\sigma'}(\rho)$ is not an NE. So suppose player i has an incentive to deviate from $\sigma_i'(\rho)$. Now by property 3 there exists a history $\rho' \in Path(G, v_0)$ such that $\sigma_i'(\rho) = \sigma_i(\rho')$. Then player i must have an incentive to deviate from $\sigma_i(\rho')$ itself. But this contradicts the subgame perfection of $\bar{\sigma}$. ∎

Now $\bar{\sigma'}$ exists on $T_{fin}(x\sharp v_0)$. Indeed, it is the strategy where $\sigma_i'(x\sharp y) = x'\sharp y'$ such that $x\sharp y$ is a parent of $x'\sharp y'$ in $T_{fin}(x\sharp v_0)$ and $C_{\bar{\sigma}}(x\sharp y) = C_{\bar{\sigma}}(x'\sharp y')$. Let $\bar{\mu'}$ denote this memoryless strategy tuple on $T_{fin}(x\sharp v_0)$ corresponding to $\bar{\sigma'}$.

LEMMA 7. *$\bar{\mu'}$ has Property 1*

PROOF. Suppose not. Then there exists a node $x\sharp y \in T_{fin}(x\sharp v_0)$ such that for any backward induction strategy profile $\bar{\mu}^+$ on $T_{fin}(x\sharp y)$, $\bar{\mu'}(x\sharp y) \neq \bar{\mu}^+(x\sharp y)$. Now we have that $\bar{\sigma'}$ is subgame perfect on (G, v_0) and bounded memory, the memory being $L_{\mathcal{A}}$. So $\bar{\mu'}(x\sharp y)$ must correspond to some equilibrium tuple $\bar{\sigma'}$ on $(G, last(x\sharp y))$ which exists in $T_{fin}(x\sharp y)$, as backward induction on $T_{fin}(x\sharp y)$ gives all the bounded memory equilibria starting at node $last(x\sharp y)$ with memory $L_{\mathcal{A}}$. But then the above cannot happen. ∎

LEMMA 8. *If a strategy tuple $\bar{\mu}$ on $\mathcal{T}_{fin}(x\sharp v_0)$ satisfies Property 1 then $\bar{\sigma}$ on (G, v_0) corresponding to $\bar{\mu}$ is an SPE.*

PROOF. Suppose not. Then there exists $\rho \in Path(G, v_0)$ such that $\bar{\sigma}(\rho)$ is not an equilibrium on $(G, last(\rho))$. Let LAR(ρ)=$x\sharp y$. Now $x\sharp y \in T_{fin}(x\sharp v_0)$ and $\bar{\sigma}(\rho)$ is bounded memory, the memory being $L_{\mathcal{A}}$. Thus $\bar{\mu}(\rho)$ corresponding to $\bar{\sigma}(\rho)$ cannot be a backward induction profile on $T_{fin}(x\sharp y)$ as backward induction on $T_{fin}(x\sharp y)$ gives all the bounded memory equilibria starting at node $last(x\sharp y)$ with memory $L_{\mathcal{A}}$. So $\bar{\mu}$ cannot satisfy Property 1. ∎

From the above set of lemmata we have the following theorem.

THEOREM 9. *A generalised Muller game (G, v_0) has a subgame perfect equilibrium if and only if there exists a strategy profile $\bar{\mu}$ on $T_{fin}(x\sharp v_0)$ that satisfies Property 1.*

PROOF. It only remains to construct the choice function $C_{\bar{\sigma}}$ satisfying Property 2 given a strategy profile $\bar{\sigma}$. We do that as follows: let $\lessdot$ be a breadth-first ordering on $T_{fin}(x\sharp v_0)$ and let $H = \emptyset$.
Till $H \neq T_{fin}(x\sharp v_0)$ do

- Let $x\sharp y$ be the minimum in the ordering $(T_{fin}(x\sharp v_0) \setminus H) \restriction \lessdot$.
- Let ρ be the path from the root to $x\sharp y$.
- Let $C_{\bar{\sigma}}(x\sharp y) = \inf(\pi_{\bar{\sigma}(\rho)}) = F$.
- There exists a path ρ' from $x\sharp y$ to a leaf node of $T_{fin}(x\sharp v_0)$ such that for all $x'\sharp y' \in \rho'$, $F \in Q_{\bar{\sigma}}(\text{LAR}(x'\sharp y'))$. Put $C_{\bar{\sigma}}(x'\sharp y') = F$ for all such $x'\sharp y' \in \rho'$. Let $H = H \cup \{x'\sharp y' \mid x'\sharp y' \in \rho'\}$.
- For all $x'\sharp y' \in T_{fin}(x\sharp v_0)$ such that $x'\sharp y' \notin \rho'$ and such that LAR($x'\sharp y'$) = LAR($x''\sharp y''$) for some $x''\sharp y'' \in \rho'$, put $C_{\bar{\sigma}}(x'\sharp y') = C_{\bar{\sigma}}(x''\sharp y'')$. Let $H = H \cup \{x'\sharp y'\}$.

It is immediate that the $C_{\bar{\sigma}}$ constructed above meets Property 2. ∎

The above theorem immediately gives us the following procedure to test if a generalised Muller game has a subgame perfect equilibrium.

Procedure:
For every backward induction strategy profile $\bar{\mu}$ on $T_{fin}(x\sharp v_0)$
  For all $x\sharp y \in T_{fin}(x\sharp v_0)$ such that $x\sharp y \neq x\sharp v_0$
    If $\bar{\mu}(x\sharp y) \neq \bar{\mu}^+(x\sharp y)$ for some backward induction strategy profile $\bar{\mu}^+$
    on $T_{fin}(x\sharp y)$, then return TRUE and exit
Return FALSE

Complexity: Let $|V| = m$. There are at most $1 + m + m^2 + \ldots + m^{m!} = (m^{m!+1} - 1)/(m - 1)$ nodes in an LAR tree. There are atmost $m \cdot m^2 \cdot \ldots m^{m!} = m^{m!(m!+1)/2}$ strategy tuples in an LAR tree. Hence the complexity of the above procedure is $\mathcal{O}((m^{m!+1} - 1)/(m - 1) \cdot m^{m!(m!+1)/2} \cdot (m^{m!+1} - 1)/(m - 1)) = \mathcal{O}(m^{2m!} \cdot m^{(m!)^2})$.

5 Discussion

Nash equilibrium and subgame perfect equilibrium are well studied in finite games. In the setting of finite games, subgame perfection is justified under the trembling hand assump-

tion and a subgame perfect profile is considered more robust than general Nash equilibrium profiles. When we move to nonzero sum games of infinite duration even coming up with an appropriate notion of rationality which justifies the trembling hand assumption is a challenging task. However, the equilibrium notions are mathematically well defined and deserves attention in their own right. In this paper rather than delve into issues concerning rationality, we have attempted to investigate equilibrium notions in the context of infinite games. We have shown that the standard technique of backward induction can be appropriately modified to compute equilibrium profile in generalised Muller games. Though the running time complexity of the procedures is not very encouraging, we would like to view this as a generic technique for solving games.

6 Acknowledgements

We are indebted to R. Ramanujam for suggesting this problem and for his constant support and encouragement. We are grateful to Dietmar Berwanger for the various discussions and for sharing his insights into rationality concerns associated with subgame perfect equilibria in infinite games. We thank the anonymous reviewers for their valuable comments and in particular for pointing out Example 1.

References

[1] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[2] K. Chatterjee. *Stochastic ω-Regular Games*. PhD thesis, University of California at Berkeley, 2007.

[3] K. Chatterjee, M. Jurdzinski, and R. Majumdar. On Nash equilibria in stochastic games. In *Proceedings of the 13th Annual Conference of the European Association for Computer Science Logic*, volume 3210 of *LNCS*, pages 26–40. Springer-Verlag, 2004.

[4] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 2005.

[5] E. Grädel and M. Ummels. Solution concepts and algorithms for infinite multiplayer games. In *New Perspectives on Games and Interaction*, volume 4 of *Texts in Logic and Games*, pages 151–178. Amsterdam University Press, 2008.

[6] Y. Gurevich and L. Harrington. Trees, automata and games. In *Proceedings of the 14th Annual Symposium on Theory of Computing*, pages 60–65. ACM Press, 1982.

[7] D. A. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–371, 1975.

[8] R. Selten. Spieltheoretische behandlung eines oligopolmodells mit nachfrageträgheit. *Zeischrift Für Die Gesamte Staatswissenschaft*, 121:301–324, 1965.

[9] M. Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In *Proceedings of the Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4337 of *LNCS*, pages 212–223. Springer, 2006.

[10] E. Zermelo. Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels,. In *Proceedings of the Fifth Congress Mathematicians*, pages 501–504. Cambridge University Press, 1913.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Modelchecking counting properties of 1-safe nets with buffers in paraPSPACE

M. Praveen and Kamal Lodaya
The Institute of Mathematical Sciences, Chennai, India

ABSTRACT. We consider concurrent systems that can be modelled as 1-safe Petri nets communicating through a fixed set of buffers (modelled as unbounded places). We identify a parameter K, which we call "benefit depth", formed from the communication graph between the buffers. We show that for our system model, the coverability and boundedness problems can be solved in polynomial space assuming K to be a fixed parameter, that is, the space requirement is $f(K)p(n)$, where f is an exponential function and p is a polynomial in the size of the input. We then obtain similar complexity bounds for modelchecking a logic based on such counting properties. This means that systems that have sparse communication patterns can be analyzed more efficiently than using previously known algorithms for general Petri nets.

1 Introduction

Many theoretical models exist for concurrent, infinite-state systems. Petri nets [19], process rewrite systems [4], lossy channel systems (LCS) [5] and networks of pushdown systems [1] are some of them. The power to express properties of the original system in sufficient detail and existence of efficient algorithms for analysis are often conflicting goals in these models. Reachability in LCS is non-primitive recursive [22] and reachability for Petri nets is decidable but with no known upper bound [18, 15].

More structure is sometimes imposed on the models to handle these conflicting goals. Communicating automata with buffers [3] is one such model. In this paper we consider a small generalization where 1-safe Petri nets (which we call components) communicate via buffers. Thus we have a system model which allows both asynchronous and synchronous communication, since 1-safe Petri nets can model the latter.

The diagram shown in Fig. 1 illustrates the kind of systems we are interested in. The boxes labelled as line 1, line 2 etc. can be thought of as assembly lines represented by 1-safe Petri nets, drawing raw materials from buffers ib_1, ib_2 etc. Output of these assembly lines are deposited into buffers ob_1, ob_2 etc. Boxes labelled master line 1 and master line 2 can be thought of as master assembly lines that use output of earlier assembly lines as their input. They deposit their output in buffers pr_1 and pr_2 respectively. We are concerned with verifying properties like $\exists c : pr_1 \leq c$ *in all reachable configurations* (boundedness) or $ob_1 + ob_2 \geq 100$ *in some reachable configuration* (coverability). For instance, the latter property might show that the two buffers are dealing with enough throughput. Karp and Miller examined these properties in the context of Petri nets [14] and Lipton and Rackoff showed them to be EXPSPACE-complete [17, 20].

As Esparza notes in his survey article [10], verification of a "logic" based on such properties, for instance LTL or CTL extended with counting properties, quickly becomes undecidable. Modalities of the form $\mathbf{EF}(M \geq M_c)$ (where M, M_c are markings) can be handled

© Praveen, Lodaya; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 347–358
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2331

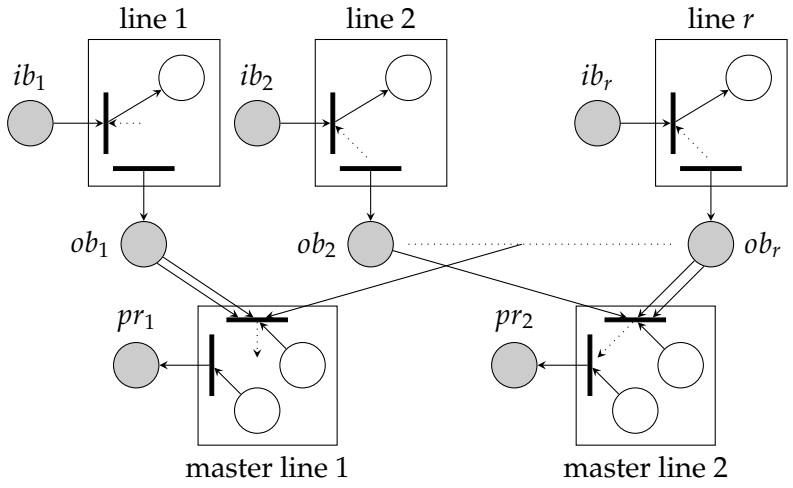


Figure 1: Illustration of communicating automata with buffers

without getting into undecidability [24]. However, a "usual" definition of a logic based on these modalities can express reachability, as in Howell, Rosier and Yen's logic [13] and in Yen's logic [24] (as was recently shown by Atig and Habermehl [2]). So we are left with positive Boolean combinations of formulae of the form $\mathbf{EF}(M \geq M_c)$ [24] for which modelchecking is EXPSPACE-hard. Rosier and Yen analyzed boundedness [21] using what we today call parameterized complexity [9] to show that the space requirement is exponential in the number of unbounded places and polynomial in the number of bounded places. If we give up counting properties, Habermehl shows that the full linear time μ-calculus can be reduced to the problem of repeated control state reachability [12] and is PSPACE-complete in the size of the formula and EXPSPACE-complete in the size of the model.

An EXPSPACE lower bound in the size of the model is not very encouraging for potential verifiers. Our first contribution is the identification of a parameter K, which we call benefit depth. A buffer p_1 can benefit by another buffer p_2 if there is a sequence of transitions that decrease tokens in p_2 and increase tokens in p_1. Benefit depth is the maximum number of buffers benefited by any one buffer. It seems reasonable that, in a sparsely communicating system, benefit depth can be low.

We show that boundedness and coverability in our models, when parameterized by benefit depth, are solvable in paraPSPACE [11]. That is, the space requirement is of the form $\mathcal{O}(f(K)p(n))$, where f is an exponential function of benefit depth and p is some polynomial of the size of the model and the marking to be covered. For constant benefit depth, boundedness and coverability can be solved in PSPACE. Thus, our results are refinements of Rosier and Yen's [21], improving them if benefit depth is less than the number of buffers (as happens in sparsely communicating systems).

As our final contribution, we define a logic which can express counting properties such as coverability *and* show that it can be modelchecked on Petri nets in paraPSPACE.

The full version of this paper may be consulted at http://www.imsc.res.in/%7Epraveen/ for detailed proofs. This conference version attempts a more intuitive treatment without

compromising precision.

2 Problem definitions

Let $\mathbb{Z}$ be the set of integers and $\mathbb{N}$ the set of natural numbers. A Petri net is a 4-tuple $N = (P, T, Pre, Post)$ where P is a set of places, T is a set of transitions and Pre and $Post$ are the incidence functions: $Pre : P \times T \to [0 \dots W]$ (arcs going from places to transitions), $Post : P \times T \to [0 \dots W]$ (arcs going from transitions to places), where $W \geq 1$.

DEFINITION 1. *Given a place p, the set of places $Ben(p) \subseteq P$ and the set of transitions $T_{ben}(p) \subseteq T$ benefited by p are those connected to p by a sequence of arcs with weight ≥ 1. Formally they are the smallest sets satisfying:*

1. *$p \in Ben(p)$.*
2. *If some $p' \in Ben(p)$ and there is a transition t with $Pre(p', t) \geq 1$, then $t \in T_{ben}(p)$.*
3. *If some transition $t \in T_{ben}(p)$ and there is a place p'' such that $Post(p'', t) \geq 1$, then $p'' \in Ben(p)$.*

$Ind(p) = P \setminus Ben(p)$ and $T_{ind}(p) = T \setminus T_{ben}(p)$ are the places and transitions not benefiting from p.

We call a function $M : P \to \mathbb{Z}$ a vector. For two vectors M_1 and M_2, we say M_1 covers M_2 (written $M_1 \geq M_2$) if for every place p, $M_1(p) \geq M_2(p)$. $M_1 > M_2$ means that M_1 covers M_2 but they are not the same.

If the range of the vector is $\mathbb{N}$, it is called a marking. At a marking M, a place p is said to have $M(p)$ tokens. A pair (N, M_0) consisting of a Petri net N and an initial marking M_0 is called a **system**. We assume a net is presented as two matrices for Pre and $Post$. In the rest of this paper, we will assume that a Petri net N has m places, n transitions and that W is the maximum of the range of Pre and $Post$. We define the size of the net to be $2mn \log W$ bits. The system has size $2mn \log W + \log |M_0|$ bits.

A transition t may be taken as a step at the vector M yielding a new vector M' given by the equation $M'(p) = M(p) - Pre(p, t) + Post(p, t)$ for all $p \in P$. The transition t is said to be fired at M if, in addition, t is enabled at M, that is, for all $p \in P$, $M(p) \geq Pre(p, t)$. Thus firing a transition leads from a marking to another marking, while stepping is a more general notion leading from a vector to a vector.

A finite transition sequence $\sigma = t_1 t_2 \dots t_r$ is a walk from an initial vector M_0 to a vector M_r if there exist intermediate vectors $M_1, M_2, \dots, M_r$ such that for all i with $1 \leq i \leq r$, we have a step from M_{i-1} to M_i using the transition t_i. We write $M_0 \xrightarrow{\sigma} M_r$. σ is a firing sequence enabled at some initial marking M_0 if the transitions are enabled at the intermediate vectors, so that $M_1, M_2, \dots, M_r$ are all markings. We write $M_0 \stackrel{\sigma}{\Longrightarrow} M_r$ and say that the marking M_r is reachable from M_0. $\mathcal{R}(N, M_0)$ is the set of markings reachable from M_0. A place is said to be c-bounded, $c \in \mathbb{N}$, in the system (N, M_0), if for all its reachable markings M, $M(p)$ is in $\{0, \dots, c\}$. The system is c-bounded if all its places are. A 1-bounded system is commonly called a 1-Safe net.

DEFINITION 2. *[Reachability, coverability, boundedness] Given a system (N, M_0) and a marking M as input data, the reachability problem is to decide if the marking M is in $\mathcal{R}(N, M_0)$; the coverability problem is to decide if there is an M' in $\mathcal{R}(N, M_0)$ such that M' covers M.*

Given a system (N, M_0), the boundedness problem is to decide if there is some $c \in \mathbb{N}$ such that the system is c-bounded.

Given a c-bounded system, the reachability and coverability problems are known to be PSPACE-complete [6]. For systems in general, which can be unbounded, Lipton showed that all three problems are EXPSPACE-hard [17]. Rackoff showed that boundedness and coverability are in EXPSPACE[20]. Reachability has been shown to be decidable [18, 15], obtaining an upper bound is a famous open problem.

2.1 A logic of properties

Inspired by Yen [24], we now formulate a logic of properties such that its model checking can be reduced to coverability (κ) and boundedness (β) problems, but is designed to avoid expressing reachability. In particular, a κ formula of the form $\tau \leq c,\ c \in \mathbb{N}$, is *not* provided and the κ and ϕ formulas are *not* closed under negation.

$$\begin{aligned}
\tau &::= p,\ p \in P \mid \tau_1 + \tau_2 \mid c\tau,\ c \in \mathbb{N} \\
\kappa &::= \tau \geq c,\ c \in \mathbb{N} \mid \kappa_1 \wedge \kappa_2 \mid \kappa_1 \vee \kappa_2 \mid \mathbf{EF}\kappa \\
\beta &::= \{\tau_1, \ldots, \tau_r\} < \omega \mid \neg\beta \mid \beta_1 \vee \beta_2 \\
\phi &::= \beta \mid \kappa \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2
\end{aligned}$$

The satisfaction of a formula ϕ by a system (N, M_0) (denoted as $N, M_0 \models \phi$) is defined below. The boolean operators work as usual. Note that every term (of type τ) gives a function $L_\tau : P \to \mathbb{N}$ such that τ is syntactically equivalent to $\sum_{p \in P} L_\tau(p)p$.

- $N, M_0 \models \tau \geq c$ if $\sum_{p \in P} L_\tau(p) M_0(p) \geq c$.
- $N, M_0 \models \mathbf{EF}\kappa$ if $\exists M \in \mathcal{R}(N, M_0)$ such that $N, M \models \kappa$.
- $N, M_0 \models \{\tau_1, \ldots \tau_r\} < \omega$ if $\exists c \in \mathbb{N} : \forall M \in \mathcal{R}(N, M_0)\ \exists j \in \{1, \ldots, r\}$ such that $\sum_{p \in P} L_{\tau_j}(p) M(p) \leq c$.

We use $\{\tau_1, \ldots, \tau_r\} = \omega$ as an abbreviation for $\neg(\{\tau_1, \ldots, \tau_r\} < \omega)$.

The formula $\{p_1, \ldots, p_r\} < \omega$ says that the given set of places is bounded according to Valk and Vidal-Naquet [23, Section 4.1]. On the other hand, $\{p_1 + \cdots + p_r\} < \omega$ says that the same set of places is uniformly bounded according to the same authors [23].*

2.2 System model

Though our results work for any Petri net, we work with the model defined below to emphasize the fact that our problem formulation strictly generalizes reachability for 1-bounded systems. The model of concurrent systems we consider in this paper consists of some 1-safe nets, called components, which can add or remove tokens to/from a set of unbounded places that we refer to as buffers.

*We thank an anonymous FSTTCS referee for pointing out this subtlety. Following their suggestion, we have slightly extended our logic beyond the submitted version to cover both kinds of boundedness.

DEFINITION 3. *A **net communicating with buffers** (we just use the word "net" below) is a Petri net $N = (C, B, T, Pre, Post)$ where the set of places $P = C \cup B$ is partitioned into a set of buffers B and component places $C = P \setminus B$, such that all places in C remain 1-bounded (regardless of the number of tokens in the buffers in an initial marking).*

In the rest of the paper, we will assume that $|C| = a$, $|B| = b$ and that $a + b = m$, where m is the total number of places. In our model, the components do not contribute to exponential space complexity. Our results can be generalized to the case where the components are declared to be c-bounded (for a constant c) rather than 1-bounded.

DEFINITION 4. *The **benefit depth** of a net is defined as $K = max\{|Ben(p) \cap B| - 1 \mid p \in B\}$.*

Benefit depth depends only on the communication pattern among buffers, even though the communication link may involve some component places. It can be computed efficiently (in NLOGSPACE).

The communication graph of the system of Fig. 1 is shown in Fig. 2. Irrespective of the number of assembly lines, benefit depth is 3 since only ob_i, pr_1 and pr_2 can benefit by decreasing tokens from ib_i. If there are interdependencies among the assembly lines, such

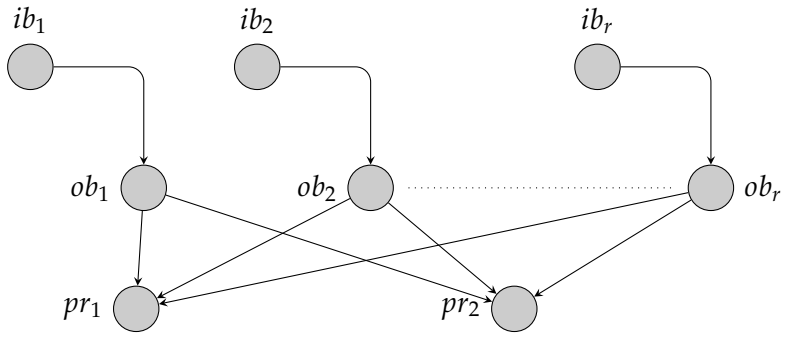


Figure 2: Communication graph of buffers of the system in Fig. 1

as a byproduct of one being the raw material of another (not shown in the figure), then benefit depth will increase. The more such dependencies (i.e., more dense the communication graph among the buffers is), the higher will be the benefit depth. Intuitively, the number of tokens in a place in $Ben(p)$ can be increased by decreasing some tokens in p through a sequence of transitions in $T_{ben}(p)$. Only those transitions use the extra tokens from p.

Our earlier definitions are modified to be well-behaved on the components. A vector will now be given by a pair of functions $C \to \{0, 1\}$ and $B \to \mathbb{Z}$; it is a marking if the second function has range $\mathbb{N}$. Walks and firing sequences will now be defined with these kinds of intermediate vectors and markings.

3 Benefit depth and coverability

Let $Q \subseteq P$ be a subset of places. For this paper we will need the inbetween notion (due to Rackoff) of σ being a Q-**run** where for the vectors $M_i, 0 \leq i < r$, $M_i(p) \geq Pre(p, t_{i+1})$ for every place p in Q. Thus a walk is a $\emptyset$-run and a firing sequence is a P-run. For two vectors

M_1 and M_2, we say $M_1 \geq_Q M_2$ if for every $p \in Q, M_1(p) \geq M_2(p)$ and $M_1(p) = M_2(p)$ for every $p \in C$. A walk σ from M_1 is said to Q-**cover** a marking M_{cov} if it is a Q-run and the final vector M_2 obtained by walking σ at M_1 satisfies $M_2 \geq_Q M_{cov}$. We say σ **covers** a marking if σ P-covers it.

We will fix for this section M_{cov} as the marking to be covered. For the purpose of complexity analysis, we will denote the maximum of the range of M_{cov} by R.

DEFINITION 5. *A Q-**covering** run is a Q-run that Q-covers M_{cov}. Let $Q_0 \subseteq Q$. A Q-run from M_0 to M_r is said to be c-**bounded for** Q_0, $c \in \mathbb{N}$, if for all intermediate vectors $M_i, 0 \leq i < r$, $M_i(p)$ is in $\{0, \ldots, c\}$ for every place p in Q_0.*

DEFINITION 6. *[20, Rackoff] Let $C \subseteq Q \subseteq P$. Define $lencov(Q, M, M_{cov})$ to be the length of the shortest Q-covering run from the vector M. If there is no such sequence, define $lencov(Q, M, M_{cov})$ to be 0. For $0 \leq i \leq b$, $\ell(i, M_{cov})$ is defined to be $max\{lencov(Q, M, M_{cov}) \mid M$ a vector, $C \subseteq Q \subseteq P$ and $|Q \setminus C| = i\}$. In this section we abbreviate $\ell(i, M_{cov})$ to $\ell(i)$. In section 5 we will abbreviate $\ell(b, M)$ to $\ell'(M)$.*

DEFINITION 7. *Let $C \subseteq Q \subseteq P$ and $p \in B$ be a buffer. Define $covind^p(Q, M, M_{cov})$ to be the length of the shortest Q-covering run in $T_{ben}(p)^*$ from the vector M. If there is no such sequence, define $covind^p(Q, M, M_{cov})$ to be 0. Let $\ell_1(i) = max\{covind^p(Q, M, M_{cov}) \mid M$ a vector, p a buffer, $|Q \cap Ben(p) \cap B| = i\}$.*

Our strategy is to segregate covering sequences into two parts, the first made of transitions in $T_{ind}(p)$ and the second one made of transitions in $T_{ben}(p)$. We need the following technical lemma, which is a generalization of the exchange lemma [7, Lemma 2.14] to Petri nets with weighted arcs.

LEMMA 8. *Let p be a place, transitions $t_{ben} \in T_{ben}(p)$ and $t_{ind} \in T_{ind}(p)$. Let $Q \subseteq P$ be some subset of places. If $t_{ben}t_{ind}$ is a Q-run from some vector M, then so is $t_{ind}t_{ben}$.*

LEMMA 9. *If $K \leq i < b$, then $\ell(i+1) \leq (W\ell_1(K) + R)^{i+1}2^a + \ell(i) + \ell_1(K)$.*

PROOF. Suppose σ is a Q_{i+1}-covering run from some vector M, with $Q_{i+1} \subseteq P$ and $|Q_{i+1} \cap B| = i + 1$. If some buffer $p \in Q_{i+1}$ has more than $W\ell_1(K) + R$ tokens at some intermediate marking M', rest of the sequence can be replaced by a Q_i-covering run σ_2' of length at most $\ell(i)$, where $Q_i = Q_{i+1} \setminus \{p\}$. Now, apply Lemma 8 repeatedly to rearrange σ_2' into $\tau_1\tau_2$, with $\tau_1 \in T_{ind}(p)^*$ and $\tau_2 \in T_{ben}(p)^*$ (see Fig. 3). Since τ_2 is a covering sequence made of transitions in $T_{ben}(p)$, it can be replaced by another one of length at most $\ell_1(K)$. ∎

The bound on $\ell(i+1)$ given by Rackoff in [20] is similar to the one in Lemma 9 but uses $\ell(i)$ in place of $\ell_1(K)$. Since $\ell_1(K)$ can be much smaller than $\ell(i)$, the bound in Lemma 9 is better. This is the fact that enables us to restrict exponential space complexity to K. The following lemma gives a recurrence relation for length of covering sequences made of transitions in $T_{ben}(p)$.

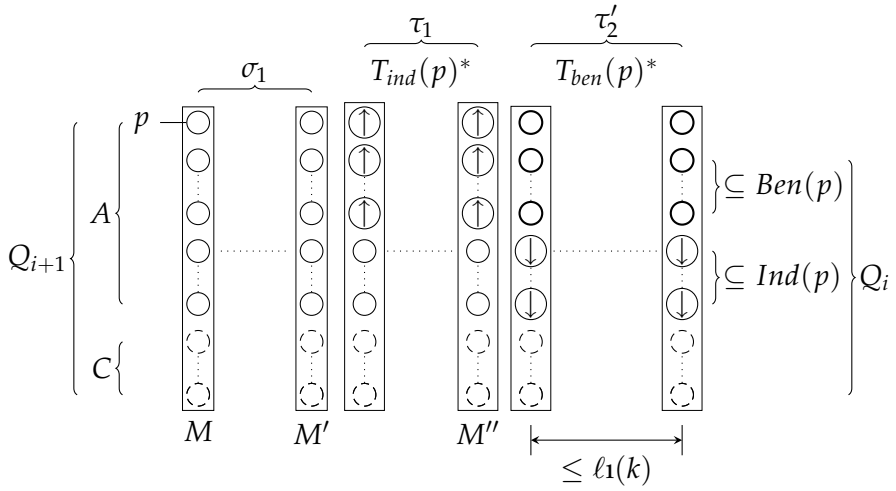


Figure 3: Sequences and bounds used in the proof of Lemma 9
↑ (resp. ↓) inside places indicates that tokens are non-decreasing (resp. non-increasing).

LEMMA 10. *$\ell_1(0) \leq 2^a$ and $\ell_1(i+1) \leq (W\ell_1(i)+R)^{i+1}2^a + \ell_1(i)$.*

PROOF. (Following [20].) For any $Q \subseteq P$, buffer p and Q-run $\sigma \in T_{ben}(p)^*$, if two intermediate vectors of the run are equal when restricted to $Q \cap Ben(p)$, the subsequence between these two vectors can be removed and the remaining sequence will still be a Q-run and retains the covering properties of σ. This is because the removed subsequence doesn't affect places in $Q \cap Ben(p)$ and doesn't increase tokens in any place in $Q \cap Ind(p)$.

The bound on $\ell_1(0)$ is due to the above observation and the fact that component places are 1-bounded and there are 2^a possible distinct vectors when restricted to C. For $\ell_1(i+1)$, suppose there is a Q_{i+1}-covering run $\sigma \in T_{ben}(p)^*$ for some buffer p with $|Q_{i+1} \cap B \cap Ben(p)| = i+1$. If some buffer $p' \in Ben(p) \cap Q_{i+1}$ has more than $W\ell_1(i)+R$ tokens at some intermediate vector M, we can apply the same kind of reasoning used in Lemma 9. ∎

It now only remains to solve the recurrence relations we have obtained and use them in a nondeterministic algorithm that guesses covering sequences to get our first main theorem.

DEFINITION 11. *Let $W' = \max\{W,2\}$, $R' = \max\{R,2\}$. Define a growth function $g : \mathbb{N} \to \mathbb{N}$ as $g(0) = W'R'2^a$ and $g(i+1) = (g(i))^{3(i+1)}2^a$.*

LEMMA 12. *$\ell(K+j) \leq (K+j)(W\ell_1(K)+R)^{K+j}2^a + j\ell_1(K) + \ell(K)$.*

LEMMA 13. *$\ell_1(i), \ell(i) \leq g(i) \leq (W'R')^{3^i i!}2^{6^i i! a}$ and $\ell(K+j) \leq (K+j)(g(K))^{3(K+j)}2^a$.*

THEOREM 14. *Suppose a net under consideration has benefit depth K. There is a nondeterministic algorithm that decides if there is a firing sequence covering M_{cov} from M_0 in space $\mathcal{O}(\log|M_0| + \log n + (\log W' + \log R')6^{K+2}K!m^3 \log m)$.*

PROOF. Since there are b buffers in the net, $\ell(b)$ gives an upper bound on the length of the shortest P-covering run. Therefore, there exists a P-covering run iff there is one of length at most $\ell(b)$. From Lemma 13 we get

$$\ell(b) \leq b(g(K))^{3b}2^a \leq m(g(K))^{3m}2^a \leq m\left((W'R')^{3^K K!}2^{6^K K!a}\right)^{3m}2^a \leq m\left((W'R')^{6^{K+1}K!a}\right)^{3m}2^a$$

Hence $\ell(b) \leq m(W'R')^{6^{K+2}K!m^2}$. A nondeterministic algorithm can guess a sequence of transitions of this length and verify that it is P-covering from M_0. The memory needed is dominated by a counter to count up to maximum $\ell(b)$ and the memory needed to store intermediate markings. The memory needed for the counter is $\mathcal{O}((\log W' + \log R')6^{K+2}K!m^2 \log m)$ and to store markings we need $\mathcal{O}(\log |M_0| + \log n + (\log W' + \log R')6^{K+2}K!m^3 \log m)$. ∎

Given a net, its benefit depth K can be computed in polynomial time. Hence, the upper bound on the memory requirement in the above theorem is space constructible and the well known Savitch's theorem can be applied to determinize the above algorithm (see any standard text on complexity theory). The memory required will still be polynomial in the size of the input net and this gives us the paraPSPACE algorithm.

For later use in section 5, we name the exponent $6^{K+2}K!m^2$ used in the above proof $expcov(1)$, and let $expcov(i) = expcov(1)^i$.

4 Benefit depth and boundedness

In this section, we will tighten Rosier and Yen's analysis [21] and prove that the complexity of boundedness problem is paraPSPACE when parameterized by benefit depth. As in coverability, we segregate transitions that reduce tokens from a place and those that do not.

DEFINITION 15. *Let $U \subseteq B$ be a subset of buffers, $Q \subseteq P$ a subset of places and M a vector. A Q-run σ from M is said to be U-**self-covering** if it can be decomposed as $\sigma_1\sigma_2$ with $M \xrightarrow{\sigma_1} M_1 \xrightarrow{\sigma_2} M_2$, $M_2 \geq M_1$ and for all $p \in U$, $M_2(p) > M_1(p)$. We call σ_2 as the **pumping portion** of the self-covering sequence.*

It is well known that a place p is unbounded iff there is a firing sequence that is U-self-covering from the initial marking[†] for some $U \subseteq P$ with $p \in U$. In the rest of this section, we will fix a non-empty subset U of places and refer to U-self-covering sequences as self-covering sequences. Let $T_{dep}(p) = \{t \in T_{ben}(p) \mid \forall p' \in Ind(p) : Pre(p', t) = 0\}$.

DEFINITION 16. *Let $C \subseteq Q \subseteq P$ and $p \in B$ be a buffer. Let $scov^p(Q, M)$ be the length of the shortest Q-run in $T_{ben}(p)^*$ that is self-covering from the vector M with the pumping portion of the sequence in $T_{dep}(p)^*$. If there is no such sequence, define $scov^p(Q, M)$ to be 0. Let $s_1(i) = \max\{scov^p(Q, M) \mid M$ a vector, $|Q \cap Ben(p) \cap B| = i\}$. Also, let $scov(Q, M)$ be the length of the shortest self-covering Q-run from the vector M and 0 if there is no such sequence. Let $s(i) = \max\{scov(Q, M) \mid M$ a vector, $|Q \cap B| = i\}$.*

LEMMA 17. *For $0 \leq i < b$, $s(i+1) \leq (W^2 s_1(K))^{poly(m)} + s_1(K) + (Ws_1(K) + 2)s(i)$ for $poly(m)$ a polynomial in m with degree independent of W, m, K.*

PROOF. Suppose that $Q = Q_{i+1} = C \cup A$ with $|A| = i + 1$ and that there is a self-covering Q_{i+1}-run σ from some vector M. If this run is $Ws_1(K)$-bounded for Q_{i+1}, the required result

[†]We thank an anonymous FSTTCS referee for pointing out a mistake here.

is a consequence of Lemma 2.2 in [21]. Otherwise some buffer $p \in Q_{i+1}$ has more than $Ws_1(K)$ tokens at some intermediate vector M'. The sequence occurring after M' can be replaced by a self-covering Q_i-run σ_2 of length at most $s(i)$, with $Q_i = Q_{i+1} \setminus \{p\}$. By repeated application of Lemma 8, rearrange the non-pumping portion of σ_2 into $\tau_1\tau_2$ and the pumping portion into $\tau_1'\tau_2'$, with $\tau_1, \tau_1' \in T_{ind}(p)^*$ and $\tau_2, \tau_2' \in T_{ben}(p)^*$ (see Fig. 4). τ_2'

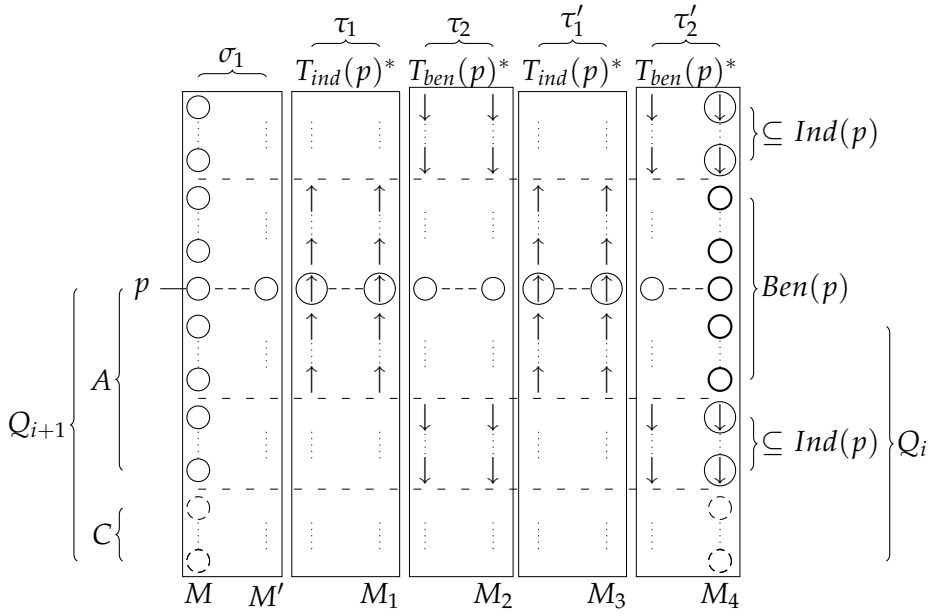


Figure 4: Sequences and bounds used in the proof of Lemma 17

is a sequence in $T_{ben}(p)^*$ that "pumps up" tokens in some of the places and hence can be replaced by another one τ_2'' of length at most $s_1(K)$. τ_2'' can however decrease tokens from places that are pumped up by τ_1', so we compensate for it by firing τ_1' $Ws_1(K)+1$ times. Putting everything together, we get $\tau_1\tau_2\tau_1'^{Ws_1(K)}\tau_2''$ is a self-covering Q_{i+1}-run from M'. ∎

The following lemmas give recurrence relations for length of self-covering sequences in $T_{ben}(p)^*$. The proofs are similar to those of corresponding lemmas in [21] with the additional fact that transitions in $T_{ben}(p)$ don't increase tokens in $Ind(p)$. As before, $W' = \max\{W, 2\}$.

LEMMA 18. *Let $C \subseteq Q \subseteq P$ and $p \in B$ a buffer. For $c \in \mathbb{N}$, suppose there is a self-covering Q-run in $T_{ben}(p)^*$ from some vector M which is c-bounded for $Q \cap Ben(p) \cap B$. If its pumping portion is in $T_{dep}(p)^*$, then a similar sequence exists whose length is at most $(W'c2^a)^{poly(K)}$ for $poly(K)$ some polynomial in K whose degree is independent of W, c, a, K.*

LEMMA 19. *$s_1(0) \leq (W'2^a)^{poly(K)}$ and $s_1(i+1) \leq (W'^2 s_1(i) 2^a)^{poly(K)}$.*

Now we give upper bounds for these recurrence relations and use them in a nondeterministic algorithm. A technical point is that the recurrence relation in Lemma 17 for $s(i)$ starts from $i = 1$ (unlike that in Lemma 9). This avoids the calculation of an upper bound for $s(u)$ using Lemma 20 below from containing terms m^K in the exponent, which is not acceptable in paraPSPACE algorithms.

LEMMA 20. *For $0 < i < b$, we have $s_1(i) \leq W'^{2(i+1)poly(K^{i+1})}2^{a(i+1)poly(K^{i+1})}$, as also $s(i) \leq 2^{i-1}(4Ws_1(K))^{i-1}(W^2s_1(K))^{poly(m)} + (4Ws_1(K))^i s(0)$.*

THEOREM 21. *There is a nondeterministic algorithm that decides if a net is bounded in space $\mathcal{O}(\log|M_0| + \log W'K^{cK}m^c a + m\log n)$ where c is some constant.*

5 The model checking algorithm

We now show that checking whether a given system (N, M_0) satisfies a given formula ϕ of the logic defined in sub-section 2.1 can be done in paraPSPACE with benefit depth as the parameter. This requires a lot of technical work. First of all, we simplify the kind of formulas that our algorithm has to handle by nondeterministically choosing a disjunct from a disjunctive subformula. We then end up with ϕ a sequence of conjuncts $\beta_1, \ldots, \beta_c, \kappa$, where each β_i is of the form $\{\tau_1, \cdots \tau_r\} < \omega$ or $\{\tau_1, \cdots \tau_r\} = \omega$ and κ consists of conjunctions of nested **EF** modalities over $\tau \geq c$ formulas. If we can check such formulas in paraPSPACE, Savitch's theorem ensures that satisfiability of ϕ can be checked in paraPSPACE.

For checking β_i, we need the following lemma. The proof of this lemma relies on some results on Karp-Miller trees, in particular on [8, Theorems 21 and 22]. Recall that every term τ gives a function $L_\tau : P \to \mathbb{N}$ such that τ is syntactically equivalent to $\sum_{p\in P} L_\tau(p)p$.

LEMMA 22. *$N, M_0 \models \{\tau_1, \ldots, \tau_r\} = \omega$ iff there exists a U-self-covering sequence for some $U \subseteq P$ such that for every $j \in \{1.\ldots, r\}$, there is a $p_j \in U$ with $L_{\tau_j}(p_j) \geq 1$.*

Hence, checking of β_i can be done in paraPSPACE by using results of section 4.

We now consider verifying the formulas κ, which are of the form $\gamma \wedge \mathbf{EF}(\kappa_1) \wedge \cdots \wedge \mathbf{EF}(\kappa_r)$, with γ having only conjunctions of $\tau \geq c$ formulas. We call γ the **content** of κ and $\kappa_1, \ldots, \kappa_r$ as the **children** of κ. Each of the children may have their own content and children, thus generating a tree with nodes Γ, with κ at the root of this tree. We will represent nodes of this tree by sequences of natural numbers, 0 being the root.

The maximum length of sequences in Γ is one more than the nesting depth of the **EF** modality in κ and we denote it by D. Let $[D] = \{0, 1, \ldots, D-1\}$. If $\alpha \in \Gamma$ is a tree node that represents the formula $\kappa(\alpha) = \gamma \wedge \mathbf{EF}(\kappa_1) \wedge \cdots \wedge \mathbf{EF}(\kappa_r)$, $content(\alpha) = \gamma$ denotes the content of the node α. Let $ratio(\tau \geq c) = max\{\lceil c/L_\tau(p)\rceil \mid L_\tau(p) \neq 0, p \in P\}$. Defining $\max(\emptyset) = 0$, we define the maximum ratio at height i in the tree by $ratio(i) = \max\{ratio(\tau \geq c) \mid \tau \geq c$ appears as a conjunct in $content(\alpha)$ for some $\alpha \in \Gamma, |\alpha| = i+1\}$. Recall from Definition 6 that b is the number of buffers and $\ell'(M)$ the length of the shortest run covering M using all the buffers $\ell(b, M)$.

DEFINITION 23. *Given a formula κ and a system (N, M_0), the bound function $f : [D] \times P \to \mathbb{N}$ is defined as follows. We use $f(j)$ for the marking defined by $f(j)(p) = f(j, p)$.*

- $f(D-1, p) = ratio(D-1)$,
- $f(D-i, p) = \max\{ratio(D-i), W\ell'(f(D-i+1)) + f(D-i+1, p)\}, 1 < i < D$,
- $f(0, p) = M_0(p)$.

A guess function $h : \Gamma \times P \to \mathbb{N}$ is any function that satisfies $h(\alpha, p) \leq f(|\alpha| - 1, p)$ for all $\alpha \in \Gamma$ and $p \in P$. If h is a guess function, $h(\alpha)$ is the marking defined by $h(\alpha)(p) = h(\alpha, p)$.

If a given system satisfies the formula $\kappa = \gamma \wedge \mathbf{EF}(\kappa_1) \wedge \cdots \wedge \mathbf{EF}(\kappa_r)$, then there exist firing sequences $\sigma_{01}, \ldots, \sigma_{0r}$ that are all enabled at the initial marking M_0 such that $M_0 \stackrel{\sigma_{0i}}{\Longrightarrow} M_{0i}$ and M_{0i} satisfies κ_i. In general, if κ generates a tree with set of nodes Γ, then there is a set of sequences $\{\sigma_\alpha \mid \alpha \in \Gamma \setminus \{0\}\}$ and set of markings $\{M_\alpha \mid \alpha \in \Gamma\}$ such that $M_\alpha \stackrel{\sigma_{\alpha j}}{\Longrightarrow} M_{\alpha j}$ for all $\alpha, \alpha j \in \Gamma$ and M_α satisfies $content(\alpha)$ for all $\alpha \in \Gamma$.

LEMMA 24. *There exist sequences $\{\mu_\alpha \mid \alpha \in \Gamma \setminus \{0\}\}$ and markings $\{M_\alpha \mid \alpha \in \Gamma\}$ such that $M_\alpha \stackrel{\mu_{\alpha j}}{\Longrightarrow} M_{\alpha j}$ for all $\alpha, \alpha j \in \Gamma$ with M_α satisfying $content(\alpha)$ and $|\mu_\alpha| \leq \ell'(f(|\alpha| - 1))$ iff there exist sequences $\{\sigma_\alpha \mid \alpha \in \Gamma \setminus \{0\}\}$ and markings $\{M'_\alpha \mid \alpha \in \Gamma\}$ (M'_0 should be equal to M_0) such that $M'_\alpha \stackrel{\sigma_{\alpha j}}{\Longrightarrow} M'_{\alpha j}$ for all $\alpha, \alpha j \in \Gamma$ with M'_α satisfying $content(\alpha)$.*

To derive an upper bound for $f(i)$ to use in a nondeterministic algorithm, let $R = \max\{ratio(\tau \geq c) \mid \tau \geq c \text{ is a subformula of } \kappa\}$, $R' = \max\{R, 2\}$ and $W' = \max\{W, 2\}$. Recall that $D - 1$ is the nesting depth of **EF** and note that boundedness and coverability can be expressed with $D \leq 2$.

LEMMA 25. *For $i \geq 2$, $f(D - i, p) \leq (i + 1)R'W\ell'(f(D - i + 1))$.*

LEMMA 26. *Recall from the end of section 3 that $expcov(i) = (6^{K+2}K!m^2)^i$. Then $\ell'(f(D - 1)) \leq m(W'R')^{expcov(1)}$ and $\ell'(f(D - i)) \leq m \prod_{j=D-i}^{D} \left((D - j + 1)W'^2R'm\right)^{expcov(i+j+1-D)}$.*

THEOREM 27. *Given a net and a formula ϕ, if the benefit depth of the net is treated as a parameter and the nesting depth D of **EF** modality in the formula is treated as a constant, then there is a* paraPSPACE *algorithm that checks if the net satisfies the given formula.*

PROOF. By Lemma 24, it is enough for a nondeterministic algorithm to guess sequences $\sigma_{\alpha j}$, $\alpha j \in \Gamma$ of lengths at most $\ell'(f(|\alpha j| - 1))$ and verify that they satisfy the formula. Using bounds given by Lemma 26 and an argument similar to the one in the proof of Theorem 14, it can be shown that the space used is exponential in K and polynomial in the size of the net and numeric constants in the formula. This gives the paraPSPACE algorithm. ∎

The space requirement of the above algorithm will have terms like m^{2D} and hence it will not be paraPSPACE if D is treated as a parameter instead of a constant.

6 Conclusion

We considered nets communicating with buffers. These are infinite-state concurrent systems allowing 1-safe Petri net components communicating through synchronization, which in turn communicate asynchronously through a fixed set of buffers. We identified the parameter benefit depth that measures the maximum number of other buffers that any one buffer can influence. We showed that based on this parameter, paraPSPACE algorithms can be obtained for the coverability and boundedness problems. Note that this does *not* yield a paraPSPACE algorithm for the reachability problem. Whether benefit depth can yield such an algorithm is open; for work of this kind we refer to Kostin [16]. We then extended the above technique to show that satisfiability of formulas of the logic given in sub-section 2.1 can be checked in paraPSPACE if the nesting depth of **EF** quantifiers in such formulas is treated as a constant.

References

[1] M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, volume 5201 of *LNCS*, pages 356–371, 2008.

[2] M. F. Atig and P. Habermehl. On Yen's path logic for Petri nets. In *RP 2009*, volume 5797 of *LNCS*, pages 51–63, 2009.

[3] D. Brand and P. Zafiropulo. On communicating finite-state machines. *JACM*, 30, 1983.

[4] D. Caucal. On the regular structure of prefix rewriting. *TCS*, 106:61–86, 1992.

[5] G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Inf. Comput.*, 124(1):20–31, 1996.

[6] A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. *Theoret. Comp. Sci.*, 147(1-2):117–136, 1995.

[7] J. Desel and J. Esparza. *Free choice Petri nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.

[8] J. Desel and W. Reisig. *Place transition Petri nets*, volume 1491 of *LNCS*. 1998.

[9] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Springer-Verlag, 1999.

[10] J. Esparza. *Decidability and complexity of Petri net problems — An introduction*, volume 1491 of *LNCS*, pages 374–428. 1998.

[11] J. Flum and M. Grohe. Describing parameterized complexity classes. *Inf. Comput.*, 187(2):291–319, 2003.

[12] P. Habermehl. On the complexity of the linear-time μ-calculus for Petri-nets. In *ATPN '97*, volume 1248 of *LNCS*, pages 102–116, 1997.

[13] R. Howell, L.E. Rosier, and H.-C. Yen. A taxonomy of fairness and temporal logic problems for petri nets. *Theoret. Comp. Sci.*, 82(2):341–372, 1991.

[14] R.M. Karp and R.E. Miller. Parallel program schemata. *JCSS*, 3(2):147–195, May 1969.

[15] S.R. Kosaraju. Decidability of reachability in vector addition systems. In *Proc. 14th STOC*, pages 267–281. ACM, 1982.

[16] A.E. Kostin. Using transition invariants for reachability analysis of Petri nets. In V. Kordic, editor, *Petri net: theory and applications*, pages 435–458. I-Tech Edu. Pub., 2008.

[17] R. Lipton. The reachability problem requires exponential space. Yale university, 1975.

[18] E.W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.

[19] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Inst. Instrumentelle Math., 1962.

[20] C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoret. Comp. Sci.*, 6:223–231, 1978.

[21] L.E. Rosier and H.-C. Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *J. Comput. Syst. Sci.*, 32(1):105–135, 1986.

[22] P. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Inf. Proc. Lett.*, 83(5):251–261, 2002.

[23] R. Valk and G. Vidal-Naquet. Petri nets and regular languages. *JCSS*, 23:299–325, 1981.

[24] H.-C. Yen. A unified approach for deciding the existence of certain petri net paths. *Inf. Comput.*, 96(1):119–137, 1992.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Synthesis of Finite-state and Definable Winning Strategies*

Alexander Rabinovich
The Blavatnik School of Computer Science, Tel Aviv University, Israel 69978
rabinoa@post.tau.ac.il

ABSTRACT. Church's Problem asks for the construction of a procedure which, given a logical specification φ on sequence pairs, realizes for any input sequence I an output sequence O such that (I, O) satisfies φ. McNaughton reduced Church's Problem to a problem about two-player ω-games. Büchi and Landweber gave a solution for Monadic Second-Order Logic of Order (MLO) specifications in terms of finite-state strategies. We consider two natural generalizations of the Church problem to countable ordinals: the first deals with finite-state strategies; the second deals with MLO-definable strategies. We investigate games of arbitrary countable length and prove the computability of these generalizations of Church's problem.

1 Introduction

Two fundamental results of classical automata theory are decidability of the monadic second-order logic of order (MLO) over $\omega = (\mathbb{N}, <)$ and computability of the Church synthesis problem. These results have provided the underlying mathematical framework for the development of formalisms for the description of interactive systems and their desired properties, the algorithmic verification and the automatic synthesis of correct implementations from logical specifications, and advanced algorithmic techniques that are now embodied in industrial tools for verification and validation.

In order to prove decidability of the monadic theory of ω, Büchi introduced finite automata over ω-words. He provided a computable reduction from formulas to finite automata.

Büchi also introduced automata which "work" on words of any countable length (ordinal) and proved that the MLO-theory of any countable ordinal is decidable (see [BS73]).

What is known as the "Church synthesis problem" was first posed by Church in [Ch63] for the case of $(\omega, <)$. The Church problem is much more complex than the decidability problem for MLO. Church uses the language of automata theory. It was McNaughton [Mc65] who first observed that the Church problem can be equivalently phrased in game-theoretic language.

Let $\alpha > 0$ be an ordinal and let $\varphi(X_1, X_2)$ be a formula, where X_1 and X_2 are set (monadic predicate) variables. The *McNaughton game* $\mathcal{G}_\varphi^\alpha$ is defined as follows.

1. The game is played by two players, called Player I and Player II.
2. A *play* of the game has α rounds.
3. At round $\beta < \alpha$: first, Player I chooses $\pi_{X_1}(\beta) \in \{0,1\}$; then, Player II chooses $\pi_{X_2}(\beta) \in \{0,1\}$.

*This work was partially supported by the ESF Research Networking Programme Games and the UK's EPSRC.

© A. Rabinovich; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 359–370
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2332

4. By the end of the play two monadic predicates $\pi_{X_1}, \pi_{X_2} \subseteq \alpha$ have been constructed[†].
5. Then, Player I wins the play if $(\alpha, <) \models \varphi(\pi_{X_1}, \pi_{X_2})$; otherwise, Player II wins.

What we want to know is: Does either one of the players have a *winning strategy* in $\mathcal{G}^\alpha_\varphi$? If so, which one? That is, can Player I choose his moves so that, whatever way Player II responds we have $\varphi(\pi_{X_1}, \pi_{X_2})$? Or can Player II respond to Player I's moves in a way that ensures the opposite?

This leads to

Game version of the Church problem Let α be an ordinal. Given an MLO formula $\varphi(X_1, X_2)$, decide whether Player I has a winning strategy in $\mathcal{G}^\alpha_\varphi$.

In [BL69], Büchi and Landweber prove the computability of the Church problem in $\omega = (\mathbb{N}, <)$. Even more importantly, they show that in the case of ω we can restrict ourselves to MLO-*definable strategies*, or equivalently, to *finite-state strategies* (see Sect. 3 for the definitions of these strategies).

THEOREM 1.1 (BÜCHI-LANDWEBER, 1969) *Let $\varphi(X_1, X_2)$ be an MLO formula. Then:*

Determinacy *One of the players has a winning strategy in the game $\mathcal{G}^\omega_\varphi$.*

Decidability *It is decidable* which *of the players has a winning strategy.*

Definable strategy *The player who has a winning strategy, also has an MLO-*definable *winning strategy.*

Synthesis *We can compute a formula $\psi(X_1, X_2)$ that defines (in ω) a winning strategy for the winning player in $\mathcal{G}^\omega_\varphi$.*

After stating their main theorem, Büchi and Landweber write:

> "We hope to present elsewhere a corresponding extension of [our main theorem] from ω to any countable ordinal."

However, despite the fundamental role of the Church problem, no such extension is even mentioned in a later book by Büchi and Siefkes [BS73], which summarizes the theory of finite automata over words of countable ordinal length.

We proved in [RS08a, Rab09] that the Büchi-Landweber theorem extends fully to all ordinals $< \omega^\omega$ and its determinacy and decidability parts extend to all countable ordinals.

In [RS08], we provided a counter-example to a full extension of the Büchi-Landweber theorem to $\alpha \geq \omega^\omega$. For every ordinal $\alpha \geq \omega^\omega$ we constructed an MLO formula $\varphi_\alpha(X_1, X_2)$ such that Player I has a winning strategy in $\mathcal{G}^\alpha_{\varphi_\alpha}$; however, he has no MLO-definable winning strategy.

For $\alpha \leq \omega^\omega$, the set of MLO-definable in α strategies is the same as the set of finite-state strategies. However, for $\alpha > \omega^\omega$, the set of MLO-definable in α strategies properly contains the set of finite-state strategies. This leads to the following two synthesis problems for $\alpha \geq \omega^\omega$:

Synthesis Problems for α
Input: an MLO formula $\varphi(X_1, X_2)$.
Task1: Decide whether one of the players has a definable winning strategy in $\mathcal{G}^\alpha_\varphi$, and if so, construct ψ which defines his winning strategy.
Task2: Decide whether one of the players has a finite-state winning strategy in $\mathcal{G}^\alpha_\varphi$, and if so, construct such a strategy.

[†]We identify monadic predicates with their characteristic functions.

The first task is the synthesis problem of definable strategy and it will be denoted by Dsynth(α); the second task is the synthesis problem of finite-state strategy and it will be denoted by Fsynth(α).

In [Rab09], we reduced the synthesis problem Dsynth(α) to Dsynth(ω^ω). However, the decidability of the latter remained open.

Two main contributions of this paper are: the computability of Dsynth(ω^ω) (and, as a consequence, the computability of Dsynth(α)), and the computability of Fsynth(α). Our results are stronger than the computability of Dsynth(α) and Fsynth(α). For every countable α we need *finite* amount of data (code of α) which determines its monadic theory (see Subsection 2.2). We prove that there is an algorithm that receives the code of an ordinal α and a formula φ and decides whether Player I has a definable or finite-state strategy in the McNaughton game $\mathcal{G}^\alpha_\varphi$.

Our proofs use both game theoretical techniques and the "composition method" developed by Feferman-Vaught, Shelah and others (see, e.g. [Sh75]).

The article is organized as follows. The next section recalls standard definitions about monadic logic of order, summarizes elements of the composition method and reviews known facts about the monadic theory of countable ordinals. In Sect. 3, we provide definitions of the finite-state and MLO-definable strategies and survey results about McNauughton games of countable length. In Section 4, we introduce special games on types and provide a reduction of these games to the McNaughton games. Section 5 contains the main results of the paper and outlines the proof of the computability of the synthesis problem for MLO-definable strategies. Finally, in Sect. 6, we discuss some open problems.

2 Preliminaries on Monadic Logic of Order

Notations and terminology We use n, k, l, m, p, q for natural numbers and $\alpha, \beta, \gamma, \delta$ for ordinals. We use $\mathbb{N}$ for the set of natural numbers and ω for the first infinite ordinal. We write $\alpha + \beta$, $\alpha\beta$, α^β for the sum, multiplication and exponentiation, respectively, of ordinals α and β. We use the expressions "*chain*" and "*linear order*" interchangeably. We use $\mathbb{P}(A)$ for the set of subsets of A.

2.1 The Monadic Logic of Order (MLO)

Syntax The syntax of the monadic second-order logic of order - MLO has in its vocabulary *individual* (first order) variables $t_1, t_2 \ldots$, monadic *second-order* variables $X_1, X_2 \ldots$ and one binary relation $<$ (the order).

Atomic formulas are of the form $X(t)$ and $t_1 < t_2$. Well-formed formulas of the monadic logic MLO are obtained from atomic formulas using Boolean connectives $\neg, \vee, \wedge, \rightarrow$, the first-order quantifiers $\exists t$ and $\forall t$, and the second-order quantifiers $\exists X$ and $\forall X$. The quantifier depth of a formula φ is denoted by $\mathrm{qd}(\varphi)$.

We use upper case letters X, Y, Z to denote second-order variables, and overlined letters $\bar{X}, \bar{Y}$ to denote finite tuples of variables.

Semantics A *structure* is a tuple $\mathcal{M} := (A^\mathcal{M}, <^\mathcal{M}, \bar{P}^\mathcal{M})$ where: $A^\mathcal{M}$ is a non-empty set, $<^\mathcal{M}$ is a binary relation on $A^\mathcal{M}$, and $\bar{P}^\mathcal{M} := \left(P_1^\mathcal{M}, \ldots, P_l^\mathcal{M}\right)$ is a *finite* tuple of subsets of $A^\mathcal{M}$.

If $\bar{P}^{\mathcal{M}}$ is a tuple of l sets, we call $\mathcal{M}$ an *l-structure*. If $<^{\mathcal{M}}$ linearly orders $A^{\mathcal{M}}$, we call $\mathcal{M}$ an *l-chain*.

Suppose $\mathcal{M}$ is an l-structure and φ a formula with free-variables among $X_1, \ldots, X_l$. We define the relation $\mathcal{M} \models \varphi$ (read: $\mathcal{M}$ *satisfies* φ) as usual, understanding that the second-order quantifiers range over *subsets* of $A^{\mathcal{M}}$.

Let $\mathcal{M}$ be an l-structure. The *monadic theory* of $\mathcal{M}$, $MTh(\mathcal{M})$, is the set of all formulas with free variables among $X_1, \ldots, X_l$ satisfied by $\mathcal{M}$.

¿From now on, we omit the superscript in '$<^{\mathcal{M}}$' and '$\bar{P}^{\mathcal{M}}$'. We often write $(A, <) \models \varphi(\bar{P})$ meaning $(A, <, \bar{P}) \models \varphi$.

2.2 The monadic theory of countable ordinals

Büchi (for instance [BS73]) has shown that there is a *finite* amount of data concerning any countable ordinal which determines its monadic theory:

THEOREM 2.1 *Let α be a countable ordinal. Write $\alpha = \omega^{\omega}\beta + \zeta$ where $\zeta < \omega^{\omega}$ (this can be done in a unique way). Then the monadic theory of $(\alpha, <)$ is determined by:*

1. *whether $\alpha < \omega^{\omega}$, and*
2. *ζ.*

We can associate with every countable α a finite *code* which holds the data required in the previous theorem. This is clear with respect to (1). As for (2), if $\zeta \neq 0$, write

$\zeta = \sum_{i \leq n} \omega^{n-i} \cdot a_{n-i}$, where $n, a_i \in \mathbb{N}$ for $i \leq n$ and $a_n \neq 0$

(this, too, can be done in a unique way), and let the sequence $(a_n, \ldots, a_0)$ encode ζ. The following is implicit in [BS73]:

THEOREM 2.2 (MONADIC DECIDABILITY THEOREM) *There is an algorithm that, given a sentence φ and the* code *of a countable ordinal α, determines whether $(\alpha, <) \models \varphi$.*

We conclude by a well-known Lemma which is easily derived from Büchi results [BS73], as well from the composition theorem (see Theorem 2.11).

LEMMA 2.3 *For every n there is m computable from n such that for every MLO sentence φ of the quantifier depth at most n and every countable ordinals $\alpha > 0$ and β:*

$$\omega^m + \beta \models \varphi \text{ if and only if } \omega^m\alpha + \beta \models \varphi$$

2.3 Elements of the composition method

Our proofs make use of the technique known as the composition method developed by Feferman-Vaught and Shelah [FV59, Sh75]. To fix notations and to aid the reader unfamiliar with this technique, we briefly review the required definitions and results. A more detailed presentation can be found in [Th97] or [Gu85].

Let $n, l \in \mathbb{N}$. We denote by $\mathfrak{Form}_l^n$ the set of formulas with free variables among $X_1, \ldots, X_l$ and of quantifier depth $\leq n$.

DEFINITION 2.4 *Let $n, l \in \mathbb{N}$ and let $\mathcal{M}, \mathcal{N}$ be l-structures. The n-*theory *of $\mathcal{M}$ is*

$$Th^n(\mathcal{M}) := \{\varphi \in \mathfrak{Form}_l^n \mid \mathcal{M} \models \varphi\}.$$

*If $Th^n(\mathcal{M}) = Th^n(\mathcal{N})$, we say that $\mathcal{M}$ and $\mathcal{N}$ are n-*equivalent *and write $\mathcal{M} \equiv^n \mathcal{N}$.*

Clearly, $\equiv^n$ is an equivalence relation. For any $n \in \mathbb{N}$ and $l > 0$, the set $\mathfrak{Form}_l^n$ is infinite. However, it contains only finitely many semantically distinct formulas. So, there are finitely many $\equiv^n$-equivalence classes of l-structures. In fact, we can compute characteristic sentences for the $\equiv^n$-equivalence classes:

LEMMA 2.5 (HINTIKKA LEMMA) *For $n, l \in \mathbb{N}$, we can compute a* finite *set $Char_l^n \subseteq \mathfrak{Form}_l^n$ such that:*

1. For every $\equiv^n$-equivalence class A there is a unique *$\tau \in Char_l^n$ such that for every l-structure $\mathcal{M}$: $\mathcal{M} \in A$ iff $\mathcal{M} \models \tau$.*

2. Every MLO formula $\varphi(X_1, \ldots X_l)$ with $\mathrm{qd}(\varphi) \leq n$ is equivalent to a (finite) disjunction of characteristic formulas from $Char_l^n$. Moreover, there is an algorithm which for every formula $\varphi(X_1, \ldots X_l)$ computes a finite set $G_\varphi \subseteq Char_l^{\mathrm{qd}(\varphi)}$ of characteristic formulas, such that φ is equivalent to the disjunction of all the formulas in G.

Any member of $Char_l^n$ we call a *(n,l)-Hintikka formula* or (n,l)-characteristic formula. We use τ, τ_i, τ^j to range over the characteristic formulas and G, G_i, G' to range over sets of characteristic formulas. Usually, we do not distinguish between φ and the corresponding set G_φ of characteristic formulas.

DEFINITION 2.6 (n-TYPE) *For $n, l \in \mathbb{N}$ and an l-structure $\mathcal{M}$, we denote by $type_n(\mathcal{M})$ the unique member of $Char_l^n$ satisfied by $\mathcal{M}$ and call it the n-*type *of $\mathcal{M}$.*

Thus, $type_n(\mathcal{M})$ determines $Th^n(\mathcal{M})$ and, indeed, $Th^n(\mathcal{M})$ is computable from $type_n(\mathcal{M})$.

DEFINITION 2.7 (SUM OF CHAINS) *Let $l \in \mathbb{N}$, $\mathcal{I} := (I, <^{\mathcal{I}})$ a chain and $\mathfrak{S} := (\mathcal{M}_\alpha \mid \alpha \in I)$ a sequence of l-chains. Write $\mathcal{M}_\alpha := (A_\alpha, <^\alpha, P_1{}^\alpha, \ldots, P_l{}^\alpha)$ and assume that $A_\alpha \cap A_\beta = \emptyset$ whenever $\alpha \neq \beta$ are in I. The* ordered sum *of $\mathfrak{S}$ is the l-chain*

$$\sum_{\mathcal{I}} \mathfrak{S} := (\bigcup_{\alpha \in I} A_\alpha, <^{\mathcal{I},\mathfrak{S}}, \bigcup_{\alpha \in I} P_1{}^\alpha, \ldots, \bigcup_{\alpha \in I} P_l{}^\alpha),$$

where: if $\alpha, \beta \in I$, $a \in A_\alpha$, $b \in A_\beta$, then $b <^{\mathcal{I},\mathfrak{S}} a$ iff $\beta <^{\mathcal{I}} \alpha$ or $\beta = \alpha$ and $b <^\alpha a$.

If the domains of the $\mathcal{M}_\alpha$'s are not disjoint, replace them with isomorphic l-chains that have disjoint domains, and proceed as before.

If $\mathcal{I} = (\{0,1\}, <)$ and $\mathfrak{S} = (\mathcal{M}_0, \mathcal{M}_1)$, we denote $\sum_{\mathcal{I}} \mathfrak{S}$ by $\mathcal{M}_0 + \mathcal{M}_1$.

The next proposition states that taking ordered sums preserves $\equiv^n$-equivalence.

PROPOSITION 2.8 *Let $n, l \in \mathbb{N}$. Assume:*

1. *$(I, <^{\mathcal{I}})$ is a linear order,*
2. *$(\mathcal{M}_\alpha^0 \mid \alpha \in I)$ and $(\mathcal{M}_\alpha^1 \mid \alpha \in I)$ are sequences of l-chains, and*
3. *for every $\alpha \in I$, $\mathcal{M}_\alpha^0 \equiv^n \mathcal{M}_\alpha^1$.*

Then, $\sum_{\alpha \in I} \mathcal{M}_\alpha^0 \equiv^n \sum_{\alpha \in I} \mathcal{M}_\alpha^1$.

This allows us to define the sum of formulas in $Char_l^n$ with respect to any linear order.

DEFINITION 2.9 *Let $n, l \in \mathbb{N}$, $\mathcal{I} := (I, <^{\mathcal{I}})$ a chain, $\mathfrak{H} := (\tau_\alpha \mid \alpha \in I)$ a sequence of (n,l)-Hintikka formulas. The* ordered sum *of $\mathfrak{H}$, (notations $\sum_{\mathcal{I}} \mathfrak{H}$ or $\sum_{\alpha \in \mathcal{I}} \tau_\alpha$), is* an *element τ of $Char_l^n$ such that:*

if $\mathfrak{S} := (\mathcal{M}_\alpha \mid \alpha \in I)$ is a sequence of l-chains and $type_n(\mathcal{M}_\alpha) = \tau_\alpha$ for $\alpha \in I$, then

$$type_n(\sum_{\mathcal{I}} \mathfrak{S}) = \tau.$$

If $\mathcal{I} = (\{0,1\}, <)$ and $\mathfrak{H} = (\tau_0, \tau_1)$, we denote $\sum_{\alpha \in \mathcal{I}} \tau_\alpha$ by $\tau_0 + \tau_1$.

The next Lemma states that the sum of two types is computable.

LEMMA 2.10 (ADDITION LEMMA) *The function which maps the pairs of characteristic formulas to their sum is recursive. Formally, $\lambda n, l \in \mathbb{N}.\lambda\tau_0, \tau_1 \in Char_l^n.\tau_0 + \tau_1$ is recursive.*

The following fundamental result of Shelah can be found in [Sh75]:

THEOREM 2.11 (COMPOSITION THEOREM) *Let $\varphi(X_1, \ldots, X_l)$ be a formula, let $n = \mathrm{qd}(\varphi)$ and let $\{\tau_1, \ldots, \tau_m\} = Char_l^n$. Then, there is a formula $\psi(Y_1, \ldots, Y_m)$ such that for every chain $\mathcal{I} = (I, <)$ and sequence $(\mathcal{M}_\alpha \mid \alpha \in I)$ of l-chains the following holds:*

$$\sum_{\alpha \in I} \mathcal{M}_\alpha \models \varphi \quad \textit{iff} \quad \mathcal{I} \models \psi(Q_1, \ldots, Q_m), \textit{ where}$$

$Q_j = \{\alpha \in I \mid M_\alpha \models \tau_j\}$. Moreover, ψ is computable from φ.

3 Finite-state and MLO-definable strategies

In the McNaughton game $\mathcal{G}_\varphi^\alpha$, at round $\beta < \alpha$, Player I has access only to $\pi_{X_2} \cap [0, \beta)$ and Player II has access only to $\pi_{X_1} \cap [0, \beta]$. Therefore, the following formalizes well the notion of a strategy in this game:

DEFINITION 3.1 (CAUSAL OPERATOR) *Let α be an ordinal, $F : \mathbb{P}(\alpha) \to \mathbb{P}(\alpha)$ maps the subsets of α into the subsets of α. We call F* causal *(resp.* strongly causal*) iff for all $P, P' \subseteq \alpha$ and $\beta < \alpha$:*
if $P \cap [0, \beta] = P' \cap [0, \beta]$ (resp. $P \cap [0, \beta) = P' \cap [0, \beta)$), then
$F(P) \cap [0, \beta] = F(P') \cap [0, \beta]$.
That is, if P and P' agree up to and including *(resp.* up to*) β, then so do $F(P)$ and $F(P')$.*

So, a winning strategy for Player I is a strongly causal $F : \mathbb{P}(\alpha) \to \mathbb{P}(\alpha)$ such that for every $P \subseteq \alpha$, $(\alpha, <) \models \varphi(F(P), P)$; a winning strategy for Player II is a causal $F : \mathbb{P}(\alpha) \to \mathbb{P}(\alpha)$ such that for every $P \subseteq \alpha$, $(\alpha, <) \models \neg\varphi(P, F(P))$.

Let $\psi(X_1, X_2)$ be a formula where X_2 is declared as the "domain" variable and X_1 as the "range" variables. Let $\mathcal{M} := (A, <)$ be a chain and let $F : \mathbb{P}(A) \to \mathbb{P}(A)$ be an operator. We say that ψ *defines* F in $\mathcal{M}$ if $\mathcal{M} \models \psi(P_1, P_2)$ iff $P_1 = F(P_2)$.

It is easy to formalize in MLO that ψ defines in $\mathcal{M}$ a causal or strongly causal operator. Hence, for every ψ there are sentences I-Player-strategy$_\psi$ and II-Player-strategy$_\psi$ such that $\alpha \models$ I-Player-strategy$_\psi$ iff ψ defines (in α) a strategy for Player I, and $\alpha \models$ II-Player-strategy$_\psi$ iff ψ defines (in α) a strategy for Player II. A play $\rho := (\rho_{X_1}(0), \rho_{X_2}(0)) \ldots (\rho_{X_1}(\beta), \rho_{X_2}(\beta)) \ldots$ is consistent with the strategy defined in α by ψ if $\alpha \models \psi(\rho_{X_1}, \rho_{X_2})$. A Player I's strategy defined by ψ is winning in $\mathcal{G}_\varphi^\alpha$ if $\alpha \models \forall X_1 X_2 \psi(X_1, X_2) \to \varphi(X_1, X_2)$. Hence, the monadic theory of an ordinal α "knows" which formulas defines in α a strategy and which definable strategies are winning in $\mathcal{G}_\varphi^\alpha$.

A formula $\psi(\bar{X}, t)$ with at most one free individual variable t is (syntactically) *bounded* if all its first-order quantifiers are of the form $\exists^{<t} y \ldots$ (short for $\exists y(y < t \wedge \ldots)$ and $\forall^{<t} y \ldots$ (short for $\forall y(y < t \to \ldots)$).

If $\psi(X_1, X_2, t)$ is syntactically bounded and does not contain the atomic formulas $X_1(t)$ and $X_2(t)$, then $\forall t\big(X_1(t) \leftrightarrow \psi\big)$ defines in every ordinal a strategy for Player I (a strongly causal operator); ψ is said to be an *explicit definition* of this strategy. Similarly, if $\psi(X_1, X_2, t)$ is syntactically bounded and does not contain the atomic formula $X_2(t)$, then $\forall t\big(X_2(t) \leftrightarrow \psi\big)$ defines in every ordinal a strategy for Player II (a causal operator).

The strategies explicitly defined by the bounded formulas can be computed by finite-state transducers. A finite state transducer consists of a finite set Q - memory states, an initial state q_{init}, next-state functions $\text{next}_1 : Q \to Q$ and $\text{next}_2 : Q \times \{0,1\} \to Q$, a limit transition function $\Delta : \mathbb{P}(Q) \to Q$, and an output function $\text{out} : Q \to \{0,1\}$.

During a play, according to a transducer, at round β, Player I first updates the state according to next_1 or Δ, outputs value according to out, and then after a move of Player II updates the state. Formally, a play $\rho := (\rho_{X_1}(0), \rho_{X_2}(0)) \dots (\rho_{X_1}(\beta), \rho_{X_2}(\beta)) \dots$ is consistent with such a strategy if there are $q_0, q'_0 \dots q_\beta q'_\beta \dots$ such that $q_0 = q_{init}$

1. If $\beta = \beta' + 1$ is a successor ordinal, then $q_\beta = \text{next}_1(q'_{\beta'})$
2. If β is a limit ordinal then $q_\beta = \Delta(L)$, where $L := \{q \in Q \mid q \text{ appears cofinally often in } q_0, q'_0 \dots q_\gamma q'_\gamma \dots (\gamma < \beta)\}$.
3. $\rho_{X_1}(\beta) = \text{out}(q_\beta)$.
4. $q'_\beta = \text{next}_2(q_\beta, \rho_{X_2}(\beta))$

It is clear that a transducer defines a strategy *st* for Player I. Moreover, *st* is definable by a transducer iff it is explicitly definable by a bounded a formula.

Every ordinal $\alpha < \omega^\omega$ is MLO-definable. It is not difficult to show that a strategy is MLO-definable in $\alpha < \omega^\omega$ iff it is finite-state strategy (equivalently is explicitly defined by a bounded formula). If for a countable ordinal α every cofinal interval (β, α) is isomorphic to α, then a strategy is finite-state iff it is MLO-definable in α. However, the set of MLO-definable strategies is larger than the set of finite-state strategies; e.g., if $n > 0$ and φ expresses "X_1 contains exactly the last element", then Player I has a definable winning strategy in $\mathcal{G}_\varphi^{\omega^\omega + n}$, but he has no finite-state winning strategy in this game.

We recall below results from [Rab09, RS08] about McNauughton games over ordinals, and results from [CH08] about reachability and safety games of length ω^ω.

THEOREM 3.2 *Let α be a* countable *ordinal, $\varphi(X_1, X_2)$ a formula.*

Determinacy *One of the players has a winning strategy in the game $\mathcal{G}_\varphi^\alpha$.*

MLO characterization of the winner *There is a sentence $win(\varphi)$ such that for every countable ordinal α: Player I wins $\mathcal{G}_\varphi^\alpha$ if and only if $\alpha \models win(\varphi)$. Furthermore, $win(\varphi)$ is computable from φ.*

Decidability *There is an algorithm that given α and φ decides* which *of the players has a winning strategy in $\mathcal{G}_\varphi^\alpha$.*

No definable winning strategy *For every $\alpha \geq \omega^\omega$, there is a formula φ such that no player has a definable winning strategy in $\mathcal{G}_\varphi^\alpha$.*

Finite-state winning strategy *If $\alpha < \omega^\omega$, then the player who has a winning strategy, also has a* finite-state *winning strategy.*

Synthesis *If $\alpha < \omega^\omega$, then we can compute a finite-state winning strategy for the winning player in $\mathcal{G}_\varphi^\alpha$.*

Hence, the Büchi-Landweber theorem extends fully to the ordinals less than ω^ω, and its determinacy and decidability parts extends to all countable ordinals.

REMARK 3.3 *1. In this paper, whenever we say that an algorithm is "given an ordinal..." or "returns an ordinal...", we mean the* code *of the ordinal. In particular, this holds for the decidability and synthesis parts of Theorem 3.2.*

2. Sometimes, like in the MLO characterization part of Theorem 3.2, we state our result only for Player I. However, in all these cases there is a duality between the players, and similar assertions hold for Player II. For every φ we can construct ψ such that Player I has a definable (respectively,

finite-state) winning strategy st in $\mathcal{G}^{\alpha}_{\psi}$ iff Player II has a definable (respectively, finite-state) winning strategy in $\mathcal{G}^{\alpha}_{\varphi}$. Moreover, this strategy is computable from (the description of) st.
3. To simplify notations, games and the Church problem were previously defined for formulas with two free variables X_1 and X_2. It is easy to generalize all definitions and results to formulas $\psi(X_1, \ldots, X_m, Y_1, \ldots Y_n)$ with many variables. In this generalization at round β, Player I chooses values for $X_1(\beta), \ldots, X_m(\beta)$, then Player II replies by choosing values for $Y_1(\beta), \ldots, Y_n(\beta)$. Note that, strictly speaking, the input to the Church problem is not only a formula, but a formula plus a partition of its free variables to Player I's variables and Player II's variables.
In [CH08] reachability games of ordinal length over finite graphs were considered. The next theorem reformulates results from [CH08] in logical terms.

Let $\vartheta(X_1, X_2)$ be a formula. Let $\vartheta^{<t}$ be the relativization of ϑ to the interval $[0, t)$, i.e., obtained from $\vartheta(X_1, X_2)$ by changing the first-order quantifiers $\exists y$ and $\forall y$ to $\exists^{<t} y$ and $\forall^{<t} y$. A *reachability* formula is a formula of the form $\exists t \vartheta^{<t}$. A *safety* formula is a formula of the form $\forall t \vartheta^{<t}$.

THEOREM 3.4 *Let φ be a reachability or safety formula. Then*
Finite-state strategy *The player who has a winning strategy in $\mathcal{G}^{\omega^\omega}_{\varphi}$ also has a finite-state winning strategy.*
Synthesis *We can compute a finite-state winning strategy for the winning player in $\mathcal{G}^{\omega^\omega}_{\varphi}$.*

4 Special Games on Types

In this section we introduce special games on types. These games play an important role in our proof that Dsynth(ω^ω) is computable. We reduce special games to safety games and derive that a winning player in these games has a definable winning strategy.

DEFINITION 4.1 (RESIDUAL) *Let $k \in \mathbb{N}$, $G \subseteq Char_2^k$ and $\tau \in Char_2^k$. Define $res_\tau(G)$ as $res_\tau(G) := \{\tau' \in Char_2^k \mid \tau + \tau' \in G\}$.*

Let F assign to every $\tau \in G$ a non-empty subset of $\mathcal{P}(res_\tau(G)) \setminus \{\emptyset\}$. The ω^ω-game on types, $Game(F, G)$, is defined as follows. There are ω^ω rounds.

Round 0: Player I sets $G_0 := G$. Player II chooses $\tau_0 \in G_0$.

Round α (for $\alpha > 0$): Let $\tau_{<\alpha} := \sum_{\beta\in\alpha} \tau_\beta$. If $\tau_{<\alpha} \notin G$, then Player II wins. Otherwise, Player I chooses $G_\alpha \in F(\tau_{<\alpha})$ and then Player II chooses $\tau_\alpha \in G_\alpha$.

Winning Conditions: Player I wins a play $G_0\tau_0 \ldots G_\beta\tau_\beta \ldots$ if $\sum_{\beta\in\alpha} \tau_\beta \in G$ for every $\alpha \leq \omega^\omega$.

The proof of the next proposition is based on a reduction of special games to safety games.

PROPOSITION 4.2 *There is an algorithm that given a game $Game(F, G)$, decides whether Player I has a winning strategy. Furthermore, if such a strategy exists, then there is definable winning strategy, and we can compute a formula $\psi(\bar{X}, \bar{Y})$ that defines in ω^ω a winning strategy for Player I.*

Since a strategy is definable in ω^ω iff it is finite-state, we can replace "definable" by "finite-state" in the above Proposition.

5 Main Results

In the next lemma and throughout this paper we often use $G \subseteq Char_2^k$ for φ defined as $\vee_{\tau\in G}\tau$. In particular, we use $\mathcal{G}^{\alpha}_{G}$, for the McNaughton game $\mathcal{G}^{\alpha}_{\varphi}$, and $win(G)$ for $win(\varphi)$, where $win(\varphi)$ is the sentence from Theorem 3.2.

LEMMA 5.1 (MAIN) *Let $G \subseteq Char_2^k$. The following are equivalent:*
1. *Player I has a definable winning strategy in $\mathcal{G}_G^{\omega^\omega}$.*
2. *There is $G' \subseteq G$ and a special game $Game(F, G')$ such that*
   (a) *$\omega^\omega \models win(G')$.*
   (b) *$\omega^\omega \models win(G_1)$ for every $\tau \in G'$ and $G_1 \in F(\tau)$.*
   (c) *Player I has a winning strategy in $Game(F, G')$.*

The implication (2)⇒(1) will be proved in Subsection 5.1. The implication (1)⇒(2) will be proved in Subsection 5.2. As a consequence, we obtain the computability of Dsynth(ω^ω).

THEOREM 5.2 (COMPUTABILITY OF DSYNTH(ω^ω)) *There is an algorithm that given a formula $\varphi(X_1, X_2)$ decides whether Player I has a definable winning strategy in the game $\mathcal{G}_\varphi^{\omega^\omega}$. Furthermore, if such a strategy exists we can compute a formula $\psi(X_1, X_2)$ that defines (in ω^ω) a winning strategy for Player I.*

PROOF. Since condition (2) of Lemma 5.1 is decidable, we obtain the decidability part of the theorem. The "furthermore part" of the theorem can be extracted from our proof of Lemma 5.1. ∎

In [Rab09] we provided reduction from Dsynth(α) to Dsynth(ω^ω). As a consequence of Theorem 5.2 and results in [Rab09] we obtain:

THEOREM 5.3 (COMPUTABILITY OF DSYNTH(α)) *1. There is an algorithm that given a formula $\varphi(X_1, X_2)$ computes a sentence $Dwin_\varphi$ such that for every countable ordinal $\alpha \geq \omega^\omega$: Player I has a definable (in α) winning strategy in $\mathcal{G}_\varphi^\alpha$ iff $\alpha \models Dwin_\varphi$.*
2. There is an algorithm that given a formula $\varphi(X_1, X_2)$ and the code of an ordinal α decides whether Player I has a definable winning strategy in $\mathcal{G}_\varphi^\alpha$, and if so, computes a formula ψ_α which defines in α such a strategy.

The next theorem states that the synthesis problem for finite-state strategies is computable. Its proof refines the proof of Theorem 5.3 and will be presented in the full paper.

THEOREM 5.4 (COMPUTABILITY OF FSYNTH(α)) *1. There is an algorithm that given a formula $\varphi(X_1, X_2)$ computes a sentence $Fswin_\varphi$ such that for every countable ordinal $\alpha \geq \omega^\omega$: Player I has a finite-state winning strategy in $\mathcal{G}_\varphi^\alpha$ if and only if $\alpha \models Fswin_\varphi$.*
2. There is an algorithm that given a formula $\varphi(X_1, X_2)$ and a code of α decides whether Player I has a finite-state winning strategy in $\mathcal{G}_\varphi^\alpha$, and if so, computes such a strategy.

5.1 Implication (2)⇒ (1) of Lemma 5.1

Terminology. (k-type of a play) For a (partial) play $\pi := (\pi_{X_1}(0), \pi_{X_2}(0)) \dots (\pi_{X_1}(\beta), \pi_{X_2}(\beta)) \dots$ $(\beta \in \alpha)$ its k-type is defined as the k-type of the chain $(\alpha, <, \pi_{X_1}, \pi_{X_2})$.

Let n be an upper bound on the quantifier depth of $win(H)$ for $H \subseteq Char_2^k$, where $win(H)$ is as in Theorem 3.2. By Lemma 2.3, we can compute m such that no sentence of the quantifier depth $\leq n$ can distinguish between multiples of ω^m.

¿From condition 2(a), and our choice of m, it follows that $\omega^m \models win(G')$ and therefore, by the synthesis part of Theorem 3.2, Player I has a definable winning strategy in $\mathcal{G}_{G'}^{\omega^m}$. We fix such a strategy $st_{G'}$. Similarly, condition 2(b) implies that for every $\tau \in G'$ and $G_1 \in F(\tau)$ Player I has a definable winning strategy in $\mathcal{G}_{G_1}^{\omega^m}$, we denote such a strategy by st_{G_1}. Condition 2(c) implies that Player I has a definable winning strategy st_F in $Game(F, G')$.

We organize our description of a winning strategy for $\mathcal{G}_G^{\omega^\omega}$ in sessions; each session is played for ω^m rounds. Each session "corresponds" to one round in $Game(F, G')$.

We show that this strategy wins G' on every multiple of ω^m.

Session 0: Play first ω^m rounds according to a definable winning strategy for $G_0 := G'$. Set τ_0 to be the k-type of the play during this session. Note that $\tau_0 \in G_0$ and this session corresponds to the play $\pi_0 := G_0\tau_0$ consistent with st_F in the game $Game(F, G')$.

Session α (for $\alpha > 0$): Let $\pi := G_0\tau_0, \ldots, G_\beta\tau_\beta \ldots$ (for $\beta < \alpha$) be the play of $Game(F, G')$ which corresponds to the previous sessions of the play.

Let G_α be defined as the response of st_F after π. Play the next ω^m rounds according to the winning strategy st_{G_α} in $\mathcal{G}_{G_\alpha}^{\omega^m}$.

Set τ_α to be the k-type of the play during this session. Note that $\tau_\alpha \in G_\alpha$ and the play $\pi G_\alpha \tau_\alpha$ is a play according to st_F.

It is clear that the above strategy is winning in $\mathcal{G}_{G'}^{\omega^\omega}$ and hence in $\mathcal{G}_G^{\omega^\omega}$.

It is easy to see that the above description of the strategy can be formalized in MLO.

5.2 Implication (1)⇒(2) of Lemma 5.1

DEFINITION 5.5 *Let $G \subseteq Char_2^k$. We say that a strategy* realizes *G on α if it wins $\mathcal{G}_G^\alpha$ and there is no $G_1 \subsetneq G$ such that it wins $\mathcal{G}_{G_1}^\alpha$.*

Note that for each k and a strategy st, the set $G \subseteq Char_2^k$ realized by st on α is unique. For every ψ and $G \subseteq Char_2^k$, there is a sentence $\mathrm{Realize}(\psi, G)$ such that for every α: ψ defines in α a strategy which realizes G iff $\alpha \models \mathrm{Realize}(\psi, G)$.

Assume that st defines a strategy and the quantifier depth of st is s. For $\tau \in Char_2^s$, let $st_\tau := \{\tau' \in Char_2^s \mid \tau + \tau' \to st\}$ be the residual of st wrt τ.

LEMMA 5.6 *Assume that st defines in ω^ω a strategy, its quantifier depth is s, and $\tau \in Char_2^s$.*

1. *If $st \wedge \tau$ is satisfiable, then st_τ defines in ω^ω a strategy.*
2. *If $\mathcal{M}_0 + \mathcal{M}_1 \models st$ and $type_s(\mathcal{M}_0) = \tau$ then $\mathcal{M}_1 \models st_\tau$.*
3. *If $\mathcal{M}_0 \models st$ and $type_s(\mathcal{M}_0) = \tau$ and $\mathcal{M}_1 \models st_\tau$, then $\mathcal{M}_0 + \mathcal{M}_1 \models st$.*
4. *If $\tau_\beta = type_s(\mathcal{M}_\beta)$, and $\mathcal{M}_0 \models st$ and $\mathcal{M}_\beta \models st_{\Sigma_{\gamma\in\beta}\tau_\gamma}$ for every $\beta \in (0, \alpha)$, then $\Sigma_{\beta\in[0,\alpha)}\mathcal{M}_\beta \models st$.*

For $k \in \mathbb{N}$ and a strategy st, we denote by $R(k, st)$ the subset of $Char_2^k$ realized by st on ω^ω. Define $F_{st}^k : R(k, st) \to \mathcal{P}(\mathcal{P}(Char_2^k)) \setminus \{\emptyset\}$ as follows:

$$F_{st}^k(\tau) := \{R(k, st_\delta) \mid \delta \in Char_2^s \text{ and } \delta \wedge st \wedge \tau \text{ is satisfiable on } \omega^\omega\}$$

The implication (1) ⇒ (2) of Lemma 5.1 immediately follows from the next lemma and the observation that st_δ wins $\mathcal{G}_{R(k,st_\delta)}^{\omega^\omega}$.

LEMMA 5.7 *Assume that st defines in ω^ω a strategy for Player I, and the quantifier depth of st is s. Then for every $k \leq s$, Player I has a winning strategy in $Game(F_{st}^k, R(k, st))$.*

PROOF. Let m be defined from $n := s + 2$ as in Lemma 2.3. In particular, st realizes $R(k, st)$ on every multiple of ω^m. Note that for $\delta \in Char_2^s$, the quantifier depth of st_δ is s. Therefore, st_δ realizes $R(k, st_\delta)$ on every multiple of ω^m.

We will describe a strategy for Player I and show that it is winning in $Game(F_{st}^k, R(k, st))$. Each round in this game corresponds to ω^m rounds in $\mathcal{G}_{R(k,st)}^{\omega^\omega}$. A play according to this strategy corresponds to a play according to the strategy st in $\mathcal{G}_{R(k,st)}^{\omega^\omega}$.

In addition to the description of the strategy we are going to define for each round α: $\delta_\alpha, v_\alpha \in Char_2^s$, and a play $\mathcal{M}_\alpha$ of length ω^m.

Round 0 Play $G_0 := R(k, st)$. Assume that Player II has replied by $\tau_0 \in G_0$ in round 0. Choose $v_1 = \delta_0 \in Char_2^s$ consistent with $\tau_0 \wedge st$ on ω^ω. Choose 2-chain $\mathcal{M}_0 := (\omega^m, X_1, X_2)$ such that $\mathcal{M}_0 \models \tau_0 \wedge \delta_0$. The structure $\mathcal{M}_0$ is a (partial) play, according to the strategy st.

Round α (for $\alpha > 0$) Assume that $\pi_{<\alpha} = G_0\tau_0 \dots G_\beta\tau_\beta \dots$ is the (partial) play up to round α and we have chosen $\delta_\beta \in Char_2^s$ at round $\beta < \alpha$.

Set $v_\alpha := \sum_{\beta\in\alpha} \delta_\beta$.

Play $G_\alpha := R(k, st_{v_\alpha})$.

Assume that Player II replies by $\tau_\alpha \in G_\alpha$ at this round.

Choose $\delta_\alpha \in Char_2^s$ to be consistent with $\tau_\alpha \wedge st_{v_\alpha}$.

Choose 2-chain $\mathcal{M}_\alpha := (\omega^m, X_1, X_2)$ such that $\mathcal{M}_\alpha \models \tau_\alpha \wedge \delta_\alpha \wedge st_{v_\alpha}$.

By the induction on α, using Lemma 5.6 and our choice of m, one can show that for every play $G_0\tau_0 \dots G_\beta\tau_\beta \dots$ which is consistent with the described strategy the following invariants hold:

1. $\delta_\alpha \wedge \tau_\alpha$ are satisfiable on ω^ω and therefore on ω^m.
2. $(\sum_{\beta\in\alpha} \delta_\beta) \wedge (\sum_{\beta\in\alpha} \tau_\beta)$ are satisfiable on ω^ω, and therefore on $\omega^m\alpha$.
3. $type_s(\sum_{\beta\in\alpha} \mathcal{M}_\beta) = \sum_{\beta\in\alpha} \delta_\beta = v_\alpha$
4. $\sum_{\beta\in\alpha} \mathcal{M}_\beta \models st$, i.e., the play $\sum_{\beta\in\alpha} \mathcal{M}_\beta$ is consistent with st.
5. $\sum_{\beta\in\alpha} \tau_\beta \in R(k, st)$.
6. $G_\alpha \in F^k_{st}(\sum_{\beta\in\alpha} \tau_\beta)$.

¿From (5)-(6) it follows that the described strategy is a winning strategy in $Game(F^k_{st}, R(k, st))$.

∎

6 Open Problems and Further Directions

The Büchi-Landweber theorem (Theorem 1.1) states that for the ω-games with MLO winning conditions, the player who has a winning strategy also has an MLO-definable winning strategy. In [RT07], we considered fragments of MLO logics. We proved that the Büchi-Landweber theorem fully extends to the first-order fragment of MLO (FOMLO) for ω-games; i.e., for every winning conditions $\varphi(X_1, X_2) \in$ FOMLO, the player who has a winning strategy in $\mathcal{G}^\omega_\varphi$, also has a FOMLO-definable winning strategy. We also proved that the theorem extends fully to the FOMLO extended by modular counting quantifiers.

In [RS08], we proved that for every ordinal $\alpha \geq \omega^\omega$ there is a FOMLO formula $\varphi_\alpha(X_1, X_2)$ such that Player I has a winning strategy in $\mathcal{G}^\alpha_{\varphi_\alpha}$; however, he has no MLO-definable winning strategy.

We plan to consider several fragments of MLO including FOMLO, FOMLO extended by the modular counting quantifiers and FOMLO extended by the quantifications over the finite sets (WMLO). For each of the above fragments $\mathcal{L}$ we address the problem of deciding for a formula $\varphi \in \mathcal{L}$ and an ordinal α, whether one of the player has $\mathcal{L}$-definable winning strategy in $\mathcal{G}^\alpha_\varphi$.

We reduced the synthesis problems to the satisfiability problem for MLO which has non-elementary complexity. We plan to analyze the complexity of the synthesis problems

when winning conditions are described by automata which have the same expressive power as MLO or by temporal logic formulas which have the same expressive power as FOMLO. For the winning conditions expressed in these formalisms we hope to prove that the synthesis problems have a reasonable complexity.

Acknowledgments

I am very grateful to Amit Shomrat for his insightful comments.

References

[BL69] J. R. Büchi, L. H. Landweber, Solving Sequential Conditions by Finite-State Strategies, Transactions of the AMS, Vol. 138 (Apr. 1969), pp. 295-311.

[BS73] J. R. Büchi, D. Siefkes, The Monadic Second-order Theory of all Countable Ordinals, Springer Lecture Notes 328 (1973), pp. 1-126.

[Ch63] A. Church, Logic, Arithmetic and Automata, Proc. Intrnat. Cong. Math. 1963, Almquist and Wilksells, Uppsala, 1963.

[CH08] J. Cristau and F. Horn. Graph Games on Ordinals. In Annual Conference on Foundations of Software Technology and Theoretical Computer Science - FSTTCS 2008, Dagstuhl Seminar Proceedings, vol. 08004, 2008.

[FV59] S. Feferman and R.L. Vaught (1959). The first-order properties of products of algebraic systems. *Fundamenta Mathematica* **47**:57–103.

[Gu85] Y. Gurevich, Monadic second-order theories, in: J. Barwise, S. Feferman (eds.), *Model-Theoretic Logics*, Springer-Verlag, 1985, pp. 479-506.

[Mc65] R. McNaughton, Finite-state infinite games, Project MAC Rep., MIT, Cambridge, Mass., Sept. 1965.

[Rab09] A. Rabinovich. The Church Problem for Countable Ordinals. Logical Methods in Computer Science, Vol 5(2), 2009.

[RT07] A. Rabinovich and W. Thomas. Logical Refinements of Church's Problem. In CSL 2007, Springer LNCS 4646, 69-83, 2007.

[RS08a] A. Rabinovich and A. Shomrat. Selection and Uniformization Problems in the Monadic Theory of Ordinals. In Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday. Springer LNCS 4800, pp. 571-588, 2008.

[RS08] A. Rabinovich and A. Shomrat. Selection in the Monadic Theory of Countable Ordinal. Journal of Symbolic Logic 73(3), pp. 783-816, 2008.

[Sh75] S. Shelah, The monadic theory of order, Annals of Mathematics, Ser. 2, Vol. 102 (1975), pp. 379-419.

[Th97] W. Thomas, Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. Volume 1261 of Lecture Notes in Computer Science, Springer, 1997, pp. 118-143.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

The Power of Depth 2 Circuits over Algebras

Chandan Saha[1], **Ramprasad Saptharishi**[2*] **and Nitin Saxena**[3]

[1]Indian Institute of Technology, Kanpur 208016, India
csaha@iitk.ac.in

[2]Chennai Mathematical Institute, Chennai 603103, India
ramprasad@cmi.ac.in

[3]Hausdorff Center for Mathematics, Bonn 53115, Germany
ns@hcm.uni-bonn.de

ABSTRACT.
We study the problem of polynomial identity testing (PIT) for depth 2 arithmetic circuits over matrix algebra. We show that identity testing of depth 3 ($\Sigma\Pi\Sigma$) arithmetic circuits over a field $\mathbb{F}$ is polynomial time equivalent to identity testing of depth 2 ($\Pi\Sigma$) arithmetic circuits over $U_2(\mathbb{F})$, the algebra of upper-triangular 2×2 matrices with entries from $\mathbb{F}$. Such a connection is a bit surprising since we also show that, as computational models, $\Pi\Sigma$ circuits over $U_2(\mathbb{F})$ are strictly 'weaker' than $\Sigma\Pi\Sigma$ circuits over $\mathbb{F}$. The equivalence further implies that PIT of $\Sigma\Pi\Sigma$ circuits reduces to PIT of width-2 commutative *Algebraic Branching Programs*(ABP). Further, we give a deterministic polynomial time identity testing algorithm for a $\Pi\Sigma$ circuit of size s over commutative algebras of dimension $O(\log s / \log\log s)$ over $\mathbb{F}$. Over commutative algebras of dimension $\text{poly}(s)$, we show that identity testing of $\Pi\Sigma$ circuits is at least as hard as that of $\Sigma\Pi\Sigma$ circuits over $\mathbb{F}$.

1 Introduction

Polynomial identity testing (PIT) is a fundamental problem in theoretical computer science. Over the last decade this problem has drawn significant attention from many leading researchers owing to its role in designing efficient algorithms and in proving circuit lower bounds. Identity testing is the following problem:

PROBLEM 1. *Given an arithmetic circuit C with input variables $x_1, \ldots, x_n$ and constants taken from a field $\mathbb{F}$, check if the polynomial computed by C is identically zero.*

Besides being a natural problem in algebraic computation, identity testing appears in important complexity theory results such as, IP = PSPACE [21] and the PCP theorem [6]. It also plays a promising role in proving super-polynomial circuit lower bound for permanent [13, 1]. Moreover, algorithms for problems like primality testing [3], graph matching [17] and multivariate polynomial interpolation [12] also involve identity testing. Several efficient randomized algorithms [20, 23, 10, 16, 2, 15] are known for identity testing. However, despite many attempts a deterministic polynomial time algorithm has remained elusive.

*Supported by MSR India PhD Fellowship

© Saha, Saptharishi, Saxena; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 371–382
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2333

Nevertheless, important progress has been made both in the designing of deterministic algorithms for special circuits, and in the understanding of why a general deterministic solution could be hard to get.

Assume that a circuit C has alternate layers of addition and multiplication gates. A layer of addition gates is denoted by Σ and that of multiplication gates is denoted by Π. Kayal and Saxena [14] gave a deterministic polynomial time identity testing algorithm for depth 3 ($\Sigma\Pi\Sigma$) circuits with constant top fan-in. As such, no other general polynomial time result is known for depth 3 circuits. A justification behind the hardness of PIT even for small depth circuits was provided by Agrawal and Vinay [4]. They showed that a deterministic black box identity test for depth 4 ($\Sigma\Pi\Sigma\Pi$) circuits implies a quasi-polynomial time deterministic PIT algorithm for circuits computing polynomials of *low* degree[†].

Thus, the non-trivial case for identity testing starts with depth 3 circuits; whereas circuits of depth 4 are *almost* the general case. At this point, it is natural to ask as to what is the complexity of the PIT problem for depth 2 ($\Pi\Sigma$) circuits if we allow the *constants* of the circuit to come from an *algebra*[‡] $\mathcal{R}$ that has dimension over $\mathbb{F}$, $\dim_{\mathbb{F}}(\mathcal{R}) > 1$. Can we relate this problem to the classical PIT problem for depth 3 and depth 4 circuits? In this paper, we address and answer this question. We assume that the algebra $\mathcal{R}$ is given in *basis form* i.e. we know an $\mathbb{F}$-basis $\{e_1, \ldots, e_k\}$ of $\mathcal{R}$ and we also know how $e_i e_j$ can be expressed in terms of the basis elements, for all i and j. Since elements of a finite dimensional algebra, given in basis form, can be expressed as matrices over $\mathbb{F}$, the problem at hand is the following.

PROBLEM 2. *Given an expression $P = \prod_{i=1}^{d} \sum_{j=0}^{n} A_{ij} x_j$ with $x_0 = 1$ and $A_{ij} \in \mathsf{M}_k(\mathbb{F})$, the algebra of $k \times k$ matrices over $\mathbb{F}$, check if P is zero using $\mathsf{poly}(n \cdot k \cdot d)$ many $\mathbb{F}$-operations.*

How hard is this problem? It is quite easy to verify that if we allow randomness then it is solvable just like the usual PIT problem (using Schwartz-Zippel test [20, 23]). So we are only interested in deterministic methods in this work.

Conventions - Whenever we say 'arithmetic circuit (or formula)' without an extra qualification, we mean a circuit (or formula) over a field. Otherwise, we explicitly mention 'arithmetic circuit (or formula) over *some* algebra' to mean that the constants of the circuit are taken from 'that' algebra. Also, by depth 3 and depth 2 circuits, we always mean $\Sigma\Pi\Sigma$ and $\Pi\Sigma$ circuits respectively. Further, we take $x_0 = 1$ throughout this paper.

1.1 The depth 2 model of computation

A depth 2 circuit C over matrices naturally defines a computational model. Assuming $\mathcal{R} = \mathsf{M}_k(\mathbb{F})$, for some k, a polynomial $P \in \mathcal{R}[x_1, \ldots, x_n]$ outputted by C can be viewed as a $k \times k$ matrix of polynomials in $\mathbb{F}[x_1, \ldots, x_n]$. We say that a polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ is *computed* by C if one of the k^2 polynomials in matrix P is f. Sometimes we say P *computes* f to mean the same. In the following discussion, we denote the algebra of upper-triangular $k \times k$ matrices by $\mathsf{U}_k(\mathbb{F})$. The algebra $\mathsf{U}_2(\mathbb{F})$ is the *smallest* non-commutative algebra with unity over $\mathbb{F}$, in the sense that $\dim_{\mathbb{F}} \mathsf{U}_2(\mathbb{F}) = 3$ and any algebra of smaller dimension is commutative. We show here that already $\mathsf{U}_2(\mathbb{F})$ captures an open case of identity testing.

[†] A polynomial is said to have low degree if its degree is less than the size of the circuit that computes it.

[‡] In this paper, an algebra is always a finite dimensional associative algebra with unity.

Ben-Or and Cleve [8] showed that a polynomial computed by an arithmetic formula E of depth d, and fan-in (of every gate) bounded by 2, can also be computed by a straight-line program of length at most 4^d using only 3 registers. The following fact can be readily derived from their result: From an arithmetic formula E of depth d and fan-in bounded by 2, we can efficiently compute the expression, $P = \prod_{i=1}^{m} \sum_{j=0}^{n} A_{ij} x_j$, where $m \leq 4^d$ and $A_{ij} \in \mathsf{M}_3(\mathbb{F})$ such that P *computes* the polynomial that E does. Thus solving Problem 2 in polynomial time even for 3×3 matrices yields a polynomial time algorithm for PIT of constant depth circuits, in particular depth 4 circuits. There is an alternative way of arguing that the choice of $\mathcal{R}$ as $\mathsf{M}_3(\mathbb{F})$ is *almost* the general case.

Given an arithmetic circuit of size s, computing a low degree polynomial, use the depth-reduction result by Allender, Jiao, Mahajan and Vinay [5] (see also [22]) to construct an equivalent bounded fan-in formula of size $s^{O(\log s)}$ and depth $O(\log^2 s)$. From this, obtain a depth 2 circuit over $\mathsf{M}_3(\mathbb{F})$ of size $4^{O(\log^2 s)} = s^{O(\log s)}$ (using Ben-Or and Cleve's result) that computes the same polynomial as the formula. Thus, derandomization of PIT for depth 2 circuits over 3×3 matrices yields a quasi-polynomial time PIT algorithm for any circuit computing a low degree polynomial. This means, in essence a depth 2 circuit over $\mathsf{M}_3(\mathbb{F})$ plays the role of a depth 4 circuit over $\mathbb{F}$ (in the spirit of Agrawal and Vinay's result).

It is natural to ask how the complexity of PIT for depth 2 circuits over $\mathsf{M}_2(\mathbb{F})$ relates to PIT for arithmetic circuits. In this paper, we provide an answer to this. We show a surprising connection between PIT of depth 2 circuits over $\mathsf{U}_2(\mathbb{F})$ and PIT of depth 3 circuits. The reason this is surprising is because we also show that, a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ is not even powerful enough to compute a simple polynomial like, $x_1x_2 + x_3x_4 + x_5x_6$!

Known related models

Identity testing and circuit lower bounds have been studied for different algebraic models. Nisan [18] showed an exponential lower bound on the size of any arithmetic formula computing the determinant of a matrix in the non-commutative *free algebra* model. The result was generalized by Chien and Sinclair [11] to a large class of non-commutative algebras satisfying polynomial identities, called PI-algebras. Identity testing has also been studied for the non-commutative model by Raz and Shpilka [19], Bogdanov and Wee [9], and Arvind, Mukhopadhyay and Srinivasan [7]. But unlike those models where the variables do not commute, in our setting the variables always commute but the *constant coefficients* are taken from an algebra $\mathcal{R}$. The motivation for studying this latter model (besides it being a natural generalization of circuits over fields) is that, it provides a different perspective to the complexity of the classical PIT problem in terms of the dimension of the underlying algebra. It seems to 'pack' the combinatorial nature of the circuit into a larger base algebra and hence opens up the possibility of using algebra structure results. The simplest nontrivial circuit in this model is a $\Pi\Sigma$ circuit over the non-commutative algebra $\mathcal{R} = \mathsf{U}_2(\mathbb{F})$, and even this, as we show, represents the frontier of our understanding.

1.2 Our Results

The results we give are of two types. Some are related to identity testing while the rest are related to the weakness of the depth 2 computational model over $\mathsf{U}_2(\mathbb{F})$ and $\mathsf{M}_2(\mathbb{F})$.

Identity testing

We show the following result.

THEOREM 3. *Identity testing for depth* 3 $(\Sigma\Pi\Sigma)$ *circuits is polynomial time equivalent to identity testing for depth* 2 $(\Pi\Sigma)$ *circuits over* $\mathsf{U}_2(\mathbb{F})$.

The above theorem has an interesting consequence on identity testing for Algebraic Branching Program (ABP) [18]. It is known that identity testing for non-commutative ABP can be done in deterministic polynomial time [19]. But no interesting result is known for identity testing of even width-2 commutative ABP's. The following result justifies this.

COROLLARY 4. *Identity testing of depth* 3 *circuits* $(\Sigma\Pi\Sigma)$ *reduces to that of width-*2 *ABPs.*

We mentioned before the prospect of using algebra structure results to solve PIT for depth 2 circuits over algebras. Our next result shows this idea at work for commutative algebras.

THEOREM 5. *Given an expression* $P = \prod_{i=1}^{d} \sum_{j=0}^{n} A_{ij}x_j$, *where* $A_{ij} \in \mathcal{R}$, *a commutative algebra of dimension* k *over* $\mathbb{F}$, *there is a deterministic algorithm to test if* P *is zero running in time* $\mathsf{poly}\left(k^k, n, d\right)$.

The above result gives a polynomial time algorithm for $k = O\left(\log s / \log\log s\right)$ where $s = O(nd)$. This result establishes that the power of depth 2 circuits over *small* algebras is primarily derived from the non-commutative nature of the algebra. However, we show that commutative algebras of polynomial dimension over $\mathbb{F}$ are much more powerful.

THEOREM 6. *Identity testing of depth* 3 $(\Sigma\Pi\Sigma)$ *circuits reduces to identity testing of depth* 2 $(\Pi\Sigma)$ *circuit* C *over a commutative algebra of dimension polynomial in the size of* C.

Our argument for proving Theorem 3 is relatively simple in nature. Perhaps the reason why such a connection was overlooked before is that, unlike a depth 2 circuit over $\mathsf{M}_3(\mathbb{F})$, we do not have the privilege of *exactly* computing a polynomial over $\mathbb{F}$ using a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$. Showing this weakness of the latter computational model constitutes the second part of our results.

Weakness of the depth 2 model over $\mathsf{U}_2(\mathbb{F})$ and $\mathsf{M}_2(\mathbb{F})$

We show that depth 2 circuits over $\mathsf{U}_2(\mathbb{F})$ are computationally weaker than depth 3 circuits.

THEOREM 7. *Let* $f \in \mathbb{F}[x_1, \ldots, x_n]$ *be a polynomial such that there are no two linear functions* l_1 *and* l_2 *(with* $1 \notin (l_1, l_2)$, *the ideal generated by* l_1 *and* l_2*) which make* $f \bmod (l_1, l_2)$ *also a linear function. Then* f *is not computable by a depth* 2 $(\Pi\Sigma)$ *circuit over* $\mathsf{U}_2(\mathbb{F})$.

Even a simple polynomial like $x_1x_2 + x_3x_4 + x_5x_6$ satisfies the condition stated in the above theorem, and so it is not computable by any depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$, no matter how large! This contrast makes Theorem 3 surprising as it establishes an equivalence of identity testing in two models of different computational strengths. We further show that the computational power of depth 2 circuits over $\mathsf{M}_2(\mathbb{F})$ is also severely restrictive. Let P_ℓ denote the partial product $P_\ell = \prod_{i=\ell}^{d} \sum_{j=0}^{n} A_{ij}x_j$, where $A_{ij} \in \mathsf{M}_2(\mathbb{F})$ and $1 \leq \ell \leq d$.

DEFINITION 8. *A polynomial f is computed by a depth 2 circuit ($\Pi\Sigma$) under a degree restriction of m if the degree of every partial product P_ℓ is bounded by m, for $1 \leq \ell \leq d$.*

THEOREM 9. *There exists a class of polynomials over $\mathbb{F}$ of degree n that cannot be computed by a depth 2 ($\Pi\Sigma$) circuit over $\mathsf{M}_2(\mathbb{F})$, under a degree restriction of n.*

The motivation behind imposing a condition like *degree restriction* comes naturally from depth 2 circuits over $\mathsf{M}_3(\mathbb{F})$. Given a polynomial $f = \sum_i m_i$, where m_i's are the monomials of f, it is easy to construct a depth 2 circuit over $\mathsf{M}_3(\mathbb{F})$ that literally forms these monomials and adds them up one by one. This computation is degree restricted, if we extend our definition of degree restriction to $\mathsf{M}_3(\mathbb{F})$. However, the above theorem shows that this simple scheme fails over $\mathsf{M}_2(\mathbb{F})$.

2 Identity testing over $\mathsf{M}_2(\mathbb{F})$

We show that PIT of depth 2 circuits over $\mathsf{M}_2(\mathbb{F})$ is at least as hard as PIT of depth 3 circuits. This implies that PIT of a width-2 commutative ABP is also 'harder' than the latter problem.

2.1 Equivalence with depth 3 identity testing

Given a depth 3 circuit, assume (without loss of generality) that the fan-in of the multiplication gates are the same. This multiplicative fan-in is referred to as the *degree* of the depth 3 circuit. For convenience, we call a matrix with linear functions as entries, a *linear* matrix.

LEMMA 10. *Let f be a polynomial over $\mathbb{F}$ computed by a depth 3 circuit C of degree d and top fan-in s. Given C, it is possible to efficiently construct a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ of size $O(ds^2)$ that computes $L \cdot f$, where L is a product of non-zero linear functions.*

PROOF. A depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ is simply a product sequence of 2×2 upper-triangular linear matrices. We show that there exists such a sequence of length $O(ds^2)$ such that the product 2×2 matrix has $L \cdot f$ as one of its entries. Since f is computed by a depth 3 circuit, $f = \sum_{i=1}^{s} P_i$, where each summand $P_i = \prod_j l_{ij}$ is a product of linear functions. Observe that a single P_i can be computed using the following product sequence of length d.

$$\begin{bmatrix} l_{i1} & \\ & 1 \end{bmatrix} \cdots \begin{bmatrix} l_{i(d-1)} & \\ & 1 \end{bmatrix} \begin{bmatrix} 1 & l_{id} \\ & 1 \end{bmatrix} = \begin{bmatrix} L' & P_i \\ & 1 \end{bmatrix}, \quad \text{where } L' = l_{i1} \cdots l_{i(d-1)}. \qquad (1)$$

The proof proceeds through induction, where Equation 1 serves as the induction basis. A generic intermediate matrix looks like $\begin{bmatrix} L_1 & L_2 g \\ & L_3 \end{bmatrix}$, where each L_i is a product of non-zero linear functions and g is a partial sum of the P_i's. Inductively double the number of summands in g as follows.

At the i-th iteration, suppose we have the matrices $\begin{bmatrix} L_1 & L_2 g \\ & L_3 \end{bmatrix}$ and $\begin{bmatrix} M_1 & M_2 h \\ & M_3 \end{bmatrix}$, each computed by a sequence of n_i linear matrices. We want a sequence that computes a polynomial of the form $L \cdot (g + h)$. Consider the following sequence,

$$\begin{bmatrix} L_1 & L_2 g \\ & L_3 \end{bmatrix} \begin{bmatrix} A & \\ & B \end{bmatrix} \begin{bmatrix} M_1 & M_2 h \\ & M_3 \end{bmatrix} = \begin{bmatrix} AL_1M_1 & AL_1M_2h + BL_2M_3g \\ & BL_3M_3 \end{bmatrix}, \qquad (2)$$

where A, B are products of linear functions. By setting $A = L_2M_3$ and $B = L_1M_2$ we have,

$$\begin{bmatrix} L_1 & L_2g \\ & L_3 \end{bmatrix} \begin{bmatrix} A & \\ & B \end{bmatrix} \begin{bmatrix} M_1 & M_2h \\ & M_3 \end{bmatrix} = \begin{bmatrix} L_1L_2M_1M_3 & L_1L_2M_2M_3(g+h) \\ & L_1L_3M_2M_3 \end{bmatrix}.$$

This way, we have doubled the number of summands in $g + h$. By induction, each L_i and M_i is a product of n_i linear functions. Therefore, the matrix $\begin{bmatrix} A & \\ & B \end{bmatrix}$ is a product of at most $2n_i$ diagonal linear matrices and the length of the sequence given in Equation 2 is bounded by $4n_i$. This process of doubling the summands needs to be repeated at most $\log s + 1$ times and so the length of the final product sequence is bounded by $d \cdot 4^{\log s} = ds^2$. ∎

PROOF. [Theorem 3] Given a depth 3 circuit computing f we can construct a depth 2 circuit D over $\mathsf{U}_2(\mathbb{F})$ that computes $L \cdot f$. The output of D can be projected appropriately so that we may assume that D outputs the matrix $\begin{bmatrix} 0 & L \cdot f \\ & 0 \end{bmatrix}$, which is zero if and only if f is zero.

To see the other direction of the equivalence, observe that the off-diagonal entry of the output of any depth 2 circuit D over $\mathsf{U}_2(\mathbb{F})$ is a sum of at most d' products of linear functions, where d' is the multiplicative fan-in of D. ∎

2.2 Width-2 algebraic branching programs

Algebraic Branching Program (ABP) is a model of computation introduced by Nisan [18].

DEFINITION 11. *An ABP is a directed acyclic graph with a source and a sink. The vertices of this graph are partitioned into levels, where edges go from level i to level $i+1$, with the source at the first level and the sink at the last level. Each edge is labelled with a homogeneous linear function of $x_1, \ldots, x_n$. The* width *of the ABP is the maximum number of vertices at any level. An ABP computes a function by summing over all paths from source to sink, the product of all linear functions by which the edges of the path are labelled.*

PROOF. [Corollary 4] In Theorem 3 we have constructed a depth 2 circuit D that computes $P = \prod_i \sum_j A_{ij}x_j$, where each $A_{ij} \in \mathsf{U}_2(\mathbb{F})$. We can make D homogeneous by introducing an extra variable z, such that $P = \prod_i (A_{i0}z + A_{i1}x_1 + \ldots + A_{in}x_n)$. By making the i^{th} linear matrix in the sequence act as the biadjacency matrix between level i and $i+1$ of the ABP, we have a width-2 ABP computing the same polynomial. ∎

3 Identity testing over commutative algebras

The main idea behind the proof of Theorem 5 is a structure theorem for finite dimensional commutative algebras involving *local rings*.

DEFINITION 12. *A ring $\mathcal{R}$ is* local *if it has a unique maximal ideal.*

In a local ring the unique maximal ideal consists of all non-units in $\mathcal{R}$. The following theorem shows how a commutative algebra decomposes into local sub-algebras. The theorem is quite well known in the theory of commutative algebras. But, as we need an effective version of this theorem, we present an appropriate proof here.

THEOREM 13. *A finite dimensional commutative algebra $\mathcal{R}$ is isomorphic to a direct sum of local rings, i.e. $\mathcal{R} \cong \oplus_{i=1}^{\ell} \mathcal{R}_i$, where $\mathcal{R}_i$ is a local ring and any non-unit in $\mathcal{R}_i$ is nilpotent.*

PROOF. If all non-units in $\mathcal{R}$ are nilpotents then $\mathcal{R}$ is a local ring and the set of nilpotents forms the unique maximal ideal. Suppose, there is a non-nilpotent non-unit z in $\mathcal{R}$. (Any non-unit z in a finite dimensional algebra is a zero-divisor i.e. $\exists y \in \mathcal{R}$ and $y \neq 0$ such that $yz = 0$.) We will later show that using z it is possible to find an *idempotent* $v \notin \{0, 1\}$ (i.e. $v^2 = v$) in $\mathcal{R}$. But at first, let us see what happens if we already have a non-trivial idempotent $v \in \mathcal{R}$. Let $\mathcal{R}v$ be the sub-algebra of $\mathcal{R}$ generated by multiplying elements of $\mathcal{R}$ with v. Since any $a = av + a(1 - v)$ and for any $b \in \mathcal{R}v$ and $c \in \mathcal{R}(1 - v)$, $b \cdot c = 0$, we get $\mathcal{R} \cong \mathcal{R}v \oplus \mathcal{R}(1 - v)$ as a non-trivial decomposition of $\mathcal{R}$. By repeating the splitting process on the sub-algebras we can eventually prove the theorem.

Now we show how to find an idempotent from a zero-divisor z. An element $a \in \mathcal{R}$ can be equivalently expressed as a matrix in $\mathsf{M}_k(\mathbb{F})$, where $k = \dim_{\mathbb{F}}(\mathcal{R})$, by treating a as the linear transformation on $\mathcal{R}$ that takes $b \in \mathcal{R}$ to $a \cdot b$. Therefore, z is a zero-divisor if and only if z as a matrix is singular. Consider the Jordan normal form of z. Since it is merely a change of basis we can assume that z is already in Jordan normal form. (We will not compute the Jordan normal form in our algorithm, it is used only for the sake of argument.) Let, $z = \begin{bmatrix} A & 0 \\ 0 & N \end{bmatrix}$, where A, N are block diagonal matrices and A is non-singular and N is nilpotent. Then, $w = z^k = \begin{bmatrix} B & 0 \\ 0 & 0 \end{bmatrix}$, where $B = A^k$ is non-singular. The claim is, there is an identity element in the sub-algebra $\mathcal{R}w$ which can be taken to be the idempotent v that splits $\mathcal{R}$. First observe that the minimum polynomial of w is $m(x) = x \cdot m'(x)$, where $m'(x)$ is the minimum polynomial of B. Also if $m(x) = \sum_{i=1}^{k} \alpha_i x^i$ then $\alpha_1 \neq 0$ as it is the constant term of $m'(x)$ and B is non-singular. Therefore, there exists an $a \in \mathcal{R}$ such that $w \cdot (aw - 1) = 0$. Hence $v = aw$ is the identity element of $\mathcal{R}w$ and is also an idempotent in $\mathcal{R}$. ∎

We are now ready to prove Theorem 5.

PROOF. [Theorem 5] Let $\{e_1, \ldots, e_k\}$ be a basis of $\mathcal{R}$ over $\mathbb{F}$. As argued before, any element in $\mathcal{R}$ can be equivalently expressed as a $k \times k$ matrix over $\mathbb{F}$. Hence, assume that $A_{ij} \in \mathsf{M}_k(\mathbb{F})$, for all i and j. Since $\mathcal{R}$ is given in basis form, the matrix representations of A_{ij}'s can be found efficiently. If every A_{ij} is non-singular, then surely $P \neq 0$. So, assume that $\exists A_{ij} = z$ such that z is a zero-divisor i.e. singular. From the proof of Theorem 13 it follows that the sub-algebra $\mathcal{R}w$, where $w = z^k$, contains an identity element v which is an idempotent. The idempotent v can be found by solving a system of linear equations over $\mathbb{F}$. Let $b_1, \ldots, b_{k'}$ be a basis of $\mathcal{R}w$, which can be easily computed from the elements $e_1 w, \ldots, e_k w$. Express v as, $v = \sum_{j=1}^{k'} \nu_j b_j$, where $\nu_j \in \mathbb{F}$ are unknowns. Since v is an identity in $\mathcal{R}w$ it satisfies the relation, $\sum_{j=1}^{k'} \nu_j b_j \cdot b_i = b_i$, for $1 \leq i \leq k'$. Expressing each b_i in terms of $e_1, \ldots, e_k$, we get a system of linear equations in the ν_j's. Find v by solving this linear system.

Since $\mathcal{R} \cong \mathcal{R}v \oplus \mathcal{R}(1 - v)$, we can split the identity testing problem into two subproblems. That is, P is zero if and only if, $Pv \in \mathcal{R}v$ and $P(1 - v) \in \mathcal{R}(1 - v)$ are both zero. Now apply the above process, recursively, on Pv and $P(1 - v)$. By decomposing the algebra each time an A_{ij} is a non-nilpotent zero-divisor, we are finally left with the easier problem

of checking if, $P = \prod_{i=1}^{d} \left(\sum_{j=0}^{n} A_{ij} x_j \right)$ is zero, where the coefficients A_{ij}'s are either nilpotent or invertible matrices. It is not hard to see that such a P is zero, if and only if the product of all those terms for which all the coefficients are nilpotent matrices is zero. If the number of such terms is greater than k then P is automatically zero (this follows from the fact that commuting nilpotent matrices can be simultaneously triangularized).

Otherwise, treat each term $\sum_{j=0}^{n} A_{ij} x_j$ as a $k \times k$ linear matrix. Since, there are at most k such linear matrices in P, the total number of linear functions occurring as entries of these linear matrices is bounded by k^3. Using a basis of these linear functions we can reduce the number of effective variables in P to k^3. Now, checking if P is zero takes only $\mathsf{poly}(k^k)$ field operations and hence the overall time complexity is bounded by $\mathsf{poly}(k^k, n, d)$. ∎

Thus, PIT of depth 2 circuits over finite dimensional commutative algebras reduces in polynomial time to that over local rings. If dimensions of these local rings are small we have an efficient algorithm. But what happens for much larger dimensions?

THEOREM 6. *Given a depth* 3 $(\Sigma\Pi\Sigma)$ *circuit* C *of degree* d *and top level fan-in* s*, it is possible to construct in polynomial time a depth* 2 $(\Pi\Sigma)$ *circuit* $\tilde{C}$ *over a local ring of dimension* $s(d-1)+2$ *over* $\mathbb{F}$ *such that* $\tilde{C}$ *computes a zero polynomial if and only if* C *does so.*

PROOF. Consider a depth 3 circuit computing a polynomial $f = \sum_{i=1}^{s} \prod_{j=1}^{d} l_{ij}$, where l_{ij}'s are linear functions. Consider the ring $\mathcal{R} = \mathbb{F}[y_1, \ldots, y_s]/\mathcal{I}$, where $\mathcal{I}$ is an ideal generated by the elements $\{y_i y_j\}_{1 \le i < j \le s}$ and $\{y_1^d - y_i^d\}_{1 < i \le s}$. Observe that $\mathcal{R}$ is a local ring, as $y_i^{d+1} = 0$ for all $1 \le i \le s$. The elements $\{1, y_1, \ldots, y_1^d, y_2, \ldots, y_2^{d-1}, \ldots, y_s, \ldots, y_s^{d-1}\}$ form an $\mathbb{F}$-basis of $\mathcal{R}$. Notice that the polynomial, $P = \prod_{j=1}^{d} \sum_{i=1}^{s} l_{ij} y_i = f \cdot y_1^d$ is zero if and only if f is zero. Polynomial P can indeed be computed by a depth 2 circuit over $\mathcal{R}$. ∎

4 Weakness of the depth 2 model

In Lemma 10, we have constructed a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ that computes $L \cdot f$ instead of f. Is it possible to drop the factor L and simply compute f? In this section, we show that in *many* cases it is impossible to find a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ that computes f.

4.1 Depth 2 model over $\mathsf{U}_2(\mathbb{F})$

The ideal of $\mathbb{F}[x_1, \ldots, x_n]$ generated by two linear functions l_1 and l_2 is denoted by (l_1, l_2). We say that l_1 is *independent* of l_2 if $1 \notin (l_1, l_2)$. Let f be a polynomial such that there are no two independent linear functions l_1 and l_2 which make $f \bmod (l_1, l_2)$ also a linear function.

PROOF. [Theorem 7] Assume on the contrary that f can be computed by a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$. That is, there is a product sequence $M_1 \cdots M_t$ of 2×2 upper-triangular linear matrices such that f is the top-right entry of the product matrix. Let $M_i = \begin{bmatrix} l_{i1} & l_{i2} \\ & l_{i3} \end{bmatrix}$, then

$$f = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} l_{11} & l_{12} \\ & l_{13} \end{bmatrix} \begin{bmatrix} l_{21} & l_{22} \\ & l_{23} \end{bmatrix} \cdots \begin{bmatrix} l_{t1} & l_{t2} \\ & l_{t3} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{3}$$

Case 1**:** Not all the l_{i1}'s are constants. Let k be the smallest such that l_{k1} is not a constant and $l_{i1} = c_i$ for all $i < k$. Let $[B\ L]^T = M_{k+1} \cdots M_t \cdot [0\ 1]^T$ and $[d_i\ D_i] = [1\ 0] \cdot M_1 \cdots M_{i-1}$. Observe that L is just a product of linear functions, and for all $1 \leq i < k$, we have the relations, $d_{i+1} = \prod_{j=1}^{i} c_j$ and $D_{i+1} = d_i l_{i2} + l_{i3} D_i$. Hence, Equation 3 simplifies as,

$$f = \begin{bmatrix} d_k & D_k \end{bmatrix} \begin{bmatrix} l_{k1} & l_{k2} \\ & l_{k3} \end{bmatrix} \begin{bmatrix} B \\ L \end{bmatrix} = d_k l_{k1} B + (d_k l_{k2} + l_{k3} D_k)\, L.$$

Suppose there is some factor l of L with $1 \notin (l_{k1}, l)$. Then $f = 0 \bmod (l_{k1}, l)$, which is not possible. Hence, L must be a constant modulo l_{k1}. For appropriate constants α, β, we have

$$f = \alpha l_{k2} + \beta l_{k3} D_k \pmod{l_{k1}}. \tag{4}$$

By inducting on k, we argue that the above relation can not be true. If l_{k3} was independent of l_{k1}, then $f = \alpha l_{k2} \bmod (l_{k1}, l_{k3})$ which is not possible. Therefore, l_{k3} must be a constant modulo l_{k1}. We then have the following (reusing α and β to denote appropriate constants):

$$\begin{aligned} f &= \alpha l_{k2} + \beta D_k \pmod{l_{k1}} \\ &= \alpha l_{k2} + \beta \left(d_{k-1} l_{(k-1)2} + l_{(k-1)3} D_{k-1} \right) \pmod{l_{k1}} \\ \implies f &= \left(\alpha l_{k2} + \beta d_{k-1} l_{(k-1)2} \right) + \beta l_{(k-1)3} D_{k-1} \pmod{l_{k1}}. \end{aligned}$$

The last equation can be rewritten in the form of Equation 4 with the term $\beta l_{k3} D_k$ replaced by $\beta l_{(k-1)3} D_{k-1}$. Notice that the expression $\left(\alpha l_{k2} + \beta d_{k-1} l_{(k-1)2} \right)$ is linear just like αl_{k2}. Hence, by using the argument iteratively we eventually get a contradiction at D_1.

Case 2**:** All the l_{i1}'s are constants. In this case, $f = d_t l_{t2} + l_{t3} D_t$. This relation is again of the form in Equation 4 (without the mod term) and so the same argument can be repeated. ∎

Some explicit examples of functions that cannot be computed are the following.

COROLLARY 14. *A depth* 2 *circuit over* $\mathsf{U}_2(\mathbb{F})$ *cannot compute the polynomial* $x_1x_2 + x_3x_4 + x_5x_6$*. Other examples include functions like the determinant and permanent polynomials.*

4.2 Depth 2 model over $\mathsf{M}_2(\mathbb{F})$

The power of depth 2 circuits is very restrictive even if the underlying algebra is $\mathsf{M}_2(\mathbb{F})$.

DEFINITION 15. *A polynomial f is said to be* r-robust *if f does not belong to any ideal generated by r linear forms. (A homogeneous linear function is called a* linear form*.)*

For instance, it can be checked that det_n and perm_n, the symbolic determinant and permanent of an $n \times n$ matrix, are $(n-1)$-robust polynomials. For any polynomial f, denote the d^{th} homogeneous part of f by $[f]_d$. Recall the definition of *degree restriction* (Definition 8).

THEOREM 16. *A polynomial f of degree n, such that $[f]_n$ is* 5*-robust, cannot be computed by a depth* 2 $(\Pi\Sigma)$ *circuit over* $\mathsf{M}_2(\mathbb{F})$ *under a degree restriction of n.*

We prove this with the help of the following lemma.

LEMMA 17. *Let f_1 be a polynomial of degree n such that $[f_1]_n$ is 4-robust. Suppose there is a linear matrix M and polynomials f_2, g_1, g_2 of degree at most n satisfying $[f_1\, f_2]^T = M \cdot [g_1\, g_2]^T$. Then, there is an appropriate invertible column operation A such that $M \cdot A = \begin{bmatrix} 1 & h_2 \\ c_3 & h_4 + c_4 \end{bmatrix}$, where c_3, c_4 are constants and h_2, h_4 are linear forms.*

We defer the proof of this lemma to the end of this section.

PROOF. [Theorem 16] Assume that there is such a sequence of matrices computing f. Without loss of generality, let the first matrix in the sequence be a row vector $\bar{v}$ and the last matrix be a column vector $\bar{w}$. Let $f = \bar{v} \cdot M_1 M_2 \cdots M_d \cdot \bar{w}$ be a sequence of minimum length computing f. Using Lemma 17, we repeatedly transform this sequence, replacing the term $M_i M_{i+1}$ by $(M_i A)(A^{-1} M_{i+1})$ for an appropriate invertible column transformation A. To begin, let $\bar{v} = [l_1\, l_2]$ for two linear functions l_1 and l_2, and $[f_1\, f_2]^T = M_1 \cdots M_d \bar{w}$. Then, $[f\ 0]^T = \begin{bmatrix} l_1 & l_2 \\ 0 & 0 \end{bmatrix} \cdot [f_1\, f_2]^T$. Applying Lemma 17, we can assume that $\bar{v} = [1\, h]$ and so $f = f_1 + h f_2$. Also, $h \neq 0$, by the minimality of the sequence. This forces $[f_1]_n = [f]_n$ to be 4-robust and the degree restriction makes $[f_2]_n = 0$.

Let $[g_1\, g_2]^T = M_2 \cdots M_d \bar{w}$. The goal is to translate the properties that $[f_1]_n$ is 4-robust and $[f_2]_n = 0$, to $[g_1]_n$ and $[g_2]_n$ respectively. We use induction and translate these properties to the vectors $M_i \cdots M_d \bar{w}$, for all $i \geq 2$. So, suppose that the relation, $[f_1\, f_2]^T = M_i \cdot [g_1\, g_2]^T$, holds in general for some i, where $[f_1]_n$ is 4-robust and $[f_2]_n = 0$.

Since $[f_1]_n$ is 4-robust, using Lemma 17 again, we can assume that

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} 1 & h_2 \\ c_3 & c_4 + h_4 \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \tag{5}$$

by reusing the symbols g_1, g_2. Observe that in the above equation if $h_4 = 0$ then $M_{i-1} M_i$ still continues to be a linear matrix (since, by induction, M_{i-1} is of the form as dictated by Lemma 17) and that would contradict the minimality of the sequence. Therefore $h_4 \neq 0$.

We claim that, in Equation 5, $c_3 = 0$. As $h_4 \neq 0$, the degree restriction forces $[g_2]_n = 0$. And since $[f_2]_n = 0$, we have the relation $c_3 [g_1]_n = -h_4 [g_2]_{n-1}$. If $c_3 \neq 0$, we have $[g_1]_n \in (h_4)$, contradicting 4-robustness of $[f_1]_n$ as then $[f_1]_n = [g_1]_n + h_2 [g_2]_{n-1} \in (h_2, h_4)$.

From the relations, $[f_2]_n = 0$, $c_3 = 0$ and $h_4 \neq 0$, it follow that $[g_2]_{n-1} = 0$. Hence, $[g_1]_n = [f_1]_n$ is 4-robust. Thus, we have translated the properties to $[g_1\, g_2]^T$, showing that $[g_1]_n$ is 4-robust and $[g_2]_n = 0$. However, since the sequence is finite, there must come a point when degree of g_1 in $[g_1\, g_2]^T = M_i \cdots M_d \bar{w}$ drops below n for some $i \geq 2$. At this point we get a contradiction. ∎

PROOF. [Lemma 17] Suppose we have the equation,

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} h_1 + c_1 & h_2 + c_2 \\ h_3 + c_3 & h_4 + c_4 \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \tag{6}$$

where $c_1, \ldots, c_4$ are constants and $h_1, \ldots, h_4$ are linear forms. On comparing degree $n+1$ terms, we have the relations, $h_1 [g_1]_n + h_2 [g_2]_n = 0$ and $h_3 [g_1]_n + h_4 [g_2]_n = 0$. If h_3 and h_4 (a similar reasoning holds for h_1 and h_2) are not proportional (i.e. not multiple of each other),

then $[g_1]_n, [g_2]_n \in (h_3, h_4)$. But this implies that, $[f_1]_n = h_1[g_1]_{n-1} + h_2[g_2]_{n-1} + c_1[g_1]_n + c_2[g_2]_n \in (h_1, h_2, h_3, h_4)$, contradicting the 4-robustness of $[f_1]_n$. Thus, h_3 and h_4 (as well as h_1 and h_2) are proportional, in the same ratio as $-[g_2]_n$ and $[g_1]_n$. Using an appropriate column operation, Equation 6 simplifies to $\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} c_1 & h_2 + c_2 \\ c_3 & h_4 + c_4 \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}$, reusing symbols g_1, g_2 and others. If $c_1 = [g_2]_n = 0$ then $[f_1]_n = h_2[g_2]_{n-1}$, contradicting robustness. Therefore, either $c_1 \neq 0$, in which case another column transformation gets the matrix to the form claimed, or $[g_2]_n \neq 0$ implying that $h_2 = h_4 = 0$. But then c_1 and c_2 both cannot be zero, $[f_1]_n$ being 4-robust, and hence a column transformation yields the desired form. ∎

5 Concluding remarks

We give a new perspective to identity testing of depth 3 arithmetic circuits by showing an equivalence to identity testing of depth 2 circuits over $\mathsf{U}_2(\mathbb{F})$. We also give a deterministic polynomial time identity testing algorithm for depth 2 circuits over commutative algebras of small dimension. Our algorithm crucially exploits an interesting structural result involving local rings. This naturally poses the following question - Can we use more algebraic insight on non-commutative algebras to solve the general problem? In fact, we have a specific non-commutative algebra in mind. The question is - Is it possible to use properties very specific to the ring of upper-triangular 2×2 matrices to solve PIT for depth 3 circuits?

Acknowledgement

This work was started when the first author visited Hausdorff Center for Mathematics, Bonn. We thank Marek Karpinski for the generous hospitality and several discussions. We also thank Manindra Agrawal for several insightful discussions on this work. And finally thanks to V Vinay for many useful comments on the first draft of this paper.

References

[1] Manindra Agrawal. Proving Lower Bounds Via Pseudo-random Generators. In *FSTTCS*, pages 92–105, 2005.

[2] Manindra Agrawal and Somenath Biswas. Primality and Identity Testing via Chinese Remaindering. In *FOCS*, pages 202–209, 1999.

[3] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Ann. of Math*, 160(2):781–793, 2004.

[4] Manindra Agrawal and V Vinay. Arithmetic circuits: A chasm at depth four. In *FOCS*, pages 67–75, 2008.

[5] Eric Allender, Jia Jiao, Meena Mahajan, and V. Vinay. Non-Commutative Arithmetic Circuits: Depth Reduction and Size Lower Bounds. *Theor. Comput. Sci.*, 209(1-2):47–86, 1998.

[6] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof Verification and the Hardness of Approximation Problems. *Journal of the ACM*, 45(3):501–555, 1998.

[7] Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. New results on noncommutative and commutative polynomial identity testing. In *IEEE Conference on Computational Complexity*, pages 268–279, 2008.

[8] Michael Ben-Or and Richard Cleve. Computing Algebraic Formulas Using a Constant Number of Registers. In *STOC*, pages 254–257, 1988.

[9] Andrej Bogdanov and Hoeteck Wee. More on noncommutative polynomial identity testing. In *CCC*, pages 92–99, 2005.

[10] Zhi-Zhong Chen and Ming-Yang Kao. Reducing Randomness via Irrational Numbers. In *STOC*, pages 200–209, 1997.

[11] Steve Chien and Alistair Sinclair. Algebras with polynomial identities and computing the determinant. In *FOCS*, pages 352–361, 2004.

[12] Michael Clausen, Andreas W. M. Dress, Johannes Grabmeier, and Marek Karpinski. On Zero-Testing and Interpolation of k-Sparse Multivariate Polynomials Over Finite Fields. *Theor. Comput. Sci.*, 84(2):151–164, 1991.

[13] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *STOC*, pages 355–364, 2003.

[14] Neeraj Kayal and Nitin Saxena. Polynomial Identity Testing for Depth 3 Circuits. *Computational Complexity*, 16(2), 2007.

[15] Adam Klivans and Daniel A. Spielman. Randomness efficient identity testing of multivariate polynomials. In *STOC*, pages 216–223, 2001.

[16] Daniel Lewin and Salil P. Vadhan. Checking Polynomial Identities over any Field: Towards a Derandomization? In *STOC*, pages 438–447, 1998.

[17] László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574, 1979.

[18] Noam Nisan. Lower bounds for non-commutative computation. In *STOC*, pages 410–418, 1991.

[19] Ran Raz and Amir Shpilka. Deterministic Polynomial Identity Testing in Non-Commutative Models. In *CCC*, pages 215–222, 2004.

[20] Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4):701–717, 1980.

[21] Adi Shamir. IP=PSPACE. In *FOCS*, pages 11–15, 1990.

[22] Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983.

[23] Richard Zippel. Probabilistic algorithms for sparse polynomials. *EUROSAM*, pages 216–226, 1979.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Deductive Verification of Continuous Dynamical Systems

Ankur Taly[1], **Ashish Tiwari**[2]*

[1] Computer Science Dept., Stanford University
ataly@stanford.edu

[2] SRI International, Menlo Park, CA 94025
tiwari@csl.sri.com

ABSTRACT. We define the notion of inductive invariants for continuous dynamical systems and use it to present inference rules for safety verification of polynomial continuous dynamical systems. We present two different sound and complete inference rules, but neither of these rules can be effectively applied. We then present several simpler and practical inference rules that are sound and relatively complete for different classes of inductive invariants. The simpler inference rules can be effectively checked when all involved sets are semi-algebraic.

1 Introduction

The deductive rule for safety verification of sequential and concurrent programs was an important milestone in the field of formal program verification [6, 10, 12]. A program can be proved safe by constructing an inductive invariant that is strong enough to prove safety. Programs can be formally viewed as discrete state transition systems. If the predicate $t(\vec{x}, \vec{y})$ states that there is a discrete transition from the state $\vec{x}$ to the state $\vec{y}$ in the discrete state transition system $\mathtt{DTS}$, and if $\mathtt{Init}$ and $\mathtt{Safe}$ are, respectively, the initial states of $\mathtt{DTS}$ and the hypothesized safe set, then the classical inference rule for safety verification is given as follows:

$$\begin{array}{llrcl}
(1) & \forall \vec{x} & \vec{x} \in \mathtt{Init} & \Rightarrow & \vec{x} \in \mathtt{Inv} \\
(2) & \forall \vec{x}, \vec{y} & \vec{x} \in \mathtt{Inv} \wedge t(\vec{x}, \vec{y}) & \Rightarrow & \vec{y} \in \mathtt{Inv} \\
(3) & \forall \vec{x} & \vec{x} \in \mathtt{Inv} & \Rightarrow & \vec{x} \in \mathtt{Safe} \\
\hline
 & & \mathtt{Reach}(\mathtt{DTS}) & \subseteq & \mathtt{Safe}
\end{array}$$

This rule essentially says that we can prove that all reachable states of $\mathtt{DTS}$ lie inside the safe set $\mathtt{Safe}$ by finding a suitable "inductive invariant" $\mathtt{Inv}$.

A valuable property of the deductive verification rule is that it is both sound and complete. Soundness here means that if a program is proved correct using the rule, then that program indeed satisfies the safety property. Completeness means that if the given program is actually safe, then there is an inductive invariant $\mathtt{Inv}$ that satisfies the Conditions (1), (2) and (3) of the deductive verification rule. The above rule, however, applies only to discrete state transition systems where the "next" states can be effectively specified.

While *discrete state transition systems* is a powerful modeling formalism, it is inadequate for modeling systems that involve physical components. Physical systems are typically

*Research supported in part by NSF grants CNS-0720721 and CSR-0917398 and NASA grant NNX08AB95A.

© Taly, Tiwari; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 383–394
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2334

modeled using differential equations as *continuous dynamical systems*. The formalisms of continuous dynamical systems and discrete transition systems can be combined to give *hybrid dynamical systems*. Hybrid systems are immensely useful in describing systems that have physical and computational components, such as embedded and control systems, as well as, systems that operate at multiple different time scales, such as biological systems.

This paper presents a deductive verification rule for continuous dynamical systems. When combined with the above rule for discrete state transition systems, we get a deductive verification rule for hybrid systems. The challenge in coming up with a deductive verification rule for continuous dynamical systems is that there is no useful notion of a "next" state. In this paper, we use "continuity" to formulate the deductive verification rule. One of the technical difficulties here is to obtain a rule that is simultaneously (1) sound, (2) complete, and (3) effectively checkable. It is easy to propose rules that compromise one or more of these three requirements. The main results of this paper are (a) two distinct sound and complete rules, but these are not directly checkable, and (b) three simpler sound and effectively checkable rules, that are relatively complete for large and useful classes of systems and invariants.

Motivation and Related Work. From a purely theoretical perspective, it is appealing to have an effective, sound, and relatively complete inference rule for safety verification of continuous systems. Recently, however, promising practical techniques have been proposed for safety verification that are directly based on using such inference rules. One such technique – that is especially effective for safety verification of continuous and hybrid systems – is *bounded verification*. Bounded verification is the dual of bounded model checking. Whereas bounded model checking searches for a bounded counter example for safety, bounded verification searches for a bounded proof for safety. The essential idea in bounded verification is to search for an inductive invariant of a given form. Note that the inference rule for safety verification requires proving the formula

$$\exists \mathtt{Inv} : \forall \vec{x}, \vec{y} : \phi(\mathtt{Inv}, \vec{x}, \vec{y}), \tag{1}$$

where ϕ is simply a conjunction of Formulas (1), (2) and (3) from the rule above. This formula involves a second-order quantification. We can eliminate this second-order quantification by restricting the form of the inductive invariant `Inv`. For example, assuming `Inv` can be written as $\psi(\vec{u}, \vec{x})$, over some unknown parameters $\vec{u}$, Formula 1 changes to

$$\exists \vec{u} : \forall \vec{x}, \vec{y} : \phi(\psi(\vec{u}, \vec{x}), \vec{x}, \vec{y}). \tag{2}$$

Formula 2 is now a first-order $\exists\forall$ formula. If this formula is valid, then we know there is an inductive invariant that proves safety. Further details on bounded verification, can be found in the work of Gulwani et al. [7] and Gulwani and Tiwari [8].

The formula $\psi(\vec{u}, \vec{x})$ can be seen as a template for the invariant. The idea of using templates is not new. In fact, it is the classical approach used to prove stability in control theory. Recently, it has also been used for safety verification for discrete programs [2, 7, 11, 18] and continuous and hybrid systems [17, 15, 1, 20, 8]. These papers use templates for performing bounded verification, but differ in the details about their use of the inference rule to construct ϕ' in Formula 2 and their use of the constraint solving technique to solve the

$\exists\forall$ constraint. Since template-based verification is not the main topic of this paper, but just used as a motivation, we do not discuss the related literature here. However, the inference rules used in the papers on verification of continuous and hybrid systems are relevant to the work in this paper and we discuss them briefly here and in the rest of the article.

If the hypothesized invariant $\texttt{Inv}$ is a polynomial equation, $p = 0$, then there is a simple way to check invariance: whenever $p = 0$, the time derivative of p, $\frac{dp}{dt}$, should also be 0. This verification rule for equational invariants was used by Sankaranarayanan et al. [17]. If the invariant is an inequality, such as $p \geq 0$, then there are several sufficient checks, such as, $\frac{dp}{dt} \geq 0$ whenever $p \geq 0$. This test is very strong: it requires that p is increasing everywhere inside the invariant set. This sound, but incomplete, test has been used by Platzer et al. [14, 13]. We can weaken the test, and check $\frac{dp}{dt} \geq 0$ only on points where $p = 0$ [15, 8], but this variant is not sound in general. This is discussed in detail later.

Outline of the Paper. We formally define continuous dynamical systems in Section 2 and present two distinct sound and complete deductive verification rules for continuous dynamical systems in Section 3. In Section 4, we first present inference rules that are interesting from a practical point of view and compromise either soundness or completeness. We then present three sound and relatively complete inference rules.

2 Continuous Dynamical System

DEFINITION 1.*[Continuous Dynamical System] A* continuous dynamical system $\texttt{CDS}$ *is a tuple* $(X, \texttt{Init}, f)$ *where* X *is a finite set of variables interpreted over the reals* $\mathbb{R}$, $\mathbf{X} = \mathbb{R}^X$ *is the set of all valuations of the variables* X, $\texttt{Init} \subseteq \mathbf{X}$ *is the set of initial states, and* $f : \mathbf{X} \mapsto \mathbf{X}$ *is a vector field that specifies the continuous dynamics.*

Note that $\mathbb{R}^{\mathrm{X}}$ is isomorphic to the n-dimensional real space $\mathbb{R}^n$ where $n = |\mathrm{X}|$ is the number of variables in X. Note also that the continuous dynamical systems we consider here are autonomous, that is, they have no inputs. We assume that f is Lipshitz, which guarantees that the ordinary differential equations $\frac{d\mathrm{X}}{dt} = f(\mathrm{X})$ have a unique solution. In fact, the following property [4] of Lipschitz vector fields will be used in the proofs.

PROPOSITION 2.*[Theorem 2.3.1, p80 [4]] Consider a Lipschitz vector field* f *and the initial value problem* $\frac{dX(t)}{dt} = f(X(t))$, $X(0) = \vec{x}_0$. *The solution of this problem, denoted by* $F(\vec{x}_0, t)$, *always exists and is unique. Moreover,* $F(\vec{x}_0, t)$ *depends continuously on the initial state* $\vec{x}_0$.

The meaning of a continuous dynamical system is simply the collection of all possible trajectories starting from an initial state. Formally, if $F(\vec{x}_0, t)$ is the solution of $\frac{d\mathrm{X}(t)}{dt} = f(\mathrm{X}(t))$, $\mathrm{X}(0) = \vec{x}_0$, then the semantics, $[[\texttt{CDS}]]$, of a continuous dynamical system $\texttt{CDS} = (\mathrm{X}, \texttt{Init}, f)$ is given as

$$[[\texttt{CDS}]] \;:=\; \{F_1 : [0,\infty) \mapsto \mathbf{X} \;\mid\; F_1(t) = F(\vec{x}_0, t), \quad \vec{x}_0 \in \texttt{Init} \;\}$$

The above semantics using flow functions is broadly referred to as the *flow semantics* [21]. One can also give a *transition semantics* using discrete state transition systems [9], but the distinction [5] is not relevant for this paper.

The set of reachable states for a continuous dynamical system CDS, Reach(CDS), is given by $\{\vec{x} \in \mathbf{X} \mid \exists F \in [[\texttt{CDS}]], \exists t \geq 0 : \vec{x} = F(t)\}$. A (safety) property, Safe, is simply a subset of the state space $\mathbf{X}$. A property Safe is an *invariant* (for the system CDS) if Reach(CDS) $\subseteq$ Safe. We are interested in solving the following problem in this paper:

DEFINITION 3. *[Safety Verification Problem] Given a continuous dynamical system* CDS *and a safety property* Safe*, determine if* Safe *is an invariant for* CDS*.*

One of the classical methods to solve the safety verification problem is based on finding stronger invariants that are also *inductive*. By introducing the extra requirement of inductiveness, the "global" test for invariance, viz. *all* reachable states are contained in Safe, reduces to a simpler "local" test, viz. every *single* transition out of Safe state goes into only a Safe state.

3 Sound and Complete Rules

In this section we present two verification rules for solving the problem described in Definition 3. Each rule replaces the global test for invariance by a local test for inductiveness.

We fix our notation and denote the given continuous dynamical system by CDS $=$ (X, Init, f) and the given safety property by Safe. The challenge in defining a local inductiveness test is that, for continuous dynamical systems, there is no clear notion of a "next" state in the flow semantics. Even if we use the transition semantics, the set of all the uncountably many next states is equal to the Reach set and hence the distinction between inductive invariants and general invariants is lost. However, using continuity, instead of using arbitrary future states, we can look at only states reachable in an ϵ-future and require that they remain inside Inv. This is formalized below in $(\mathcal{A}2)$.

DEFINITION 4. *[Inductive Invariant] A set* $Inv \subset \mathbb{R}^X$ *is an* inductive invariant *for a given continuous dynamical system* $CDS := (X, Init, f)$ *if the following conditions hold:*

$$(\mathcal{A}1) \quad Init \subseteq Inv$$
$$(\mathcal{A}2) \quad \forall \vec{x} \in Inv : \exists t_0 > 0 : \forall 0 \leq t < t_0 : F(\vec{x}, t) \in Inv$$

where F *is the solution of the initial value problem* $\frac{dX(t)}{dt} = f(X(t)), X(0) = \vec{x}$.

A closed set that is an inductive invariant in the above sense contains all the reachable states and hence it is indeed an invariant.

PROPOSITION 5. *Let* Inv *be a closed inductive invariant for the continuous dynamical system* $CDS := (X, Init, f)$*. Then,* $Reach(CDS) \subseteq Inv$*.*

However, Definition 4 is not directly useful for checking inductiveness because (a) it uses quantifier alternation ($\forall\exists\forall$) and (b) it uses the solution F of the differential equations. For most interesting applications, it may be difficult, if not impossible, to compute F analytically. Fortunately, there are two different ways in which we can check for inductiveness without using F. Before describing them, we first concretize the specification language for CDS and Safe.

$$(S1)\quad \texttt{Init}(\vec{x}) \Rightarrow p(\vec{x}) \geq 0 \qquad (T1)\quad \texttt{Init}(\vec{x}) \Rightarrow p(\vec{x}) \geq 0$$

$$(S2)\quad p(\vec{x}) = 0 \Rightarrow f(\vec{x}) \in T(p \geq 0)(\vec{x}) \qquad (T2)\quad p = 0 \Rightarrow (\bigwedge_{i=1}^{k-1} L_f^{(i)}(p) = 0 \Rightarrow L_f^{(k)}(p) \geq 0) \text{ for } k = 1, 2, \ldots$$

$$(S3)\quad \frac{p(\vec{x}) \geq 0 \Rightarrow \texttt{Safe}(\vec{x})}{\texttt{Reach}(\texttt{CDS}) \subseteq \texttt{Safe}} \qquad (T3)\quad \frac{p(\vec{x}) \geq 0 \Rightarrow \texttt{Safe}(\vec{x})}{\texttt{Reach}(\texttt{CDS}) \subseteq \texttt{Safe}}$$

Figure 1: Inference rules for safety verification of continuous system $\texttt{CDS} := (\texttt{X}, \texttt{Init}, f)$ and safety property $\texttt{Safe} \subseteq \mathbf{X}$.

Since we are interested in computability, henceforth, we assume that the continuous dynamical system $\texttt{CDS} := (\texttt{X}, \texttt{Init}, f)$ and the safe set $\texttt{Safe}$ are specified using polynomials. Let $L := \{\mathbb{Q}, +, -, *, \geq, >, =\}$ be a language containing all rational constants $\mathbb{Q}$, function symbols $+, -, *$ and predicates $\geq, >, =$. These symbols are interpreted over the reals in the usual way. We fix $\texttt{X}$ to be the set of variables. A term over variables $\texttt{X}$ will just be a polynomial in the ring $\mathbb{Q}[\texttt{X}]$. Atomic formulas consist of polynomial equalities and inequalities. A set $\texttt{S} \subseteq \mathbb{R}^n$ is *semi-algebraic* if it represents the solutions of a (quantifier-free) formula. A $\texttt{CDS} := (\texttt{X}, \texttt{Init}, f)$ is a *polynomial CDS* if $\texttt{Init}$ is semi-algebraic and the vector field is specified using only polynomials from $\mathbb{Q}[\texttt{X}]$.

For simplicity of presentation, we will initially restrict the set $\texttt{Inv}$ to be of the form $p \geq 0$ for some polynomial p. We will later extend the results to boolean combinations. Since we are restricting $\texttt{Inv}$ to be in a certain class, we will lose completeness. However, we are interested in "relative completeness"; that is, if there is an inductive invariant in the restricted class, then the deductive verification rule should be applicable.

We are now ready to present the two different ways for checking inductiveness without using F. First, we use a result in Control Theory, called Nagumo's theorem, that says that a set $\texttt{Inv}$ is an invariant only if, at every point $\vec{x}$ on the boundary of $\texttt{Inv}$, the vector field $f(\vec{x})$ at that point points "inwards". Formally, the set of vectors that point "inwards" at point $\vec{x}$ define the tangent cone at $\vec{x}$.

DEFINITION 6. *[Tangent Cone, Definition 3.1 in [3]] Let $S \subset \mathbb{R}^n$ be a closed set. Let $\vec{x} \in \mathbb{R}^n$. The tangent cone to S at $\vec{x}$ is the set*

$$T(S)(x) \quad := \quad \{\vec{z} \in \mathbb{R}^n \mid \liminf_{\alpha \to 0} \frac{d(\vec{x} + \alpha\vec{z}, S)}{\alpha} = 0\} \tag{3}$$

where $d(\vec{x}, S) := \inf_{\vec{y} \in S} ||\vec{x} - \vec{y}||$ is the distance of $\vec{x}$ from S and $||\cdot||$ is any norm in $\mathbb{R}^n$.

Figure 1 (Left) presents an inference rule for safety verification of continuous systems. Note that Condition (S2) says that for every point on the boundary of $\texttt{Inv}$, the vector field f is in the tangent cone at that point. Nagumo's theorem states that for closed sets $\texttt{Inv}$, Condition (S2) from Figure 1 is equivalent to Condition $(\mathcal{A}2)$ from Definition 4. We refer the reader to the review article by Blanchini for details [3].

The key idea behind the second approach for automating the test of Condition $(\mathcal{A}2)$ is the use of *Lie* derivatives. Intuitively, we can check that trajectories do not leave $p \geq 0$ by

checking that $\frac{dp}{dt}$ is greater-than zero whenever $p = 0$. Technically, the derivative of p with respect to time, $\frac{dp}{dt}$, is called the *Lie derivative*, $L_f(p)$, of p with respect to the vector field f. It can be computed using the chain rule, as shown below. Let us define the notation $L_f^{(n)}(p)$ to denote the n-th derivative of p with respect to time. Formally,

$$L_f^{(n)}(p) \;:=\; \begin{cases} \sum_{x \in \mathrm{X}} \frac{\partial p}{\partial x}\frac{dx}{dt} \;:=\; \vec{\nabla} p \cdot f := (\frac{\partial p}{\partial x_1}, \frac{\partial p}{\partial x_2}, \ldots) \cdot (\frac{dx_1}{dt}, \frac{dx_2}{dt}, \ldots) & \text{if } n = 1 \\ \frac{dL_f^{(n-1)}(p)}{dt} & \text{otherwise} \end{cases} \tag{4}$$

where the time-derivative, $\frac{d}{dt}$, is always computed using the chain rule as $\frac{dg}{dt} = \vec{\nabla} g \cdot f$. If f is specified using polynomials (i.e., $\frac{dx}{dt}$ is a polynomial for every variable x) and if p is a polynomial in $\mathbb{Q}[\mathrm{X}]$, then Equation 4 shows that $L_f^{(n)}(p)$ is a polynomial in $\mathbb{Q}[\mathrm{X}]$ and it can be symbolically computed. The second inference rule for checking inductiveness is shown in Figure 1(Right). Note that Condition (T2) requires that, *for all* k, the k-th derivative be non-negative whenever the first $k-1$ derivatives are zero.

The two deductive verification rules given in Figure 1 are both sound and (relatively) complete. For lack of space, proofs are not provided.

THEOREM 7. *[Soundness] Let* $\mathtt{CDS} := (\mathtt{X}, \mathtt{Init}, f)$ *be a continuous dynamical system and* $\mathtt{Safe} \subseteq \mathbf{X}$ *be a safety property. If there is a set* $\mathtt{Inv}$ *that satisfies Conditions (S1), (S2) and (S3) from Figure 1(Left), or alternatively, it satisfies Conditions (T1), (T2) and (T3) from Figure 1(Right), then* $\mathtt{Reach}(\mathtt{CDS}) \subseteq \mathtt{Safe}$.

THEOREM 8. *[Relative Completeness] Let* $\mathtt{CDS} := (\mathtt{X}, \mathtt{Init}, f)$ *be a CDS and* $\mathtt{Safe}$ *be a closed set such that* $\mathtt{Reach}(\mathtt{CDS}) \subseteq \mathtt{Safe}$. *If there is an inductive invariant* $p \geq 0$ *such that* $p \geq 0 \Rightarrow \mathtt{Safe}$, *then* $p \geq 0$ *also satisfies Conditions (S1), (S2) and (S3) from Figure 1(Left), as well as, Conditions (T1), (T2) and (T3) from Figure 1(Right).*

The inference rules in Figure 1 can be generalized to also handle Boolean combinations of predicates of the form $p \geq 0$ (see Discussion in Section 4) and these generalized rules will be complete for all semi-algebraic invariants.

Comparing the two inference rules. Since the two sets of conditions in Figure 1 are both sound and relatively complete for showing inductive invariance, it is tempting to assume that they are "essentially the same". These two tests are indeed "globally equivalent": if every point on the boundary satisfies Condition (S2), then every point on the boundary also satisfies Condition (T2), and vice-versa. However, the two tests are distinct tests and they are *not* "locally equivalent"; that is, they may disagree on individual points.

Example 1 *Consider the constant vector field* $f((x,y)) = (1,0)$ *and consider the candidate invariant region,* $-x^2 - y^2 + 2y \geq 0$. *The candidate invariant set is a circle of radius* 1 *centered at* $(0,1)$ *and hence clearly the vector field is tangential to the invariant set at the origin; that is,* $(1,0) \in T(-x^2 - y^2 + 2y \geq 0)((0,0))$. *Hence Condition (S2) evaluates to true for point* $(0,0)$. *However, the derivative test fails at* $(0,0)$*: though* $\frac{dp}{dt}$ *at* $(0,0)$ *is* 0, *the second derivative is negative (everywhere):* $\frac{dp}{dt} = -2x\frac{dx}{dt} - (2y-2)\frac{dy}{dt} = -2x$, $\frac{d^2p}{dt^2} = -2\frac{dx}{dt} = -2$. *This shows that Condition (T2) fails at* $(0,0)$. *Thus, Condition (S2) and Condition (T2) give different answers at the point* $(0,0)$. *However, they both agree globally that the candidate invariant set here is not an invariant.*

$$\begin{array}{ll} (A1) & \texttt{Init}(\vec{x}) \Rightarrow p(\vec{x}) \geq 0 \\ (A2) & p(\vec{x}) = 0 \Rightarrow L_f(p)(\vec{x}) \geq 0 \\ (A3) & \underline{p(\vec{x}) \geq 0 \Rightarrow \texttt{Safe}(\vec{x})} \\ & \texttt{Reach(CDS)} \subseteq \texttt{Safe} \end{array} \qquad \begin{array}{ll} (B1) & \texttt{Init}(\vec{x}) \Rightarrow p(\vec{x}) \geq 0 \\ (B2) & p(\vec{x}) = 0 \Rightarrow L_f(p)(\vec{x}) > 0 \\ (B3) & \underline{p(\vec{x}) \geq 0 \Rightarrow \texttt{Safe}(\vec{x})} \\ & \texttt{Reach(CDS)} \subseteq \texttt{Safe} \end{array}$$

Figure 2: An unsound, but relatively complete, rule (left) and a sound, but incomplete, rule (right) for safety verification of polynomial CDS $\texttt{CDS} := (\texttt{X}, \texttt{Init}, f)$ and safety property $\texttt{Safe} \subseteq \mathbf{X}$.

Although the verification rules in Figure 1 are both sound and relatively complete, they are not computationally feasible as there is no easy way to verify Condition (S2) and Condition (T2): the former involves reasoning about the tangent cone, whereas the latter is an infinite set of conditions. We will next present computable conditions and prove their soundness or completeness by comparing them to Condition (S2) or Condition (T2).

4 Practical Rules for Safety Verification of Polynomial CDS

In this section, we present inference rules that can be applied in practice for performing safety verification of continuous systems. We shall also point to the literature where these rules have been used. The rules will compromise either soundness or completeness.

Figure 2 presents two approximations of the inference rule in Figure 1(Right). First, instead of performing the infinitely many checks in Condition (T2)– one for each k – we can just perform the check for $k = 1$ and ignore the other checks. This gives an unsound, but relatively complete, inference rule, shown in Figure 2(Left). The following example shows the unsoundness and was mentioned to us by Andre Platzer.

Example 2 *Consider the system* $\texttt{CDS} := (\{x\}, \{x = 0\}, f)$ *where* $f(x) = 1$ *and the safety property* $-x^2 \geq 0$. *Since initially* $x = 0$ *and since* $\frac{dx}{dt} = f(x) = 1$, x *takes positive values and hence the safety property is violated. However, the rule in Figure 2(Left) can be applied successfully using* $-x^2$ *as* p. *Condition (A2) is verified because the following is a theorem in the theory of reals:* $-x^2 = 0 \Rightarrow -2x * 1 \geq 0$. *This example shows that the rule in Figure 2(Left) is unsound.*

Example 2 suggests that we can regain soundness by replacing the check $L_f(p) \geq 0$ by the stronger test $L_f(p) > 0$. This gives us the inference rule in Figure 2(Right). However, we lose completeness.

Example 3 (Incompleteness) *Consider the system* $\texttt{CDS} := (\{x\}, \{x = 0\}, f)$ *where* $f(x) = 0$ *and the safety property* $x \geq 0$. *Since initially* $x = 0$ *and since* $\frac{dx}{dt} = f(x) = 0$, *clearly* CDS *is safe with respect to the given safety property. In fact, there is an inductive invariant* $x \geq 0$ *(of the form* $p \geq 0$*) that can prove this safety property. However, the rule in Figure 2 fails: for any* $p \in \mathbb{Q}[x]$, $L_f(p)$ *is always* 0, *and it is never strictly positive (as required by Condition (B2)).*

The rules in Figure 2 are commonly used. Despite the unsoundness, the inference rule in Figure 2(Left) has been used in the work by Gulwani and Tiwari [8] and Prajna and Jadbabaie [15]. The sound, but incomplete, variant in Figure 2(Right) has been used by Prajna, Jadbabaie and Pappas [16].

$$\frac{(C1)\ \ \texttt{Init} \Rightarrow p \geq 0 \quad (C2)\ \ p = 0 \Rightarrow L_f(p) \geq 0 \quad (C2')\ \ p = 0 \Rightarrow \vec{\nabla} p \neq 0 \quad (C3)\ \ p \geq 0 \Rightarrow \texttt{Safe}}{\texttt{Reach}(\texttt{CDS}) \subseteq \texttt{Safe}}$$

$$\frac{(D1)\ \ \texttt{Init} \Rightarrow p \geq 0 \quad (D2)\ \ p = 0 \Rightarrow L_f(p) \geq 0 \quad (D2')\ \ p = 0 \wedge \vec{\nabla} p = 0 \Rightarrow \neg \texttt{neg}(p, \vec{\mathrm{x}}, f(\vec{\mathrm{x}})) \quad (D3)\ \ p \geq 0 \Rightarrow \texttt{Safe}}{\texttt{Reach}(\texttt{CDS}) \subseteq \texttt{Safe}}$$

Figure 3: Sound inference rules for safety verification of polynomial CDS $\texttt{CDS} := (\mathrm{X}, \texttt{Init}, f)$ and safety property $\texttt{Safe} \subseteq \mathbf{X}$ that are also complete for a certain class of invariants.

Inference Rule Complete for Smooth Invariants

The case that leads to unsoundness or incompleteness is when $p(\vec{\mathrm{x}}) = 0$ and $L_f(p)(\vec{\mathrm{x}}) = 0$. Intuitively, one expects that the condition $L_f(p)(\vec{\mathrm{x}}) = 0$ should hold only when the vector field is "tangential" to the invariant set $p \geq 0$. Unfortunately, it also holds in some degenerate cases. One such degenerate case is when $\vec{\nabla} p = 0$. The inference rule in Figure 3(Left) explicitly rules out such cases. Let us say that the boundary of a set $p \geq 0$ is *smooth* if, $\vec{\nabla} p(\vec{\mathrm{u}}) \neq 0$ for all points $\vec{\mathrm{u}}$ s.t. $p(\vec{\mathrm{u}}) = 0$. Condition (C2′) in Figure 3(Left) explicitly checks that the boundary of the invariant set is smooth. With this additional check, the inference rule in Figure 3(Left) can be shown to be sound.

Example 4 *Consider the dynamical system from Example 2. We cannot use the rule in Figure 3 on it. If we use $-x^2$ as p, Condition (C2′) becomes $-x^2 = 0 \Rightarrow -2x \neq 0$ which is false over the reals.*

The inference rule in Figure 3(Left) is not only sound, but also complete for invariants $p \geq 0$ whose boundary is smooth. This is the case, for example, when p is linear, which is a particularly useful class [8]. Figure 3(Left) fails on invariants with non-smooth boundaries.

Example 5 *Consider the system $\texttt{CDS} := (\{x, y, z\}, \texttt{Init}, f)$, where $\texttt{Init}$ is the set $x^2 + y^2 \leq z^2$ and the vector field f is given by $f((x, y, z)) := (-x, -y, -z)$. Thus, at every point, the vector field points to the origin and the initial set is a cone. We wish to prove that the set $\texttt{Init}$ is safe; i.e., $\texttt{Safe} = \texttt{Init}$. The inductive invariant $z^2 - x^2 - y^2 \geq 0$ can prove safety. However, there is no polynomial p such that $p \geq 0$ satisfies Conditions (C1), (C2), (C2′) and (C3). Suppose p is such a polynomial. Then, since the set $\texttt{Init}$ is equal to the set $\texttt{Safe}$, the set $\texttt{Inv} := \{\vec{\mathrm{u}} \mid p(\vec{\mathrm{u}}) \geq 0\}$ has to be necessarily equal to these two sets (by Condition (C1) and (C3)). But then, Condition (C2′) will fail because at the boundary point $(0, 0, 0)$ the gradient of p cannot be nonzero.*

Inference Rule Complete for Quadratic Invariants

We can generalize Condition (C2′) to require that, at all points where $p = 0$ and $\vec{\nabla} p = 0$, the vector field f is "pointing inside" (Figure 3(Right)). Before we outline the test for "pointing inside", we need the following definition.

DEFINITION 9. *[Homogeneous decomposition, $\texttt{zero}, \texttt{pos}, \texttt{neg}$] A polynomial $p \in \mathbb{Q}[X]$ is a* homogeneous polynomial of degree *k if the total degree of each monomial in p is k. A* homogeneous decomposition *of p is obtained by writing p as $\sum_{i=1}^{n} p_i$, where p_i is homogeneous with degree k_i and $k_i < k_j$ for $i < j$. Let $p(\vec{x} + \vec{y})_i$ denote the i-th homogeneous*

$$(F1) \qquad \texttt{Init} \Rightarrow p \geq 0$$

$$(F2) \quad p = 0 \Rightarrow \neg \texttt{neg}(p, \vec{x}, f) \vee \bigvee_{k=2}^{n} (\texttt{kneg}(p, \vec{x}, f, k) \wedge \bigvee_{l<k} (\exists g : \texttt{pos}(p_l, f, g) \wedge \bigwedge_{j<l} \texttt{zero}(p_j, f, g)))$$

$$(F3) \qquad p \geq 0 \Rightarrow \texttt{Safe}$$

$$\texttt{Reach}(\texttt{CDS}) \subseteq \texttt{Safe}$$

Figure 4: Sound, and relatively complete, deductive rule for solving the safety verification problem for polynomial CDS $\texttt{CDS} := (\texttt{X}, \texttt{Init}, f)$ and safety property $\texttt{Safe} \subseteq \mathbf{X}$.

component of $p(\vec{x}+\vec{y})$ when viewed as a polynomial in $\vec{y}$ (with coefficients in $\mathbb{Q}[\vec{x}]$). The predicates $\texttt{zero}, \texttt{pos}, \texttt{neg}$ and $\texttt{kneg}$ are defined as follows:

$$\begin{aligned} \texttt{pos}(p, \vec{x}, \vec{u}) &:= \bigvee_{k=1}^{n} (p(\vec{x}+\vec{y})_k(\vec{u}) > 0 \wedge \bigwedge_{i=1}^{k-1} p(\vec{x}+\vec{y})_i(\vec{u}) = 0) \\ \texttt{kneg}(p, \vec{x}, \vec{u}, k) &:= (p(\vec{x}+\vec{y})_k(\vec{u}) < 0 \wedge \bigwedge_{i=1}^{k-1} p(\vec{x}+\vec{y})_i(\vec{u}) = 0) \end{aligned}$$

$$\texttt{zero}(p, \vec{x}, \vec{u}) := \bigwedge_{i=1}^{n} p(\vec{x}+\vec{y})_i(\vec{u}) = 0 \qquad \texttt{neg}(p, \vec{x}, \vec{u}) := \bigvee_{i=1}^{n} \texttt{kneg}(p, \vec{x}, \vec{u}, i)$$

If p is a polynomial and $\vec{x}, \vec{u}$ are two points such that $p(\vec{x}) = 0$, then
(a) $\texttt{pos}(p, \vec{x}, \vec{u})$ is equivalent to $\exists \alpha_0 > 0 : \forall (0 < \alpha \leq \alpha_0) : p(\vec{x} + \alpha\vec{u}) > 0$.
(b) $\texttt{zero}(p, \vec{x}, \vec{u})$ is equivalent to the fact that $p(\vec{x} + \alpha\vec{u}) = 0$ for all α.
(c) $\texttt{neg}(p, \vec{x}, \vec{u})$ is equivalent to $\exists \alpha_0 > 0 : \forall (0 < \alpha \leq \alpha_0) : p(\vec{x} + \alpha\vec{u}) < 0$.

Using the predicate $\texttt{neg}$, the inference rule in Figure 3(Right) checks that, for every point $\vec{x}$ such that $p(\vec{x}) = 0$ and $\vec{\nabla} p(\vec{x}) = 0$, it is the case that moving along the direction of the vector field $f(\vec{x})$ at the point $\vec{x}$, we move inside the invariant set $p \geq 0$. Figure 3(Right) generalizes the rule in Figure 3(Left). We will later see that it is complete for quadratic p.

Inference Rule Complete for Convex Invariants

Figure 4 presents an inference rule that generalizes the above two rules and can be shown to be complete for a larger class of invariants that includes linear, smooth and quadratic invariants. The rule in Figure 4 checks that for each point $\vec{x}$ on the boundary ($p(\vec{x}) = 0$), either we move inside the set $p \geq 0$ as we move from $\vec{x}$ along the vector field direction $f(\vec{x})$, or we move outside but there is a direction g such that if we go along g, we can make $p = 0$ "sufficiently quickly"; see illustration in Figure 5. The following example illustrates the notation from Definition 9 and the inference rule in Figure 4.

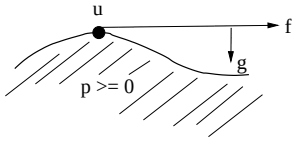


Figure 5: Illustration

Example 6 *Consider $\texttt{CDS} := (\{x_1, x_2\}, \texttt{Init}, f)$, where $\texttt{Init}$ is given by $x_1 = 2, x_2 = 0$ and $f(x_1) = x_2, f(x_2) = -x_1$. Let p be $-x_1^2 - x_2^2 + 4$. The set $p \geq 0$ is an inductive invariant of*

this `CDS`. *Let* $\vec{u}$ *be a point on the boundary; i.e.,* $p(\vec{u}) = 0$. *Moving the origin to* $\vec{u}$, *we get the new polynomial* $p(\vec{u}+\vec{x}) = -(u_1+x_1)^2 - (u_2+x_2)^2 + 4$ *which is equal to* $(-u_1^2 - u_2^2 + 4) - 2u_1x_1 - 2u_2x_2 - x_1^2 - x_2^2$. *Since* $p(\vec{u}) = 0$, *the new polynomial simplifies to* $-2u_1x_1 - 2u_2x_2 - x_1^2 - x_2^2$. *This has two homogeneous components:*

$$\begin{array}{lll} p_1 := p(\vec{u}+\vec{x})_1 := -2u_1x_1 - 2u_2x_2 & & \text{homogeneous with degree } k_1 = 1 \\ p_2 := p(\vec{u}+\vec{x})_2 := -x_1^2 - x_2^2 & & \text{homogeneous with degree } k_2 = 2 \end{array}$$

We now verify that $-x_1^2 - x_2^2 + 4 \geq 0$ *satisfies Condition (F2):*

$$p_1(f(\vec{u})) := -2u_1u_2 + 2u_2u_1 = 0 \qquad p_2(f(\vec{u})) := -u_1^2 - u_2^2$$

Since $p(\vec{u}) = 0$, *which is* $-u_1^2 - u_2^2 + 4 = 0$, *implies* $p_1(f(\vec{u})) = 0$ *and* $p_2(f(\vec{u})) < 0$, *we have* $\texttt{kneg}(p, \vec{u}, f(\vec{u}), 2)$ *holds. Clearly,* $\texttt{zero}(p, \vec{u}, f(\vec{u}))$ *and* $\texttt{pos}(p, \vec{u}, f(\vec{u}))$ *do not hold. Thus, we see that the direction* $f(\vec{u})$ *takes the point outside of the invariant set (as in Figure 5). However, Condition (F2) is true since for direction* $g := (-u_1, -u_2)$, $\texttt{pos}(p_1, f(\vec{u}), g)$ *holds:*

$$p_1(f(\vec{u})+\vec{x})_1 := -2u_1x_1 - 2u_2x_2, \qquad p_1(f(\vec{u})+\vec{x})_1(g) := 2u_1^2 + 2u_2^2 = 2*4 = 8 > 0$$

The rule in Figure 4 is complete for the class of invariants `Inv` that are `convex`.

DEFINITION 10. *The predicate* $\texttt{convex}(p \geq 0)$ *holds for the set* $p \geq 0$ *if, for any points* $\vec{u}$ *and* $\vec{v}$, *if* $p(\vec{u}) \geq 0$ *and* $p(\vec{v}) \geq 0$, *then* $p(\vec{u} + \alpha\vec{v}) \geq 0$ *for all* $0 \leq \alpha \leq 1$.

For example, the set $-x^2 - y^2 + 1 \geq 0$ is `convex`, but the set $-x^2 - y^2 + 1 = 0$, which can be encoded as $-(-x^2 - y^2 + 1)^2 \geq 0$, is not `convex`.

Soundness and Relative Completeness

THEOREM 11. *[Soundness] Let* $\texttt{CDS} := (X, \texttt{Init}, f)$ *be a CDS and* `Safe` *be a safety property. If* $p \in \mathbb{Q}[X]$ *is a polynomial that satisfies Conditions (C1), (C2), (C2') and (C3) of Figure 3(Left), or alternatively Conditions (D1), (D2), (D2') and (D3) of Figure 3(Right), or alternatively, Conditions (F1), (F2) and (F3) of Figure 4, then* $\texttt{Reach}(\texttt{CDS}) \subseteq \texttt{Safe}$.

THEOREM 12. *[Relative Completeness] Let* $\texttt{CDS} := (X, \texttt{Init}, f)$ *be a CDS and* `Safe` *be a closed set such that* $\texttt{Reach}(\texttt{CDS}) \subseteq \texttt{Safe}$. *Let* $p \geq 0$ *be an inductive invariant such that* $p \geq 0 \Rightarrow \texttt{Safe}$. *Then, the following claims are true.*
(1) If $p = 0 \Rightarrow \vec{\nabla} p \neq 0$, *then* $p \geq 0$ *satisfies Conditions (C1), (C2), (C2') and (C3).*
(2) If p *is quadratic, then* $p \geq 0$ *satisfies Conditions (D1), (D2), (D2') and (D3).*
(3) If $p \geq 0$ *is* `convex`, *then* $p \geq 0$ *satisfies Conditions (F1), (F2) and (F3).*

Theorem 12 shows that the rules in Figure 3 are complete for a large class of practically useful invariants, namely, linear, quadratic, and convex invariants. Note that for a polynomial CDS and a semi-algebraic safe set, given a p, the inference rules in Figure 3 are formulas in the first-order theory of the reals, which is decidable [19]. It appears to be extremely difficult to come up with a simple and effective rule that is sound and complete for the class of *all* invariants of the form $p \geq 0$.

Example 7 *The set* $-(-x^2 - y^2 + 2y)^2 \geq 0$, *which geometrically is the circumference of a circle, is not* `convex`. *In fact, inference rules in Figure 3 and Figure 4 will all fail to prove that this set is an inductive invariant under the dynamics given by* $\frac{dx}{dt} = 1 - y$, $\frac{dy}{dt} = x$.

Discussion The rule in Figure 4 is related to the earlier rules via the observation that $p(\vec{u}+\vec{x})_1(f(\vec{u}))$ is equal to $L_f(p)(\vec{u})$. In the special case when $\vec{\nabla}p \neq 0$, the role of the witness direction g (in Figure 4) can be performed by $\vec{\nabla}p$. Thus, Figure 4 is also relatively complete for "smooth" sets and hence it is more powerful than the rules in Figure 3.

The rules above are complete for larger classes that what have been identified above. For example, the rule in Figure 3(Right) is complete for all p such that $p(\vec{x}) = 0 \wedge \vec{\nabla}(p)(\vec{x}) = 0 \Rightarrow (p(\vec{x}+\vec{y})_2(f(\vec{x})) \neq 0$, but we do not explore those results here.

Since Condition (F2) is based on Nagumo's criterion, which holds more generally, we can now easily generalize Condition (F2) from $p \geq 0$ to more general boolean combinations of polynomial inequalities. Let $\mathtt{In}(p, \vec{x}, f)$ be a predicate that denotes Condition (F2) applied to polynomial p at point $\vec{x}$ with vector field f. When the candidate invariant is $p_1 \geq 0 \ \wedge \ p_2 \geq 0$, Condition (F2) generalizes to $(p_1(\vec{x}) = 0 \wedge p_2(\vec{x}) > 0 \Rightarrow \mathtt{In}(p_1, \vec{x}, f)) \wedge (p_1(\vec{x}) > 0 \wedge p_2(\vec{x}) = 0 \Rightarrow \mathtt{In}(p_2, \vec{x}, f)) \wedge (p_1(\vec{x}) = 0 \wedge p_2(\vec{x}) = 0 \Rightarrow \mathtt{In}(p_1, \vec{x}, f) \wedge \mathtt{In}(p_2, \vec{x}, f))$. Similarly, when the candidate invariant is $p_1 \geq 0 \ \vee \ p_2 \geq 0$, then Condition (F2) generalizes to $(p_1(\vec{x}) = 0 \wedge p_2(\vec{x}) < 0 \Rightarrow \mathtt{In}(p_1, \vec{x}, f)) \wedge (p_1(\vec{x}) < 0 \wedge p_2(\vec{x}) = 0 \Rightarrow \mathtt{In}(p_2, \vec{x}, f)) \wedge (p_1(\vec{x}) = 0 \wedge p_2(\vec{x}) = 0 \Rightarrow \mathtt{In}(p_1, \vec{x}, f) \vee \mathtt{In}(p_2, \vec{x}, f))$.

Hybrid Systems Since hybrid systems extend CDSs with discrete transitions, and since the rule to handle discrete transitions is standard, the sound inference rules for hybrid systems can be obtained by combining the rule for continuous systems with the rule for discrete transitions. However, when using the rule for continuous systems, we can use any rule whose soundness is proved using Condition $(\mathcal{A}2)$ (such as rule in Figure 1(Right)), but we cannot use a rule whose soundness is proved using Condition (S2) (such as rule in Figure 4). The reason is that, as mentioned in Section 3, Condition (T2) is locally sound, whereas Condition (S2) is locally unsound, but only globally sound. In hybrid systems, due to the possibility of the presence of discrete transitions from the boundary, we need a sound condition that can verify invariance locally at every point.

Example 8 *We build a hybrid system to exploit the difference illustrated in Example 1. Consider a hybrid system that has only one mode, with dynamics* $f((x,y)) = (1,0)$ *and a discrete transition given by,* $x := -x$ *whenever* $x^2 + (y-1)^2 = 1 \wedge x > 0$. *Suppose initially,* $x^2 + (y-1)^2 \leq 1$ *and we want to show that this initial set is also an inductive invariant. We note that this set is not an invariant because there are trajectories leaving the invariant set from points* $(0,1)$ *and* $(0,0)$. *But Condition (S2) holds at both these points, and it also holds on all boundary points* from where there is no discrete transition. *The invariant set is inductive with respect to the discrete transitions. This shows that one has to be careful when generalizing rules based on Condition (S2) to hybrid systems.*

5 Conclusions

We presented several inference rules for safety verification of continuous systems and analyzed their soundness and relative completeness. We have a finite and sound rule that is also complete for the class of invariants containing convex and certain smooth semi-algebraic sets. It remains a challenge to discover an effectively checkable and sound rule that is complete for all semi-algebraic invariants.

References

[1] A. Abate, A. Tiwari, and S. Sastry. Box invariance for biologically-inspired dynamical systems. In *Proc. IEEE Conf. on Decision and Control, CDC*, pages 359–364, 2007.

[2] D. Beyer, T. Henzinger, R. Majumdar, and A. Rybalchenko. Invariant synthesis for combined theories. In *VMCAI*, volume 4349 of *LNCS*, pages 378–394, 2007.

[3] F. Blanchini. Set invariance in control. *Automatica*, 35:1747–1767, 1999.

[4] K. Burns and M. Gidea. *Differential Geometry and Topology: With a view to dynamical systems*. Chapman & Hall, 2005.

[5] P. Cuijpers and M. Reniers. Lost in translation: Hybrid-time flows vs real-time transitions. In *Proc. 11th HSCC*, volume 4981 of *LNCS*, pages 116–129. Springer, 2008.

[6] R. W. Floyd. Assigning meaning to programs. In *Proc. Symp. in Appl. Math*, 1967.

[7] S. Gulwani, S. Srivastava, and R. Venkatesan. Program analysis as constraint solving. In *Proc. ACM Conf. on Prgm. Lang. Desgn. and Impl. PLDI*, pages 281–292, 2008.

[8] S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *Proc. 20th CAV*, volume 5123 of *LNCS*, pages 190–203. Springer, 2008.

[9] T. A. Henzinger. A theory of hyrid automata. In *Proc. 11th IEEE Logic in Comp. Sci. LICS*, pages 278–292, 1996.

[10] C. A. R. Hoare. An axiomatic basis of computer programming. *Comm. ACM*, 12(10):576–580, 1969.

[11] Deepak Kapur. Automatically generating loop invariants using quantifier elimination. In *Deduction and Applications*, 2005.

[12] R. M. Keller. Formal verification of parallel programs. *Comm. of the ACM*, 19(7), 1976.

[13] A. Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reasoning*, 41(2):143–189, 2008.

[14] A. Platzer and E. M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In *CAV*, volume 5123 of *LNCS*, pages 176–189. Springer, 2008.

[15] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *HSCC*, volume 2993 of *LNCS*, pages 477–492, 2004.

[16] S. Prajna, A. Jadbabaie, and G. Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE Trans Aut Control*, 52(8), 2007.

[17] S. Sankaranarayanan, H. Sipma, and Z. Manna. Constructing invariants for hybrid systems. In *HSCC*, volume 2993 of *LNCS*, pages 539–554, 2004.

[18] S. Sankaranarayanan, H. Sipma, and Z. Manna. Non-linear loop invariant generation using gröbner bases. In *POPL*, pages 318–329, 2004.

[19] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, 1948.

[20] A. Tiwari. Generating box invariants. In *Proc. HSCC*, LNCS 4981. Springer, 2008.

[21] A. J. van der Schaft and J. M. Schumacher, editors. *An introduction to hybrid dynamical systems*, volume 251 of *Lecture Notes in Ctrl. and Inf. Sci.* Springer, 2000.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Recurrence and Transience for Probabilistic Automata

M. Tracol[1]*, C. Baier[2], M. Grösser[2]

[1] LRI
University Paris-South
France
tracol@lri.fr

[2] Technische Universität Dresden
Germany
{baier|groesser}@tcs.inf.tu-dresden.de

ABSTRACT.
In a context of ω-regular specifications for infinite execution sequences, the classical Büchi condition, or repeated liveness condition, asks that an accepting state is visited infinitely often. In this paper, we show that in a probabilistic context it is relevant to strengthen this infinitely often condition. An execution path is now accepting if the *proportion* of time spent on an accepting state does not go to zero as the length of the path goes to infinity. We introduce associated notions of recurrence and transience for non-homogeneous finite Markov chains and study the computational complexity of the associated problems. As Probabilistic Büchi Automata (PBA) have been an attempt to generalize Büchi automata to a probabilistic context, we define a class of Constrained Probabilistic Automata with our new accepting condition on runs. The accepted language is defined by the requirement that the measure of the set of accepting runs is positive (probable semantics) or equals 1 (almost-sure semantics). In contrast to the PBA case, we prove that the emptiness problem for the language of a constrained probabilistic Büchi automaton with the probable semantics is decidable.

1 Introduction

In a context of system analysis, ω-regular specifications are used to evaluate the long term properties of a system [14]. An ω-regular specification can be decomposed into a safety part and a liveness part. Typically, if the system is an elevator reacting to a user, an ω-regular specification can ensure that the system will never do something "wrong" (for instance having its door open while moving), and that the system will eventually do something "good" after a stimulus (for instance the elevator should stop on level i after a finite number of steps if the user asks to). The avoidance of the "wrong" event is the safety part, and the "eventually good" event is the liveness part. The liveness can be violated only in the limit. As underlines [6], a weakness of the classical definition is that the requirements can be satisfied by evolutions of the system which are quite unsatisfactory because no bound can be put on the response time. For instance, as the elevator is used by different users, they may have to wait an increasing and maybe unbounded amount of time to reach their level. Alternative

*This research was funded by the French program on Computer Security **ANR-07-SESU-013**

© Tracol, Baier, Grösser; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 395–406
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2335

definitions for liveness have been proposed, in order to bound the distance between consecutive responses [6, 3]. In [6], the authors present the alternative notion of *finitary liveness*: finitary liveness assumes the existence of an unknown bound b such that every stimulus is followed by a response within b transitions. In this paper, instead of asking for a bound on the number of steps between "good" events, we will ask that the *proportion* of "good" events on a run does not go to zero as the length of the run goes to infinity.

In [4], the authors consider ω-regular properties on Markov chains, and in [2], the authors extend Büchi automata to a probabilistic context. They introduce the class $PBA^{>0}$ of Probabilistic Büchi Automata, which can be seen as a resolution of the non-determinism on a Büchi automaton by a probabilistic choice. In [2], as for the classical Büchi condition, a run is accepted if it visits infinitely often an accepting state, and a word is accepted if the probability of the set of associated accepted runs is non zero. This definition leads to a class of languages which is closed under the elementary operations of union, intersection and complementation. Moreover, the class of languages defined by PBA$^{>0}$ strictly subsumes the class of ω-regular languages. Unfortunately, working on these objects is difficult since basic problems such as the emptiness problem for the language of an automaton in $PBA^{>0}$ is undecidable [1].

In this paper, we consider alternative definitions of accepting runs. We introduce the notion of the *Support* of a run: a state s is in the support of a run r if the portion of time the state s is visited by r between time 0 and T, does not go to zero as T goes to infinity. We introduce the class $CPBA$ of *Constrained Probabilistic Büchi Automata*. A run on an automaton in $CPBA$ is accepting if there exists an accepting state in its support. As for PBA$^{>0}$, a word is accepted by a CPBA$^{>0}$ if the probability of the set of associated accepted runs is non zero.

We show that the class of languages associated to CPBA$^{>0}$ is not closed under complementation, however the emptiness problem is now in PSPACE. As it is done in [1] for the class PBA, we consider the class $CPBA^{=1}$ of Constrained Probabilistic Büchi Automata with an *almost sure semantics*. We prove that solving the emptiness problem of the language of an automaton in $CPBA^{=1}$ is equivalent to solving the same problem on an automaton in the class $PBA^{=1}$.

The fact that with positive probability an accepting state is in the support of a run can be seen as a recurrence property, by analogy with the classical homogeneous Markov chain theory. We define notions of recurrence and transience for non homogeneous probabilistic processes, in a context of Finite Probabilistic Tables (FPT, [16, 15]). An FPT can be seen as a non-homogeneous Markov chain on a finite state space with a finite number of transition functions. The main results of the paper are:

- Notions of weak and strong transience and recurrence for non homogeneous Markov processes.
- The study of the computational complexity of the associated problems, in particular the PSPACE-completeness of the strong recurrence problem, and the undecidability of the two states strong recurrence problem.
- The decidability of the emptiness problem for the languages of automata in $CPBA^{>0}$ and $CPBA^{=1}$.
- The study of the expressivity of our new classes of automata. In particular the set of the complement of languages of automata in $PBA^{=1}$ is expressible with automata in

$CPBA^{>0}$.

The paper is organized as follows: In section 2 we briefly recall the basic notions of finite probabilistic tables and define (constrained) probabilistic automata on infinite words. In section 3 we define the notions of transience and recurrence on finite probabilistic tables and study the computational complexity of the associated problems. In section 4 we consider the classes $CPBA^{>0}$ and $CPBA^{=1}$ and possible generalizations. Section 5 concludes the paper.

2 Preliminaries

Throughout the paper, we assume some familiarity with classical automata theory on infinite words [9]. We will use the notion of Finite Probabilistic Table (FPT) as a general framework for probabilistic automata. An FPT is the "structural part" of a probabilistic automaton, on which no acceptance condition has been made precise. An FPT can also be seen as a particular kind of non-homogeneous Markov chain, where only a finite number of transition functions are available. If S is a finite set, we write $\Delta(S)$ for the set of probability distributions on S.

DEFINITION 1. *[Finite Probabilistic Tables [16]] A* Finite Probabilistic Table *(FPT), is a tuple $\mathcal{T} = (S, \Sigma, \{M^a, a \in \Sigma\}, \alpha)$, where S is a finite set (representing the states), $\alpha \in \Delta(S)$ is the initial distribution, Σ is a finite set (representing the alphabet), and for all $a \in \Sigma$, M^a is a Markov matrix of order $|S|$ (M^a represents the transition probabilities from state to state related to the symbol a).*

We write $M^a = (m^a_{s_i,s_j})_{i,j\in\{1,\dots,|S|\}}$. The component $m^a_{s,t}$ corresponds to the probability of going from state s to state t when the transition matrix M^a is chosen. If $w = a_1...a_l \in \Sigma^*$, we write M^w for the product $M^{a_1} \cdot ... \cdot M^{a_l}$, whose components are the $m^w_{s_i,s_j}$. Often, we will use the notation δ for the transition function: if $w \in \Sigma^*$ and $s, t \in S$, $\delta(s, w)(t)$ is the probability to arrive in t if we start on s and read w. In other words, $\delta(s, w)(t) = m^w_{s,t}$. We generalize the notation and write $\delta(s, w)$ for the set of states $t \in S$ such that $\delta(s, w)(t) > 0$. Finally, if $A \subseteq S$ (resp. $\alpha \in \Delta(S)$), $\delta(A, w)$ (resp. $\delta(\alpha, w)$) is the set of states $t \in S$ such that there exists $s \in A$ with $\delta(s, w)(t) > 0$ (resp. $s \in S$ s.t. $\alpha(s) > 0$ and $\delta(s, w)(t) > 0$). We will often define an FPT as a tuple $\mathcal{T} = (S, \Sigma, \delta, \alpha)$, since we can compute easily δ and the $M^a, a \in \Sigma$ one from the other.

Let $\mathcal{T} = (S, \Sigma, \delta, \alpha)$ be an FPT. A *run* on $\mathcal{T}$, or a run on S and Σ, is an alternating sequence $s_0a_1s_1a_2...$, finite or infinite, of states in S and letters in Σ. The *trace* of a run r, written $Tr(r)$, is the sequence of its letters, and $Inf(r)$ is the set of states which appear infinitely often in r. Given a finite run $r = s_0a_1s_1...a_ns_n$ we denote by $|r| = n$ the length of r and by $r_{|k} = s_0a_1s_1...a_ks_k$ its prefix of length k. Similarly for a finite word $w \in \Sigma^*$, $|w|$ is the length of w and $w_{|k}$ denotes its prefix of length k. We write Ω for the set of infinite runs on $\mathcal{T}$. If $n \in \mathbb{N}$, X_n is the random variable on Ω which associates to a run r its n-th state. The set of cones of the form $C_w = \{r \in \Omega | Tr(r_{|n}) = w\}$, for $w \in \Sigma^n$, induces a σ-field $\mathcal{F}$ on Ω which is the smallest σ-field with respect to which all the $X_n, n \geq 0$, are measurable. The initial distribution α on S, and an infinite word $w = a_1a_2... \in \Sigma^\omega$, uniquely determine a probability measure $\mathbb{P}^\alpha_w$ on $\mathcal{F}$ such that $X_n, n \geq 0$ is a non-homogeneous Markov chain on $(\Omega, \mathcal{F}, \mathbb{P}^\alpha_w)$, with $\mathbb{P}^\alpha_w(X_0 = s) = \alpha(s)$, and $\mathbb{P}^\alpha_w(X_{n+1} = t | X_n = s) = \delta(s, a_{n+1})(t)$ for all

$n \in N$ and $s, t \in S$. (See [11, 13, 18, 7]). We may forget the α in the notation when clear from the context.

DEFINITION 2. *[Support of an infinite sequence] Let Σ be a finite alphabet, and $w = a_0, a_1, ... \in \Sigma^\omega$. Let $\rho = b_0 b_1 ... b_l \in \Sigma^*$. We call the* proportion *of ρ in w the limit-sup of the proportion of time spent reading ρ when reading $a_1, ..., a_n$:*

$$prop(\rho, w) = \overline{Lim}_{n\to\infty} \frac{|\{i \in [1; n-l] \text{ s.t. } a_i = b_0 \wedge ... \wedge a_{i+l} = b_l\}|}{n}.$$

The support *of the sequence w, written $Supp(w)$, is the set of words $\rho \in \Sigma^*$ such that $prop(\rho, w) > 0$.*

For instance, if we consider a run r on an automaton $\mathcal{A}$ as an infinite sequence on $S \cup \Sigma$, the set of states in the support of r can be seen as the set of states on which r spends a non negligible amount of time. It is a subset of $Inf(r)$, and the inclusion is strict in general. Instead of imposing acceptance conditions on the set of states that are visited infinitely often in a run, in this paper we will impose acceptance conditions on the set of states that are visited with a "non negligible" portion, i.e. that are in the support of the run. This gives rise to the class of constrained probabilistic automata.

A probabilistic automaton is just a pair $\mathcal{A} = (\mathcal{T}, \mathsf{Acc})$ where $\mathcal{T} = (S, \Sigma, \delta, \alpha)$ is an FPT and Acc is an acceptance condition. We consider here acceptance conditions of the following types: Büchi, where $\mathsf{Acc} \subseteq S$ is a subset of final states, Street and Rabin, where $\mathsf{Acc} = \{(H_1, K_1), \ldots, (H_n, K_n)\}$ is a set of acceptance pairs and Muller, where $\mathsf{Acc} \subseteq 2^S$ is a set of final sets. Given a subset $T \subseteq S$ of states we call T accepting according to a

- Büchi acceptance condition $\mathsf{Acc} \subseteq S$, if $T \cap \mathsf{Acc} \neq \emptyset$. In the sequel we will denote a Büchi acceptance condition by F.
- Rabin acceptance condition $\mathsf{Acc} = \{(H_1, K_1), \ldots, (H_n, K_n)\}$, if there exists $1 \leq i \leq n$ such that $T \cap H_i = \emptyset$ and $T \cap K_i \neq \emptyset$.
- Streett acceptance condition $\mathsf{Acc} = \{(H_1, K_1), \ldots, (H_n, K_n)\}$, if for every $1 \leq i \leq n$ it holds that $T \cap H_i \neq \emptyset$ or $T \cap K_i = \emptyset$.
- Muller acceptance condition $\mathsf{Acc} \subseteq 2^S$, if $T \in \mathsf{Acc}$.

As indicated above we will distinguish between two types of automata, namely

- (classical) probabilistic automata, where a run is called accepting for $w \in \Sigma^\omega$ iff $Tr(r) = w$ and $Inf(r)$ is accepting and
- constrained probabilistic automata, where a run is called accepting for $w \in \Sigma^\omega$ iff $Tr(r) = w$ and $Supp(r)$ is accepting.

For both types of automata we distinguish two semantics, the probable semantics, where the accepted language of $\mathcal{A}$ is:

$$\mathcal{L}^{>0}(\mathcal{A}) = \{w \in \Sigma^\omega | \mathbb{P}_w(\{r | r \textit{ is accepting for } w\}) > 0\}$$

and the almost-sure semantics where the accepted language of $\mathcal{A}$ is:

$$\mathcal{L}^{=1}(\mathcal{A}) = \{w \in \Sigma^\omega | \mathbb{P}_w(\{r | r \textit{ is accepting for } w\}) = 1\}.$$

Given an automaton with a Büchi acceptance condition, we call a (classical) probabilistic automaton a PBA and we call a constrained probabilistic automaton a CPBA (the analogous notations apply to Street (PSA, CPSA), Rabin (PRA, CPRA) and Muller (PMA, CPMA) automata). The class of PBA is denoted PBA. In the following, when K is a class of automata and $x \in \{> 0, = 1\}$, we write $\mathcal{C}_{K^x}$ for the associated class of languages. We will some-

times write PBA^x, resp. PBA^x, to denote a PBA, resp. the class of PBA, with the associated semantics given by x.

By [2, 1], $\mathcal{C}_{PBA^{>0}}$ is closed under union, intersection and complementation. However, the emptiness problem of a PBA$^{>0}$ is shown to be undecidable. On the other hand, the emptiness problem for an automaton in $PBA^{=1}$ is shown to be decidable, but the class $\mathcal{C}_{PBA^{=1}}$ is not any more closed under complementation.

Remarks: Taking an *inf* limit instead of a *sup* limit in the definition of the support, we could express the fact that the proportion of time spent on a particular set of states stays bounded away from zero as the length increases. The two different possible definitions for the support of a run would lead to different classes of automata, which recognize different languages and can express different properties of interest. However, we will see that the same algorithms can be used on both classes of automata, for the natural problems such as the emptiness problem. In this paper, we will use the *limit-sup* to define the support of a run, but the results could be easily adapted to handle the *inf limit* case.

3 Finite non-homogeneous Markov chains

We are interested in basic questions concerning our models of probabilistic automata (PBA, $CPBA$), such as the emptiness problem of the language of a given automaton, or the universality problem for this language. Such problems can be presented in the general framework of finite non-homogeneous Markov chains. In the past, researcher working in this domain seem to have been mostly interested in considerations on the ergodic properties of such chains ([15, 17]). In general they did not take into account the fact that the number of transition functions of the process may be finite, which is crucial when dealing with probabilistic automata. We start with some remarks on homogeneous Markov chains, and next we study severall problems of interest concerning non-homogenous Markov chains.

3.1 Recurrence and transience for non-homogeneous Markov chains

Homogeneous Markov chains

We fix $X_i, i \geq 0$ a homogeneous Markov chain on a finite state space S. If $\alpha \in \Delta(S)$, $\mathbb{P}^\alpha$ is the probability distribution on the set of runs on the chain with initial distribution α. Recall, [11], that a state $s \in S$ is called *recurrent* if $\mathbb{P}^s(\{r | s \in Inf(r)\}) > 0$. Otherwise it is called *transient*. Note that we sometimes identify s with the Dirac distribution $\mu_s \in \Delta(S)$ with $\mu_s(s) = 1$.

THEOREM 3. *[Recurrence and the ergodic theorem, [11]] Given a homogeneous Markov chain with finite state space S and $s \in S$, s is recurrent iff $\mathbb{P}^s(\{r | s \in Inf(r)\}) = 1$, iff $\mathbb{P}^s(\{r | s \in Supp(r)\}) > 0$, iff $\mathbb{P}^s(\{r | s \in Supp(r)\}) = 1$.*

Thus, in the homogeneous case, a state s is recurrent if almost all the runs on the chain visit infinitely often s, or equivalently if almost all the runs spend a non negligible amount of time on s. We will see in the next subsection that this equivalence does not hold in the context of non-homogeneous Markov chains. Notice that the notion of finitary liveness of [6]

is not adapted to the probabilistic context. Indeed, even if s is recurrent, on a homogeneous Markov chain, for almost all the runs on the chain, the distance between two consecutive occurrences of s is not bounded.

Non-homogeneous Markov chains

For the following we fix an FPT $\mathcal{T} = (S, \Sigma, \delta, \alpha)$.

Accessibility: a state $s \in S$ is said to be *accessible* in $\mathcal{T}$ if there exists $n \in \mathbb{N}$ and a word $\rho \in \Sigma^n$ such that $\delta(\alpha, \rho)(s) > 0$. That is, with positive probability the process can be in state s after a finite number steps. By simple reachability considerations, we can compute the set $Acc(\mathcal{T})$ of the accessible states in $\mathcal{T}$ in time polynomial in the size of the FPT.

Given a homogeneous Markov chain on S and $s \in S$, theorem 3 shows that $\mathbb{P}^s(\{r | s \in Inf(r)\}) > 0$ iff $\mathbb{P}^s(\{r | s \in Supp(r)\}) > 0$. This is not the case in the context of non-homogeneous Markov chains, which motivates the two following definitions for recurrence.

DEFINITION 4. *[Strong Recurrence, Weak Recurrence] Let $X_n, n \in \mathbb{N}$ be a non homogeneous Markov chain on a finite state space S, and $s \in S$. Let $\mathbb{P}$ be the probability distribution on the set of runs of the chain. We say that s is* weakly recurrent *(resp.* strongly recurrent*), if*

$$\mathbb{P}(\{r | s \in Inf(r)\}) > 0 \text{ (resp. } \mathbb{P}(\{r | s \in Supp(r)\}) > 0)$$

Otherwise, s is said to be weakly transient (resp. strongly transient).

Given an FPT $\mathcal{T} = (S, \Sigma, \delta, \alpha)$, several algorithmic problems may arise, concerning transience and recurrence. The first question is whether we can find $w \in \Sigma^\omega$ such that a given state $s \in S$ is weakly, or strongly, recurrent, for the associated non-homogeneous Markov chain on $\mathcal{T}$.

Problem 1 (Weak recurrence (resp. strong recurrence))
Input: *An FPT $\mathcal{T} = (S, \Sigma, \delta, \alpha)$, $F \subseteq S$.*
Question: *Is there $w \in \Sigma^\omega$ such that*

$$\mathbb{P}^\alpha_w[\{r | F \cap Inf(r) \neq \emptyset\}] > 0. \text{ (resp. } \mathbb{P}^\alpha_w[\{r | F \cap Supp(r) \neq \emptyset\}] > 0).$$

The undecidability of the emptiness problem for PBA$^{>0}$ [2], implies that the weak recurrence problem is undecidable. In contrast, we will see that the strong recurrence problem is PSPACE-complete (theorem 10). We cannot generalize our approach to several states, as we will prove that the following problem is undecidable (theorem 15):

Problem 2 (Two states strong recurrence)
Input: *An FPT $\mathcal{T} = (S, \Sigma, \delta, \alpha)$, $s, t \in S$.*
Question: *Is there $w \in \Sigma^\omega$ s.t. $\mathbb{P}^\alpha_w[\{r | s \in Supp(r) \text{ and } t \in Supp(r)\}] > 0$?*

Consider now the *universal* analog of the weak recurrence problem (resp. of the strong recurrence problem): do we have that for all $w \in \Sigma^\omega$, $\mathbb{P}_w[\{r | s \in Inf(r)\}] > 0$? (resp. $\mathbb{P}_w[\{r | s \in Supp(r)\}] > 0$). By contraposition, these problems can be reformulated as follows.

Problem 3 (Universal weak recurrence (resp. universal strong recurrence))
Input: *An FPT $\mathcal{T} = (S, \Sigma, \delta, \alpha)$, $F \subseteq S$.*
Question: *Is there $w \in \Sigma^\omega$ such that*

$$\mathbb{P}^\alpha_w[\{r | F \cap Inf(r) = \emptyset\}] = 1. \text{ (resp. } \mathbb{P}^\alpha_w[\{r | F \cap Supp(r) = \emptyset\}] = 1).$$

By the results of [1], since $\mathcal{C}_{PBA^{>0}}$ is closed under complementation, the universal weak recurrence problem is undecidable. We will show later that $\mathcal{C}_{CPBA^{>0}}$ is not closed under complementation, hence we cannot conclude directly for the complexity of the universal strong recurrence problem.

The condition $\mathbb{P}_w[\{r|F \cap Inf(r) \neq \emptyset\}] > 0$ (as well as the condition $\mathbb{P}_w[\{r|F \cap Supp(r) \neq \emptyset\}] > 0$), can be seen as a Büchi condition. One can be interested in the co-Büchi condition: a run is accepted if no state in F is visited infinitely often. The associated problems in our context are the following.

Problem 4 (Weak transience (resp. strong transience))
Input: *An FPT* $\mathcal{T} = (S, \Sigma, \delta, \alpha)$, $F \subseteq S$.
Question: *Is there* $w \in \Sigma^\omega$ *such that*

$$\mathbb{P}^\alpha_w[\{r|F \cap Inf(r) = \emptyset\}] > 0. \text{ (resp. } \mathbb{P}^\alpha_w[\{r|F \cap Supp(r) = \emptyset\}] > 0.)$$

The weak transience and strong transience problems are both PSPACE -complete (theorem 14). As before, we can consider the universal versions of these problems.

Problem 5 (Universal weak transience (resp. universal strong transience))
Input: *An FPT* $\mathcal{T} = (S, \Sigma, \delta, \alpha)$, $F \subseteq S$.
Question: *Is there* $w \in \Sigma^\omega$ *such that*

$$\mathbb{P}^\alpha_w[\{r|F \cap Inf(r) \neq \emptyset\}] = 1. \text{ (resp. } \mathbb{P}^\alpha_w[\{r|F \cap Supp(r) \neq \emptyset\}] = 1.)$$

The universal weak and strong transience problems are PSPACE-complete (theorem 12). The following of the section is devoted to the proofs of the complexity of the previous problems.

3.2 Computational complexity of the recurrence problems.

Our decision procedures will often rely on the notion of *probabilistic loop*, which correspond to the set of homogeneous Markov chains that one can define on an FPT.

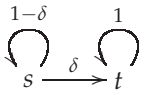
DEFINITION 5.*[Probabilistic loop] A* probabilistic loop *in* $\mathcal{T}$ *is a couple* (C, ρ)*, where* $C \subseteq S$ *and* $\rho \in \Sigma^*$ *are such that* $\delta(C, \rho) \subseteq C$.

If $F \subseteq S$*, a* probabilistic loop around F *in* $\mathcal{T}$ *is a probabilistic loop* (C, ρ) *in* $\mathcal{T}$ *such that for all* $s \in C$*, there exists* ρ'_s *a prefix of* ρ*, such that* $\delta(s, \rho'_s) \cap F \neq \emptyset$.

A probabilistic loop (C, ρ) in $\mathcal{T}$ induces an homogeneous Markov chain $X_n, n \in \mathbb{N}$ with state space C and transitions probabilities given, for all $s, t \in C$, by $\mathbb{P}[X_{n+1} = t | X_n = s] = \delta(s, \rho)(t)$. Let A be the set of states in C which are recurrent for this chain. The *Support* of the loop (C, ρ) is the set of states t in S such that there exists $s \in A$ and ρ' a prefix of ρ with $\delta(s, \rho')(t) > 0$.

We consider first the strong recurrence problem. We fix an instance $\mathcal{T} = (S, \Sigma, \delta, \alpha)$, $F \subseteq S$, of the strong recurrence problem. We can assume that $F = \{s\}$, with no loss on generality. We will prove in this subsection that s is strongly recurrent for a non-homogeneous Markov chain on the probabilistic table iff s is accessible and there exists a probabilistic loop around s in $\mathcal{T}$. This will imply the PSPACE completeness of the strong recurrence problem. The next example shows that this equivalence does not hold in general, if Σ is infinite.

Example 1 Let $S = \{s, t\}$. For $\delta \in]0; 1]$ consider the Markov matrix $M_\delta = \begin{pmatrix} 1-\delta & \delta \\ 0 & 1 \end{pmatrix}$. The graph

of the associated Markov chain is: 

Suppose that the chain is initiated on state s: $\alpha = \{s\}$. Consider now the family of matrices $\mathcal{M} = \{M_{1/2^i}, i \in \mathbb{N}\}$. It is not difficult to see that for any finite product of matrices in $\mathcal{M}$, the associated homogeneous Markov chain $X_n, n \geq 0$ on S is aperiodic and t is the only state in the support of the stationary distribution. By theorem 3, this implies that s is transient for the (homogeneous) chain. This implies that there exists no probabilistic loop around s in $\mathcal{T}$. However, if we consider the non-homogeneous Markov chain $X_n, n \geq 0$ on S whose transitions probabilities are given by the matrices $M_{1/2}, M_{1/2^2}, M_{1/2^3}, ...$, then $\mathbb{P}^{\alpha}_{1/2,1/2^2...}[\{r|\forall n \in \mathbb{N}\ X_n(r) = s\}] > 0$, and in particular $\mathbb{P}^{\alpha}_{1/2,1/2^2...}[\{r|s \in Supp(r) > 0\}] > 0$, which proves that s is strongly recurrent for the (non-homogeneous) chain.

We give a couple of definitions and lemma to prove our theorem. The notion of *filter* will allow us to build a probabilistic loop around a state s by aggregating the successors of this state.

DEFINITION 6. *[Filters] Let S be a finite state space, and Σ be a finite alphabet. A* filter *on S and Σ is a finite sequence of couples on $S \cup \{\cdot\}$ and $\Sigma \cup \{\cdot\}$, where $\cdot$ is a special symbol denoting an "indefinite place".*

A filter can be seen as a word in $((S \cup \{\cdot\})(\Sigma \cup \{\cdot\}))^*$. Two filters x and y will be said to coincide, written $x = y$, if they have the same length and at each place either they have the same elements, or at least one has got an empty place. If u and v are two filters on S and Σ, then uv is the natural concatenated filter: For instance, if $w = a_1...a_l \in \Sigma^*$ and $s \in S$, then (s, w, s) is the filter $(s, a_1), (\cdot, a_2), ..., (\cdot, a_l), (s, \cdot)$.

We start with a combinatorial lemma. The proportion $prop(w, r)$ of a filter w in a run r is naturally defined the same way as we defined the proportion of a subword in a run, using a limit-sup.

LEMMA 7. *Let S be a finite state space and Σ be a finite alphabet. Let r be a run on S and Σ, and let u be a filter on S and Σ. Suppose $prop(u, r) > 1/N$, where $N \in \mathbb{N}$ and $N > |u|$. Then there exists another filter v on S and Σ such that $prop(uvu, r) > 1/(2 \cdot N)$. Moreover, we can choose v such that $|v| \leq 2 \cdot N$.*

We will apply recursively the following lemma to build a probabilistic loop around s.

LEMMA 8. *Let $\rho \in \Sigma^*$. Suppose $\mathbb{P}^{\alpha}_{w}[\{r|prop((s, \rho), r) > 0\}] > 0$, and let $t \in \delta(s, \rho)$. Then, there exists $\rho' \in \Sigma^*$ such that:*

$$s \in \delta(t, \rho'), \text{ and } \mathbb{P}^{\alpha}_{w}[\{r|prop((s, \rho\rho'), r) > 0\}] > 0.$$

THEOREM 9. *Let $\mathcal{T} = (S, \Sigma, \delta, \alpha)$, $s \in S$, be an instance of the strong recurrence problem. Then the following are equivalent:*

- *There exists $w \in \Sigma^\omega$ such that s is strongly recurrent for the associated non-homogeneous Markov chain on $\mathcal{T}$.*
- *s is accessible, and there exists a probabilistic loop around s in $\mathcal{T}$.*

Moreover, in the positive case, the letters of the trace of the loop can all be taken in the support of w.

PROOF. (sketch) Notice that one way is easy: if there exists $\rho_0 \in \Sigma^n$ such that $\delta(\alpha, \rho_0)(s) > 0$ and if there exists a probabilistic loop (C, ρ) around s, then $\mathbb{P}^{\alpha}_{\rho_0 \cdot \rho^\omega}(\{r | s \in Supp(r)\}) > 0$, and s is strongly recurrent for the chain associated to $w = \rho_0 \cdot \rho^\omega$.

We prove now that the strong recurrence problem is PSPACE complete. First, we know that we can compute in PTIME if s is accessible from α. Thus, the strong recurrence problem is PTIME equivalent to the problem of finding if there exists a probabilistic loop around s. We reduce the problem of Finite Intersection of Regular Languages, which is known to be PSPACE complete [12], to our strong recurrence problem.

Problem 6 (Finite Intersection of Regular Languages)
Input: $\mathcal{A}_1, ..., \mathcal{A}_l$ a family of deterministic automata (on finite words) on the same finite alphabet Σ.
Question: Do we have $\mathcal{L}(\mathcal{A}_1) \cap ... \cap \mathcal{L}(\mathcal{A}_l) = \emptyset$?

THEOREM 10. *The strong recurrence problem is PSPACE complete.*

We consider now the complexity of the co-Büchi problems.

PROPOSITION 11. *Let $\mathcal{T} = (S, \Sigma, \delta, \alpha)$ be an FPT, and $F \subseteq S$. Then the following are equivalent:*

1. *$\exists w \in \Sigma^\omega$ s.t. $\mathbb{P}^{\alpha}_{w}[\{r | F \cap Inf(r) = \emptyset\}] > 0$.*
2. *$\exists w \in \Sigma^\omega$ s.t. $\mathbb{P}^{\alpha}_{w}[\{r | F \cap Supp(r) = \emptyset\}] > 0$.*
3. *There exists an accessible probabilistic loop on S whose support does not contain any state in F*

THEOREM 12. *The universal weak and strong transience problems (problem 5) are PSPACE complete.*

PROOF. As for the strong recurrence problem, we can build a nondeterministic Turing machine which finds a relevant probabilistic loop in PSPACE. For the PSPACE hardness, we can also reduce the finite intersection of regular languages problem to these problems.

PROPOSITION 13. *Let $\mathcal{T} = (S, \Sigma, \delta, \alpha)$ be an FPT, and $F \subseteq S$. Then the following are equivalent:*

1. *$\exists w \in \Sigma^\omega$ s.t. $\mathbb{P}_{w}[\{r | F \cap Inf(r) \neq \emptyset\}] = 1$.*
2. *$\exists w \in \Sigma^\omega$ s.t. $\mathbb{P}_{w}[\{r | F \cap Supp(r) \neq \emptyset\}] = 1$.*
3. *There exists ρ_0 and ρ in Σ^* such that $(\delta(\alpha, \rho_0), \rho)$ is a probabilistic loop around F.*

PROOF. $3 \Rightarrow 2$ and $2 \Rightarrow 1$ are simple. Suppose 1: $\exists w \in \Sigma^\omega$ s.t. $\mathbb{P}^{\alpha}_{w}[\{r | F \cap Inf(r) \neq \emptyset\}] > 0$. Write $w = a_1 a_2 ....$ For $i \in \mathbb{N}$, let $H_i = \delta(\alpha, w_{|i}) = \bigcup_{s | \alpha(s) > 0} \delta(s, w_{|i})$.

Since S is finite, there exists $H \subseteq S$ such that infinitely often, $H_i = H$. Let $i_0 \in \mathbb{N}$ such that $H_{i_0} = H$. Let $t \in H$. Then $\mathbb{P}^{\alpha}_{w}[\{r | X_{i_0}(r) = t\}] > 0$. Since $\mathbb{P}^{\alpha}_{w}[\{r | F \cap Inf(r) \neq \emptyset\}] = 1$, F must be reachable from t after a finite number of steps. That is, there exists $l_t \in \mathbb{N}$ such that $\delta(t, a_{i_0+1} a_{i_0+2} ... a_{i_0+l_t})(F) > 0$. Let $l_0 = Max_{t \in H} l_t$, and $l \geq l_0$ such that $\delta(s, w_{|i_0+l}) = H$. Then $\rho_0 = w_{|i_0}$ and $\rho = a_{i_0+1}, ..., a_{i_0+l}$ satisfy the conditions of 3.

THEOREM 14. *The weak transience and strong transience problems (problem 4) are PSPACE complete.*

PROOF. The proof of the fact that these problems are in PSPACE is the same as for the strong recurrence problem: a nondeterministic Turing machine can guess ρ_0 and ρ and verify in PSPACE the requirements. Concerning the PSPACE hardness, we point out that the

exact same reduction as for the strong recurrence problem is also a reduction for the Intersection of Regular Languages problem to our problem.

We can reduce the emptiness problem for an automaton in $PBA^{>0}$ to problem 2:

THEOREM 15. *Problem 2 is undecidable.*

4 Probabilistic automata

In this section we study our new classes of constrained probabilistic automata using the results from the previous section. We start our discussion with the class $CPBA^{>0}$. As a CPBA is structurally an FPT plus a set of final states, we can use the results of the last section, and the notion of probabilistic loop. A probabilistic loop on a CPBA will be *accepting* if its support contains an accepting state. For the following, we fix a CPBA $\mathcal{A} = (\mathcal{T}, F)$, where $\mathcal{T} = (S, \Sigma, \delta, \alpha)$ is an FPT and $F \subseteq S$. The past section yields the following theorem.

THEOREM 16. *The following are equivalent:*

1. $\mathcal{L}^{>0}(\mathcal{A}) \neq \emptyset$.
2. *$\mathcal{A}$ accepts a lasso shape word.*
3. *There exists an accessible and accepting probabilistic loop on $\mathcal{A}$.*

PROOF. $1 \Leftrightarrow 3$ comes from theorem 9. $2 \Rightarrow 1$ is direct. Suppose 3. If $x \in \Sigma^*$ is such that $\delta(\alpha, x)(s) > 0$ and $y \in \Sigma^*$ is the trace of the loop, the word $x \cdot y^\omega$ is a lasso shape word and belongs to the language of the automaton.

COROLLARY 17. *The emptiness problem of the language of a CPBA with the probable semantics is PSPACE complete.*

PROPOSITION 18. *$\mathcal{C}_{CPBA^{>0}}$ is not closed under complementation.*

In particular, the proof shows that the set of ω-regular languages is not a subset of the set of languages definable by automata in $CPBA^{>0}$. The following proposition, using a construction of [10], shows that the class of the complements of languages of automata in $PBA^{=1}$ is a subset of the class of languages recognized by automata in $CPBA^{>0}$.

PROPOSITION 19. *If $\mathcal{A} \in PBA$, there exists $\mathcal{A}' \in CPBA$ such that $|\mathcal{A}'| \leq |\mathcal{A}| + 1$ and $\mathcal{L}^{>0}(\mathcal{A}') = \mathcal{L}^{=1}(\mathcal{A})^c$. Moreover, the inclusion $\{\mathcal{L}^{=1}(\mathcal{A})^c | \mathcal{A} \in PBA\} \subseteq \mathcal{C}_{CPBA^{>0}}$ is strict.*

As a corollary, since the emptiness of the language of a CPBA$^{>0}$ can be decided in PSPACE, this proves that the universality problem of the language of an automaton in $PBA^{=1}$ can be decided in PSPACE. The following proposition shows that the emptiness problems on the classes $PBA^{=1}$ and $CPBA^{=1}$ are equivalent. Given a probabilistic automaton $\mathcal{A}$ with final states set F, we can consider the language $\mathcal{L}^{PBA^{=1}}(\mathcal{A})$ of the set of words accepted by $\mathcal{A}$ when $\mathcal{A}$ is considered in $PBA^{=1}$, and also the language $\mathcal{L}^{CPBA^{=1}}(\mathcal{A})$ of the set of words accepted by $\mathcal{A}$ when $\mathcal{A}$ is considered in $CPBA^{=1}$.

PROPOSITION 20. *Let $\mathcal{A}$ be a probabilistic automaton with final states set F. Then:*
- $\mathcal{L}^{PBA^{=1}}(\mathcal{A}) = \emptyset$ *iff* $\mathcal{L}^{CPBA^{=1}}(\mathcal{A}) = \emptyset$.
- $\mathcal{L}^{PBA^{=1}}(\mathcal{A}) = \Sigma^\omega$ *iff* $\mathcal{L}^{CPBA^{=1}}(\mathcal{A}) = \Sigma^\omega$.

PROOF. Follows directly from propositions 11 and 13, as the condition on the probabilistic loop around the final state set is a structural condition, which does not depend on considerations on Inf or Supports sets of the runs.

By theorem 12 and theorem 14, the complexity of the emptiness problem and the universality problem of the language of an automaton in $CPBA^{=1}$ or in $PBA^{=1}$ is in PSPACE. This improves the previous results of [1] which showed using different tools that the emptiness problem for the language of an automaton in $PBA^{=1}$ is in EXPTIME. Note that the upcoming paper [5] shows PSPACE-completeness for the emptiness problem and the universality problem of the language of an automaton in $PBA^{=1}$.

PROPOSITION 21. *The class of languages $\mathcal{C}_{PBA^{=1}}$ is a subclass of $\mathcal{C}_{CPBA^{=1}}$, and the inclusion is strict.*

PROOF. The inclusion follows from the construction of a layered automaton, as in proposition 19. We can show that $\{w | a \in Supp(w)\} \in \mathcal{C}_{CPBA^{=1}} - \mathcal{C}_{PBA^{=1}}$.

We have seen that in contrast to (classical) probabilistic automata, for constrained probabilistic automata, the emptiness problem for Büchi acceptance under the probable semantics becomes decidable. However, for Street, resp. Muller acceptance condition, the emptiness problem for the probable semantics is undecidable. Surprisingly, for Rabin (and thus parity) acceptance, we can prove as for theorem 10 that the problem is decidable.

THEOREM 22. *The emptiness problem for CPSA, resp. CPMA, under the probable semantics is undecidable.*

PROOF. With $\mathsf{Acc} = \{(\{s\}, S), (\{t\}, S)\}$, resp. $\mathsf{Acc} = \{T : \{s, t\} \subseteq T \subseteq S\}$, problem 2 (two states strong recurrence) reduces to the emptiness problem for CPSA$^{>0}$, resp. CPMA$^{>0}$. As theorem 15 shows the undecidability of problem 2, the claim follows.

THEOREM 23. *The emptiness problem for CPRA under the probable semantics is decidable.*

Remarks: If we use the alternative definition for the support of a run, such that a state s is in the support of a run if the *Inf* limit of the time spent on s is non zero, we get different classes of automata, with different languages. However, the emptiness problem and all the natural problems can still be solved using the same tools. For instance, the language of an associated PCA automaton is still non empty iff there exists an accessible and accepting probabilistic loop. Thus, the complexity of the problems we studied does not change.

5 Conclusion

This paper presents an alternative definition to the the classical "infinitely often" Büchi condition. We presented several notions of recurrence and transience on finite probabilistic tables and gave the precise computational complexity of several of the associated problems. We used these results to prove the decidability of basic problems on new classes of

probabilistic automata on infinite words. Several theoretical questions are still open, e.g., the complexity of the universal strong recurrence problem. The possibility to find classes of probabilistic automata on which the basic problems such as the emptiness problem are computable, and which may be used to specify relevant properties in a system verification context, could motivate future work. Another issue is in the context of infinite duration games, where we can change the classical ω-regular condition of [8], or the extensions of [6], by our notion of acceptance.

Bibliography

[1] C. Baier, N. Bertrand, and M. Grösser. On Decision Problems for Probabilistic Büchi Automata. In *Proc. of FOSSACS '08*, volume 4962 of *LNCS*. Springer, 2008.

[2] C. Baier and M. Grösser. Recognizing omega-regular Languages with Probabilistic Automata. In *Proc. of LICS '05*, pages 137–146. IEEE CS Press, 2005.

[3] M. Bojanczyk and T. Colcombet. Bounds in omega-regularity. In *Proc. of LICS '06*, pages 285–296. IEEE CS Press, 2006.

[4] D. Bustan, S. Rubin, and M.Y. Vardi. Verifying omega-Regular Properties of Markov Chains. In *Proc. of CAV '04*, volume 3114 of *LNCS*, pages 189–201. Springer, 2004.

[5] R. Chadha, A.P. Sistla, and M. Viswanathan. Power of randomization in automata on infinite strings. In *to appear in Proc. of the 20th International Conference on Concurrency Theory (CONCUR'09)*, Lecture Notes in Computer Science. Springer, 2009.

[6] K. Chatterjee, T.A. Henzinger, and F. Horn. Stochastic Games with Finitary Objectives.

[7] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[8] L. de Alfaro and TA Henzinger. Concurrent omega-regular games. In *Logic in Computer Science, 2000. Proceedings. 15th Annual IEEE Symposium on*, pages 141–154, 2000.

[9] E. Grädel, W. Thomas, and T. Wilke. Automata, Logics, and Infinite Games: A Guide to Current Research, volume 2500 of LNCS. *Notes in Comp. Sci. Springer*, 2002.

[10] M. Größer. *Reduction Methods for Probabilistic Model Checking*. PhD thesis, Technische Universität Dresden, 2008.

[11] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer-Verlag, 1983.

[12] D. Kozen. Lower bounds for natural proof systems. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 254–266, 1977.

[13] V.G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall/CRC, 1995.

[14] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: specification*. Springer.

[15] A. Paz. Ergodic theorems for infinite probabilistic tables. *Ann. Math. Statist*, 41:539–550, 1970.

[16] A. Paz. *Introduction to probabilistic automata*. Academic Press New York, 1971.

[17] E. Seneta. *Non-Negative Matrices and Markov Chains*. Springer, 2006.

[18] M.Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Foundations of Computer Science, 1984., 26th Annual Symposium on*, pages 327–338, 1985.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Structure and Specification as Sources of Complexity

Anuj Dawar
University of Cambridge Computer Laboratory
anuj.dawar@cl.cam.ac.uk

1 Introduction

Computational complexity is often described as the study of what makes certain computational problems inherently difficult to solve. Of course, it has proved to be extremely difficult to establish unconditional lower bounds, but the theory has provided us with important tools for identifying intractable problems. If one were to pick out the most important contribution that complexity theory has made to the theory and practice of computing, it is arguably in introducing the notion of NP-completeness. The ability to identify NP-complete problems and to construct reductions are skills that are taught to virtually all students of computer science. However, while thousands of problems have been identified as NP-complete, and we have a strong, if informal, understanding of what makes a problem hard, this does not amount to a *theory* of complexity. We understand that an exponential, unstructured search space leads to difficulty, but we do not have an account of what kind of structure in the search space allows for tractable solutions. This is a distinct problem from our inability to prove lower bounds, i.e. to explain why NP-complete problems are truly intractable. It is the problem of explaining what makes certain problems NP-complete in the first place. It may even be argued that, at this point, we do not know what such a theory of difficulty might look like.

In this talk, I review results from descriptive complexity that relate to this issue. The best known results of descriptive complexity are about the characterisations of complexity classes in terms of logical definability. I would argue that one important contribution of these results is the separation they provide between the *specification* of a decision problem and the *structure* against which this specification is checked. The first is usually formalised as a sentence in some suitable formal logic, while the latter is usually a relational structure of some kind. This separation allows some insight into sources of complexity. One can measure the richness of the language in which specifications are written and one can measure the density of the structures considered. These are two aspects of work in descriptive complexity that I will consider. In these notes to accompany the talk, I briefly present some definitions and the main results. Many of these are historical, but I take them up to recent work and provide pointers to the literature. After presenting some background and definitions I briefly consider the complexity of specification languages in Section 2 and of structures at some length in Section 3. The former leads to some recent work on the question of characterisations of P, while the latter leads to connections with paramterized complexity.

© Dawar; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 407–416
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2336

2 Complexity of Specification

The general situation we consider is of a problem where an instance is a structure (such as a graph) and the problem to be decided is given by a formula in some logic (typically an extension of first-order logic). Indeed, in the examples I consider in this paper, I confine myself to decision problems on graphs. Consider, for example, some classical NP-complete problems on graphs: INDEPENDENT SET, DOMINATING SET, 3-COLOURABILITY and HAMILTONICITY. In the first two cases, the input is a graph together with an integer parameter, while in the second case it consists of a graph alone. As we shall see, it matters whether we consider the integer parameter to be part of the specification of the problem, or the instance.

Suppose then that we are given a graph G and a formula φ of first-order logic in the language with one binary relation. How hard is it to decide whether $G \models \varphi$? There are essentially two versions of this question that interest us here (called the *data complexity* and the *combined complexity* of first-order logic, respectively by Vardi in [34]).

In the first, we ask how complex can be the set of graphs that satisfy a fixed first-order sentence. The answer is that it is always decidable in logarithmic space by a straightforward algorithm (and, indeed the set is in fact in AC^0 [1]). Moreover, there are problems in L which one can easily show are not definable by any first-order sentence. In particular, there is no sentence that defines the graphs with an even number of vertices or the connected graphs (see [15, 26] for proofs). It is also not difficult to show that the Hamiltonian graphs, or the 3-colourable graphs are *provably* not first-order definable. The conclusion one can draw from this is that the expressive power of first-order logic is rather weak. This is one reason that research in finite model theory has focused on extensions of the logic.

On the other hand, it is easy to write, for each k a first-order sentence that defines the graphs that contain an independent set of k vertices, or a dominating set with k vertices. Thus, if one considers the combined complexity of first-order logic, i.e. the following decision problem: given a graph G and a first-order formula φ, determine whether $G \models \varphi$, then it is clearly hard. In fact, the problem is PSpace-complete. In terms of parameterized complexity, taking the length of φ as parameter, the problem is AW[*]-complete. Moreover, restricting the first-order sentences to a fixed-number of quantifier alternations yields complete problems at every level of the W-hierarchy and thus the problem of evaluating first-order sentences in graphs is central to parameterized complexity. I return to connections with parameterized complexity in the next section. Futher details may also be found in the excellent text [19].

Searching for a specification language more expressive than first-order logic, the logician may turn first to second-order logic. Here, it is known since the work of Fagin [17] that the existential fragment is rich enough to express all (and only) the problems in NP. It follows that second-order logic expresses all decision problems in the polynomial hierarchy [32]. From the complexity-theoretic point of view, the interesting logics are intermediate in expressive power between first and second-order logic. In particular, it remains an open question whether there is a logic that expresses exactly the polynomial-time decidable properties of graphs.

Immerman [25] and Vardi [34] showed that LFP, the extension of first-order logic with inductive definitions expesses exactly the polynomial-time properies of *ordered* graphs but

this is too weak in general. An extension of LFP with a mechanism for counting was proposed by Immerman, but shown to be too weak in [3]. Since then, a number of further logic have been proposed that all properly extend the expressive power of LFP with counting and for which it remains an open question whether they can express all polynomial-time properties. They include the language of *choiceless polynomial-time with counting* of Blass, Gurevich and Shelah [2] and the language of *specified symmetric choice* of Gire and Hoang [22, 12]. A significant recent development in this direction is the proposal to extend LFP with linear algabraic operators [8]. The mutual interrelationship between these various extensions also remains to be explored (see [13] for related results). A useful recent survey on the problem of characterising P is given by Grohe in [24].

3 Resticted Graph Classes

We now turn our attention to the combined complexity of first-order logic and to the question about how constraints on the *structure* can limit the search space and make hard problems tractable. As mentioned above, the problem of deciding, given a graph G and a first-order sentence φ whether $G \models \varphi$ is PSpace-complete, while for any fixed φ, the class of graphs that satisfy it is in L. To be more precise, if φ has length l and m distinct variables and G is a graph on n vertices, then $G \models \varphi$ can be decided in time $\mathcal{O}(ln^m)$ and space $\mathcal{O}(m \log n)$. In [33], Stolboushkin and Taitslin asked whether there is a constant c such that every first-order sentence defines a problem decidable in time $\mathcal{O}(n^c)$. They conjectured that this was not the case and noted that a proof of the conjecture would imply a separation of P from PSpace. A more uniform version of their question would ask for a computable function that maps φ to a $\mathcal{O}(n^c)$ clocked algorithm for deciding the models of φ. The existence of such a function would imply that the problem of deciding whether $G \models \varphi$ was fixed-parameter tractable. Since this problem is AW[*]-complete (see [19] for details) this would imply the collapse of the edifice of parameterized complexity.

Indeed, many natural problems that are hard from the point of view of parameterized complexity can be naturally formulated in first-order logic. As an example, consider two problems mentioned above: INDEPENDENT SET and DOMINATING SET. They are complete for $W[1]$ and $W[2]$ respectively and, as noted above, naturally expressed by a (parameter-dependent) first-order formula.

A subject of intensive investigation in recent years has been the fixed-parameter tractability of otherwise hard problems, when the class of input graphs is restricted. A typical example is the fixed-paramter tractability of DOMINATING SET when restricted to planar graphs. Indeed, for many interesting restrictions on graphs, one can show that first-order satisfaction is itself fixed-parameter tractable and as a result the tractability of a whole host of other individual problems follows. In the rest of this section, we briefly survey results that establish the fixed-parameter tractability of first-order satisfaction on a number of such classes. The classes we examine are all classes of *sparse* graphs. That is, though the classes may be defined in other terms, they have the property that the number of edges in a graph in the class as a function of the number of vertices does not grow very fast. It should be remarked that there are other classes of graphs (such as those of bounded cliquewidth) which are not sparse in this sense, but where it is known that first-order logic (and, indeed, even monadic

second-order logic) admit fixed-parameter tractable algorithms for the satisfaction problem (see [6, 5]).

Sparse Classes The relationships between various classes of sparse graphs that have been studied are depicted in Figure 1.

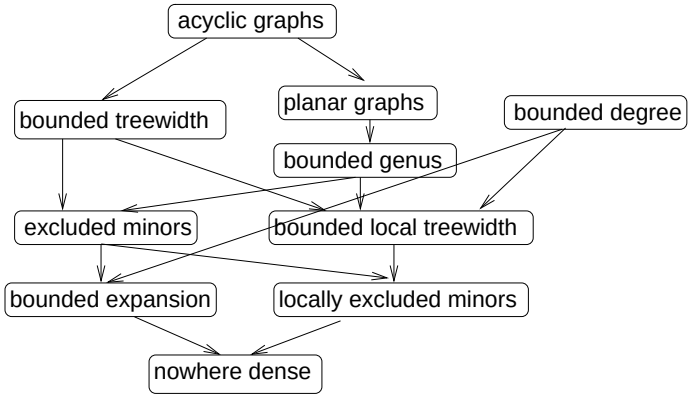


Figure 1: Relationships between sparse graph classes.

Among the restrictions given in Figure 1, those of acyclicity and planarity are of a different character to the others in that they apply to single graphs. We can say of graph G that it is acyclic or planar. When we apply this restriction to a class $\mathcal{C}$, we mean that all structures in the class satisfy it. The other conditions in the figure only make sense in relation to classes of graphs. Thus, it makes little sense to say of a single finite graph that it is of bounded degree (it is necessarily so). When we say of a class $\mathcal{C}$ that it is of bounded degree, we mean that there is a uniform bound on the degree of all graphs in $\mathcal{C}$.

The arrows in Figure 1 should be read as implications. Thus, any graph that is acyclic is necessarily planar. Similarly, any class of acyclic graphs has bounded treewidth. The arrows given in the figure are *complete* in the sense that when two restrictions are not connected by an arrow (or sequence of arrows) then the first does not imply the second and separating examples are known in all such cases.

The restrictions of acyclicity, planarity and bounded degree are self-explanatory. We say that a class of graphs $\mathcal{C}$ has bounded genus if there is a fixed orientable surface S such that all graphs in $\mathcal{C}$ can be embedded in S (see [27]). In particular, as planar graphs are embeddable in a sphere, any class of planar graphs has bounded genus. The treewidth of a graph is a measure of how tree-like it is (see [14]). In particular, trees have treewidth 1, and so any class of acyclic graphs has treewidth bounded by 1. The measure plays a crucial role in the graph structure theory developed by Robertson and Seymour in their proof of the graph minor theorem. We say that a graph G is a minor of H (written $G \preceq H$) if G can be obtained from a subgraph of H by a series of edge contractions (see [14] for details). We say that a class of graphs $\mathcal{C}$ excludes a minor if there is some G such that for all $H \in \mathcal{C}$ we have $G \not\preceq H$. In particular, this includes all classes $\mathcal{C}$ which are closed under taking minors and which do not include all graphs. If G is embeddable in a surface S then so are all its minors. Since, for any fixed integer k, there are graphs that are not of genus k, it follows that

any class of bounded genus excludes some minor.

The notion of bounded local treewidth was introduced as a common generalisation of classes of bounded treewidth and bounded genus. A variant, called the diameter width property was introduced in [16] while bounded local treewidth is from [20]. Recall that the r-neighbourhood of a vertex v in a graph G, denoted $N^r_G(v)$, is the subgraph of G induced by the set of vertices at distance at most r from v. We say that a class of graphs $\mathcal{C}$ has bounded local treewidth if there is a nondecreasing function $t : \mathbb{N} \to \mathbb{N}$ such that for any graph $G \in \mathcal{C}$, any vertex v in G and any r, the treewidth of $N^r_G(v)$ is at most $t(r)$. It is clear that any class of graphs of bounded treewidth has bounded local treewidth (indeed, bounded by a constant function t). Similarly, any class of graphs of degree bounded by d has local treewidth bounded by the function d^r, since the number of elements in $N^r_G(v)$ is at most d^r. The fact that classes of bounded genus also have bounded local treewidth follows from a result of Eppstein [16].

We say that a class of graphs $\mathcal{C}$ locally excludes minors if there is a nondecreasing function $t : \mathbb{N} \to \mathbb{N}$ such that for any graph $G \in \mathcal{C}$, any vertex v in G and any r, the clique $K_{t(r)}$ is not a minor of the graph $N^r_G(v)$. This notion is introduced in [9] as a natural common generalisation of bounded local treewidth and classes with excluded minors. Classes of graphs with bounded expansion were introduced by Nešetřil and Ossona de Mendez [30] as a common generalisation of classes of bounded degree and proper minor-closed classes. A class of graphs $\mathcal{C}$ has bounded expansion if there is a function $t : \mathbb{N} \to \mathbb{N}$ such that for any graph $G \in \mathcal{C}$, any subgraph H of G and any minor H' of H obtained from H by contracting neighbourhoods of radius at most r, the average degree in H' is bounded by $t(r)$. In particular, classes that exclude a minor have bounded expansion witnessed by a constant function f.

Finally, we say that a class $\mathcal{C}$ of graphs is *nowhere dense* if there is a function $t : \mathbb{N} \to \mathbb{N}$ such that for each r, the graph $K_{t(r)}$ cannot be obtained as a minor of any $G \in \mathcal{C}$ by contracting neighbourhoods of radius at most r. This notion is introduced by Nešetřil and Ossona de Mendez in [28, 29]. They present convincing arguments to show that this is the natural upper limit to well-behaved classes of graphs based on sparseness conditions.

Automata and Locality The following is a sampling of results on the fixed-parameter tractability of the first-order satisfaction problem on classes of sparse graphs. In each of these, l is the length of the formula φ, n is the size of the graph G and f is some computable function.

1. If $\mathcal{T}_k$ is the class of graphs of treewidth at most k, then $G \models \varphi$ is decidable in time $\mathcal{O}(f(l)n)$. Indeed this is true not just for first-order φ but even in monadic second-order logic by [4].
2. If $\mathcal{D}_k$ is the class of graphs of degree at most k, then $G \models \varphi$ is decidable in time $\mathcal{O}(f(l)n)$. This is established by Seese in [31].
3. If LTW_t is the class of graphs of local treewidth bounded by a function t, then $G \models \varphi$ is decidable in time $\mathcal{O}(f(l)n^2)$ by a result of Frick and Grohe [20].
4. If $\mathcal{M}_k$ is the class of graphs excluding K_k as a minor, then $G \models \varphi$ is decidable in time $\mathcal{O}(f(l)n^5)$ by results of Flum and Grohe [18].

5. If LEM_t is the class of graphs with locally excluded minors given by t, then $G \models \varphi$ is decidable in time $\mathcal{O}(f(l)n^6)$ by a result of Dawar et al. [9].

These results are established by a combination of two essential methods. One is sometimes called the method of *automata* or the method of *decompositions*. The other is based on the *locality* of first-order logic. These two basic methods are best illustrated by the first two results on the list above.

For two graphs G and H and tuples of vertices $\mathbf{u}$ and $\mathbf{v}$ we write $(G, \mathbf{u}) \equiv_m (H, \mathbf{v})$ to denote that any formula $\varphi(\mathbf{x})$ with quantifier depth at most m is true of $\mathbf{u}$ in G if, and only if, it is true of $\mathbf{v}$ in H. Two key facts about this equivalence relation are (1) that, for any fixed m and fixed length of tuple, it has finite index and (2) that it is a congruence with respect to a certain gluing operation. That is, if $\mathbf{v}$ is a tuple of vertices inducing the same subgraph in both G and H, let $G \oplus_{\mathbf{v}} H$ denote the graph obtained by taking the disjoint union of G and H while identifying the vertices in $\mathbf{v}$. Then, it can be shown that the $\equiv_m$ equivalence class of $(G \oplus_{\mathbf{v}} H, \mathbf{v})$ is determined by the classes of $(G, \mathbf{v})$ and $(H, \mathbf{v})$ respectively. Since graphs in $\mathcal{T}_k$ can be constructed from a finite collection of graphs (i.e. the graphs with at most k vertices) using this gluing operation (and some vertex renaming operations needed for technical reasons), we can use dynamic programming to determine the $\equiv_m$-class of an arbitrary graph in $\mathcal{T}_k$ in linear time from its tree decomposition.

Abstractly, the method of decompositions can be formulated as follows. Suppose $\mathcal{C}$ is a class of graphs such that there is a finite class $\mathcal{B}$ and a finite collection of operations Op such that:

- $\mathcal{C}$ is contained in the closure of $\mathcal{B}$ under the operations in Op;
- there is a polynomial-time algorithm which constructs, given any $G \in \mathcal{C}$ an Op-decomposition of G over $\mathcal{B}$; and
- for each m, the equivalence relation $\equiv_m$ is an *effective congruence* with respect to all the operations $o \in$ Op (by which we mean that the $\equiv_m$ class of $o(G_1, \ldots, G_s)$ can be computed from the classes of $G_1, \ldots, G_s$),

then, satisfaction of first-order formulas for graphs in $\mathcal{C}$ is fixed-parameter tractable.

More generally, instead of requiring $\mathcal{B}$ to be finite, it suffices that first-order satisfaction is itself fixed-parameter tractable on $\mathcal{B}$. Indeed, result (4) above, on classes of graphs that exclude a minor, is obtained by considering a tree-decomposition of graphs in such a class *over* a class of bounded local treewidth and then using the result (3).

Another possible relaxation of the method is to replace $\equiv_m$ by some other sequence $\sim_m$ of congruence relations. The properties required to make this work are that for every first-order formula φ there is an m such that φ is invariant under $\sim_m$ and that for each m, $\sim_m$ is a relation of finite index. In this context, it should be noted that taking $G \sim_m H$ to denote that G and H cannot be distinguished by any formula of *length* at most m does not yield a congruence relation even with respect to disjoint union. Indeed, it was shown in [10] that there is no elementary function e such that $G_1 \sim_{e(m)} H_1$ and $G_2 \sim_{e(m)} H_2$ implies $G_1 \oplus G_2 \sim_m H_1 \oplus H_2$.

In contrast, the proof of result (2) above is based on the *locality* of first-order logic. This property essentially says that the truth of a formula φ in a graph G can be determined by examining local neighbourhoods inside G. A precise statement is given by Gaifman's locality theorem [21] the statement of which requires some definitions.

For every integer $r \geq 0$, let $\delta(x,y) \leq r$ denote the first-order formula expressing that the distance between x and y in the Gaifman graph is at most r. Let $\delta(x,y) > r$ denote the negation of this formula. Note that the quanfier rank of $\delta(x,y) \leq r$ is bounded by r. A *basic local sentence* is a sentence of the form

$$(\exists x_1) \cdots (\exists x_n) \left(\bigwedge_{i \neq j} \delta(x_i, x_j) > 2r \wedge \bigwedge_i \psi^{N^r(x_i)}(x_i) \right), \tag{1}$$

where ψ is a first-order formula with one free variable. Here, $\psi^{N^r(x_i)}(x_i)$ stands for the relativization of ψ to $N^r(x_i)$; that is, the subformulas of ψ of the form $(\exists x)(\theta)$ are replaced by $(\exists x)(\delta(x,x_i) \leq r \wedge \theta)$, and the subformulas of the form $(\forall x)(\theta)$ are replaced by $(\forall x)(\delta(x,x_i) \leq r \rightarrow \theta)$.

THEOREM 1.*[Gaifman Locality] Every first-order sentence is equivalent to a Boolean combination of basic local sentences.*

We call the Boolean combination of basic local sentences that is equivalent to a given first-order sentence φ a *Gaifman normal form* of φ. Since the proof of Theorem 1 (see for instance [15, Thm 2.5.1]) gives an effective construction of the Gaifman normal form from φ, to prove (2), it suffices to consider how a basic local sentence can be evaluated. Since, in a graph of bounded degree, there is a bound on the size of neighbourhoods, we can easily (in linear time) label elements by whether or not they satisfy the formulas $\psi^{N^r(x_i)}(x_i)$. The problem then reduces to determining in a vertex-coloured graph whether there is a large enough r-scattered set of a given colour. This can be done easily enough on graphs of bounded degree. However, Frick and Grohe [20] show that this can be solved in a somewhat more general setting giving an abstract method of locality. See [23, Sec. 4] for a very readable account.

The abstract formulation of the method of locality is as follows. Suppose we have a function, associating an integer parameter k_G with each graph G. Suppose further that we have an algorithm which, given a graph G and a formula φ decides $G \models \varphi$ in time $g(l, k_G)n^c$ for some computable g and some constant c. Finally, let $\mathcal{C}$ be a class of graphs of *bounded local k*. That is, there is a computable function t such that for every $G \in \mathcal{C}$ and every vertex v in G, $k_{N^r_G(v)} < t(r)$. Then, there is an algorithm which decides $G \models \varphi$ in time $f(l)n^{c+1}$ for some computable f.

It is this general localisation principle that gives us (3) from (1) above. It may seem that (5) follows from (4) by a similar application of the method of locality. However, while the result in [18] gives, for each k, a fixed-parameter tractable algorithm for deciding $G \models \varphi$ for classes that exclude K_k as a minor, it is not clear from the proof that the parameter dependence is computable from k. The proof relies on Robertson-Seymour decompositions which do not yield computable bounds. Thus, the result in [9] relies on rather different decompositions.

Nowhere-Dense Classes As of this writing, it remains an open question whether the fixed-parameter tractability of first-order satisfaction can be pushed beyond the classes of locally excluded minors. In particular, the box at the bottom of Figure 1, containing the

nowhere-dense classes, is an interesting case. This property was identified by Nešetřil and Ossona de Mendez in [28, 29]. They show that it is closely related to a property of classes of graphs called *quasi-wideness* in [7]. They give strong evidence that this property is the natural limit for methods which rely on the sparsity of graphs. To be precise, they associate the following parameter with any infinite class $\mathcal{C}$ of graphs.

$$d_{\mathcal{C}} = \lim_{r\to\infty} \limsup_{G\in\mathcal{C}_r} \frac{\log ||G||}{\log |G|},$$

where $\mathcal{C}_r$ denotes the collection of graphs that can be obtained as minors of a graph in $\mathcal{C}$ by contracting neighbourhoods of radius at most r. As usual, $||G||$ and $|G|$ denote the number of edges and the number of vertices in G respectively. The remarkable result they then prove is what they call the *trichotomy theorem* [29] which states that $d_{\mathcal{C}}$ only takes values 0, 1 and 2. Moreover, the nowhere-dense classes are exactly the ones where it does not take value 2.

So, could it be that first-order satisfaction is fixed-parameter tractable on all nowhere-dense classes? The connection with quasi-wideness provides some clues. It is easy to establish that problems such as INDEPENDENT SET are fixed-parameter tractable on such classes. A paper in the present volume [11] shows that variations on the DOMINATING SET problem are also fixed-parameter tractable. However, it remains a challenge to extend this to all first-order definable properties. In particular, such a result would generalise the tractability of first-order logic on excluded minor classes, which depends on deep decomposition theorems. In contrast, the results in [11] depend on rather more straightforward combinatorial properties of nowhere-dense classes.

References

[1] D.M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC_1. *Journal of Computer and System Sciences*, 41:274–306, 1990.

[2] A. Blass, Y. Gurevich, and S. Shelah. On polynomial time computation over unordered structures. *Journal of Symbolic Logic*, 67(3):1093–1125, 2002.

[3] J-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[4] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. Elsevier, 1990.

[5] B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33:125–150, 2000.

[6] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101:77–114, 2000.

[7] A. Dawar. Homomorphism preservation on quasi-wide classes. *J. Compute and System Sciences*, 2009. to appear. See arXiv:0811.4497v1 [cs.LO].

[8] A. Dawar, M. Grohe, B. Holm, and B. Laubner. Logics with rank operators. In *Proc. 24th IEEE Symp. on Logic in Computer Science*, pages 113–122, 2009.

[9] A. Dawar, M. Grohe, and S. Kreutzer. Locally excluding a minor. In *Proc. 22nd IEEE Symp. on Logic in Computer Science*, pages 270–279, 2007.

[10] A. Dawar, M. Grohe, S. Kreutzer, and N. Schweikardt. Model theory makes formulas large. In *ICALP'07:Proc. 34th International Colloquium on Automata, Languages and Programming*, LNCS. Springer, 2007.

[11] A. Dawar and S. Kreutzer. Domination problems in nowhere-dense classes of graphs. In *FSTTCS 2009*, 2009.

[12] A. Dawar and D. Richerby. Fixed-point logics with nondeterministic choice. *Journal of Logic and Computation*, 13:503–530, 2003.

[13] A. Dawar, D. Richerby, and B. Rossman. Choiceless polynomial time, counting and the Cai-Fürer-Immerman graphs. *Annals of Pure and Applied Logic*, 152:31–50, 2008.

[14] R. Diestel. *Graph Theory*. Springer, 3rd edition, 2005.

[15] H-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 2nd edition, 1999.

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

[17] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol 7*, pages 43–73, 1974.

[18] J. Flum and M. Grohe. Fixed-parameter tractability, definability, and model checking. *SIAM Journal on Computing*, 31:113 – 145, 2001.

[19] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[20] M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *Journal of the ACM*, 48:1184–1206, 2001.

[21] H. Gaifman. On local and non-local properties. In J. Stern, editor, *Proceedings of the Herbrand Symposium Logic Colloquium '81*, pages 105–135. North-Holland, 1982.

[22] F. Gire and H. Hoang. An extension of fixpoint logic with a symmetry-based choice construct. *Information and Computation*, 144:40–65, 1998.

[23] M. Grohe. Logic, graphs, and algorithms. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, pages 357–422. Amsterdam University Press, 2007.

[24] M. Grohe. The quest for a logic capturing PTIME. In *Proc. 22nd IEEE Symp. on Logic in Computer Science*, pages 267–271, 2008.

[25] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[26] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[27] B. Mohar and C. Thomassen. *Graphs on Surfaces*. Johns Hopkins University Press, 2001.

[28] J. Nešetřil and P. Ossona de Mendez. First-order properties of nowhere dense structures. *Journal of Symbolic Logic*, 2009. to appear.

[29] J. Nešetřil and P. Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 2009. to appear.

[30] J. Nešetřil and P. Ossona de Mendez. The grad of a graph and classes with bounded expansion. In *International Colloquium on Graph Theory*, pages 101 – 106, 2005.

[31] D. Seese. Linear time computable problems and first-order descriptions. *Math. Struct. in Comp. Science*, 6:505–526, 1996.

[32] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1976.

[33] A. Stolbouskin and M. Taitslin. Is first order contained in an initial segment of PTIME? In *Computer Science Logic 94*, volume 933 of *LNCS*. Springer-Verlag, 1995.

[34] M. Y. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM Symp. on the Theory of Computing*, pages 137–146, 1982.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Priced Timed Automata: Theory & Tools

Kim G. Larsen*
Department of Computer Science, Aalborg University
Denmark
kgl@cs.aau.dk

ABSTRACT. Priced timed automata are emerging as useful formalisms for modeling and analysing a broad range of resource allocation problems. In this extended abstract, we highlight recent (un)decidability results related to priced timed automata as well as point to a number of open problems.

1 Introduction

The model of timed automata, introduced by Alur and Dill [2, 3], has by now established itself as a classical formalism for describing the behaviour of real-time systems. A number of important properties has been shown decidable, including reachability, model checking and several behavioural equivalences and preorders.

By now, real-time model checking tools such as UPPAAL [11, 39] and KRONOS [30] are based on the timed automata formalism and on the substantial body of research on this model that has been targeted towards transforming the early results into practically efficient algorithms — *e.g.* [9, 15, 8, 13] — and data structures — *e.g.*[38, 37, 14, 14].

More recently, model-checking tools in general and UPPAAL in particular have been applied to solve realistic scheduling problems by a reformulation as reachability problems — *e.g.* [34, 35, 1, 41]. Aiming at *optimal* scheduling, *priced timed automata* [12, 5] are emerging as a useful formalism for formulating and solving a broad range of resource allocation problems of importance in applications areas such as, *e.g.*, embedded systems.

2 Optimal Reachability and Optimal Safety

Within the model of priced timed automata, the cost variables serve purely as *evaluation functions* or *observers*, *i.e.*, the behaviour of the underlying timed automatoa may in no way depend on the cost variables. As a consequence of this restriction – and in contrast to the related models of constant slope and linear hybrid automata – a number of optimization problems have been shown decidable for priced timed automata including minimun-cost reachability [12, 4, 20], optimal (minimum and maximum cost) reachability in multi-priced settings [40].

EXAMPLE 1. *Consider the timed automaton of Fig. 1(a) with two clocks x and y, and label set $\{a, b, c, d, e\}$. Note that no time can elapse in the middle location due to the invariant $(y = 0)$. The a, d and e transitions have guards $x \leq 2$ and $x = 2$ respectively. It is clear that no matter which*

*Partially funded by the VKR Center of Excellence, MT-LAB.

© Larsen; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 417–425
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2337

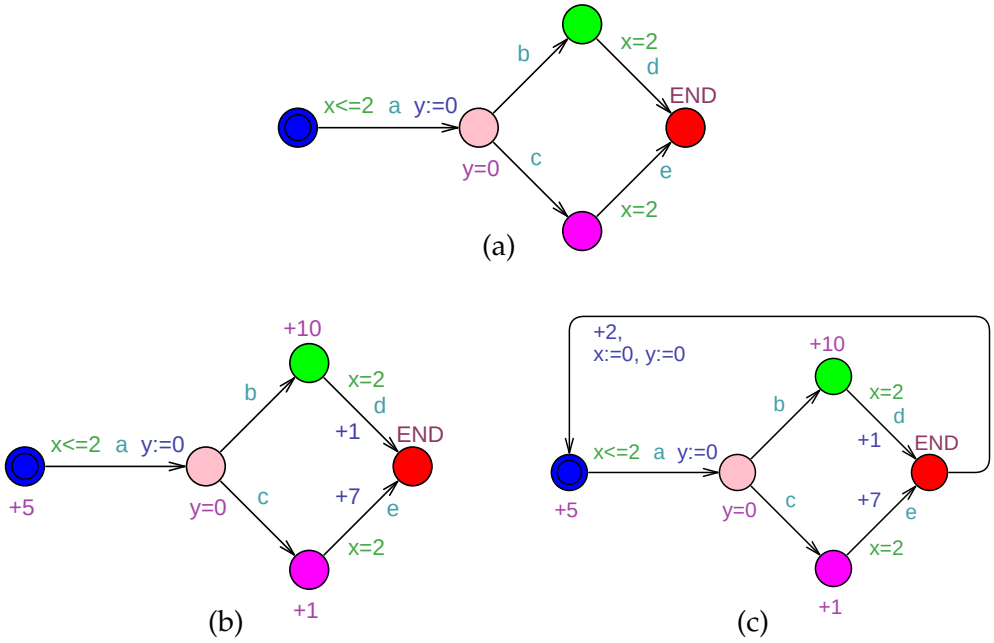


Figure 1: (a) A Small Timed Automaton. (b) A Small Priced Timed Automaton. (c) A Small Cyclic Priced Timed Automaton.

execution – differing in the delay in initial location and whether to choose the b or c transition – the minimal time for reaching the END *location is* 2.

Now consider the priced timed automaton of Fig. 1(b). Here the decoration of $+10$ *on a location indicates that cost increases by* 10 *per time unit in the location; a decoration* -7 *on a transition indicates that taking the transition increases overall cost by* 7*. Let us compute the minimum cost that is required for reacing location* END*. There are two families of executions: those that follow the b edge and those that follow the c edge. Furthermore, in each family, there is a single parameter* $t \leq 2$ *being the time elapsed in the initial location before the a edge is taken. Hence the minimum cost is:*

$$\inf_{0 \leq t \leq 2} \min \begin{pmatrix} 5t + 10(2-t) + 1 \\ 5t + (2-t) + 7 \end{pmatrix} = 9$$

where $5t + 10(2-t) + 1$ *and* $5t + (2-t) + 7$ *give the cost of executions following the b respectively the c edge.*

Dually, computability of cost-optimal infinite schedules have been established covering optimal infinite schedules in terms of minimal (or maximal) *cost per time ratio* in the limit have been obtained in [21, 22] and optimal infinite schedules in terms of minimal (or maximal) *discounted total cost* [33].

EXAMPLE 2. *Now reconsider the cyclic priced timed automaton of Fig. 1(c). Due to the simplicity of the cycle (both x and y are reset at the looping transition), the optimal schedule in terms of minimal cost per time unit has value* $(9+2)/2 = 5.5$.

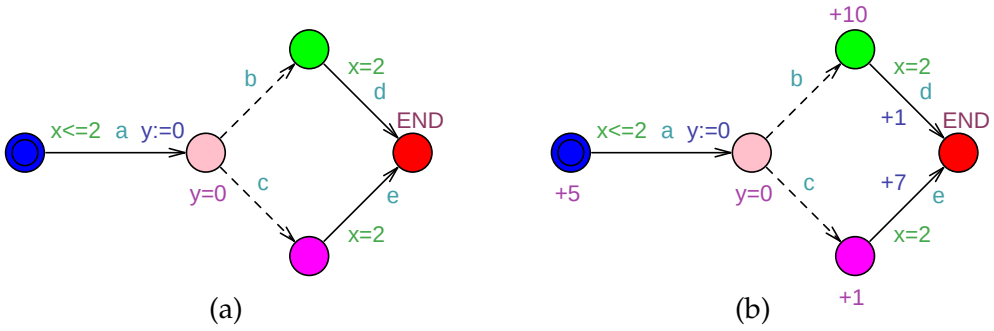


Figure 2: (a) A Small Timed Game. (b) A Small Priced Timed Game.

In terms of tool support UPPAAL Cora [36, 16, 17, 42] provides an efficient method for computing cost-optimal or near-optimal solutions to reachability questions, implementing a symbolic A* algorithm based on a new data strucutre (so-called priced zones) allowing for efficient symbolic state-representation with additional cost-information.

3 Model-Checking

Cost-extended versions of temporal logics such as CTL (branching-time) and LTL (linear-time) appear as a natural "generalizations" of the above optimization problems. Just as TCTL and MTL provide extensions of CTL and LTL with time-constrained modalities, WCTL and WMTL are extensions with cost-constrained modalities interpreted with respect to priced timed automata. Unfortunately, the addition of cost now turns out to come with a price: whereas the model-checking problems for timed automata with respect to TCTL and MTL are decidable, it has been shown in [31] that model-checking with respect to WCTL is undecidable for priced timed automata with three clocks or more. In contrast [27, 28] shows that model checking with respect to WCTL is decidable under the single clock assumption. Decidability of WCTL for priced timed automata with two clocks is still an open (and hard) problem.

EXAMPLE 3. *Reconsider the priced timed automaton of Fig. 1(b) (with $x \leq 2$ added as an invariant to the initial location). Then the properties that i) there is a run leading to the location* END *with cost no more than* 9 *and that ii) all runs will lead to* END *within cost* 17 *may be expressed as the WCTL formula* $\mathsf{EF}_{c\leq 9}\mathsf{END}$ *and* $\mathsf{AF}_{c\leq 17}\mathsf{END}$*, respectively.*

4 Priced Timed Games

The models we have considered so far are *closed* in the sense that all transitions under control of the the user of the system. This is not sufficient to model embedded systems, where interaction with an uncontrollable environment is crucial, or systems with some imprecisions. These can be modelled using (two-player) *timed games* [7], in which some actions are

triggered by the environment. The aim is to *control*, or *guide*, the system so that it is guaranteed to be safe or correct regardless of the way the environment interferes. An example of a timed game is depicted in Fig. 2. Here, a *winning strategy* clearly exists achieving the objective of reaching location END. Such winning reachability strategies, as well as time-optimal strategies [6], may be computed using efficient zone-based algorithms. Tool support is now available in UPPAAL Tiga [10] applying a symbolic on-the-fly algorithm.

It is natural to extend the timed game framework with cost information, hence making it possible to model uncertainty as well as consumption of resource, and to ask for strategies which obtain a stated objective in an optimal manner given the particular cost decoration. The model of *priced timed games* is the obvious combination of timed games and priced timed automata.

EXAMPLE 4. *Consider the example of the priced timed game of Fig. 2. Now we may want to compute the minimal cost for reaching the final location* END *regardless of whether the environment chooses to take the b or the c edge. As the system cannot control this choice, the minimum cost is given by the formula:*

$$\inf_{0\leq t\leq 2} \max \begin{pmatrix} 5t + 10(2-t) + 1 \\ 5t + (2-t) + 7 \end{pmatrix} = 14.33$$

As for model checking priced timed automata, optimal winning strategies for priced timed games have proved much more difficult than simple optimal reachability and safety. In particular in [32] it has been shown that the problem of determining cost-optimal winning reachability strategies for priced timed games is not computable. In [19] it has been shown that these negative results hold even for priced timed (game) automata with no more than three clocks.

Decidability has been shown for classes of priced timed games with strong zone-like conditions on the evolution of cost [23, 24] and for one-clock priced timed games [29]. Again the case of two clocks is as yet unsettled.

5 Energy-Games

In [26] we began the study of a new class of resource scheduling problems, namely that of constructing infinite schedules or strategies subject to boundary constraints on the accumulation of resources, so-called *energy-games* or *energy-schedules*.

More specifically, we consider priced timed automata with *positive* as well as *negative* price-rates. This extension allows for the modelling of systems where resources are not only consumed but also occasionally produced or regained. In [26] three infinite scheduling problems was considered: *lower-bound* where the energy level never must go below zero, *interval-bound* where energy level must be maintained within a given interval, and *weak* upper bound, which does not prevent energy-increasing behaviour from proceeding once the upper bound is reached but merely maintains the energy level at the upper bound.

For one-clock priced timed automata both the lower-bound and the lower-weak-upper-bound problems are shown decidable (in polynomial time) [26], whereas the interval-bound problem is proved to be undecidable in a game setting. Decidability of the interval-bound

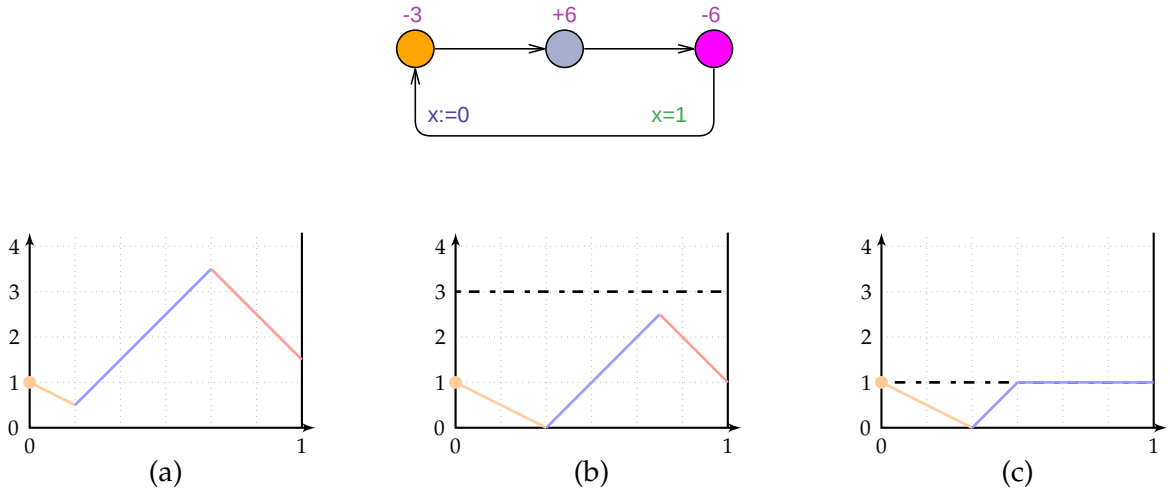


Figure 3: One-clock priced timed automaton and three types of infinite schedules: lower-bound (a), lower-upper-bound (b) and lower-weak-upper-bound (c).

problem for one-clock priced timed automata as well as decidability of all of the considered scheduling problems for priced timed automata with two or more clocks are still unsettled.

EXAMPLE 5. *Consider the priced timed automaton in Fig. 3 with infinite behaviours repeatedly delaying in the three locations for a total duration of one time unit. The negative weights (-3 and -6) indicate rates by which energy will be consumed, and the positive rate ($+6$) indicates the rate by which energy will be gained. Thus, for a given iteration the effect on the energy remaining will highly depend on the distribution of the one time unit over the three locations. The three types of schedules given an inital energy level of one are illustrated.*

Most recently, the decidability of [26] for the lower-bound problem has been extended to the setting of "$1\frac{1}{2}$" priced timed automata and with prices growing either linearly (*i.e.* $\dot{p} = k$) or exponentially (*i.e.* $\dot{p} = kp$) [25]. By "$1\frac{1}{2}$-clock" priced timed automata we refer to one-clock priced timed automata augmented with discontinuous (discrete) updates (*i.e.*, $p := p + c$) of the price on edges: discrete updates can easily be encoded using a second clock but do not provide the full expressive power of two clocks.

Surprisingly, the presence of discrete updates makes the lower-bound problem significantly more intricate. In particular, whereas region-stable strategies suffice in the search for infinite lower-bound schedules for one-clock priced timed automata, this is no longer the case when discrete updates are permitted as can be seen from Fig. 4. Not being able to rely on the classical region construction, the key to our decidability result is the notion of an *energy function* providing an abstraction of a path in the priced timed automaton (Fig. 5).

Acknowledgement

The author would like to thank Patricia Bouyer, Uli Fahrenberg, Nicolas Markey, Jiri Srba and Claus Thrane for numerous fruitful discussions on priced timed automata.

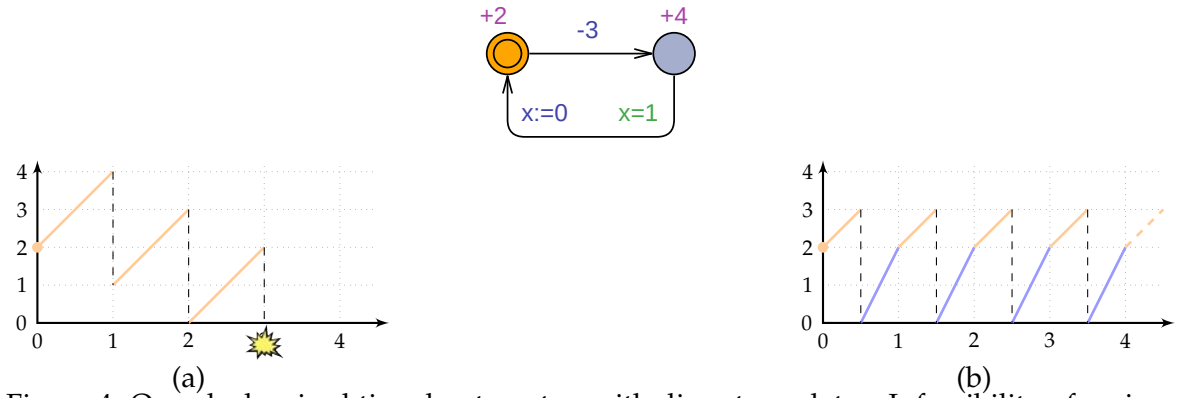
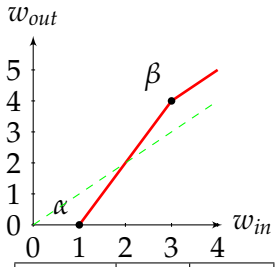
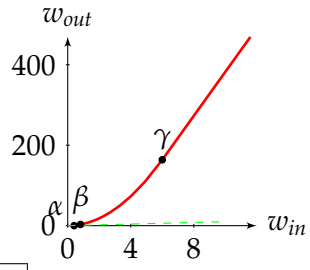


Figure 4: One-clock priced timed automaton with discrete updates. Infeasibility of region-stable lower-bound schedule (a) and optimal lower-bound schedule (b).



point	w_{in}	w_{out}
α	1	0
β	3	4

(a)



pt	w_{in}	w_{out}
α	≈ 0.406006	0
β	≈ 0.812012	3
γ	6	≈ 163.794

(b)

Figure 5: Energy functions for the two-location path of the priced timed automaton of Figure 4 with linear rates (a) and exponential rates (b).

References

[1] Y. Abdeddaïm, A. Kerbaa, and O. Maler. Task graph scheduling using timed automata. In *IPDPS*, page 237. IEEE Computer Society, 2003.

[2] R. Alur and D. L. Dill. Automata for modeling real-time systems. In M. Paterson, editor, *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, 1990.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[4] R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In M. D. Di Benedetto and A. L. Sangiovani-Vincentelli, editors, *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer-Verlag, Mar. 2001.

[5] R. Alur, S. L. Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In Benedetto and Sangiovanni-Vincentelli [18], pages 49–62.

[6] E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata.

In F. W. Vaandrager and J. H. van Schuppen, editors, *HSCC*, volume 1569 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.

[7] E. Asarina, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier, 1998.

[8] G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi. Uppaal implementation secrets. In W. Damm and E.-R. Olderog, editors, *FTRTFT*, volume 2469 of *Lecture Notes in Computer Science*, pages 3–22. Springer-Verlag, 2002.

[9] G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelánek. Lower and upper bounds in zone based abstractions of timed automata. In K. Jensen and A. Podelski, editors, *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 312–326. Springer-Verlag, 2004.

[10] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. Uppaal-tiga: Time for playing games! In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer-Verlag, 2007.

[11] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *SFM*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer-Verlag, 2004.

[12] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. W. Vaandrager. Minimum-cost reachability for priced timed automata. In Benedetto and Sangiovanni-Vincentelli [18], pages 147–161.

[13] G. Behrmann, T. Hune, and F. W. Vaandrager. Distributing timed model checking - how the search order matters. In E. A. Emerson and A. P. Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 216–231. Springer-Verlag, 2000.

[14] G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In N. Halbwachs and D. Peled, editors, *CAV*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353. Springer-Verlag, 1999.

[15] G. Behrmann, K. G. Larsen, and R. Pelánek. To store or not to store. In W. A. H. Jr. and F. Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 433–445. Springer-Verlag, 2003.

[16] G. Behrmann, K. G. Larsen, and J. I. Rasmussen. Priced timed automata: Algorithms and applications. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, editors, *FMCO*, volume 3657 of *Lecture Notes in Computer Science*, pages 162–182. Springer-Verlag, 2004.

[17] G. Behrmann, K. G. Larsen, and J. I. Rasmussen. Optimal scheduling using priced timed automata. *SIGMETRICS Performance Evaluation Review*, 32(4):34–40, 2005.

[18] M. D. D. Benedetto and A. L. Sangiovanni-Vincentelli, editors. *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings*, volume 2034 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[19] P. Bouyer, T. Brihaye, and N. Markey. Improved undecidability results on weighted timed automata. *Inf. Process. Lett.*, 98(5):188–194, 2006.

[20] P. Bouyer, Th. Brihaye, V. Bruyère, and J.-F. Raskin. On the optimal reachability problem on weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, Oct. 2007.

[21] P. Bouyer, E. Brinksma, and K. G. Larsen. Staying alive as cheaply as possible. In R. Alur and G. J. Pappas, editors, *HSCC*, volume 2993 of *Lecture Notes in Computer*

Science, pages 203–218. Springer-Verlag, 2004.

[22] P. Bouyer, E. Brinksma, and K. G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):2–23, Feb. 2008.

[23] P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In K. Lodaya and M. Mahajan, editors, *FSTTCS*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer-Verlag, 2004.

[24] P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Synthesis of optimal strategies using hytech. *Electr. Notes Theor. Comput. Sci.*, 119(1):11–31, 2005.

[25] P. Bouyer, U. Fahrenberg, K. G. Larsen, and N. Markey. Timed automata with observers under energy constraints. Under submission, 2009.

[26] P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In F. Cassez and C. Jard, editors, *FORMATS*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008.

[27] P. Bouyer, K. G. Larsen, and N. Markey. Model-checking one-clock priced timed automata. In H. Seidl, editor, *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'07)*, volume 4423 of *Lecture Notes in Computer Science*, pages 108–122, Braga, Portugal, Mar. 2007. Springer.

[28] P. Bouyer, K. G. Larsen, and N. Markey. Model checking one-clock priced timed automata. *Logical Methods in Computer Science*, 4(2:9), June 2008.

[29] P. Bouyer, K. G. Larsen, N. Markey, and J. I. Rasmussen. Almost optimal strategies in one clock priced timed games. In S. Arun-Kumar and N. Garg, editors, *FSTTCS*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer-Verlag, 2006.

[30] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In A. J. Hu and M. Y. Vardi, editors, *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer-Verlag, 1998.

[31] T. Brihaye, V. Bruyère, and J.-F. Raskin. Model-checking for weighted timed automata. In Y. Lakhnech and S. Yovine, editors, *FORMATS/FTRTFT*, volume 3253 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2004.

[32] T. Brihaye, V. Bruyère, and J.-F. Raskin. On optimal timed strategies. In P. Pettersson and W. Yi, editors, *FORMATS*, volume 3829 of *Lecture Notes in Computer Science*, pages 49–64. Springer-Verlag, 2005.

[33] U. Fahrenberg and K. G. Larsen. Discount-optimal infinite runs in priced timed automata. *Electr. Notes Theor. Comput. Sci.*, 2008. To be published.

[34] A. Fehnker. Scheduling a steel plant with timed automata. In *RTCSA*, pages 280–286. IEEE Computer Society, 1999.

[35] T. Hune, K. G. Larsen, and P. Pettersson. Guided synthesis of control programs using uppaal. *Nord. J. Comput.*, 8(1):43–64, 2001.

[36] K. G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In G. Berry, H. Comon, and A. Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 493–505. Springer-Verlag, 2001.

[37] K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: compact data structure and state-space reduction. In *IEEE Real-Time Systems Symposium*, pages 14–24. IEEE Computer Society, 1997.

[38] K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Clock difference diagrams. *Nord. J. Comput.*, 6(3):271–298, 1999.

[39] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *STTT*, 1(1–2):134–152, 1997.

[40] K. G. Larsen and J. I. Rasmussen. Optimal reachability for multi-priced timed automata. *Theor. Comput. Sci.*, 390(2-3):197–213, 2008.

[41] O. Maler. Timed automata as an underlying model for planning and scheduling. In M. Fox and A. M. Coddington, editors, *AIPS Workshop on Planning for Temporal Domains*, pages 67–70, 2002.

[42] J. I. Rasmussen, G. Behrmann, and K. G. Larsen. Complexity in simplicity: Flexible agent-based state space exploration. In O. Grumberg and M. Huth, editors, *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 2007.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Fighting Bit Rot with Types (Experience Report: Scala Collections)

M. Odersky[1], **A. Moors**[2]*

[1] EPFL, Switzerland
martin.odersky@epfl.ch

[2] K.U.Leuven, Belgium
adriaan.moors@cs.kuleuven.be

ABSTRACT. We report on our experiences in redesigning Scala's collection libraries, focussing on the role that type systems play in keeping software architectures coherent over time. Type systems can make software architecture more explicit but, if they are too weak, can also cause code duplication. We show that code duplication can be avoided using two of Scala's type constructions: higher-kinded types and implicit parameters and conversions.

1 Introduction

Bit rot is a persistent problem in most long-running software projects. As software systems evolve, they gain in bulk but lose in coherence and clarity of design. Consequently, maintenance costs increase and adaptations and fixes become more complicated. At some point, it's better to redesign the system from scratch (often this is not done and software systems are left to be limping along because the risk of a redesign is deemed to high).

At first glance it seems paradoxical that bits should rot. After all, computer programs differ from other engineering artefacts in that they do not deteriorate in a physical sense. Software systems rot not because of rust or material fatigue, but because their requirements change. Modifying a software system is comparatively easy, so there's a low threshold to accepting new requirements, and adaptations and extensions are common. However, if not done right, every such change can obscure the original architectural design by introducing a new special case.

Two aspects of software systems tend to accelerate bit rot: lack of explicit design and code duplication. If the design of a system is not made explicit in detail it risks being undermined by changes down the line, in particular from contributors who are new to the system. Code duplication, on the other hand, is problematic because necessary adaptations might apply to one piece of code but might be overlooked in a duplicate.

In this paper we explore how a strong static type discipline affects bit rot, using the Scala collection library as a case study. A collections library is interesting because it provides a wide variety of operations, spread over several different interfaces of collections, and over an even larger number of implementations. While there is a high degree of commonality

*Supported by a grant from the Flemish IWT.

© Odersky, Moors; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 427–451
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2338

among collection interfaces and implementations, the details vary considerably. Thus, extracting the commonalities is at the same time necessary and non-trivial.

At first glance, a static type system looks like a good basis for a robust collections library because it can make design decisions explicit and checkable. On the other hand, if the static type system is not flexible enough to capture some common pattern, it might force conceptually sharable code to be repeated at each type instance. In the Scala collections we experienced both of these effects. The first Scala collection library was designed with a standard repertoire of generics and nominal inheritance and subtyping, close to what is found in Java or C#. This made a number of constraints explicit, but forced some code to be duplicated over many classes. As the number of contributors to the code base grew, this duplication caused a loss of consistency, because additions were either not done in the most general possible context, or necessary specialisations in subclasses were missed.

We recently set out to redesign the collection libraries with the aim of obtaining at the same time better architectural coherence and better extensibility. The redesign makes critical use of two advanced forms of polymorphism available in Scala: higher-kinded types and implicit parameters and conversions. Higher-kinded types allow to abstract over the constructor of a collection, independently of its element type. Implicits give a library author the means to define new type theories which are adapted to the domain at hand. Both played important roles in cleaning up the collections design.

In this paper we explain the architecture of the original collections library, and how we addressed its shortcomings in the new Scala 2.8 collections. We then present the architecture of Scala 2.8 collection framework, and show how it can be extended with new kinds of collections. We also explain how higher-kinded types and implicits help in making the new design explicit and checkable and in keeping extensions uniform and concise.

Related work The generalisation of first-order polymorphism to a higher-order system was a natural step in lambda calculus [6, 18, 2]. This theoretical advance has since been incorporated into functional programming languages. For instance, the Haskell programming language [8] supports higher-kinded types, and integrates them with type classes [9], the Haskell approach to ad-hoc polymorphism. However, to the best of our knowledge, Scala is the only object-oriented language to integrate support for higher-kinded types. We call this feature "type constructor polymorphism" [13]. Altherr et al. have proposed integrating this into Java [4].

Implicits serve two purposes in Scala: they allow for retroactive extension using the "pimp-my-library" pattern [15], and they extend the language with support for ad-hoc polymorphism. Implicits are the minimal addition to an object-oriented language that is required to encode Haskell's type classes, and thus support that style of ad-hoc polymorphism. They are more local than type classes in that the applicability of an implicit is controlled by scope rules, similarly to the modular type class proposal for ML [5]. A type-class like extension has also been proposed for Java [21].

Ad-hoc polymorphism is similar to parametric polymorphism in the sense that it allows operations to be applicable at varying types, except that, whereas parametrically polymorphic operations are truly indifferent to the concrete type that they are applied to, ad-hoc polymorphic operations take the specific type into account and vary their behaviour accord-

ingly. Java's static overloading is a minimal implementation of this abstraction mechanism, whereas Haskell type classes [20] allow for expressing much richer abstractions.

The literature on the design of collection frameworks has traditionally concentrated on the Smalltalk language. The "blue book" [7] contains a description of Smalltalk's original collection hierarchy. Cook [3] analyses the interfaces inherent in that library which are often not expressed directly in Smalltalk's single-inheritance hierarchy. Ducasse and Schärli describe the use of traits to refactor the Smalltalk collection libraries [1]. Our experience confirms their conclusion that composition of traits is an important asset in the design of such complex libraries. Scala traits differ from their formulation [19] in that Scala traits combine aspects of symmetric trait composition with aspects of linear mixin composition. Nevertheless, the applicability of both forms of traits for modelling collections stays the same. Of course, Smalltalk is dynamically typed, so none of the previously cited related works addresses the question how to type collections statically. Naftalin and Wadler describe Java's generic collections [14], which are largely imperative, and do not offer higher-order functional operations, so that they pose less challenges to the type system.

Structure of the paper Section 2 gives a quick introduction of the parts of Scala necessary to understand the examples in the rest of this paper. Section 3 presents the original collection framework as it existed before the redesign and highlights its shortcomings. The next two sections introduce key abstractions that form the foundation the new collections library. Section 4 shows how to reduce code duplication by abstracting over the representation type of the collection, as well as over how to traverse and build it. Section 5 refines this to abstractions over type constructors. However, neither approach suffices. Section 6 illustrates that we need ad-hoc polymorphism — piece-wise defined type functions — and introduces implicits as a solution. Section 7 discusses in detail how implicits express piecewise defined type functions and integrates them with builders. Section 8 outlines the Scala 2.8 collections hierarchy, and shows how new collection implementations can be integrated in the framework, illustrating the kind of code re-use that is achieved. Section 9 explains how the pre-existing primitive classes for arrays and strings can be integrated in the collections framework. Section 10 concludes.

2 Syntactic Preliminaries

In Scala [16, 17], a class can inherit from one other class and several other traits. A trait is a class that can be composed with other traits using mixin composition. Mixin composition is a restricted form of multiple inheritance, which avoids ambiguities by linearising the graph that results from composing traits that are themselves composites of traits. The difference between an abstract class and a trait is that the latter can be composed using mixing inheritance[†]. We will use "class" to refer to traits and classes alike, but, for brevity, we will use **`trait`** instead of **`abstract class`** in listings.

Identifiers in Scala may consist of symbolic as well as regular identifier characters. Method calls like `xs.++(ys)` or `xs.take(5)` have more lightweight equivalents: `xs ++ ys` and `xs take 5`.

[†]The restrictions imposed on traits to allow mixin composition are not relevant for this paper.

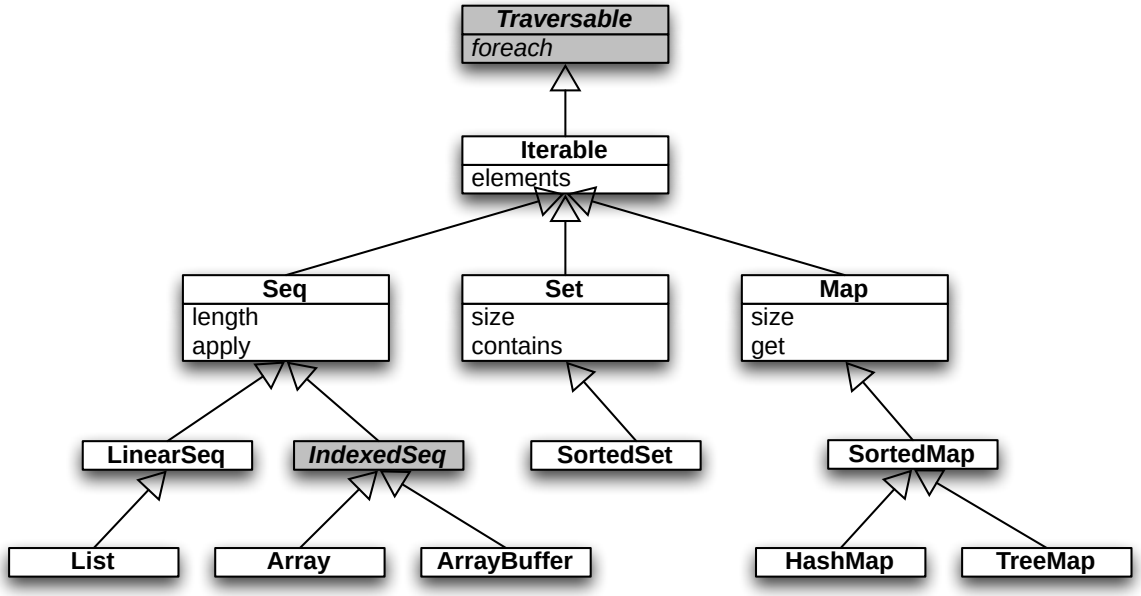


Figure 1: Some Scala Collection Classes (2.8-specific classes are shaded)

Functions are first-class values in Scala. The type of functions from T to U is written $T \Rightarrow U$. A function literal is also written with an infix "$\Rightarrow$", e.g. `(x: Int)⇒ x + 1` for anonymous successor function over type `Int`. Type inference often allows to elide the argument type of a function literal, as in `x ⇒ x + 1`. Alternatively, and even shorter, the parameter position may be marked with an underscore, as in `(_ + 1)`. Internally, functions are represented as objects with `apply` methods. For instance, each of the three above function literals is expanded to the object

```
new Function1[Int, Int] {
  def apply(x: Int) = x + 1
}
```

Conversely, function application notation `f(e)` is available for every object `f` with an `apply` method, and is in each case equivalent to `f.apply(e)`.

3 Status Quo

Scala collections are characterised by four properties: they are *object-oriented*, *optionally persistent*, *generic*, and *higher-order*.

Object-oriented Collections form a hierarchy, sharing common operations in base traits. Figure 1 gives an outline of the collections hierarchy as it existed in Scala until version 2.7, including some of the new classes from Scala 2.8, which have been shaded. At the top of the original hierarchy is trait `Iterable`, which represents a collection by means of an `elements` method that allows iterating over its elements. Specialisations of `Iterable` are

`Set` for sets, `Map` for maps, and `Seq` for sequences. Some of these abstractions have further specialisations.

For example, the classes `SortedSet` and `SortedMap` represent sets or maps which are *sorted*, meaning that their iterators return their elements in the natural order of the element type. Each collection abstraction has multiple implementations. Sequences in trait `Seq` can be linked lists, arrays, list buffers, array buffers, or priority queues, to name but a few. The class hierarchy gives rise to a subtyping relation (`<:`) between collections. For instance, `Set` is a subtype of `Iterable`, so that a set can be passed wherever an iterable is expected.

Most operations on collections are represented as methods in the collection classes. For instance, to retrieve an iterator for the elements in `c`, one calls `c.iterator`. The length of a sequence `s` can be queried using `s.length`, and `s(i)` (short for `s.apply(i)`) returns its `i`'th element.

Optionally persistent Most collection abstractions in the library exists in two forms: mutable and immutable. Immutable collections are also called "persistent"; they offer operations that create new collections from existing ones incrementally, leaving the original collections unchanged. For instance, `xs ++ ys` creates a new sequence which consists of all elements of sequences `xs`, followed by all elements of sequence `ys`. The sequences `xs` and `ys` remain unchanged. Or, `m + (k -> v)` creates a new map that augments map `m` with a new key/value binding. The original map remains again unchanged. Mutable collections introduce operations that change the collection in place. For instance, `m.update(k, v)` updates a map at key `k` with the new value `v` (this can be expressed shorter as `m(k) = v`).

Generic Most collections are parametric in the type of their elements. For instance, the type of lists with pairs of integers and strings as elements is `List[(Int, String)]`, and `Map[String, List[String]]` represents a map that takes keys of type `String` to values of type `List[String]`. The interaction between subtyping and generics is controlled by variance annotations. Most persistent collection types are covariant, whereas all mutable collections are nonvariant.

Variance defines a subtyping relation over parameterised types based on the subtyping of their element types. For example, **`class`** `List[+T]` introduces the type constructor `List`, whose type parameter is *covariant*. This means that `List[A]` is a subtype of `List[B]` iff `A` is a subtype of `B`. With a *contravariant* type parameter, this is inverted, so that **`class`** `OutputChannel[-T]` entails that `OutputChannel[A]` is a subtype of `OutputChannel[B]` iff `A` is a *supertype* of `B`. Without an explicit variance annotation, type arguments must be equal for the constructed types to be comparable.

Some collections restrict their type parameter. Sets backed by red-black trees, for example, are only defined for element types that can be ordered.

Higher-order Many operations on collections take functions as arguments. Examples are: `c.foreach(f)`, which applies the side-effecting function `f` to each element in `c`, the collection of the elements in `c` that satisfy the predicate `p` can be computed as `c.filter(p)`,

```
trait Iterable[+A] {
  def filter(p: A ⇒ Boolean): Iterable[A] = ...
  def partition(p: A ⇒ Boolean) = (filter(p(_)), filter((!p(_))))
  def map[B](f: A ⇒ B): Iterable[B] = ...
}

trait Seq[+A] extends Iterable[A] {
  override def filter(p: A ⇒ Boolean): Seq[A] = ...
  override def partition(p: A ⇒ Boolean) = (filter(p(_)), filter((!p(_))))
  override def map[B](f: A ⇒ B): Seq[B] = ...
}
```

Listing 1: Some methods of the `Iterable` and `Seq` traits

and `c.map(f)` produces a new collection with the same size as `c`, where each element is the result of applying `f` to the corresponding element of `c`.
These operations are defined uniformly for all collections. When they return a collection result, it is usually of the same class as the collection on which the operation was applied. For instance if `xs` is a list then `xs map (_ + 1)` would yield another list, but if `xs` was an array, the same call would again yield an array. The following interaction with Scala REPL shows that this relationship holds for static types as well as computed values.

```
scala> val xs = List("hello", "world", "!")
xs:  List[java.lang.String] = List(hello, world, !)

scala> xs map (_.length)
res0:  List[Int] = List(5, 5, 1)

scala> val ys = Array("hello", "world", "!")
ys:  Array[java.lang.String] = Array(hello, world, !)

scala> ys filter (_.length > 1)
res1:  Array[java.lang.String] = Array(hello, world)
```

Base traits like `Iterable` offer the same operations as their concrete implementations, but with the base trait as result type. For instance, the following REPL interactions show that applying `map` on an `Iterable` will yield `Iterable` again as the static result type (even though the computed value is a subtype).

```
scala> val zs: Iterable[String] = xs
zs:  Iterable[String] = List(hello, world, !)

scala> zs map (_.length)
res2:  Iterable[Int] = List(5, 5, 1)
```

Ideally, a collections framework should also be highly *extensible*. It should be easy to add new kinds of collections, or new implementations of existing collections. However, the combination of genericity and immutable higher-order operations makes it difficult to achieve good extensibility. Consider the `filter` method of trait `Iterable` in Listing 1, which must be specialised in `Seq` so that it returns a `Seq`. Every other subclass of `Iterable` needs a

similar re-implementation. In the original collection library such implementations had to be provided explicitly by the implementer of a collection class.

Methods that could in principle be implemented uniformly over all collections also need to be re-implemented. Consider for example the `partition` method of Listing 1, which splits a collection into two sub-collections of elements according to whether they satisfy a predicate `p`. This method could in principle be implemented just once in `Iterable`. However, to produce the correct static return type, `partition` still has to be re-implemented for every subclass, even though its definition in terms of `filter` is the same in each class.

The problem becomes even more challenging with an operation like `map`, also shown in Listing 1. The `map` method does not return exactly the same *type* as the type it was invoked on. It preserves the *type constructor*, but may apply it to a different element type.

Overall, these re-implementations pose a significant burden on collection implementers. Taking sequences as an example, this type of collection supports about a hundred methods, of which 20 return the collection type itself as some part of its result, like `filter` and `partition` do, and of which another 10 return the collection type constructor at a different element type, like `map` does. Every new collection type would have to re-implement at least these 30 methods.

In practice, this made maintaining and extending the library quite difficult. As the collection implementation evolved and the number of its contributors increased, it lost more and more of its consistency. Some operations would be added only to a specific subclass, even though they could in principle apply to more general collection types such as `Iterable`. Sometimes, a specific implementation would fail to re-implement some of the methods of the general collection class it inherited from, leading to a loss of type precision. We observed a pronounced "broken windows" effect: classes of the library that already contained ad-hoc methods would quickly attract more such methods and become more disorganised, whereas classes that started in a clean state tended to stay that way. Over the course of some years the coherence of the collection design deteriorated to a state where we felt a complete redesign was needed.

The intention was that the collection library redesign should largely keep to the original APIs in order to maintain a high degree of backwards compatibility, and also because the basic structure of these APIs proved to be sound. At the same time, the redesign should provide effective guards against the kind of bit rot that affected the previous framework. In the rest of this paper we explain how this goal was achieved and which of Scala's more advanced type constructs were instrumental in this.

4 Abstracting over the Representation Type

To avoid code duplication, collection classes such as `Traversable` or `Seq` inherit most of their concrete method implementations from an implementation trait. These implementation traits, which are denoted by the `Like` suffix, form a shadow hierarchy of the client-facing side of the collections that were depicted in Figure 1. For example, `SeqLike` is the implementation trait for `Seq` and `TraversableLike` underlies `Traversable`.

Listing 2 outlines the core implementation trait, `TraversableLike`, which backs the new root of the collection hierarchy, `Traversable`. The type parameter `Elem` stands for the

```
package scala.collection
trait TraversableLike[+Elem, +Repr] {
  protected[this] def newBuilder: Builder[Elem, Repr] // deferred
  def foreach[U](f: Elem ⇒ U)                         // deferred

  def filter(p: Elem ⇒ Boolean): Repr = {
    val b = newBuilder
    foreach { elem ⇒ if (p(elem)) b += elem }
    b.result
  }
}
```

Listing 2: An outline of **trait** `TraversableLike`

```
package scala.collection.generic
class Builder[-Elem, +To] {
  def +=(elem: Elem): this.type = ...
  def result(): To = ...
  def clear() = ...
  def mapResult[NewTo](f: To ⇒ NewTo): Builder[Elem, NewTo] = ...
}
```

Listing 3: An outline of the `Builder` class.

element type of the traversable whereas the type parameter `Repr` stands for its representation. An actual collection class, such as `List`, can simply inherit the appropriate implementation trait, and instantiate `Repr` to `List`. Thus, clients of `List` never see the type of the underlying implementation trait. There are no constraints on `Repr`, so that it might be instantiated to a type that is not a subtype of `Traversable`. Therefore, classes outside the collections hierarchy such as `String` and `Array` can still make use of all operations defined in this implementation trait.

The two fundamental operations in `Traversable` are `foreach` and `newBuilder`. Both operations are deferred in class `TraverableLike` to be implemented in concrete subclasses. The `foreach` operation takes a function parameter that is applied to every element in the traversable collection. The result of the function paraneter is ignored, so functions are applied for their side effect only. The `newBuilder` operation creates a "builder" object, from which new collections can be constructed. All other methods on of `Traversable` access the collection using `foreach`. If they construct a new collection, they always do so through a builder.

Listing 3 presents a slightly simplified outline of the `Builder` class. One can add an element `x` to a builder `b` with `b += x`. There's also syntax to add more than one element at once, for instance `b += (x, y)` to add the two elements `x` and `y`, or `b ++= xs` to add all elements in the collection `xs`. The `result()` method returns a collection from a builder. The state of the builder is undefined after taking its result, but it can be reset into a new empty state using `clear()`. Builders are generic in both the element type `Elem` and in the type `To` of collections they return.

Often, a builder can refer to some other builder for assembling the elements of a collection, but then would like to transform the result of the other builder, to give a different type, say. This task is simplified by the method `mapResult` in class `Builder`. For instance, assuming a builder `bldr` of `ArrayBuffer` collections, one can turn it into a builder for `Array`s like this:

```
bldr mapResult (_.toArray)
```

Given these abstractions, the trait `TraversableLike` can define operations like `filter` in the same way for all collection classes, without compromising efficiency or precision of type signatures. First, it relies on the `newBuilder` method to create an empty builder that's appropriate for the collection at hand, then, it uses `foreach` to traverse the existing collection, appending every `elem` that meets the predicate `p` to the builder. Finally, the builder's `result` is the filtered collection.

5 Abstracting over the Collection Type Constructor

While abstracting over the representation type suffices to factor out exactly what varies in `filter`'s result type across the collection hierarchy, it cannot capture the variation in `map`'s result type. Recall that `map` is an operation that derives a collection from an existing one by applying a user-supplied function to each of its elements. For example, if the given function `f` goes from `String` to `Int`, and `xs` is a `List[String]`, `xs map f` should yield a `List[Int]`. Likewise, if `ys` is an `Array[String]`, then we expect `ys map f` to produce an `Array[Int]`.

To provide a precise abstract *declaration* of `map` at the top of the collection hierarchy – let alone a single *implementation* – we must refine the technique we developed in the previous section. To make concrete only what distinguishes the individual subclasses, we must be able to abstract over precisely what varies in these examples, and not more. Thus, we cannot simply abstract over the representation type, as the variation (of `map`'s result type) across the hierarchy is restricted to the type *constructor* that represents the collection – it does not fix the type of its elements. The element type depends on the function supplied to `map`, not on `map`'s location in the collection hierarchy. In other words, abstracting over the representation type is too coarse, since we must be able to vary the element type in the `map` method.

More concretely, we need to factor out the type constructors `List` and `Array`. Thus, instead of abstracting over the representation type, we abstract over the collection type constructor. Abstracting over type constructors requires higher-order parametric polymorphism, which we call *type constructor polymorphism* in Scala [13]. This higher-order generalisation of what is typically called "genericity" in object-oriented languages, allows to declare type parameters, such as `Coll`, that themselves take (higher-order) type parameters, such as `x` in the following snippet:

```
trait TraversableLike[+Elem, +Coll[+x]] {
  def map[NewElem](f: Elem ⇒ NewElem): Coll[NewElem]
  def filter(p: Elem ⇒ Boolean): Coll[Elem]
}
```

Now, `List[T]` may extend `TraversableLike[T, List]` in order to specify that mapping or filtering a list again yields a list, whereas the type of the elements depends on the operation. Of course, `filter`'s type can still be expressed as well.

Thus, with type constructor polymorphism, we can give a single declaration of `map` that can be specialised without redundancy in `List` and `Array`. Moreover, as discussed in Section 7, we can even provide a single implementation, where the only variation between the different concrete subclasses is how to build that concrete collection.

However, important corner cases in the collection hierarchy exhibit variations that are less uniform than the above examples. In turns out type constructor polymorphism is too uniform to express the required ad-hoc variations. The next section discusses the general case and presents the kind of polymorphism that our design hinges on.

6 Ad-hoc Polymorphism with Implicits

The examples from the previous section led us to believe that `map`'s result type is a simple "straight-line" function from the concrete collection type (e.g., `List` or `Array`) and the type of the transformed elements to the type of the resulting collection. We assumed we could simply apply the type constructor of the generic class that represents the collection to the result type of the mapped function, as mapping a function from `Int` to `String` over an `Array` of `Int`s yields an `Array[String]`.

We shall collect the variations in `map`'s type signature using a triple of types that relates the original collection, the transformed elements, and the resulting collection. Type constructor polymorphism is restricted to type functions of the shape `(CC[_], T, CC[T])`, for any type constructor[‡] `CC` and any type `T`. This section discusses several important examples that deviate from this pattern, and introduces implicits as a way of expressing them.

The regularity of transforming arrays and lists breaks down when we consider more specialised collections, such as a `BitSet`, which must nonetheless fit in our hierarchy. Consider the following interaction with the Scala REPL:

```
scala> BitSet(1,2,3) map (_ + 1)
res0:  scala.collection.immutable.BitSet = BitSet(2, 3, 4)
```

With a little bit of foresight in `Iterable`, we can capture this pattern. However, it quickly goes awry when we consider an equally desirable transformation:

```
scala> BitSet(1,2,3) map (_.toString+"!")
res1:  scala.collection.immutable.Set[java.lang.String] = Set(1!, 2!,
   3!)
```

Because the result type of `toString` is `String` and not `Int`, the result of the map cannot be a `BitSet`. Instead a general `Set[String]` is returned. One might ask why the second `map` should be admitted at all. Could one not restrict `map` on `BitSet` to mappings from `Int` to `Int`? In fact, such a restriction would be illegal because `BitSet` is declared to be a subtype of `Set[Int]` (and there are good modelling reasons why it should be). `Set[Int]` provides a `map` operation which takes arbitrary functions over `Int`, so by the Liskov substitution principle [10] every subtype of `Set[Int]` must provide the same operation.

This means that our type function for calculating `map`'s result type must now include the following triples: `(BitSet, Int, BitSet)`, and `(BitSet, String, Set[String])`, and in fact, for every type `T` different from `Int`, `(BitSet, T, Set[T])`. A type function

[‡]More precisely, for any type constructor `CC` with one unbounded type parameter.

that includes only the first triple `(BitSet, Int, BitSet)` can be expressed using type constructor polymorphism, but the other ones are out of reach.
Finally, consider transforming maps:

```
scala> Map("a" -> 1, "b" -> 2) map { case (x, y) ⇒ (y, x) }
res2:  scala.collection.immutable.Map[Int,java.lang.String] = Map(1 ->
   a, 2 -> b)

scala> Map("a" -> 1, "b" -> 2) map { case (x, y) ⇒ y }
res3:  scala.collection.immutable.Iterable[Int] = List(1, 2)
```

The first function swaps two arguments of a key/value pair. The result of mapping this function is again a map, but now going in the other direction. In fact, the original yields the inverse of the original map, provided it is invertible. The second function, however, maps the key/value pair to an integer, namely its value component. In that case, we cannot form a `Map` from the results, but we can still form an `Iterable`, which is the base trait of `Map`.

The irregular triples `(Map[A, B], (A, B)⇒(B, A), Map[B, A])` and — assuming `T` is not `(A, B)` — `(Map[A, B], (A, B)⇒T, Iterable[T])` summarise these type signatures, for arbitrary types `A`, `B`, and `T`.

Instead of admitting these ad-hoc type relations between the type of the collection, the transformation and the result, we could restrict `map` to recover the regularity that is supported by type constructor polymorphism. However, in doing so, we must respect the Liskov substitution principle. This requires somehow "announcing" these restrictions abstractly in the top-level type, `Iterable`. Expressing these restrictions quickly becomes unwieldy so that this is not a viable alternative.

Shoehorning the collection hierarchy into what is supported by type constructor polymorphism would lead to an imprecise interface, code duplication, and thus, in the long term, bit rot. To avoid these problems, we shall use the type system to express the required piece-wise defined type functions precisely.

Piece-wise defined type functions are reminiscent of Java's static overloading, as an individual case ("piece") of the type function corresponds to an overloaded method. However, Java's static overloading can only express fairly trivial piece-wise defined type functions, rendering it unsuitable for our purposes. Haskell's type classes [20] provide a sufficiently expressive, and principled solution. Scala introduces implicits, which, together with Scala's object-oriented constructs, support ad-hoc polymorphism in much the same way as type classes.

Implicits

The foundations of Scala's implicits are quite simple. A method's last argument list may be marked as implicit. If such an implicit argument list is omitted at a call site, the compiler will, for each missing implicit argument, search for the implicit value with the most specific type that conforms to the type of that argument. For a value to be eligible, it must have been marked with the **`implicit`** keyword, and it must be in the implicit scope at the call site. For now, the implicit scope may simply be thought of as the scope of a regular value, although it is actually broader.

```
abstract class Monoid[T] {
  def add(x: T, y: T): T
  def unit: T
}

object Monoids {
  implicit object stringMonoid extends Monoid[String] {
    def add(x: String, y: String): String = x.concat(y)
    def unit: String = ""
  }
  implicit object intMonoid extends Monoid[Int] {
    def add(x: Int, y: Int): Int = x + y
    def unit: Int = 0
  }
}
```

Listing 4: Using implicits to model monoids

```
def sum[T](xs: List[T])(implicit m: Monoid[T]): T =
  if(xs.isEmpty) m.unit
  else m.add(xs.head, sum(xs.tail))
```

Listing 5: Summing lists over arbitrary monoids

Listing 4 introduces implicits by way of a simple example. It defines an abstract class of monoids and two concrete implementations, `StringMonoid` and `IntMonoid`. The two implementations are marked with an **`implicit`** modifier. Listing 5 implements a `sum` method, which works for arbitrary monoids. `sum`'s second parameter is marked **`implicit`**. Because of that, `sum`'s recursive call does not need to pass along the `m` argument explicitly; it is instead provided automatically by the Scala compiler.

After having entered the code snippets in Listings 4 and 5 into the Scala REPL, we can bring the implicit values in the `Monoid` object into scope with **`import`** `Monoids._`. This makes the two implicit definitions of `stringMonoid` and `intMonoid` eligible to be passed as implicit arguments, so that one can write:

```
scala> sum(List("a", "bc", "def"))
res0:  java.lang.String = abcdef

scala> sum(List(1, 2, 3))
res1:  Int = 6
```

These applications of `sum` are equivalent to the following two applications, where the formerly implicit argument is now given explicitly.

```
sum(List("a", "bc", "def"))(stringMonoid)
sum(List(1, 2, 3))(intMonoid)
```

Implicits are closely related to Haskell's type classes. Where Scala uses a regular class such as `Monoid`, Haskell would use a type class. Implicit values such as `stringMonoid` and

`intMonoid` correspond to instance declarations in Haskell. Implicit parameters correspond to contexts in Haskell. Conditional instance declarations with contexts in Haskell can be modelled in Scala by implicit functions that themselves take implicit parameters. For instance, here is a function defining an implicit lexicographical ordering relation on lists which have element types that are themselves ordered.

```
implicit def listOrdering[T](xs: List[T])(implicit elemOrd: Ordering[T]) =
  new Ordering[List[T]] {
    def compare(xs: List[T], ys: List[T]) = (xs, ys) match {
      case (Nil, Nil) ⇒ 0
      case (Nil, _) ⇒ -1
      case (_, Nil) ⇒ 1
      case (x :: xs1, y :: ys1) ⇒
        val ec = elemOrd.compare(x, y)
        if (ec != 0) ec else compare(xs1, ys1)
    }
  }
```

7 Implicits for Scala's collections

The most interesting application of implicits in our design of Scala's collections library is in the typing of methods like `map`, which require expressive ad-hoc polymorphism. We have seen that the result type of `BitSet`'s `map` method can be specified in terms of triples that relate the source collection, the target element type, and the resulting collection: `(BitSet, Int, BitSet),(BitSet, T, Set[T])`. These triples define a piece-wise function on types, encoded as the implicit instances of the trait `CanBuildFrom` in Listing 6.

The listing first defines the trait `CanBuildFrom`, which takes three type parameters: the `Collection` type parameter indicates the collection from which the new collection should be built, the `NewElem` type parameter indicates the new element type of the collection to be built, and the `Result` type parameter indicates the type of that collection itself. The trait has a single deferred method, `apply`, which produces a `Builder` object that constructs a `Result` collection from `NewElem` elements.

The listing then shows the `map` method in class `TraversableLike`. This method is defined for every function result type `B` and every collection type `To` such that there exists an implicit value of `CanBuildFrom[Repr, B, To]`, where `Repr` is the representation type of the current collection. In other words, the triple `(Repr, B, To)` must be populated by a `CanBuildFrom` value. We'll come back to the implementation of `map` later in this section.

Two such `CanBuildFrom` values are shown in the companion objects — the objects that are co-defined with the classes of the same name – of classes `Set` and `BitSet`. Scala's scope rules for implicits include the companion object of a type in the implicit scope for that type. More precisely, when searching for an implicit value of type T, we consider all types S that form part of T, as well as all the supertypes of any such part S. The companion objects of all these types may contain implicit definitions which are then in the implicit scope for T. For instance, when searching for an implicit value of type `CanBuildfrom[BitSet, Int, ?To]`, the `BitSet` object is in the implicit scope because `BitSet` forms part of the type of the requested implicit. The `Set` object is also in the implicit scope because `Set` is a superclass of

```
trait CanBuildFrom[-Collection, -NewElem, +Result] {
  def apply(from: Collection): Builder[NewElem, Result]
}

trait TraversableLike[+A, +Repr] {
  def repr: Repr = ...
  def foreach[U](f: A ⇒ U): Unit = ...
  def map[B, To](f: A⇒B)(implicit cbf: CanBuildFrom[Repr, B, To]): To = {
    val b = cbf(repr) // get the builder from the CanBuildFrom instance
    for (x <- this) b += f(x) // transform element and add
    b.result
  }
}
trait SetLike[+A, +Repr] extends TraversableLike[A, Repr] {  }
trait BitSetLike[+This <: BitSetLike[This] with Set[Int]] extends SetLike[
    Int, This] {}

trait Traversable[+A] extends TraversableLike[A, Traversable[A]]
trait Set[+A] extends Traversable[A] with SetLike[A, Set[A]]
class BitSet extends Set[Int] with BitSetLike[BitSet]

object Set {
  implicit def canBuildFromSet[B] = new CanBuildFrom[Set[_], B, Set[B]] {
    def apply(from: Set[_]) = ...
  }
}

object BitSet {
  implicit val canBuildFromBitSet = new CanBuildFrom[BitSet, Int, BitSet] {
    def apply(from: BitSet) = ...
  }
}

object Test {
  val bits = BitSet(1, 31, 15)
  val shifted = bits map (x ⇒ x + 1)
  val strings = bits map (x ⇒ x.toString)
}
```

Listing 6: Encoding the CanBuildFrom type-relation for BitSet

`BitSet`.

Consider now the `Test` object in Listing 6. It contains two applications of `map` on the `BitSet` value `bits`. In the first case, the implicit parameter of the `map` method has a type of the form `CanBuildfrom[BitSet, Int, ?To]` because the collection on which the `map` is performed is a `BitSet` and the result type of the new collection is `Int`. Both shown `CanBuildFrom` values are in the implicit scope, and both match the type pattern that is searched. In this case, the `canBuildFromBitSet` value in object `BitSet` is the more specific of the two, and will be selected.

Implicit resolution uses Scala's standard member resolution rules for overloading in order to disambiguate between several applicable implicits, such as `canBuildFromSet` and `canBuildFromBitSet` in the example above. Member resolution orders equivalent members according to where they are defined in the subclassing hierarchy, with definitions in class `A` preceding over those in class `B` if `A` is a subclass of `B`. This ordering is extended to companion objects, which can be seen as forming a parallel hierarchy to the corresponding class hierarchy, somewhat like meta-classes in Smalltalk.

In the second case, the implicit parameter of the `map` method has a type of the form `CanBuildfrom[BitSet, String, ?To]` because the result type of the second function argument is `String`. In this case, only the implicit value in `Set` is applicable and will be selected.

Type inference takes the availability of an implicit value into account. Thus, when inferring the type arguments for `map`, the `To` type parameter is constrained by the search for the applicable implicit. In the example, `shifted` gets type `BitSet` since the implicit value `canBuildFromBitSet` is selected, and for that to be a valid argument for `map`'s `cbf` implicit type parameter, its `To` type parameter must be `BitSet`. The definition of `strings`, on the other hand, passes `canBuildFromBitSet` to the `map` application, with `Set[String]` as third type pararameter. Consequently, `Set[String]` is also the result type of that application.

The second application of `map`, stored in `strings`, explains why `CanBuildFrom`'s first type parameter is contravariant[§]: an implicit of `CanBuildFrom[BitSet, String, ?To]` is required, where `?To` is a type inference variable. We have that `BitSet` is a subtype of `Set`. By contravariance of `CanBuildFrom`, this means that `CanBuildFrom[Set, String, ?To]` is a subtype of `CanBuildFrom[BitSet, String, ?To]`. Hence, `CanBuildFrom[Set, String , ?To]` can be substituted for the required `CanBuildFrom[BitSet, String, ?To]`, and `?To` is inferred to be `Set[String]`.

Finding Builders at Run Time

We have seen that `map` can be given a precise type signature in `TraversableLike`, but how do we implement it? Since `map` has a value of type `CanBuildFrom[From, Elem, To]`, the idea is to let the implicit `canBuildFrom` values produce builder objects of type `Builder[Elem, To]` that construct collections of the right kind.

However, there is one minor snag. Since implicit resolution is performed at compile time, it cannot take dynamic types into account. Nonetheless, we expect a `List` to be created

[§]The variance of the other type parameters will become apparent in the next section.

when the dynamic type is `List`, even if the static type information is limited to `Iterable`. This is illustrated by the following interaction with the Scala REPL:

```
scala> val xs: Iterable[Int] = List(1, 2, 3)
xs:  Iterable[Int] = List(1, 2, 3)

scala> xs map (x ⇒ x * x)
res0:  Iterable[Int] = List(1, 4, 9)
```

If `CanBuildFrom` solely relied on the triple of types `(Iterable[Int], Int, Iterable[Int])` to provide a builder, it could not do better than to statically select a `Builder[Int, Iterable[Int]]`, which in turn could not build a `List`. Thus, we add a run-time indirection that makes this selection more dynamic.

The idea is to give the `apply` method of `CanBuildfrom` access to the dynamic type of the original collection via its `from` argument. An instance `cbf` of `CanBuildFrom[Iterable[Int], Int, Iterable[Int]]`, is essentially a function from an `Iterable[Int]` to a `Builder[Int, Iterable[Int]]`, which constructs a builder that is appropriate for the dynamic type of its argument. This is shortly explained in more detail. We first discuss how `map` is implemented in terms of this abstraction.

The implementation of `map` in Listing 6 is quite similar to the implementation of `filter` shown in Listing 2. The interesting difference lies in how the builder is acquired: whereas `filter` called the `newBuilder` method of class `TraversableLike`, `map` uses the instance of `CanBuildFrom` that is passed in as a witness to the constraint that a collection of type `To` with elements of type `B` can be derived from a collection with type `Repr`. This nicely brings together the static and the dynamic aspects of implicits: they express rich relations on types, which may be witnessed by a run-time entity. Thus, static implicit resolution resolves the constraints on the types of `map`, and virtual dispatch picks the best dynamic type that corresponds to these constraints.

Most instances of `CanBuildFrom` use the same structure for this virtual dispatch, so that we can implement it in `GenericTraversableTemplate`, the higher-kinded implementation trait for all traversables, as shown in Listing 7.

Let's see what happens for a concrete call `xs.map(f)`, where `f` has static type `A ⇒ B`, and `xs`'s static type is a subtype of `GenericTraversableTemplate[A, CC]`. The compiler will statically select an instance of `CanBuildFrom[CC[A], B, CC[B]]` for the implicit argument `cbf`. The call `cbf(this)` in `map` will actually be `cbf(xs)`, which, assuming `cbf` was a standard instance of `GenericCanBuildFrom[B]`, evaluates to `xs.genericBuilder[B]`, and finally `xs.companion.newBuilder[B]`. Thus, whatever the dynamic type of `xs`, it must simply implement `companion` to point to its factory companion object, and implement the `newBuilder` method there.

8 Scala 2.8 Collections Hierarchy

In this section we give an architectural summary of the 2.8 collections framework and discuss how it can be extended by implementers of new collection classes.

Figure 1 gives an overview of some common collection classes. Classes that were added in the 2.8 framework are shaded in that figure. At the top of the collection hierarchy is now

```
trait GenericCompanion[+CC[X] <: Traversable[X]] {
  def newBuilder[A]: Builder[A, CC[A]]
}

trait GenericTraversableTemplate[+A, +CC[X] <: Traversable[X]] {
  // The factory companion object that builds instances of class CC.
  def companion: GenericCompanion[CC]

  // The builder that builds instances of CC at arbitrary element types.
  def genericBuilder[B]: Builder[B, CC[B]] = companion.newBuilder[B]
}


trait TraversableFactory[CC[X] <: Traversable[X] with
                                  GenericTraversableTemplate[X, CC]]
                                       extends GenericCompanion[CC] {
  // Standard CanBuildFrom instance for a CC that's a traversable.
  class GenericCanBuildFrom[A] extends CanBuildFrom[CC[_], A, CC[A]] {
    def apply(from: CC[_]) = from.genericBuilder[A]
  }
}
```

Listing 7: `GenericCanBuildFrom`

class `Traversable`, which implements all accesses to its data via its `foreach` method. Class `Traversable` is extended by class `Iterable`, which implements all traversals by means of an iterator. `Iterable` is further extended by classes `Seq`, `Set`, and `Map`. Each of these classes has further subclasses that capture some particular trait of a collection. For instances, sequences `Seq` are split in turn into `LinearSeq` for linear access sequences such as lists and `IndexedSeq` for random access sequences such as arrays.

All collection classes are kept in a package `scala.collection`. This package has three subpackages: `mutable`, `immutable`, and `generic`. Most collections exist in three forms, depending on their mutability.

A collection in package `scala.collection.immutable` is guaranteed to be immutable for everyone. That means one can rely on the fact that accessing the same collection value over time will always yield a collection with the same elements.

A collection in package `scala.collection.mutable` is known to have some operations that change the collection in place.

A collection in package `scala.collection` can be either mutable or immutable. For instance, `collection.Seq[T]` is a superclass of both `collection.immutable.Seq[T]` and `collection.mutable.Seq[T]`. Generally, the root collections in package `scala.collection` define the same interface as the immutable collections, and the mutable collections in package `scala.collection.mutable` typically add some destructive modification operations to this immutable interface. The difference between root collections and immutable collections is that a user of an immutable collection has a guarantee that nobody can mutate the collection, whereas users of root collections have to assume modifications by

```
package mycollection
import collection.generic.{CanBuildFrom, GenericTraversableTemplate,
    GenericCompanion, SeqFactory}
import collection.mutable.{Builder, ArrayBuffer}

class Vector[+A](buf: ArrayBuffer[A])
  extends collection.immutable.IndexedSeq[A]
     with collection.IndexedSeqLike[A, Vector[A]]
     with GenericTraversableTemplate[A, Vector] {
  override def companion: GenericCompanion[Vector]  = Vector
  def length = buf.length
  def apply(idx: Int) = buf.apply(idx)
}

object Vector extends SeqFactory[Vector] {
  implicit def canBuildFrom[A]: CanBuildFrom[Vector[_], A, Vector[A]] =
    new GenericCanBuildFrom[A]
  def newBuilder[A]: Builder[A, Vector[A]] =
    new ArrayBuffer[A] mapResult (buf ⇒ new Vector(buf))
}
```

Listing 8: A sample collection implementation.

others, even though they cannot do any modifications themselves.

The `generic` package contains building blocks for implementing various collections. Typically, collection classes defer the implementations of some of their operations to classes in `generic`. Users of the collection framework, on the other hand, should need to refer at classes in `generic` only in exceptional circumstances.

Integrating new collections

The collection framework is designed to make it easy to add new kinds of collections to it. As an example, Listing 8 shows a simple yet complete implementation of immutable vectors.

The `Vector` trait inherits from three other traits. It inherits from `scala.collection.immutable.IndexedSeq` to specify that `Vector` is a subtype of a random access sequence and is immutable. It inherits most of implementations of its methods from the `IndexedSeqLike` trait, specialising the representation type to `Vector[A]`. Finally, `Vector` mixes in `GenericTraversableTemplate`, and instantiates the type parameter that abstracts over the collection type constructor to `Vector`.

Only three abstract methods remain to be implemented. Two of these, `length` and `apply`, are related to querying an existing sequence, while the third, `companion`, is involved in creating new sequences. The method `length` yields the length of the sequence, and `apply` returns an element at a given index. These two operations are implemented in trait `Vector`. For simplicity's sake they simply forward to the same operation of an underlying `ArrayBuffer`. Of course, the actual implementation of immutable vectors is considerably

more refined algorithmically, and more efficient.

The third method, `companion`, is declared in `GenericTraversableTemplate`. `Vector` defines it to refer to its companion object, which specifies the `CanBuildFrom` case for `Vector`. This ensures that calling `map` on a `Vector` yields a `Vector`. As discussed in Section 7, the implicit value that populates the `CanBuildFrom` relation on types is an instance of `GenericCanBuildFrom`, which delegates the creation of the `Vector`-specific builder to the `newBuilder` method. This method creates an array buffer (which is a specialised kind of builder), and transforms results coming out of this buffer into instances of `Vector`. That's the minimal functionality required for instances of `GenericTraversableTemplate`.

As an added convenience, the `Vector` object inherits from class `SeqFactory` which makes available a large set of creation methods for vectors.

With the setup as described in Listing 8 the `Vector` class is fully integrated into the collections hierarchy. It inherits all methods defined on indexed sequences and all construction methods for such sequences can be applied to it. The following REPL script shows some of the operations that are supported. First, here are some ways to construct vectors:

```
import mycollection.Vector

scala> val v = Vector(1, 2, 3)
v:  mycollection.Vector[Int] = Vector(1, 2, 3)

scala> val ev = Vector.empty
ev:  mycollection.Vector[Nothing] = Vector()

scala> val zeroes = Vector.fill(10)(0)
zeroes:  mycollection.Vector[Int] = Vector(0, 0, 0, 0, 0, 0, 0, 0, 0,
   0)

scala> val squares = Vector.tabulate(10)(x ⇒ x * x)
squares:  mycollection.Vector[Int] = Vector(0, 1, 4, 9, 16, 25, 36,
   49, 64, 81)

scala> val names = Vector("Jane", "Bob", "Pierre")
names:  mycollection.Vector[java.lang.String] = Vector(Jane, Bob,
   Pierre)

scala> val ages = Vector(21, 16, 24)
ages:  mycollection.Vector[Int] = Vector(21, 16, 24)
```

To continue, here are some operations on vectors.

```
scala> val persons = names zip ages
persons:  mycollection.Vector[(java.lang.String, Int)] =
   Vector((Jane,21), (Bob,16), (Pierre,24))

scala> val (minors, adults) = persons partition (_._2 <= 18)
minors:  mycollection.Vector[(java.lang.String, Int)] =
   Vector((Bob,16))
adults:  mycollection.Vector[(java.lang.String, Int)] =
   Vector((Jane,21), (Pierre,24))
```

```
scala> val adultNames = adults map (_._2)
adultNames:  mycollection.Vector[Int] = Vector(21, 24)

scala> val totalAge = ages reduceLeft (_ + _)
totalAge:  Int = 61
```

To summarise: To fully integrate a new collection class into the framework one needs to pay attention to the following points:

1. Decide whether the collection should be mutable or immutable.
2. Pick the right base classes for the collection.
3. Inherit from the right template trait to implement most collection operations.
4. If one wants `map` and similar operations return instances of the collection type, provide an implicit builder factory in the companion object.
5. If the collection should have dynamic type adaptation for `map` and operations like it, one should also inherit from `GenericTraversableTemplate`, or implement equivalent functionality.

A simpler scheme is also possible if one does not need bulk operations like `map` or `filter` to return the same collection type. In that case one can simply inherit from some general collection class like `Seq` or `Map` and implement any additional operations directly.

9 Dealing with Arrays and Strings

The integration of arrays into the Scala collection library has turned out to be very challenging. This has mostly to do with the clash between requirements and the constraints imposed by Java and the JVM. On the one hand, arrays play an important role for interoperation with Java, which means that they need to have the same representation as in Java. This low-level representation is also useful to get high performance out of arrays. But on the other hand, arrays in Java are severely limited.

First, there's actually not a single array type representation in Java but nine different ones: one representation for arrays of reference type and another eight for arrays of each of the primitive types `byte`, `char`, `short`, `int`, `long`, `float`, `double`, and `boolean`. Unfortunately, `java.lang.Object` is the most specific common type for these different representations, even though there are some reflective methods to deal with arrays of arbitrary type in `java.lang.reflect.Array`. Second, there's no way to create an array of a generic type; only monomorphic array creations are allowed. Third, arrays only support operations for indexing, updating, and getting their length.

Contrast this with what we would like to have in Scala: Arrays should slot into the collections hierarchy, supporting the roughly one hundred methods that are defined on sequences. And they should certainly be generic, so that one can create an `Array[T]` where `T` is a type variable.

The previous collection design dealt with arrays in an ad-hoc way. The Scala compiler wrapped and unwrapped arrays when required in a process called boxing and unboxing, similarly to what is done to treat primitive numeric types as objects. Additional "magic" made generic array creation work. An expression like **`new`** `Array[T]` where `T` is a type

parameter was converted to **new** `BoxedAnyArray[T]`. `BoxedAnyArray` was a special wrapper class which *changed its representation* depending on the type of the concrete Java array to which it was cast. This scheme worked well enough for most programs but the implementation "leaked" for certain combinations of type tests and type casts, as well as for observing uninitialised arrays. It also could lead to unexpectedly low performance. Some of the problems have been described by David MacIver [11] and Matt Malone [12]. Moreover, boxed arrays were unsound when combined with covariant collections. In summary, the old array implementation technique was problematic because it was a leaky abstraction that was complicated enough so that it would be very tedious to specify where the leaks were to be expected.

The obvious way to reduce the amount of "magic" needed for arrays is to have two representations: one that corresponds closely to a Java array and another that forms an integral part of Scala's collection hierarchy. Implicit conversions can be used to transparently convert between the two representations. A possible downside of having two array types would be that it forces programmers to choose the kind of array to work with. That choice would not be clear-cut: the Java-like arrays would be fast and interoperable whereas the Scala native arrays would support a much nicer set of operations on them. With a choice like this, one would expect different components and libraries to make different decisions, which would result in incompatibilities and brittle, complex code. In a word, an ideal environment for future bit rot.

Fortunately, the introduction of implementation traits in 2.8 collections offers a way out of that dilemma of choice. Arrays can be integrated into this framework using *two* implicit conversions. The first conversion maps an `Array[T]` to an object of type `ArrayOps`, which is a subtype of type `IndexedSeqLike[T, Array[T]]`. Using this conversion, all sequence operations are available for arrays at the natural types. In particular, methods will always yield arrays instead of `ArrayOps` values as their results. Because the results of these implicit conversions are so short-lived, modern VM's can eliminate them altogether using escape analysis, so we expect the calling overhead for these added methods to be essentially zero.

So far so good. But what if we need to convert an array to a real `Seq`, not just call a `Seq` method on it? This is handled by another implicit conversion, which takes an array and converts it into a `WrappedArray`. `WrappedArray`s are mutable, indexed sequences that implement all sequence operations in terms of a given Java array. The difference between a `WrappedArray` and an `ArrayOps` object is apparent in the type of methods like `reverse`: Invoked on a `WrappedArray`, `reverse` again returns a `WrappedArray`, but invoked on an `ArrayOps` object, it returns an `Array`. The conversion from `Array` to `WrappedArray` is invertible. A dual implicit conversion goes from `WrappedArray` to `Array`. `WrappedArray` and `ArrayOps` both inherit from an implementation trait `ArrayLike`. This is to avoid duplication of code between `ArrayOps` and `WrappedArray`; all operations are factored out into the common `ArrayLike` trait.

Avoiding ambiguities. The two implicit conversions from `Array` to `ArrayLike` values are disambiguated according to the rules explained in Section 7. Applied to arrays, this means that we can prioritise the conversion from `Array` to `ArrayOps` over the conversion from `Array` to `WrappedArray` by placing the former in the standard `Predef` object (which is vis-

ible in all user code) and by placing the latter in a class `LowPriorityImplicits`, which is inherited by `Predef`. This way, calling a sequence method will always invoke the conversion to `ArrayOps`. The conversion to `WrappedArray` will only be invoked when an array needs to be converted to a sequence.

Integrating Strings. Strings pose similar problems as arrays in that we are forced to pick an existing representation which is not integrated into the collection library and which cannot be extended with new methods because Java's `String` class is **`final`**. The solution for strings is very similar as the one for arrays. There are two prioritised implicit conversions that apply to strings. The low-priority conversion maps a string to an immutable indexed sequence of type `scala.collection.immutable.IndexedSeq`. The high-priority conversion maps a string to a (short-lived) `StringOps` object which implements all operations of an immutable indexed sequence, but with `String` as the result type. The previous collection framework implemented only the first conversion. This had the following undesirable effect:

```
"abc" != "abc".reverse.reverse
```

This unintuitive behaviour occurred because the result of the double `reverse` in previous Scala collections was a `Seq` instead of a `String`, so Java's built-in operation of equality an strings failed to recognise it as equal to the string. In the new collection framework, the high-priority conversion to `StringOps` will be applied instead, so that `"abc".reverse.reverse` yields a `String` and the equality holds.

Generic Array Creation and Manifests. The only remaining question is how to implement generic array creation. Unlike Java, Scala allows an instance creation **`new`** `Array[T]` where `T` is a type parameter. How can this be implemented, given the fact that there does not exist a uniform array representation in Java? The only way to do this is to require additional run-time information which describes the type `T`. Scala 2.8 has a new mechanism for this, which is called a `Manifest`. An object of type `Manifest[T]` provides complete information about the type `T`. Manifest values are typically passed in implicit parameters, and the compiler knows how to construct them for statically known types `T`. There exists also a weaker form named `ClassManifest` which can be constructed from knowing just the top-level class of a type, without necessarily knowing all its argument types. It is this type of runtime information that's required for array creation.

10 Conclusion

As this paper is written we are about to release Scala 2.8 with its new collections library. So it is too early to tell whether the new design withstands bit rot better than the old one did. Nevertheless, we have reasonable grounds for hoping that this will be the case.

The new collection design is far more regular than the old one and makes many aspects of its structure more explicit. Mutability aspects are consistently expressed by placing collections in the right package. Reusable method implementations are separated from client interfaces in implementation classes. This allowed us to have simple and intuitive types

like `Seq[String]` or `Map[String, Int]` for clients yet have implementation classes expose their representation as in additional type parameter that can be instantiated as needed by implementers. There is a common universal framework of builders and traversal methods. Code duplication is almost completely absent (There is still a certain amount of duplicated boilerplate code in the definition of so called *views*, which are by-name transforms of existing collections, but these views are typically not extended by third parties). Arrays and strings are cleanly integrated into the collections framework with implicit conversions instead of requiring special compiler support.

Getting this design right was very hard, however. It took us about a year to go from a first sketch to the final implementation. In doing this work, we also encountered some dead ends. Initially, we anticipated that most of the flexibility and opportunities for code-reuse of the framework would come from higher-kinded types. In retrospect this turned out to be a false assumption, because requirements on the element type of collections varied from collection to collection. So common methods on collections had to be defined piece-wise. They would return a specialised collection for some element types, and a more general "fall-back" collection for other element types. In the course of the project, we learned how to use implicits to define these piece-wise functions. More generally, we came to appreciate how implicits can encode rich user-defined type theories. So, in the end higher-kinded types played a smaller role than anticipated and implicits played a much larger role.

Nevertheless, type constructor polymorphism did find a useful application niche in the collections framework, where it came to generate factories for collection classes. This application worked out fine because setting up a factory by inheriting from a factory class which takes higher-kinded type parameters is done on a case-by-case basis. Collections which pose additional constraints on the higher-kinded type parameter can simply choose not to inherit from `TraversableFactory` and implement the required methods themselves. By contrast, implementation classes follow a subtyping hierarchy; any specification made higher up in the hierarchy needs to hold up for all inheriting classes. So the lesson drawn is not that higher-kinded types per se are of limited utility, but that they sometimes interact in awkward ways with a rich subtyping hierarchy. In some sense this is a new facet of the fragile baseclass problem.

Acknowledgments The final architecture and implementation of collections was mainly done by Odersky but several people have shaped the design in important ways. The new libraries would not exist without their contributions. Matthias Zenger wrote Scala's original collection libraries for sets, maps, buffers, and other types. Many of his design decisions have survived the redesign. Some have been generalised, such as his partition into mutable and immutable collection packages, which now applies uniformly for all kinds of collections including sequences. Sean McDirmid added projections to the original collection libraries, a concept which has been taken up in the redesign under the name of views. Adriaan Moors developed higher-kinded types in Scala, which gave the primary motivation for the collection redesign, even though in the end their role is more narrow than originally anticipated. Adriaan was also the first to explore builders as a fundamental abstraction for Scala's collections. David McIver proposed builders as implicit parameters and `Traversable` as a generalisation of `Iterable`. Miles Sabin contributed the bidirectional wrappers that con-

vert between Java collections and Scala collections. Phil Bagwell, Gilles Dubochet, Burak Emir, Erik Engbrecht, Stepan Koltsov, Stéphane Micheloud, Tony Morris, Jorge Ortiz, Paul Phillips, David Pollak, Tiark Rompf, Lex Spoon, and many others have contributed to specific collection classes or made important suggestions for improvements.

Bibliography

[1] Andrew P. Black, Nathanael Schärli, and Stéphane Ducasse. Applying traits to the Smalltalk collection classes. In Ron Crocker and Guy L. Steele Jr., editors, *OOPSLA*, pages 47–64. ACM, 2003.

[2] Kim B. Bruce, Albert R. Meyer, and John C. Mitchell. The semantics of second-order lambda calculus. *Inf. Comput.*, 85(1):76–134, 1990.

[3] William R. Cook. Interfaces and specifications for the Smalltalk-80 collection classes. In *OOPSLA*, pages 1–15, 1992.

[4] Vincent Cremet and Philippe Altherr. Adding type constructor parameterization to Java. *Journal of Object Technology*, 7(5):25–65, June 2008. Special Issue: Workshop on FTfJP, ECOOP 07. http://www.jot.fm/issues/issue_2008_06/article2/.

[5] Derek Dreyer, Robert Harper, Manuel M. T. Chakravarty, and Gabriele Keller. Modular type classes. In Martin Hofmann and Matthias Felleisen, editors, *POPL*, pages 63–70. ACM, 2007.

[6] J.Y. Girard. Interpretation fonctionelle et elimination des coupures de l'arithmetique d'ordre superieur. These d'Etat, Paris VII, 1972.

[7] Adele Goldberg and David Robson. *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley, 1983.

[8] Paul Hudak, Simon L. Peyton Jones, Philip Wadler, Brian Boutel, Jon Fairbairn, Joseph H. Fasel, María M. Guzmán, Kevin Hammond, John Hughes, Thomas Johnsson, Richard B. Kieburtz, Rishiyur S. Nikhil, Will Partain, and John Peterson. Report on the programming language Haskell, a non-strict, purely functional language. *SIGPLAN Notices*, 27(5):R1–R164, 1992.

[9] Mark P. Jones. A system of constructor classes: Overloading and implicit higher-order polymorphism. *J. Funct. Program.*, 5(1):1–35, 1995.

[10] Barbara Liskov. Keynote address - data abstraction and hierarchy. In *OOPSLA '87: Addendum to the proceedings on Object-oriented programming systems, languages and applications (Addendum)*, pages 17–34, New York, NY, USA, 1987. ACM.

[11] David MacIver. Scala arrays, 2008. Blog post at `http://www.drmaciver.com/2008/06/scala-arrays`.

[12] Matt Malone. The mystery of the parameterized array, 2009. Blog post at `http://oldfashionedsoftware.com/2009/08/05/the-mystery-of-the-parameterized-array`.

[13] Adriaan Moors, Frank Piessens, and Martin Odersky. Generics of a higher kind. In Gail E. Harris, editor, *OOPSLA*, pages 423–438. ACM, 2008.

[14] Maurice Naftalin and Philip Wadler. *Java Generics and Collections*. O'Reilly Media, Inc., 2006.

[15] Martin Odersky. Pimp my library, 2006. Blog post at `http://www.artima.com/weblogs/viewpost.jsp?thread=179766`.

[16] Martin Odersky, Philippe Altherr, Vincent Cremet, Iulian Dragos, Gilles Dubochet, Burak Emir, Sean McDirmid, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Lex Spoon, Erik Stenman, and Matthias Zenger. An Overview of the Scala Programming Language (2. edition). Technical report, 2006.

[17] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala*. Artima, 2008.

[18] John C. Reynolds. Towards a theory of type structure. In Bernard Robinet, editor, *Symposium on Programming*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1974.

[19] Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz, and Andrew P. Black. Traits: Composable units of behaviour. In *ECOOP*, pages 248–274, 2003.

[20] Philip Wadler and Stephen Blott. How to make ad-hoc polymorphism less ad-hoc. In *POPL*, pages 60–76, 1989.

[21] Stefan Wehr, Ralf Lämmel, and Peter Thiemann. JavaGI : Generalized interfaces for Java. In Erik Ernst, editor, *ECOOP*, volume 4609 of *Lecture Notes in Computer Science*, pages 347–372. Springer, 2007.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Iterative Methods in Combinatorial Optimization

R. Ravi[1]*
Tepper School of Business
Carnegie Mellon University
Pittsburgh, USA
ravi@cmu.edu

ABSTRACT. We describe a simple iterative method for proving a variety of results in combinatorial optimization. It is inspired by Jain's iterative rounding method (FOCS 1998) for designing approximation algorithms for survivable network design problems, and augmented with a relaxation idea in the work of Lau, Naor, Salvatipour and Singh (STOC 2007) on designing an approximation algorithm for its degree bounded version. At the heart of the method is a counting argument that redistributes tokens from the columns to the rows of an LP extreme point. This token argument was further refined to fractional assignment and redistribution in work of Bansal, Khandekar and Nagarajan on degree-bounded directed network design (STOC 2008).
In this presentation, we introduce the method using the assignment problem, describe its application to showing the integrality of Edmond's characterization (1971) of the spanning tree polyhedron, and then extend the argument to show a simple proof of the Singh and Lau's approximation algorithm (STOC 2007) for its degree constrained version, due to Bansal, Khandekar and Nagarajan. We conclude by showing how Jain's original proof can also be simplified by using a fractional token argument (joint work with Nagarajan and Singh).
This presentation is extracted from an upcoming monograph on this topic co-authored with Lau and Singh.

1 Introduction

Iterative methods are an important tool in the growing toolkit available for designing approximation algorithms based on linear programming relaxations. First we motivate our method via the assignment problem. Through this problem we highlight the basic ingredients and ideas of the method and provide an outline of how a typical result proved using this method is structured. In the following sections, we apply this method to the classical minimum spanning tree problem, and extend it to derive an approximation algorithm for the degree-bounded version. In the last section, we present an application to re-derive an old result of Jain on LP extreme points for survivable network design problems.

The Assignment Problem: Consider the classical assignment problem: Given a bipartite graph $G = (U \cup V, E)$ with $|U| = |V|$ and weight function $w : E \to \mathbb{R}_+$, the objective is to match every vertex in U with a distinct vertex in V to minimize the total weight (cost) of the matching. This is also called the minimum weight bipartite perfect matching problem in the literature, and is a fundamental problem in combinatorial optimization.

One approach to the assignment problem is to model it as a linear programming problem. A linear program is a mathematical formulation of the problem with a system of linear

*Supported in part by NSF grant CCF-0728841.

© Ravi; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 453–469
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2339

constraints which can contain both equalities and inequalities, and also a linear objective function that is to be maximized or minimized. In the assignment problem, we associate a *variable* x_{uv} for every $(u, v) \in E$. Ideally, we would like the variables to take one of two values, zero or one (hence in the ideal case, they are binary variables). When x_{uv} is set to one, we intend the model to signal that this pair is matched; when x_{uv} is set to zero, we intend the model to signal that this pair is not matched. The following is a linear programming formulation of the assignment problem.

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{u,v} w_{uv}\, x_{uv} & \\
\text{subject to} & \displaystyle\sum_{v:\{u,v\}\in E} x_{uv} = 1 & \forall\, u \in U \\
& \displaystyle\sum_{u:\{u,v\}\in E} x_{uv} = 1 & \forall\, v \in V \\
& x_{uv} \geq 0 & \forall\, \{u, v\} \in E
\end{array}
$$

The objective function is to minimize the total weight of the matching, while the two sets of linear equalities ensure that every vertex in U is matched to exactly one vertex in V in the assignment and vice-versa.

A fundamental result in the Operations Research literature [8] is the polynomial time solvability (as well as the practical tractability) of linear programming problems. There is also a rich theory of optimality (and certificates for it) that has been developed (see e.g., the text by Chvatal [3]). Using these results, we can solve the problem we have formulated above quite effectively for even very large problem sizes.

Returning to the formulation however, our goal is to find a "binary" assignment of vertices in U to vertices in V, but in the solution returned, the x-variables may take fractional values. Nevertheless, for the assignment problem, a celebrated result that is a cornerstone of combinatorial optimization [2] states that for any set of weights that permit a finite optimal solution, there is always an optimal solution to the above LP (linear program) that takes binary values in all the x-variables.

Such *integrality* results of LPs are few and far between, but reveal rich underlying structure for efficient optimization over the large combinatorial solution space [13]. They have been shown using special properties of the constraint matrix of the problem (such as total unimodularity), or of the whole linear system including the right hand side (such as total dual integrality). This article is about a simple and fairly intuitive method that is able to re-prove many (but not all) of the results obtained by these powerful methods. One advantage of our approach is that it can be used to incorporate additional constraints that make the problem computationally hard, and allow us to derive good approximation algorithms with provable performance guarantee for the constrained versions.

2 Iterative Algorithm

Our method is iterative. Using the following two steps, it works inductively to show that the LP has an integral optimal solution.

- If any x_{uv} is set to 1 in an optimal solution to the LP, then we take this pair as matched in our solution, and delete them both to get a smaller problem, and proceed to the next iteration.
- If any variable x_{uv} is set to 0 in the optimal solution, we remove the edge (u, v) to again get a smaller problem (since the number of edges reduces by 1) and proceed to the next iteration.

We continue the above iterations till all variables have been fixed to either 0 or 1. Given the above iterative algorithm, there are two claims that need to be proven. Firstly, that the algorithm works correctly, i.e., it can always find a variable with value 0 or 1 in each iteration and secondly, the matching selected is an optimal (minimum weight) matching. Assuming the first claim, the second claim can be proved by a simple inductive argument. The crux of the argument is that in each iteration our solution pays exactly what the fractional optimal solution pays. Moreover, the fractional optimal solution when restricted to the residual problem remains feasible for the residual problem. This allows us to apply an inductive argument to show that the matching we construct has the same weight as the fractional optimal solution, and is thus optimal. For the first claim, it is not clear a-priori that one can always find a variable with value 1 or 0 at every step. However, we use the important concept of the extreme point (or vertex) solutions of linear program to show that the above iterative algorithm works correctly.

DEFINITION 1. *Let $P = \{x : Ax = b, x \geq 0\} \subseteq \mathbb{R}^n$. Then $x \in \mathbb{R}^n$ is an* **extreme point solution** *of P if there does not exist a non-zero vector $y \in \mathbb{R}^n$ such that $x + y, x - y \in P$.*

Extreme point solutions are also known as vertex solutions and are equivalent to basic feasible solutions [3]. The following basic result shows that there is always an optimal extreme point solution to bounded linear programs.

LEMMA 2. *Let $P = \{x : Ax = b, x \geq 0\}$ and assume that the optimum value $\min\{c^T x : x \in P\}$ is finite. Then for any feasible solution $x \in P$, there exists an extreme point solution $x' \in P$ with $c^T x' \leq c^T x$.*

The following **"Rank lemma"** is an important ingredient in the correctness proofs of all iterative algorithms.

LEMMA 3. *Let $P = \{x : Ax = b, x \geq 0\}$ and let x be an extreme point solution of P such that $x_i > 0$ for each i. Then the number of variables is equal to the number of linearly independent constraints of A, i.e. the rank of A.*

2.1 Contradiction Proof Idea: Lower Bound > Upper Bound

We give an outline of the proof that at each iteration there exists a variable with value 0 or 1. Suppose for contradiction that $0 < x_e < 1$ for every edge e. We use this assumption to derive a lower bound on the number of variables of the linear program. Let n be the remaining vertices in U (or V, they have the same cardinality) at the current iteration. Then each vertex in U must have two edges incident on it, since $\sum_{v \in V:(u,v) \in E} x_{uv} = 1$ and $x_{uv} < 1$ for each $(u, v) \in E$. Thus the total number of edges is at least $2n$. This is a lower bound on the number of variables of the linear program, since we have one variable for each edge.

On the other hand, using the Rank Lemma, we derive an upper bound on the number of variables of the linear program. In the linear program for bipartite matching, we have only $2n$ constraints (one for each vertex in $U \cup V$). Moreover, these $2n$ constraints are dependent since the sum of the constraints for vertices in U equals the sum of the constraints for vertices in V. Hence, the number of linearly independent constraints is at most $2n - 1$. By the Rank Lemma, the number of variables is at most $2n - 1$. This provides us an upper bound on the number of variables. Since our upper bound is strictly smaller than the lower bound, we obtain the desired contradiction. Therefore, in an extreme point solution of the linear program for bipartite matching, there must exist a variable with value 0 or 1, and thus the iterative algorithm works. The number of iterations can be simply bounded by the number of edges in the bipartite graph.

3 Outline of the Approach

We now give a brief outline of the approach to designing algorithms with this approach. The method can be used to prove the integrality of the LP relaxation of a well-studied problem, and once this is well understood, the iterative proof of integrality can be extended to design approximation algorithms for NP-hard variants of the basic problems. Both components follow the natural outline described below.

1. **Linear Programming Formulation:** We start by giving a linear programming relaxation for the optimization problem we study. If the problem is polynomially solvable, this relaxation will be one with integral extreme points and that is what we will set out to show. If the problem is NP-hard, we state an approximation algorithmic result which we then set out to prove.
   (a) **Solvability:** Sometimes the linear programming relaxation we start with will be exponential in size. We then show that the linear program is solvable in polynomial time. Usually, this would entail providing a polynomial time *separation oracle* for the program using the formalism of the ellipsoid method [7]. Informally, the separation oracle is a procedure that certifies that any given candidate solution for the program is either feasible or not and in the latter case provides a separating hyperplane which is a violated inequality of the formulation. In programs with an exponential number of such inequalities that are implicity described, the design of the separation oracle is itself a combinatorial optimization problem, and we sketch the reduction to one.
2. **Characterization of Extreme Point Solution:** We then give a characterization result for the optimal extreme point solutions of the linear program based on the Rank Lemma 3. This part aims to show that any maximal set of independent tight constraints at this extreme point solution can be captured by a sparse structure. Sometimes the proof of this requires the use of the *uncrossing* technique [2] in combinatorial optimization.
3. **Iterative Algorithm:** We present an iterative algorithm for constructing an integral solution to the problem from the vertex solution. The algorithm has two simple steps.
   (a) If there is a variable in the optimal vertex solution that is set to a value of 1, then include the element in the integral solution.
   (b) If there is a variable in the optimal vertex solution that is set to a value of 0, then

remove the corresponding element.

In each of the above cases, at each iteration, we reduce the problem and arrive at a *residual* version and iterate until all variables have been set this way. In designing approximation algorithms we also use the rounding and relaxation steps as stated earlier.

4. **Analysis:** We then analyze the algorithm. This involves arguing the following two facts. First, we establish that the algorithm runs correctly and second, that it returns an optimal solution.
   (a) **Correctness:** We show that the iterative algorithm is correct by arguing that there is always a 1-element or a 0-element to pick in every iteration. This crucially uses the characterization of tight constraints at this optimal extreme point solution. The argument here also follows the same contradiction proof idea (lower bound $>$ upper bound): We assume for a contradiction that there is no 1-element or 0-element and get a large lower bound on the number of nonzero variables in the optimal extreme point solution. On the other side, we use the sparsity of the independent tight constraints to show an upper bound on the number of such constraints. This then contradicts the rank lemma that insists that both these numbers are equal, and proves that there is always a 1- or 0-element.
   (b) **Optimality:** We finally show that the iterative algorithm indeed returns an optimal solution using a simple inductive argument. The crux of this argument is to show that the extreme point solution induced on the residual problem remains a feasible solution to this residual problem.

3.1 Approximation Algorithms for NP-hard Problems

The above framework can be naturally adapted to provide an approximation algorithm via the iterative method. In particular, for this, the iterative algorithm above typically has one or both of two additional steps: *Rounding* and *Relaxation*.

1. **Rounding:** Fix a threshold $\alpha \geq 1$. If there is a variable x_i which in the optimal extreme point solution has a value of at least $\frac{1}{\alpha}$ then include the corresponding element in the solution.
   Adding this rounding step does not allow us to obtain optimal integral solution but only near-optimal solutions. Using the above step, typically one obtains an approximation ratio of $\frac{1}{\alpha}$ for covering problems addressed using this framework.
2. **Relaxation:** Fix a threshold β. If there is a constraint $\sum_i a_i x_i \leq b$ such that $\sum_i a_i \leq b + \beta$ then remove the constraint in the residual formulation.
   The iterative relaxation step removes a constraint and hence this constraint can be violated in later iterations. But the condition on the removal of the constraints ensures that the constraint is only violated by an additive amount of β. This step enables us to obtain *additive* approximation algorithms for a variety of problems.

To summarize, for designing approximation algorithms, we first study the exact optimization problem in the above framework. We then use the above two steps in various combinations to derive strong approximation algorithms for constrained versions of these exact problems.

4 Minimum Spanning Trees

In an instance of the Minimum Spanning Tree (MST) problem we are given an undirected graph $G = (V, E)$, edge costs given as $c : E \rightarrow \mathbb{R}$, and the task is to find a spanning tree of minimum total edge cost.

4.1 Linear Programming Relaxation

An exact linear formulation for the convex hull of integral spannign trees is the subtour elimination LP which is related to the study of the Traveling Salesman Problem. For $S \subseteq V$, define $E(S)$ to be the set of edges with both endpoints in S. For a spanning tree, there are at most $|S| - 1$ edges in $E(S)$, where $|S|$ denotes the number of vertices in S. Insisting on this for every set by using the constraint (2) eliminates all the potential subtours that can be formed in the LP solution: this is how the formulation gets its name.

$$\begin{aligned}
&\text{minimize} && \sum_{e \in E} c_e x_e && && (1)\\
&\text{subject to} && x(E(S)) \leq |S| - 1 && \forall\, \emptyset \neq S \subset V && (2)\\
& && x(E(V)) = |V| - 1 && && (3)\\
& && x_e \geq 0 && \forall\, e \in E && (4)
\end{aligned}$$

We will present an iterative algorithm which will prove that the subtour LP is integral.

THEOREM 4. *Every extreme point solution to the subtour LP is integral and corresponds to the characteristic vector of a spanning tree.*

Before we give the iterative algorithm and proof of Theorem 4, we show that one can optimize over the subtour LP in polynomial time. We show this giving a polynomial time separation oracle for the constraints in subtour LP. Polynomial time solvability now follows from results on the equivalence of separation and optimization [7].

THEOREM 5. *There is a polynomial time separation oracle for the subtour LP.*

PROOF. The separation oracle, given a fractional solution x, needs to find a set $S \subseteq V$ such that $x(E(S)) > |S| - 1$ if such a set exists. It is easy to check the equality $x(E(V)) = |V| - 1$. Thus, checking the inequality for S is equivalent to checking if $\min_S \{|S| - 1 - x(E(S))\} < 0$. Using $x(E(V)) = |V| - 1$ we obtain that it is enough to check $\min_S \{|S| - 1 + x(E(V)) - x(E(S)\} < |V| - 1\}$ or equivalently if $\min_S \{|S| + x(E(V)) - x(E(S)\} < |V|\}$.

We set up a min-cut problem in a new digraph D with a new source s and new sink t. We also have one node in the digraph per edge e in the support (i.e., with $x_e > 0$) and a node per vertex of G. The source s has an arc to every edge e with capacity x_e. For every edge $e = i, j$ in G, its corresponding node in D has two arcs of infinite capacity, one to each of the vertices i and j. Finally, every vertex i has an arc of unit capacity to t. To find a violated cut, we need to check if the min $s - t$ cut is smaller than $|V|$.

Suppose there is a violated set S with $x(E(S)) > |S| - 1$. Then consider the cut formed by including on the side of s, all the nodes of D corresponding to edges in $E(S)$ as well as the

vertices of S. The set of arcs coming out of this cut are those coming out of the vertices of S and hence have capacity $|S|$. All edges not in $E(S)$, namely in $E(V) - E(S)$, must now have their incoming arc from s in the cut for a total capacity contribution of $x(E(V)) - x(E(S))$. Thus a violated cut will have cut value less than $|V|$.

Conversely, suppose the min-cut solution returns one of value less than $|V|$: we show how to extract a violated set from it. Since every node in D corresponding to an edge e of G has both its outgoing arcs with infinite capacity, if such a node (say $e = i, j$) is in the s-side of the min cut, then both its successors (i.e. both i and j) must also be in the s-side of the minimum cut. Similarly, if we take all the vertices of G in the s-side of the min-cut, all edges of G which do not have both their endpoints in this set will have to lie on the t-side of this cut. If the min-cut found has the set S' in the s-side of the cut, the capacity of the cut is precisely $|S|$ (from the unit arcs going from these nodes to t) plus the x-value of all edges that do not have both end points in S, namely $x(E(V)) - x(E(S'))$ as required. If this min-cut value is less than $|V|$, we can see that S' is a violating set. ∎

4.2 Characterizations of Extreme Point Solutions via the Uncrossing Technique

In this subsection, we analyze the extreme point solution to the subtour LP. Recall that an extreme point solution is the unique solution defined by n linearly independent tight inequalities, where n is the number of variables in the linear program. There are exponentially many inequalities in the subtour LP, and an extreme point solution may satisfy many inequalities as equalities. To analyze an extreme point solution, an important step is to find a "good" set of tight inequalities defining it. If there is an edge e with $x_e = 0$, this edge can be removed from the graph without affecting the feasibility and the objective value. So henceforth assume every edge e has $x_e > 0$.

The uncrossing technique is a powerful technique and we shall use it to find a *good* set of tight inequalities for an extreme point solution in the subtour LP. Let $E(X, Y)$ denotes the set of edges with one endpoint in X and the other endpoint in Y, and let $E(X) = E(X, X)$ denote the set of edges of G induced in $X \subseteq V(G)$. For a set $F \subseteq E$, let $\chi(F)$ denote the vector in $\mathbb{R}^{|E|}$ that has an 1 corresponding to each edge $e \in F$, and 0 otherwise. This vector is called the *characteristic vector* of F. The following proposition is straightforward.

PROPOSITION 6. *For $X, Y \subseteq V$,*

$$\chi(E(X)) + \chi(E(Y)) \leq \chi(E(X \cup Y)) + \chi(E(X \cap Y)),$$

and equality holds if and only if $E(X \setminus Y, Y \setminus X) = \emptyset$.

PROOF. Observe that

$$\chi(E(X)) + \chi(E(Y)) = \chi(E(X \cup Y)) + \chi(E(X \cap Y)) - \chi(E(X \setminus Y, Y \setminus X))$$

and proof follows immediately. ∎

Given an extreme point solution x to the subtour LP, let $\mathcal{F} = \{S \mid x(E(S)) = |S| - 1\}$ be the family of tight inequalities for an extreme point solution x in the subtour LP. The following lemma shows that this family is closed under intersection and union.

LEMMA 7. *If $S, T \in \mathcal{F}$ and $S \cap T \neq \emptyset$, then both $S \cap T$ and $S \cup T$ are in $\mathcal{F}$. Furthermore, $\chi(E(S)) + \chi(E(T)) = \chi(E(S \cap T)) + \chi(E(S \cup T))$.*

PROOF. Observe that

$$\begin{aligned} |S| - 1 + |T| - 1 &= x(E(S)) + x(E(T)) \\ &\leq x(E(S \cap T)) + x(E(S \cup T))) \\ &\leq |S \cap T| - 1 + |S \cup T| - 1 \\ &= |S| - 1 + |T| - 1. \end{aligned}$$

The first equality follows from the fact that $S, T \in \mathcal{F}$. The second inequality follows from Proposition 6. The third inequality follows from the constraints for $S \cap T$ and $S \cup T$ in the subtour LP. The last equality is because $|S| + |T| = |S \cap T| + |S \cup T|$ for any two sets S, T. Equality must hold everywhere and we have $x(E(S \cap T)) + x(E(S \cup T)) = |S \cap T| - 1 + |S \cup T| - 1$. Thus, we must have equality for constraints for $S \cap T$ and $S \cup T$, i.e., $x(E(S \cap T)) = |S \cap T| - 1$ and $x(E(S \cup T)) = |S \cup T| - 1$, which implies that $S \cap T$ and $S \cup T$ are also in $\mathcal{F}$. Moreover, equality holds for Proposition 6 and thus $\chi(E(S \setminus T, T \setminus S)) = \emptyset$ and $\chi(E(S)) + \chi(E(T)) = \chi(E(S \cap T)) + \chi(E(S \cup T))$. ∎

Denote by $span(\mathcal{F})$ the vector space generated by the set of vectors $\{\chi(E(S)) \mid S \in \mathcal{F}\}$. Call two sets X, Y *intersecting* if $X \cap Y$, $X - Y$ and $Y - X$ are nonempty. A family of sets is *laminar* if no two sets are intersecting. The following lemma says that an extreme point solution is characterized by tight inequalities whose corresponding sets form a laminar family. This is a crucial structure theorem on the extreme point solutions for the subtour LP.

LEMMA 8. *If $\mathcal{L}$ is a maximal laminar subfamily of $\mathcal{F}$, then $span(\mathcal{L}) = span(\mathcal{F})$.*

PROOF. Suppose, by way of contradiction, that $\mathcal{L}$ is a maximal laminar subfamily of $\mathcal{F}$ but $span(\mathcal{L}) \subset span(\mathcal{F})$. For any $S \notin \mathcal{L}$, define $intersect(S, \mathcal{L})$ to be the number of sets in $\mathcal{L}$ which intersect S, i.e. $intersect(S, \mathcal{L}) = |\{T \in \mathcal{L} \mid S \text{ and } T \text{ are intersecting}\}|$. Since $span(\mathcal{L}) \subset span(\mathcal{F})$, there exists a set S with $\chi(E(S)) \notin span(\mathcal{L})$. Choose such a set S with minimum $intersect(S, \mathcal{L})$. Clearly, $intersect(S, \mathcal{L}) \geq 1$; otherwise $\mathcal{L} \cup \{S\}$ is also a laminar subfamily, contradicting the maximality of $\mathcal{L}$. Let T be a set in $\mathcal{L}$ which intersects S. Since $S, T \in F$, by Lemma 7, both $S \cap T$ and $S \cup T$ are in $\mathcal{F}$. Also, both $intersect(S \cap T, \mathcal{L})$ and $intersect(S \cup T, \mathcal{L})$ are smaller than $intersect(S, \mathcal{L})$, which will be proved next in Proposition 9. Hence, by the minimality of $intersect(S, \mathcal{L})$, both $S \cap T$ and $S \cup T$ are in $span(\mathcal{L})$. By Lemma 7, $\chi(E(S)) + \chi(E(T)) = \chi(E(S \cap T)) + \chi(E(S \cup T))$. Since $\chi(E(S \cap T))$ and $\chi(E(S \cup T))$ are in $span(\mathcal{L})$ and $T \in \mathcal{L}$, the above equation implies that $\chi(E(S)) \in span(\mathcal{L})$, a contradiction. It remains to prove Proposition 9.

PROPOSITION 9. *Let S be a set that intersects $T \in \mathcal{L}$. Then $intersect(S \cap T, \mathcal{L})$ and $intersect(S \cup T, \mathcal{L})$ are smaller than $intersect(S, \mathcal{L})$.*

PROOF. Since $\mathcal{L}$ is a laminar family, for a set $R \in \mathcal{L}$ with $R \neq T$, R does not intersect T (either $R \subset T$, $T \subset R$ or $T \cap R = \emptyset$). So, whenever R intersects $S \cap T$ or $S \cup T$, R also intersects S. Also, T intersects S but not $S \cap T$ or $S \cup T$. Therefore, $intersect(S \cap T, \mathcal{L})$ and $intersect(S \cup T, \mathcal{L})$ are smaller than $intersect(S, \mathcal{L})$ ∎

This completes the proof of Lemma 8. ∎

4.3 Iterative 1-edge-finding Algorithm

In this section, we give an iterative procedure to find a minimum spanning tree from an optimal extreme point solution of the subtour LP. The algorithm is shown in Figure 1. To create the residual problem, the chosen edge e is contracted from G to identify its endpoints to result in the graph G/e.

Iterative 1-edge-finding MST Algorithm

1. Initialization $F \leftarrow \emptyset$.
2. While $V(G) \neq \emptyset$ do
   (a) Find an optimal extreme point solution x of the subtour LP and remove every edge e with $x_e = 0$ from G.
   (b) Find an edge $e = \{u, v\}$ such that $x_e = 1$ and update $F \leftarrow F \cup \{e\}$, $G \leftarrow G/e$.
3. Return F.

Figure 1: Iterative 1-edge-finding MST Algorithm

4.4 Correctness and Optimality

LEMMA 10. *For any extreme point solution x of the subtour LP with $x_e \geq 0$ for each edge e there exists an edge f such that $x_f = 1$.*

PROOF. We assign one token for each edge e in the support E, for a total of $|E|$ tokens. We will redistribute the tokens so that each set in $\mathcal{L}$ will receive one token and there are some extra tokens left. This implies that $|E| > |\mathcal{L}|$, giving us the contradiction to Lemma 8 and the Rank Lemma that together imply that $|E| = |\mathcal{L}|$.

For each edge e, we redistribute x_e to the smallest set containing both the endpoints. Now, we show that each set in $\mathcal{L}$ can collect at least one token, and demonstrate some extra leftover fractional edge tokens giving us the contradiction.

Let S be any set in $\mathcal{L}$ with children $R_1, \ldots, R_k$. We have

$$x(E(S)) = |S| - 1$$

and for each i,

$$x(E(R_i) = |R_i| - 1$$

Subtracting, we obtain

$$x(E(S)) - \sum_i x(E(R_i)) = |S| - \sum_i |R_i| + k - 1.$$

This implies that

$$x(A) = |S| - \sum_i |R_i| + k - 1$$

where $A = E(S) \setminus (\cup_i E(R_i))$. Now S obtains exactly x_e fractional token for each edge e in A. If $A = \emptyset$, then $\chi(E(S)) = \sum_i \chi(E(R_i))$ which contradicts the independence of these sets of

constraints in $\mathcal{L}$. Moreover, $x(A)$ is an integer and hence it is at least one, giving S the unit token it needs.

Since every edge is not integral, we have the extra fractional token values of $(1 - x_e)$ for every edge as unused tokens giving the contradiction. ∎

THEOREM 11. *The Iterative MST Algorithm returns a minimum spanning tree in polynomial time.*

PROOF. This is proved by induction on the number of iterations of the algorithm. Note that if the algorithm finds a 1-edge e, for any spanning tree T' of $G' = G/e$, we can construct a spanning tree $T = T' \cup \{e\}$ of G. Hence, the residual problem is to find a minimum spanning tree on G/e, and the same procedure is applied to solve the residual problem recursively.

Since $x_e = 1$, the restriction of x to $E(G')$, denoted by x_{res}, is a feasible solution to the subtour LP for G'. Inductively, the algorithm will return a spanning tree F' of G' of cost at most the optimal value of the subtour LP for G', and hence $c(F') \leq c \cdot x_{res}$. Therefore,

$$c(F) = c(F') + c_e \text{ and } c(F') \leq c \cdot x_{res}$$

which imply that

$$c(F) \leq c \cdot x_{res} + c_e = c \cdot x$$

as $x_e = 1$. Hence, the spanning tree returned by the algorithm is of cost no more than the cost of an optimal LP solution x, which is a lower bound on the cost of a minimum spanning tree. This shows that the algorithm returns a minimum spanning tree of the graph. ∎

5 Minimum Bounded-Degree Spanning Trees

We next turn to the study of the MINIMUM BOUNDED-DEGREE SPANNING TREE (MBDST) problem. In an instance of the MBDST problem we are given a graph $G = (V, E)$, edge cost given by $c : E \to \mathbb{R}$, a degree upper bound B_v for each $v \in V$ and the task is to find a spanning tree of minimum cost which satisfies the degree bounds. We prove the following theorem originally due to Singh and Lau.

THEOREM 12. *There exists a polynomial time algorithm which given an instance of the MBDST problem returns a spanning tree T such that $deg_T(v) \leq B_v + 1$ and cost of the tree T is smaller than the cost of any tree which satisfies the degree bounds.*

We prove Theorem 12 using the iterative relaxation technique.

5.1 Linear Programming Relaxation

We use the following standard linear programming relaxation for the MBDST problem, which we denote by $LP_{mbdst}(G, \mathcal{B}, W)$. In the following we assume that degree bounds are given for vertices only in a subset $W \subseteq V$. Let $\mathcal{B}$ denote the vector of all degree bounds B_v, one for each vertex $v \in W$.

$$\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} c_e x_e & & & & (5)\\
\text{subject to} \quad & x(E(V)) &=& |V| - 1 & & (6)\\
& x(E(S)) &\leq& |S| - 1 & \forall \emptyset \neq S \subset V & (7)\\
& x(\delta(v)) &\leq& B_v & \forall v \in W & (8)\\
& x_e &\geq& 0 & \forall e \in E & (9)
\end{aligned}$$

Separation over the inequalities in the above linear program can be carried out in polynomial time and follows from Theorem 5. An alternative is to write a compact reformulation of the above linear program which has polynomially many variables and constraints.

5.2 Characterization of Extreme Point Solutions

We first give a characterization of an extreme point solution of $LP_{mbdst}(G, \mathcal{B}, W)$. We remove all edges with $x_e = 0$ and focus only on the support of the extreme point solution and the tight constraints from (6)-(8). Let $\mathcal{F} = \{S \subseteq V : x(E(S)) = |S| - 1\}$ be the set of tight constraints from (6)-(7). From an application of Rank Lemma 3 and the characterization of extreme point solution to the spanning tree polyhedron (Lemma 8), we have the following characterization.

LEMMA 13. *Let x be any extreme point solution of $LP_{mbdst}(G, \mathcal{B}, W)$ with $x_e > 0$ for each edge $e \in E$. Then there exists a set $T \subseteq W$ and a laminar family $\mathcal{L}$ such that*

1. *$x(\delta(v)) = B_v$ for each $v \in T$ and $x(E(S)) = |S| - 1$ for each $S \in \mathcal{L}$.*
2. *The vectors $\{\chi(E(S)) : S \in \mathcal{L}\} \cup \{\chi(\delta(v)) : v \in T\}$ are linearly independent.*
3. *$|\mathcal{L}| + |T| = |E|$.*

5.3 An Additive One Approximation Algorithm

We now present an iterative algorithm which returns a tree of optimal cost and violates the degree bound within an additive error of one. This algorithm removes degree constraints one by one, and eventually reduces the problem to a minimum spanning tree problem. This can be thought of as a simple extension of the 1-edge-finding iterative MST algorithm presented earlier. The algorithm is given in Figure 2.

MBDST Algorithm

1. While $W \neq \emptyset$ do
   (a) Find an optimal extreme point solution x of $LP_{mbdst}(G, \mathcal{B}, W)$ and remove every edge e with $x_e = 0$ from G. Let the support of x be E.
   (b) **(Relaxation)** If there exists a vertex $v \in W$ with $deg_E(v) \leq B_v + 1$, then update $W \leftarrow W \setminus \{v\}$.
2. Return E.

Figure 2: Additive One MBDST Algorithm

5.4 Correctness and Performance Guarantee

In the next lemma we prove that in each iteration, the algorithm can find some vertex for which the degree constraint can be removed. Observe that once all the degree constraints are removed we obtain the linear program for the minimum spanning tree problem which we showed in Section 4 to be integral. Hence, the algorithm returns a tree. Moreover, at each step we only relax the linear program. Hence, the cost of the final solution is at most the cost of the initial linear programming solution. Thus the tree returned by the algorithm has optimal cost. A simple inductive argument also shows that the degree bound is violated by at most an additive one. The degree bound is violated only when we remove the degree constraint and then $deg_E(v) \leq B_v + 1$. Thus, in the worst case, if we include all the edges incident at v in T, the degree bound of v is violated by at most an additive one.

It remains to show that the iterative relaxation algorithm finds a degree constraint to remove at each step. From Lemma 13 we have that there exists a laminar family $\mathcal{L} \subseteq \mathcal{F}$ and $T \subseteq W$ such that $|\mathcal{L}| + |T| = |E|$ and constraints for sets in $\mathcal{L}$ are linearly independent. Observe that if $T = \emptyset$ then only the spanning tree inequalities define the solution x. Hence, x must be integral. In the other case, we show that there must be a vertex in W whose degree constraint can be removed.

LEMMA 14. *Let x be an extreme point solution to $LP_{mbdst}(G, \mathcal{B}, W)$ such that $x_e > 0$. Let $\mathcal{L}$ and $T \subseteq W$ correspond to the tight set constraints and tight degree constraints defining x as given by Lemma 13. If $T \neq \emptyset$ then there exists some vertex $v \in W$ with $deg_E(v) \leq B_v + 1$.*

PROOF. We use the fractional token argument as in the integrality proof of the 1-edge-finding iterative MST algorithm we presented earlier.

Suppose for the sake of contradiction, we have $T \neq \emptyset$ and $deg_E(v) \geq B_v + 2$ for each $v \in W$. We now show a contradiction by a token argument. We give one token for each edge in E. We then redistribute the token such that each vertex in T and each set in $\mathcal{L}$ gets one token and we still have extra tokens left. This will contradict $|E| = |T| + |\mathcal{L}|$. The token redistribution is as follows. Each edge $e \in E$ gives as before x_e tokens to the smallest set in $\mathcal{L}$ containing both endpoints of e, and $(1 - x_e)/2$ to each of its endpoints for the degree constraints.

We have already argued earlier that the x_e assignment suffices to obtain one token per member in the laminar family (see the proof of Lemma 10).

Thus it suffices to show that each vertex with a tight degree constraint gets one token. Let $v \in T$ be such a vertex. Then v receives $(1 - x_e)/2$ tokens for each edge incident at v for a total of

$$\sum_{e \in \delta(v)} \frac{1 - x_e}{2} = \frac{deg_E(v) - B_v}{2} \geq 1,$$

where the first equality holds since $\sum_{e \in \delta(v)} x_e = B_v$ and the inequality holds since $deg_E(v) \geq B_v + 2$ by Step 1b of the algorithm.

To finish the proof, we argue that there is some extra token left for contradiction. If $V \notin \mathcal{L}$ then there exists an edge e which is not contained in any set of $\mathcal{L}$ and the x_e token for that edge gives us the contradiction. Similarly, if there is a vertex $v \in W \setminus T$ then v also collects one token which it does not need and we get the desired contradiction. Moreover, if

there is a vertex $v \in V \setminus T$ then each edge e incident at v must have $x_e = 1$ else $(1 - x_e)/2 > 0$ tokens are extra. Note that $e \in span(\mathcal{L})$ for each e with $x_e = 1$, since e is a tight set of size two. We have

$$2\chi(E(V)) = \sum_{v \in V} \chi(\delta(v)) = \sum_{v \in T} \chi(\delta(v)) + \sum_{v \in V-T} \chi(\delta(v)) = \sum_{v \in T} \chi(\delta(v)) + \sum_{v \in V-T} \sum_{e \in \delta(v)} \chi(e).$$

We have argued that $V \in \mathcal{L}$ and $e \in span(\mathcal{L})$ for each edge $e \in \delta(v)$ for $v \in V - T$. Since $T \neq \emptyset$, this implies the linear independence of the tight constraints in T and those in $\mathcal{L}$, giving us the contradiction. ∎

5.5 Historical Notes

Edmonds [4] gave the integral linear programming relaxation for minimum spanning tree problem that we presented. There is a long line of work of successively improving the performance guarantees for the degree-bounded minimum-cost spanning tree problem. The algorithm with additive guarantee of one for the unweighted case was first given by Fürer and Raghavachari [5]. The additive algorithm with violation 2 (with both upper and lower degree bounds) was presented by Goemans [6]. The algorithm with additive violation of 1 was first presented by Singh and Lau [14], also for the case with upper and lower bounds on the degree. The fractional token proof which we used for the additive one proof was first presented by Bansal et al. [1].

6 Survivable Network Design Problem

The survivable network design problem generalizes the minimum Steiner tree problem, the minimum Steiner forest problem, and the minimum k-edge-connected subgraph problem, etc. Hence the result in this section also applies to these problems.

6.1 Linear Programming Relaxation

To formulate the problem as a linear program, we represent the connectivity requirements by a *skew supermodular* function. A function $f : 2^V \rightarrow \mathbb{Z}$ is called skew supermodular if at least one of the two following conditions hold for any two subsets $S, T \subseteq V$.

$$\begin{aligned} f(S) + f(T) &\leq f(S \cup T) + f(S \cap T) \\ f(S) + f(T) &\leq f(S \backslash T) + f(T \backslash S) \end{aligned}$$

It can be verified (with some simple case analysis) that the function f defined by $f(S) = max_{u \in S, v \notin S} r_{uv}$ for each subset $S \subseteq V$ is a skew supermodular function. Hence, one can write the following linear programming relaxation for the survivable network design problem, denoted by LP_{sndp}.

$$\begin{array}{lll} \text{minimize} & \sum_{e \in E} c_e x_e & \\ \text{subject to} & x(\delta(S)) \geq f(S) & \forall S \subseteq V \\ & 0 \leq x_e \leq 1 & \forall e \in E \end{array}$$

This linear program for the case of minimum Steiner networks can be solved in polynomial time by using a minimum cut algorithm as a separation oracle. Designing a separation oracle for more general skew submodular functions as right hand sides needs more work - details can be found in the original paper of Jain [9].

6.2 Characterization of Extreme Point Solutions

For a subset $S \subseteq V$, the corresponding constraint $x(\delta(S)) \geq f(S)$ defines a vector in $\mathbb{R}^{|E|}$: the vector has an 1 corresponding to each edge $e \in \delta(S)$, and a 0 otherwise. We call this vector the characteristic vector of $\delta(S)$, and denote it by $\chi(\delta(S))$. Recall that two sets X, Y are intersecting if $X \cap Y$, $X - Y$ and $Y - X$ are nonempty, and that a family of sets is laminar if no two sets are intersecting. It is not hard to verify the two inequalities below using the submodularity of the cut function.

$$x(\delta(X)) + x(\delta(Y)) \geq x(\delta(X \cap Y)) + x(\delta(X \cup Y)) \text{ and}$$

$$x(\delta(X)) + x(\delta(Y)) \geq x(\delta(X - Y)) + x(\delta(Y - X)).$$

For any two subsets X and Y, when f is skew supermodular, it follows from standard uncrossing arguments, as in the case of spanning trees, that an extreme point solution to LP_{sndp} is characterized by a laminar family of tight constraints. The Lemma below then follows from these uncrossing arguments and the Rank Lemma (Lemma 3).

LEMMA 15. *Let the requirement function f of LP_{sndp} be skew supermodular, and let x be an extreme point solution to LP_{sndp} with $0 < x_e < 1$ for every edge $e \in E$. Then, there exists a laminar family $\mathcal{L}$ such that:*

1. *$x(\delta(S)) = f(S)$ for each $S \in \mathcal{L}$.*
2. *The vectors $\chi(\delta(S))$ for $S \in \mathcal{L}$ are linearly independent.*
3. *$|E| = |\mathcal{L}|$.*

6.3 Iterative Algorithm

Jain's iterative rounding algorithm is in Figure 3.

Iterative Algorithm for Minimum Steiner Network

1. Initialization $F \leftarrow \emptyset$, $f' \leftarrow f$;
2. While $f' \neq \emptyset$ do
   (a) Find an optimal extreme point solution x to LP_{sndp} with cut requirement f' and remove every edge e with $x_e = 0$.
   (b) If there exists an edge e with $x_e \geq 1/2$, then add e to F.
   (c) For every $S \subseteq V$: update $f'(S) \leftarrow f(S) - |\delta_F(S)|$.
3. Return $H = (V, F)$.

Figure 3: Minimum Steiner Network Algorithm

6.4 Correctness and Performance Guarantee

Jain proved an important theorem about the extreme point solutions of LP_{sndp}.

THEOREM 16. *[Jain] Suppose f is an integral skew submodular function and x is an extreme point solution to LP_{sndp}. Then there exists an edge $e \in E$ with $x_e \geq \frac{1}{2}$.*

Assuming Theorem 16, then the iterative algorithm will terminate successfully, and it can be shown by a straightforward inductive argument that the returned solution is a 2-approximate solution.

THEOREM 17. *Algorithm 3 is a 2-approximation algorithm for the* SURVIVABLE NETWORK DESIGN *problem.*

PROOF. The proof is by induction on the number of iterations executed by the algorithm. For the base case that requires only one iteration, the theorem follows since it rounds up a single edge e with $x_e \geq \frac{1}{2}$. For the induction step, let e' be the edge with $x_{e'} \geq \frac{1}{2}$ in the current iteration, which is guaranteed to exist by Theorem 16. Let f' be the residual requirement function after this iteration and let H' be the set of edges picked in subsequent iterations for satisfying f'. The key observation is that the current solution x restricted to $E - e'$ is a feasible solution for satisfying f', and thus by the induction hypothesis, the cost of H' is at most $2\sum_{e \in E-e'} c_e x_e$. Consider $H := H' \cup e'$ which satisfies f (by the definition of f'). The cost of H is:

$$cost(H) = cost(H') + c_{e'} \leq 2 \sum_{e \in E-e'} c_e x_e + c_{e'} \leq 2 \sum_{e \in E} c_e x_e,$$

where the last inequality follows because $x_{e'} \geq \frac{1}{2}$. This implies that the cost of H is at most twice the cost of an optimal fractional solution, which is a lower bound of the optimal cost, and thus the theorem follows. ∎

We now give a simple proof of Jain's theorem above using the fractional token idea from the previous sections. This proof is due to Nagarajan et al. [12].

PROOF. We first prove that $x_e \geq \frac{1}{2}$ for some edge $e \in E$ in any extreme point solution x to LP_{SNDP}. Suppose that $0 < x_e < \frac{1}{2}$ for each $e \in E$. Then we will show that $|E| > |\mathcal{L}|$, contradicting Lemma 15. The proof is by a fractional token counting argument. We give one token to each edge in E, and then we will reassign the tokens such that we can collect one token for each member in $\mathcal{L}$ and still have extra tokens left, giving us the contradiction that $|E| > |\mathcal{L}|$. Each edge $e = uv$ is given one token which is reassigned as follows.

1. **(Rule 1)** Let $S \in \mathcal{L}$ be the smallest set containing u and $R \in \mathcal{L}$ be the smallest set containing v. Then e gives x_e tokens each to S and R.
2. **(Rule 2)** Let T be the smallest set containing both u and v. Then e gives $1 - 2x_e$ tokens to T.

We now show that each set S in $\mathcal{L}$ receives at least one token. Let S be any set with children $R_1, \ldots, R_k$ where $k \geq 0$ (if S does not have any children then $k = 0$). We have the following equalities.

$$\begin{aligned} x(\delta(S)) &= f(S) \\ x(\delta(R_i)) &= f(R_i) \;\; \forall\, 1 \leq i \leq k \end{aligned}$$

Subtracting we obtain,

$$x(\delta(S)) - \sum_i x(\delta(R_i)) = f(S) - \sum_{i=1}^{k} f(R_i). \tag{10}$$

We divide the edges involved into three types, where

$$\begin{aligned} A &= \{e : |e \cap (\cup_i R_i)| = 0, |e \cap S| = 1\} \\ B &= \{e : |e \cap (\cup_i R_i)| = 1, |e \cap S| = 2\} \\ C &= \{e : |e \cap (\cup_i R_i)| = 2, |e \cap S| = 2\}. \end{aligned}$$

Then (10) can be rewritten as:

$$x(A) - x(B) - 2x(C) = f(S) - \sum_{i=1}^{k} f(R_i). \tag{11}$$

Observe that $A \cup B \cup C \neq \emptyset$; otherwise the characteristic vectors $\chi(\delta(S)), \chi(\delta(R_1)), \ldots, \chi(\delta(R_k))$ are linearly dependent. For each edge $e \in A$, S receives x_e tokens from e by Rule 1. For each edge $e \in B$, S receives $1 - x_e$ tokens from e by Rule 1 and Rule 2. For each edge $e \in C$, S receives $1 - 2x_e$ tokens from e by Rule 2. Hence, the total tokens received by S are exactly,

$$\begin{aligned} 0 &< \sum_{e \in A} x_e + \sum_{e \in B} (1 - x_e) + \sum_{e \in C} (1 - 2x_e) \\ &= x(A) + |B| - x(B) + |C| - 2x(C) \\ &= |B| + |C| + f(S) - \sum_{i=1}^{k} f(R_i), \end{aligned}$$

where the last equality follows from (11). Since f is integral, the right hand side is at least one, and thus every set $S \in \mathcal{L}$ receives at least one token in the reassignment.

It remains to show that there are some unassigned tokens, which would imply the contradiction that $|E| > |\mathcal{L}|$. Let R be any maximal set in $\mathcal{L}$. Consider any edge $e \in \delta(R)$. The fraction of the token by Rule 2 for edge e is unassigned, as there is no set with $|T \cap e| = 2$, and gives us the desired contradiction. ∎

References

[1] N. Bansal, R. Khandekar and V. Nagarajan, *Additive guarantees for degree bounded directed network design*, in Proceedings of the Fourtieth Annual ACM Symposium on Theory of Computing (STOC), 2008.

[2] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, John Wiley and Sons, New York (1998).

[3] V. Chvatal, *Linear Programming*, Freeman, 1983.

[4] J. Edmonds, *Matroids and the Greedy Algorithm.* Mathematical Programming, 1:125–136, 1971.

[5] M. Fürer and B. Raghavachari, *Approximating the minimum-degree Steiner tree to within one of optimal*, J. of Algorithms 17(3):409-423, 1994.
[6] M.X. Goemans, *Minimum Bounded-Degree Spanning Trees,* Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, 273–282.
[7] M. Grotschel, L. Lovasz, A. Schrijver *The Ellipsoid Method and its Consequences in Combinatorial Optimization,* Combinatorica 1 (1981), 169-197.
[8] F.S. Hillier and G.J. Lieberman, *Introduction to Operations Research* (6th Ed.), Mcgraw-Hill, 1995.
[9] K. Jain, *A factor* 2 *approximation algorithm for the generalized Steiner network problem*, Combinatorica, **21**, pp.39-60, 2001. Preliminary version in *Proc. 39th IEEE FOCS*, 1998.
[10] L.C. Lau, S. Naor, M. Salavatipour and M. Singh, *Survivable network design with degree or order constraints*, Proceedings of the 40th ACM Symposium on Theory of Computing, 651-660, 2007.
[11] L.C. Lau, R. Ravi and M. Singh, *Iterative Methods in Combinatorial Optimization*, In Preparation, 2009.
[12] V. Nagarajan, R. Ravi and M. Singh, *Unified Analysis of LP Extreme Points for Steiner Network and Traveling Salesman*, submitted (2009).
[13] A. Schrijver, *Combinatorial Optimization - Polyhedra and Efficiency*, Springer-Verlag, New York, 2005.
[14] Mohit Singh and Lap Chi Lau, *Approximating Minimum Bounded Degree Spanning Tress to within One of Optimal*, Proceedings of 39th ACM Symposium on Theory of Computing, 661-670, 2007.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.

LIPIcs Leibniz International Proceedings in Informatics

Randomness extractors – applications and constructions

Avi Wigderson
Institute for Advanced Study
Princeton, NJ
avi@ias.edu

ABSTRACT. Randomness extractors are efficient algorithms which convert weak random sources into nearly perfect ones. While such purification of randomness was the original motivation for constructing extractors, these constructions turn out to have strong pseudorandom properties which found applications in diverse areas of computer science and combinatorics. We will highlight some of the applications, as well as recent constructions achieving near-optimal extraction.

Introduction

The quest to purify the randomness in "weak" random sources (of biased and correlated bits) was initiated in the papers of Blum [1] and Santha and Vazirani [16].

The amount of randomness in a distribution for this purpose is captured by the notion of min-entropy, first suggested in this context by Chor and Goldreich [2] and Zuckerman [22]. We say that a random variable has min entropy $\geq k$ if its probability of it hitting any specific value is at most 2^{-k}.

Purifying the randomness from such distributions is captured by the notion of extractors, first defined in the seminal paper of Nisan and Zuckerman [13]. A (k, ϵ)-extractor is a function $E : \{0,1\}^n \times \{0,1\}^d \mapsto \{0,1\}^m$ such that for every random variable X with min entropy k, the distribution of $E(X, U_d)$ has statistical distance $\leq \epsilon$ from the uniform distribution, where U_d denotes a random variable independent of X and uniform on $\{0,1\}^d$. The input U_d is called a *seed* and is thought of as being much shorter (in bits) than X. It is not hard to see that a seed is essential for an extractor to work in this general setting. Such extractors are often called "seeded extractors", to distinguish them from "seedless extractors" (such determinsitic seedless extractors can work only when additional structure is imposed on the source, and will not be discussed here). An excellent survey of seeded extractors is [15].

An extractor has three important parameters. The first is the seed length d, which we wish to minimize. The second is the output length m, which we want to maximize (we want to have $m \approx k$). The third parameter we wish to minimize is the 'error' ϵ – the statistical distance of the output of the extractor from the uniform distribution. It can be shown, using the probabilistic method, that a random function gives an extractor which is optimal in all three parameters, which allows (roughly) $m = k$ and $d = \log(n/\epsilon^2)$. A random function, however, is not satisfactory since in applications we need to be able to compute the extractor efficiently. An extractor which is efficiently computable is called *explicit*. Below we list the progress on explicit constructions, as well as the numerous applications of such explicit extractors.

© Wigderson; licensed under Creative Commons License-NC-ND.
Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 471–473
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2340

Constructions Since the 80's there many works devised a variety of techniques to construct explicit extractors of better and better parameters (see [15] for a complete list of references). The first paper to give an explicit extractor which was optimal (up to constant factors) both in seed length and in entropy output was the work of Lu, Reingold, Vadhan and Wigderson [12]. The first to achive this for the error parameter as well were Guruswami, Umans and Vadhan [8], in an elegant construction based on list-decodable Parvaresh-Vardy codes [14], which is also much simpler than [12]. An alternative construction, with the same parameters based on the resolution of the Kakeya conjecture in finite fields [4], was give by Dvir and Wigderson [5]. In all of these the output m was a constant fraction (arbitrarily close to 1) of k. This year Dvir, Kopparty, Saraf and Sudan [6] managed to extract $m = (1 - o(1))k$ for the first time, as byproduct of tight analysis of the Kakeya conjecture. Achieving $m = k$ and removing the large constant factor in the seed length remain challenging openquestions, of relevance to some of the applications.

Applications Extractors posses remarkable pseudorandom properties, which have found applications in a remarkably diverse areas. We list here only some of them, with sample references of each, noting that there are many others.

- Probabilistic algorithms with weak randomness [20, 22, 18]
- Derandomizing small-space computations [13, 10]
- List-decodable error-correcting codes [17]
- Expanders beating the eigenvalue bound (and the applications of these) [21]
- Lossless expanders (and the applications of these) [3]
- Sampling and Hashing [7, 9]
- Cryptography [19]
- Pseudorandom generators [18]
- Metric embeddings [11]

References

[1] Manuel Blum. Independent unbiased coin flips from a correlated biased source: a finite state Markov chain. *Proc. of the 25th FOCS*, 425–433, 1984.

[2] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, April 1988. Special issue on cryptography.

[3] Mike Capalbo, Omer Reingold, Salil Vadhan, and Avi Wigderson. Randomness conductors and constant degree expansion beyond the degree/2 barrier. In *Proceedings of the 34th Annual ACM STOC*, 659–668, 2002.

[4] Zeev Dvir. On the size of Kakeya sets in finite fields. *J. AMS*, 22, 1093–1097, 2009.

[5] Zeev Dvir, Avi Wigderson. Kakeya sets, new mergers and old extractors. *Proc. of the 49th FOCS*, 625–633, 2008.

[6] Zeev Dvir, Swastik Kopparty, Shubhangi Saraf and Madhu Sudan. Extensions to the method of multiplicity, with applications to Kakeya sets and mergers. proc. of FOCS '09, 2009, to appear.

[7] Oded Goldreich. A sample of samplers. *ECCC*, TR97-020, 1997.

[8] Venkat Guruswami, Chris Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. In *Proceedings of the 22nd Conference on Computational Complexity*, pages 96–108, 2007.

[9] Oded Goldreich and Avi Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. *Random Structures & Algorithms*, 11(4):315–343, 1997.

[10] Oded Goldreich and Avi Wigderson. Derandomization that is rarely wrong from short advice that is typically good. *Proc. of the 6th RANDOM conference*, 209–223, 2002.

[11] Pyotr Indik. Uncertainty principles, extractors and explicit embeddings of L2 into L1. *Proc. of the 39th STOC*, 2007.

[12] Chi-Jen Lu, Omer Reingold, Salil Vadhan, and Avi Wigderson. Extractors: Optimal up to constant factors. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing"*, 602–611, 2003.

[13] Noam Nisan and David Zuckerman. Randomness is Linear in Space. *JCSS*, 43–52, 1996.

[14] Farzad Parvaresh and Alexander Vardy. Correcting errors beyond the guruswami-sudan radius in polynomial time. In *Proceedings of the 46th FOCS*, pages 285–294, Washington, DC, USA, 2005. IEEE Computer Society.

[15] Ronen Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the EATCS*, 77:67–95, 2002.

[16] Miklos Santha and Umesh Vazirani. Generating quasi-random sequences from semi-random sources. *JCSS*, 48(4), 860–879, 2001

[17] Amnon Ta-Shma and David Zuckerman. Extractor codes. *Proc. of the 33rd STOC*, 193–199, 2001.

[18] Luca Trevisan. Extractors and pseudorandom generators. *JACM*, 33, 75–87, 1986.

[19] Salil Vadhan. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *Cryptology*, 17(1), 43–77, 2004.

[20] Umesh Vazirani and Vijay vazirani. Random polynomial time is equal to semi-random polynomial time. *Proc. 26th FOCS*, 417–428, 1985.

[21] Avi Wigderson and David Zuckerman. Expanders that beat the eigenvalue abound: explicit construction and applications. *Combinatorica*, 19(1), 125–138, 1999.

[22] David Zuckerman. Simulating BPP using a weak random source. *Algorithmica*, 16(4/5), 1996.

This work is licensed under the Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 License.