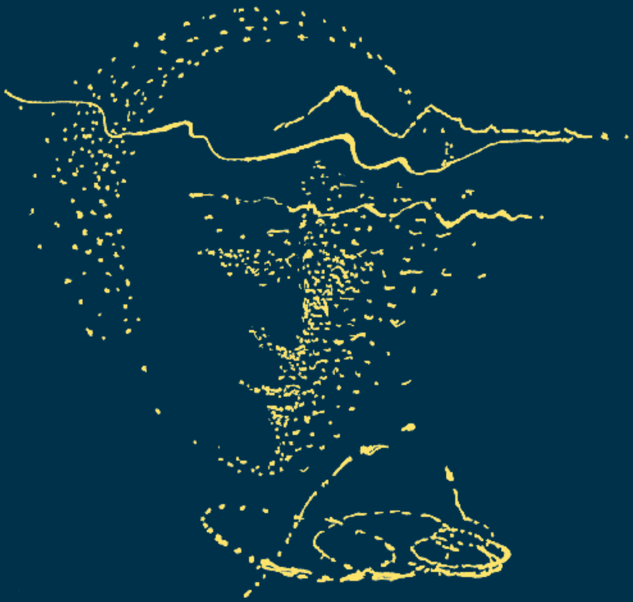
Festschrift

LNCS 11011

Hans-Joachim Böckenhauer
Dennis Komm
Walter Unger (Eds.)

Adventures Between Lower Bounds and Higher Altitudes

Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday



Springer

Hans-Joachim Böckenhauer · Dennis Komm
Walter Unger (Eds.)

Adventures Between Lower Bounds and Higher Altitudes

Essays Dedicated to Juraj Hromkovič
on the Occasion of His 60th Birthday

Springer

Editors
Hans-Joachim Böckenhauer
ETH Zürich
Zürich
Switzerland

Walter Unger
RWTH Aachen University
Aachen
Germany

Dennis Komm
ETH Zürich
Zürich
Switzerland

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-98354-7 ISBN 978-3-319-98355-4 (eBook)
https://doi.org/10.1007/978-3-319-98355-4

Library of Congress Control Number: 2018950465

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Cover illustration: The drawing on the front cover is the work of Ingrid Zámečníková, Slovakia. Used with permission.
Photograph on p. V: The photograph of the honoree was taken by Jan Lichtensteiger, Switzerland. Used with permission.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Juraj Hromkovič, 2017

Preface

This book is dedicated to Juraj Hromkovič on the occasion of his 60th birthday. Juraj was born on August 24, 1958, in Bratislava, Czechoslovakia. His first encounter with computer science took place at the Jur Hronec high school (Gymnázium Jura Hronca) in Bratislava in 1973, which he today refers to as a paradise for computer scientists. One of the reasons for that, and this seems like something to keep a note of when describing his further career, was a highly motivated teacher called Ondrej Demáček. From 1977 to 1982, Juraj studied computer science at Comenius University in Bratislava, where he received his first PhD (Dr. rer. nat.) in 1982, his second PhD (CSc) in 1986, and his habilitation in the area of Theoretical Cybernetics and Mathematical Informatics in 1989. From 1982 until 1985, Juraj worked as an assistant at the Department of Theoretical Cybernetics at Comenius University, where his early research focused on the theory of automata and Turing machines and communication complexity. Although his area of research broadened by magnitudes since that time, an echo of these first steps can still be heard in his ongoing research and teaching focus. Especially his results on two-way and multi-head deterministic and nondeterministic automata made an impact in this area.

In 1989, Juraj became associate professor at Comenius University, and in the same year he followed the invitation by Burkhard Monien to become a visiting professor at Paderborn University, where he stayed until 1994. During this period, his two daughters were born, and he obtained his third PhD in mathematics and physics in 1990. Also, during his Paderborn years, he added the areas of communication problems on graphs, parallel computation, and graph embedding to his research fields, with remarkable contributions to the area of information dissemination in networks in particular.

In 1994, Juraj became professor at Kiel University, where he stayed for three years. He continued his work on parallel computation, systolic computation, and communication problems. One of Juraj's most notable contributions from this time is the monograph on communication complexity.

In 1997, Juraj was appointed to the chair of Computer Science 1 "Algorithms and Complexity" at RWTH Aachen University, where he further broadened his research interests, now including the theory of approximation algorithms and algorithmics for hard problems. In particular, he established a new research area by introducing the stability of approximation algorithms. He also became the head of the examiners board, where he campaigned for fair examinations. His keen sense of fairness in dealing with the administrative problems is still known among students from Aachen. When he noticed the lack of algorithmic education at the schools of his daughters, or any school for that matter, he started his project "Informatik für Kids" (Computer Science for Kids). Furthermore, his ambition to write textbooks reached a new level.

Being an expert in the fields of both algorithms and education, it came to no surprise that ETH Zurich asked him to take the chair for "Information Technology and

Education." Since 2004, Juraj has been working at ETH, bringing computer science to schools (in Switzerland and far beyond), writing plenty of textbooks, and all the while continuing research in his virtually unbounded field of expertise. On top of this, he stayed active in several theoretical areas of computer science, and made important contributions to reoptimization problems and to online algorithms with advice.

The very broad list of research areas hides the fact that there is one important underlying interest in all of Juraj's research: He is at all times investigating the information needed to solve hard problems, measured both in a qualitative and quantitative way. For example: How much information has to be exchanged between automata that have to collaborate in solving one global question? How much information is hidden inside the unknown input of online problems? What is the impact of additional information, e.g., an optimal solution of a neighboring instance or the distance to an easy instance, on the hardness of optimization problems? What is the gain of information by using nondeterministic steps instead of deterministic algorithms to solve hard problems? Or, in Juraj's words: "We are investigating the information content of problems." Through his textbooks, Juraj passes on his deep insight into the structure of problems and their solutions, thus becoming a source of inspiration and motivation for his readers.

Looking back, Juraj authored and co-authored over 200 publications, including 20 monographs and textbooks (some of them in several languages: English, German, Russian, Japanese, Slovak, Spanish). He supervised 20 PhD dissertations and two habilitations, is managing editor of the Journal of Interconnection Networks and editor of several high-level international journals, co-chair and organizer of prestigious international symposia, and member of many program committees and several steering committees of international conferences. In addition, more than 20 invited talks at international conferences and over 100 invited talks in different institutions illustrate his international reputation. Juraj's work is also honored by the following selected list of awards and memberships: First Class Pribina Cross State Award of the Slovak Republic, State Award of the Slovak Republic "Goodwill Envoy," member of Academia Europaea, member of Slovak Academic Society, and First Award of the Slovak Literature Foundation for scientific literature.

The aforementioned list of achievements covers only the tip of a mountain of work Juraj did and is doing. When working closely with Juraj, it becomes more and more mysterious how he manages to do all of this at once. A somewhat widely accepted yet not formally proven theory states that Juraj is in fact not a single person, but twins; twins that both sleep two hours a week at most. But jokes aside, it should be mentioned that, even though his workload is so extremely high, no one has ever witnessed him being in a bad mood, not spreading his underlying infectious optimism – and this while writing a number of textbooks per year that would constitute a very successful research career were they replaced (instead of accompanied) by research papers.

Still, Juraj's reputation and standing among the computer science community only became fully visible to us when we received the overwhelming feedback from his colleagues regarding our question as to whether they would like to contribute to this book. We never expected to receive positive answers by almost everyone we asked; and it was taken very seriously to contribute something – and that was our major condition – that Juraj would find truly interesting.

One unifying fact about Juraj is not apparent from the above list of facts: his ambition for mountain trailing, which he started already in his early days in Bratislava. This urge to reach any mountain top reflects his ambition to solve hard computational problems or to tackle any other problem. When approaching such hard problems, one has to focus on them – as Juraj does – with an open mind and without disturbance. To this end, Juraj organizes his mountain workshops, where – often successful – attempts to solve hard problems are carried out and combined – often successfully – with mountain hiking and climbing; Juraj's enthusiastic affection inspires others – always successfully – to follow him and his ambition.

As we have just outlined in a very vague and highly incomplete way, since the beginning of his career, Juraj has been working on a large number of different areas of computer science, being an active member of many research communities. Therefore, this book is subdivided into three parts, namely, "Automata and Formal Languages," "Algorithmics," and "Computer Science Education." We could have easily added further parts, including communication complexity, network algorithms, alpine hiking, and many more.

Dear Juraj

This book is for you; a very small "thank you," compared to your caring, friendship, smiles, support, ideas, research discussions, and simply the privilege of knowing you. We wish you all the best from our hearts — and many more theoretical and real mountains to climb.

Acknowledgement

We would like to thank Peter Widmayer for supporting us from the start, and Alfred Hofmann from Springer for his support in realizing this project. We are also very grateful to Ingrid Zámečníková for the inspiring illustration on this book's cover and Jan Lichtensteiger, who took the portrait photo of Juraj displayed at the beginning of this book. Moreover, we are very thankful to Elisabet Burjons, Fabian Frei, Christos Kapoutsis, Tobias Kohn, Rastislav Královič, Tobias Mömke, Britta Peis, Giovanni Serafini, and Stefan Wolf, who all helped us enormously in reviewing the papers. Last but not least, we would like to thank once again all the people who have contributed to this book.

June 2018

Hans-Joachim Böckenhauer
Dennis Komm
Walter Unger

Prologue

The Three Parts of Juraj

Harry Buhrman

Algorithms and Complexity Group, University of Amsterdam, Netherlands
harry.buhrman@cwi.nl

I met Juraj in Aachen where he invited me to give a seminar talk in the late 1990's on quantum communication complexity, a topic that I had just ventured into and had written a survey paper on. I did not know that Juraj was intimately familiar with the topic of classical communication complexity. Nor did I know he had written a thick book on the topic. Something that happened more often in my encounters with Juraj: he had already written a textbook on topics we discussed. Polite and modest, Juraj's subtle and friendly gesture was to congratulate me on my survey and send me a copy of his book. My visit to Juraj's group in Aachen was very inspiring, his genuine interest in my research and our mutual interest in theoretical computer science made me instantly comfortable around him.

During that visit Juraj also sparked my interest in the educational side of science and of computer science in particular. Also in this field he had already made deep and important contributions, as well as written several books and texts. This inspired me to try and teach some of his material in the elementary school where my kids were going to in Amsterdam. I was able to teach twice a semester to kids in the ages of 7–9 years old. What a wonderful and rewarding experience this was. Unfortunately I could not find enough time to keep this up. But I noticed that computer science eduction in the Netherlands is far behind that of Germany and Switzerland.

After Juraj moved to the ETH in Zurich I visited him several times and we discovered another joint interest: the mountains! Also here Juraj was doing amazing iron man like competitions that I did not even know were possible. Running for 100 km or more on one day in the mountains. Nevertheless, Juraj always organised a small and energetic hike in one of the afternoons. For me these trips were always very special with a mix of science, eduction and hiking. I hope to have many more in the future.

I know now it is time to continue in Juraj's spirit and set up a program for teaching quantum computing and quantum physics and quantum computer science in elementary schools and high schools. Quantum mechanics appears to be very counterintuitive to most of us. This may very well be because we are not used to quantum mechanical effects, like, e.g., superposition and entanglement, because we never encounter them in our everyday life, nor in our early eduction. It might be that teaching quantum mechanics to very young kids takes away the counterintuitive aspects and gives them an intuitive understanding that people who get acquainted with quantum physics at a later age will not be able to develop. I hope to get as much advice as I can from Juraj on this!

Informatics is Not Just for Engineers

Hans Hinterberger

Department of Computer Science, ETH Zurich, Switzerland
hinterbe@inf.ethz.ch

This is a good occasion to sum up some impressions that accumulated over time.

First, many academics at ETH appreciate it that during the past years Juraj as the chair of Information Technology and Education at the Department of Computer Science invested a lot of time and a great amount of energy into an effective teaching of informatics. And the members of the department were thankful that the messages were carried out into the public at large.

Most important, however, is the realization that groundbreaking actions were necessary and that this fact has been communicated. Actions that lead to proper instruction in informatics at all schooling levels up to and including high school ("Gymnasium"). This is extremely important as instructors at ETH still have to spend way too much time teaching students elementary concepts of informatics. Contrary to many "reformers," Juraj's efforts did not stop at talking about the problems, but he initiated work on them and gave life to the ABZ with all its associated activities. In doing this, a path was laid out that lead to many successes, but not always without hindrance.

Second, it made life for many people much easier when Juraj, at the time of Walter Gander's retirement, accepted one of the orphaned groups into his chair of Information Technology and Education. Furthermore, it was really great that through this the support my PhD students and postdocs were able to continue after their boss also retired, which allowed them to do further work in their field of specialization.

Third, Juraj's efforts to ensure that retired members of the department can continue to use office space and infrastructure are greatly appreciated. It allows them to stay in contact and makes it possible to provide support should the need arise.

Dear Juraj: Congratulations to your sixtieth birthday. May the years to come bring you all the best and good health. Many happy returns. It has always been a pleasure to work and stay in contact with you. I truly hope that this can and will continue in this spirit.

Reflections of Juraj's First Graduate Student

Dana Pardubská

Comenius University, Bratislava, Slovakia
pardubska@dcs.fmph.uniba.sk

We met each other when we both were students – I was in the second, Juraj in the fifth grade. When I was finishing my Master's degree, I joined his selective course. I cannot remember exactly the name, something like "Selected Topics in Informatics"; this was the proper lecture at the right time. Every week some area and/or result were introduced to us. He used to explain us the relevance of information and the related proof techniques. This lecture affected me the most during my studies. I was affected not only by the content, but also by his approach to the subject and to us, his students. I am thankful for helping me to discover the beauty of theoretical computer science and also for being my Diploma and Doctoral advisor.

I am not the only one who remembers with pleasure on our workshop in Štrba – cross country skiing during the day and research in the evening. He collected a group of mostly young people from different fields and let them talk about their work. Probably only later on we appreciated the importance of these meetings and evening discussions which brought us knowledge and motivation from different areas.

These are memories of times when Juraj was still in Slovakia. Then he left into the world. He was at the University of Paderborn when I was finishing my PhD thesis. I had a chance to see him interacting with his students and colleagues there. I have also spent time with him and his family and have perceived also this side of his personality. It was inspiring.

How did I perceive him? That time and now – large-minded, passionate, hard-working, always willing to help. A visionary with his own view, with the ability to recognize the important. Pacifying, encouraging. A hiker and runner with a smile on his face. A friend.

He has achieved a lot of success, has written a lot of articles, books. He is engaged in many activities concerning teaching of informatics, and many projects aimed at helping students of various age groups to understand the history and foundations of informatics. He was awarded many awards and guided many of his students and colleagues in their personal and professional growth. Nevertheless he remained to be the same Juraj, the same friend and a human with the capital "H".

Happy birthday!

The Uniqueness of Being Juraj

Sarah Springman

Rector, ETH Zurich
sarah.springman@sl.ethz.ch

Juraj is completely unique – the mould has been thrown away… he has "roots" and "wings." He is rooted in his passionate belief of the absolutely incontrovertible imperative to educate all young people to think logically and to be able to program in simple coding language. This is an essential contribution to levelling out access to, and driving forward, modern educational opportunities in Switzerland. His many courses attract children and students alike. They stimulate and engage people from all backgrounds and both genders – as well as the parents and the occasional Federal Councillor (Bundesrat)! His conviction about the "right things to do" is incomparable. He leads his close-knit team to that they soar with "wings": a talent that he exhibits too when he is scampering up to the top of any mountain and down again!

Happy Anniversary, Juraj, and "toi toi toi!" from Sarah Springman and team.

Reminiscences of Juraj as a Predecessor and a Successor

Wolfgang Thomas

Chair of Computer Science 7, RWTH Aachen University, Germany
thomas@cs.rwth-aachen.de

It is somewhat rare that two scientists share their time as colleagues in two different institutions. As it happened, Juraj and I can serve as an example – we were colleagues at two universities, in Kiel and in Aachen. I became acquainted with Juraj when I was chairing an appointment committee in Kiel to fill a new position in theoretical computer science. Well, it was not too difficult to see that he simply was the best candidate, and so he came to Kiel and we enjoyed a most fruitful time as colleagues working in complementary branches of theoretical computer science. That my high esteem for Juraj had been right was confirmed five years later when Juraj got a call to head a chair at the technical university of Aachen. Another two years later I went to Aachen myself, now succeeding rather than preceding him. So our careers were nicely entangled, and only when Juraj moved further to Zürich and towards the study of the educational aspects of computer science, I had to follow his work from a distance. It is wonderful to see that on the occasion of his 60th birthday the present Festschrift appears, documenting his amazing scope of scientific work – and together with my sincere congratulations I am sending Juraj all my best wishes for many years of continuation of these activities – in the same productive and insightful way as we know it from the past decades.

Hiking with Juraj

Erich Valkema

Kiel University, Germany
erich.valkema@web.de

My first contact with Juraj Hromkovič was established in the eighties, when I was asked to write a referee report on a paper of Juraj who at that time was living in Czechoslovakia behind the iron curtain.

In the nineties Juraj became a member of the teaching staff of our computer science department at Kiel University. Many students attended his lectures on different kinds of algorithms, on communication complexity, on parallel computing, and many other topics not only because of the interesting contents but also due to Juraj's teaching skills.

As his scientific and teaching merits will be appreciated much better by other colleagues around the world, I will focus this short address on my personal experiences with Juraj and his family when we stayed together at our department in Kiel.

During this time Juraj turned out to be not only a good colleague of high scientific standing but he became beyond that a really true friend of mine. We enjoyed ice-skating on lakes with both our families and hiking in the country-side around Kiel. So on his 38th birthday we took a walk of 38 kilometers and I also remember quite well two scientific meetings combined with mountain hiking in Austria with Juraj. I am quite sure, that he is still able to walk as many kilometers per day as his age tells him.

I am very pleased that our good relationship did not end when Juraj moved away from Kiel and is still flourishing and will hopefully do so for a long time to come.

For his 60th birthday my wife and I wish him and his family all the best.

Greetings from the Baltic Coast

Thomas Wilke

Kiel University, Germany
thomas.wilke@email.uni-kiel.de

I got to know Juraj in 1994, when he became a professor at Kiel University. At the time, I was a postdoc with Wolfgang Thomas. In 2004, I visited Juraj in Zürich as part of my sabbatical.

Juraj has great ideas and is a diligent worker. He knows very well how to motivate students and colleagues, simply by his very own enthusiasm for the subject matter involved. In fact, although I was working on questions in logic in computer science in 1994, Juraj got me interested in a fundamental question on regular expressions and finite-state automata. From the discussions with Juraj and Sebastian Seibert – at the time one of Wolfgang Thomas's PhD students – a joint paper resulted.

Juraj has a very sharp mind and strong opinions. This is what helped me often enough to sharpen my own position on a topic. I am interested in teaching cryptography; the book on this Juraj co-authored is what I consult first when I have new ideas and want to see how they relate to what others have to say.

Juraj is very supportive. This is what I experienced at several stages of my own academic life. On one occasion, when I was trying to set up a network for computer science teachers in Schleswig-Holstein and foster computer science as a school subject, Juraj agreed to come to Kiel, give a public talk, and engage in discussions with teachers and schools, bringing in all his experience and excellence in this field.

A sixtieth birthday is a good occasion and a Festschrift is the perfect location to say that it is good that Juraj has been around and is still around. I use this occasion to wish Juraj a good life.

Contents

Automata and Formal Languages

Determinism and Nondeterminism in Finite Automata with Advice

Pavol Ďuriš[1], Rafael Korbaš[1], Rastislav Královič[1](✉), and Richard Královič[2]

[1] Faculty of Mathematics, Physics, and Informatics, Comenius University, Bratislava, Slovakia
{duris,kralovic}@dcs.fmph.uniba.sk
[2] Google, Inc., Zürich, Switzerland
ri.kralovic@gmail.com

Abstract. We consider the model of finite automata with advice introduced by Küçük et al. We show that there are languages, in particular the language of palindromes, that cannot be recognized by DFA regardless of the size of advice. Also, we show that a DFA cannot utilize more than exponential advice. We initiate the study of NFA with advice: we show that, unlike the DFA, NFA can recognize all languages with advice of exponential size. On the other side of the spectrum, with constant advice, DFA are as powerful as NFA. We show that for any growing function f, there are languages that can be recognized by NFA with advice $f(n)$, but cannot be recognized by DFA regardless of advice size. We also ask what languages can be recognized with polynomial advice. For NFA we show that this class is not closed under complementation, and that it contains all bounded languages. Bounded languages over one-letter words can even be recognized by DFA with polynomial advice. We also give examples of languages that cannot be recognized by NFA with polynomial advice. Finally, we show that increasing advice helps for NFA, and for any advice of size $f(n) \leq n$ we show that there is a language that can be recognized by a DFA with advice $f(n)$, but cannot be recognized by an NFA with advice $o(f(n))$.

1 Introduction

Uniformity, i.e., the fact that a single, finitely-described, device is used to process instances of arbitrary size, is a central property shared by all computation models deemed feasible. Understanding the role this restriction plays in the inherent limitations of feasible computation models is one of the fundamental directions in theoretical computer science. Models that are naturally defined as non-uniform (like circuits), usually come with uniformity as an add-on requirement, and the uniform and non-uniform versions can be compared. Turing machines are naturally uniform, and their non-uniform version was introduced in the seminal paper of Karp and Lipton [5] in the form of advice machines. The machine,

Research partially supported by grant VEGA 2/0165/16.

together with the input word x of length n, is also provided an advice string $\alpha(n)$, that does not need to be computable, but remains the same for all words of length n. It is well known how fruitful the line of research investigating this notion has been in understanding the fundamentals of computation. However, since the main questions concerning Turing machines still remain unsolved, it is natural to focus on their restricted versions for gaining better insight. While there had been previous attempts to study non-uniform versions of automata (e.g., Ibarra and Ravikumar [4] considered 2-way automata with growing sets of states), Damm and Holzer [1] proposed the first model of finite automata with advice along the lines of [5]: a one-tape finite automaton for which each input word is prefixed by the advice string (see Definition 1). Since the advice is on the same tape as the input, the automaton can use only constant advice. The class of languages recognized by these devices is denoted REG/k where k is the length of advice string, extending the notation REG for the class regular languages. Even $\mathrm{REG}/1$ can recognize some non-recursive languages (e.g., unary languages), and there is a strict hierarchy $\mathrm{REG}/(k-1) \subsetneq \mathrm{REG}/k$. In order to overcome the limitation to constant amount of advice, Tadaki et al. [7] consider advice of length n written on separate track (see Definition 2). We denote the class of languages recognized by these automata $\widehat{\mathrm{REG}}/n$ to distinguish them from the previous model. In [7] it is shown that $\widehat{\mathrm{REG}}/n = \mathrm{1DLIN}/O(n)$, i.e., the class of languages recognized by a linear-time 1-tape deterministic Turing machine with advice written on a separate track. Hence, the power of the Turing machine to write information to the tape does not help in this case. The advice written on a separate track overcomes the shortcomings of the model from [1], but it does not allow to study other than linear size of advice. Freivalds [3], with the motivation to study the amount of nonconstructiveness in non-constructive arguments, proposes a model of finite automata that use separate tapes to store the advice. In his model (see Definition 3), the advice may be split into several tapes. However, the advice string of length m must be valid for all words of lengths up to m. He considers deterministic automata with two-way input and advice tapes. We denote the class of these languages $\mathscr{F}(\mathrm{DFA})/f(n)$. Freivalds shows that $\mathscr{F}(\mathrm{DFA})/o(\log n) = \mathrm{REG}$, but there are some non-recursive languages that can be recognized with polylogarithmic advice. On the other hand $\mathscr{F}(\mathrm{DFA})/(n2^n)$ contains all languages, and there are languages that cannot be recognized with advice $o(2^n)$. We adopt the model by Küçük et al. [6] (see Definition 4) that combines the models of Freivalds, and Tadaki et al. Denoted by $\mathscr{L}(\mathrm{DFA})/f(n)$, the advice of length $f(n)$ for a one-way deterministic FA is written on separate tapes (in our results we consider only a single advice tape), and the advice is specific for the inputs of given length. Küçük et al. showed that $\mathscr{L}(\mathrm{DFA})/\mathit{exp}(\texttt{2w-input})$ contains all languages, a hierarchy $\mathscr{L}(\mathrm{DFA})/(n^k) \subsetneq \mathscr{L}(\mathrm{DFA})/(n^{k+1})$, and a separation $\mathscr{L}(\mathrm{DFA})/\mathit{poly} \subsetneq \mathscr{L}(\mathrm{DFA})/\mathit{poly}(\texttt{2w-input})$. They also showed that the language of palindromes, $L_{\mathsf{PAL}} \notin \mathscr{L}(\mathrm{DFA})/\mathit{poly}$. They asked a question whether exponential advice allows to recognize all languages (with one-way input), and, in particular, whether $L_{\mathsf{PAL}} \in \mathscr{L}(\mathrm{DFA})/\mathit{exp}$.

Our Contribution

We answer the question from [6], and show that L_{PAL} cannot be recognized by a DFA regardless of the advice size, i.e., $L_{\mathsf{PAL}} \notin \mathscr{L}(\text{DFA})/\star$ (Corollary 1). Moreover, we show that DFA cannot utilize more than exponential advice (Theorem 3). Then we extend the model from [6] to nondeterministic FA, and show that $\mathscr{L}(\text{NFA})/\mathit{exp}$ contains all languages (Theorem 4). We also show that for constant advice the nondeterminism doesn't help, since $\mathscr{L}(\text{NFA})/k = \text{REG}/k$ (Theorem 5). Since NFA can recognize any language with exponential advice, it is natural to ask which languages are in $\mathscr{L}(\text{NFA})/\mathit{poly}$. We show that $L_{\mathsf{PAL}} \notin \mathscr{L}(\text{NFA})/\mathit{poly}$ (Corollary 2) whereas $\mathit{co}L_{\mathsf{PAL}} \in \mathscr{L}(\text{NFA})/\mathit{poly}$ (Theorem 8), so $\mathscr{L}(\text{NFA})/\mathit{poly}$ is not closed under complement. Moreover, since $\mathscr{L}(\text{DFA})/\star$ is obviously closed under complement, $\mathit{co}L_{\mathsf{PAL}}$ is an example of a language that can be recognized nondeterministically with polynomial advice, but cannot be recognized deterministically regardless of the advice size. We extend this observation to show that for any growing function f, there is a language that can be recognized by a NFA with advice $O(f(n))$, but cannot be recognized by DFA regardless of advice (Theorem 6). Further, we show that any bounded language can be recognized by NFA with polynomial advice (Theorem 9), and if the language is of the form $L \subseteq a_1^\star \cdots a_k^\star$, it can even be recognized deterministically with polynomial advice (Theorem 10). Finally, we show a hierarchy of advice lengths for NFA (Theorem 11), and even stronger result for sublinear advice stating that for any advice of size $f(n) \leq n$ there is a language that can be recognized by a DFA with advice $f(n)$, but cannot be recognized by an NFA with advice $o(f(n))$ (Theorem 12).

2 Notation and Definitions

Let kDFA (resp. kNFA) denote a k-tape one-way deterministic (resp. nondeterministic) finite automaton. We use the standard model of multi-tape automata (see e.g., [2]) where the input words on each tape are delimited by special symbols, and in each step, the automaton decides, based on the symbols on the tapes, and the current state, which heads to advance and how to change the state. We say that a tuple $(w_1, \ldots, w_k) \in (\Sigma^\star)^k$ is the input of the kDFA automaton A if each w_i is written on the respective tape. For an automaton A, let $\mathscr{L}(A)$ be the language recognized by A. Let $\mathscr{L}(k\text{DFA}) \subseteq (\Sigma^\star)^k$ (resp. $\mathscr{L}(k\text{NFA})$) be the family of languages recognized by the respective automata. The symbol Σ denotes a finite alphabet, $\Sigma_n := \{0, 1, \ldots, n-1\}$. Unless stated otherwise, our automata will be 1-way. For technical clarity, we recall the definitions of the various approaches mentioned in the introduction. Damm and Holzer introduced the advice string as a prefix of the input word:

Definition 1 (Damm and Holzer [1]). *Let Σ be a finite alphabet, and let $\alpha\colon \mathbb{N} \to \Sigma^\star$ such that $\forall n, |\alpha(n)| \leq f(n)$. For a language L, let $\alpha L = \{\alpha(|w|)\#w \mid w \in L\}$. Then*

$$\text{REG}/f(n) := \{L \subseteq \Sigma^\star \mid \exists \alpha, \textit{ and a } \text{DFA } A\colon \alpha L = \mathscr{L}(A) \cap \alpha\Sigma^\star\}.$$

Tadaki et al. considered the advice written on a separate track:

Definition 2 (Tadaki et al. [7]). *For two words* $u = u_1 \dots u_n \in \Sigma_1, v = v_1 \dots v_n \in \Sigma_2$ *let* $\left[\begin{smallmatrix} u \\ v \end{smallmatrix}\right] = (u_1, v_1)(u_2, v_2) \dots (u_n, v_n) \in \Sigma_1 \times \Sigma_2$. *Let* Σ *be a finite alphabet, and let* $\alpha \colon \mathbb{N} \to \Sigma^\star$ *such that for each* $n \in \mathbb{N}$, $|\alpha(n)| = n$. *For a language* L, *let* $\left[\begin{smallmatrix} \alpha \\ L \end{smallmatrix}\right] = \left\{ \left[\begin{smallmatrix} \alpha(|w|) \\ w \end{smallmatrix}\right] \mid w \in L \right\}$. *Then*

$$\widehat{\mathrm{REG}}/n := \left\{ L \subseteq \Sigma^\star \mid \exists \alpha, \textit{and a } \mathrm{DFA}\ A \colon \begin{bmatrix} \alpha \\ L \end{bmatrix} = \mathscr{L}(A) \cap \left(\begin{bmatrix} \alpha \\ \Sigma^\star \end{bmatrix} \right) \right\}.$$

Freivalds considered 2-way multitape machines with advice on several tapes, such that the advice $\alpha(n)$ is valid for all words of length at most n:

Definition 3 (Freivalds [3]). *Let, for each* $1 \le i \le k$, $\alpha_i \colon \mathbb{N} \to \Sigma^\star$ *such that* $\forall n$, $\sum_{i=1}^{k} |\alpha_i(n)| \le f(n)$. *Then*

$$\begin{aligned} \mathscr{F}(\mathrm{DFA})/f(n) := \{ L \subseteq \Sigma^\star \mid \exists k, \alpha_1, \dots, \alpha_k, \ &\textit{and a 2-way } (k+1)\mathrm{DFA}\ A, \\ w \in L \Rightarrow \forall m \ge |w| \colon \ &(w, \alpha_1(m), \dots, \alpha_k(m)) \in \mathscr{L}(A) \textit{ and} \\ w \notin L \Rightarrow \forall m \ge |w| \colon \ &(w, \alpha_1(m), \dots, \alpha_k(m)) \notin \mathscr{L}(A) \}. \end{aligned}$$

We adopt the approach from Küçük et al., where the advice is on separate tapes (in general, we allow multiple advice tapes) and is specific for words of given length:

Definition 4. *Let, for each* $1 \le i \le k$, $\alpha_i \colon \mathbb{N} \to \Sigma^\star$ *such that* $\forall n, |\alpha_i(n)| = f(n)$. *For a language* L, *let* $L_\alpha = \{(w, \alpha_1(n), \dots, \alpha_k(n)) \mid w \in L, n = |w|\} \subseteq (\Sigma^\star)^{k+1}$. *Then*

$$\begin{aligned} \mathscr{L}(\mathrm{DFA})/f(n)_k := \{ L \subseteq \Sigma^\star \mid \exists \alpha_1, \dots, \alpha_k, \ \textit{and a } (k+1)\mathrm{DFA}\ A \colon \\ L_\alpha = \mathscr{L}(A) \cap \Sigma_\alpha^\star \}. \end{aligned}$$

We say that a $(k+1)$DFA A *recognizes language* L *with advice* α *if* $L_\alpha = \mathscr{L}(A) \cap \Sigma_\alpha^\star$. *We can leave out* k *if* $k = 1$. *The class* $\mathscr{L}(\mathrm{NFA})/f(n)_k$ *is defined in a similar way.*

We write $\mathscr{L}(\mathrm{DFA})/\star_k$ if the size of the advice is unlimited, $\mathscr{L}(\mathrm{DFA})/exp_k$ if it is at most exponential, and $\mathscr{L}(\mathrm{DFA})/poly_k$ if it is at most polynomial in the input length. We can further modify the automaton by giving specifications of the form `rt-tape` meaning that the tape is realtime, i.e., the head moves in every step to the right, and the automaton stops after the head reaches the end of the tape, or `2w-tape` meaning that the head is 2-way. So, e.g., $\mathscr{L}(\mathrm{DFA})/o(n)(\texttt{rt-input}, \texttt{2w-advice})$ describes deterministic automata with realtime input tape and 2-way advice tape of sublinear size. Note that the requirement $|\alpha(n)| = f(n)$ comes with no loss of generality, since the advice can always be padded with some new symbol. Also note, that we do not specify the cardinality of the advice alphabet. While this may be of importance in some cases, e.g., when studying advice of constant size, it has no effect on our results.

3 Results

The model quickly becomes extremely powerful: with two advice tapes, or exponential advice and 2-way input, all languages can be recognized. On the other hand, a DFA cannot recognize some very simple languages even with fairly large advice, and there is a hierarchy showing that additional advice size increases the power. The following statements have been proven in [6]:

Theorem 1 (Küçük et al. [6]).

1. $\mathscr{L}(\mathrm{DFA})/\mathit{exp}_2 = \mathscr{L}(\mathrm{DFA})/\mathit{exp}(\texttt{2w-input}, \texttt{rt-advice}) = 2^{\Sigma^\star}$.
2. *For all k, $\mathscr{L}(\mathrm{DFA})/(n^k) \subsetneq \mathscr{L}(\mathrm{DFA})/(n^{k+1})$.*
3. $L_{\mathsf{PAL}} \notin \mathscr{L}(\mathrm{DFA})/\mathit{poly} \cup \mathscr{L}(\mathrm{DFA})/\star(\texttt{rt-input})$ *where L_{PAL} is the language of palindromes $L_{\mathsf{PAL}} = \{ww^R \mid w \in \Sigma_2^\star\}$.*

In this paper, we focus our attention on machines with one advice tape, and 1-way input tape. In [6], the authors asked if $L_{\mathsf{PAL}} \in \mathscr{L}(\mathrm{DFA})/\star$. We show that it is not the case. In fact, our proof applies not only to L_{PAL}, but to a slightly more general class of languages described in the following definition. Informally, the words consist of two parts of fixed lengths: an arbitrary request string, and a response. There may be several responses to a given request. The required property is that for any two requests there is a string that is a valid response for exactly one of them.

Definition 5. *Let $\{R_n\}_{n=1}^\infty$ be a family of relations $R_n \subseteq \Sigma_2^n \times \Sigma^{f(n)}$ for some $f\colon \mathbb{N} \to \mathbb{N}$, such that $\forall x_0, x_1 \in \Sigma_2^n, x_0 \neq x_1$, there is a $y \in \Sigma^{f(n)}$ such that $R_n(x_i, y)$ and $\neg R_n(x_{1-i}, y)$ for some $i \in \{0,1\}$. Let L_R be the language*

$$L_R := \{xy \mid x \in \Sigma_2^\star,\ |y| = f(|x|),\ R_{|x|}(x,y)\}.$$

We call L_R a prefix-sensitive language *for relation R.*

Example 1. Examples of prefix-sensitive languages are some well studied languages like

- $L_{\mathsf{PAL}} = \{ww^{\mathsf{R}} \mid w \in \Sigma^\star\}$,
- $\mathit{NUM}_{\leq} := \{x\#y \mid x, y \in \Sigma_2^\star, |x| = |y|, [x]_2 \leq [y]_2\}$ where $[x]_2 \in \mathbb{N}$ denotes the number with binary representation x,
- $\mathit{NUM}_{<} := \{x\#y \mid x, y \in \Sigma_2^\star, |x| = |y|, [x]_2 < [y]_2\}$,
- $\{ww \mid w \in \Sigma^\star\}$.

Theorem 2. *Let L be a prefix-sensitive language. Then $L \notin \mathscr{L}(\mathrm{DFA})/\star$.*

Before proving the theorem, let us introduce some notation. Let A be any 2-tape DFA with the set of states Q, where $s := |Q|$, and advice α. For a fixed n we shall consider words of length $n+f(n)$, and denote $m := |\alpha(n+f(n))|$. For an $i \in \{1, \ldots, m\}$ and a $q \in Q$, we say that A is in *internal configuration* (i, q) if the advice head of A is reading the i-th advice symbol and A is in state q. We define the *internal configuration graph G of A* as the graph whose vertices are (i, q) for all such i, q. For each vertex (i, q), there are two outgoing directed edges, labelled by symbols 0 and 1, respectively. Each of the edges may be additionally labelled by $+$. These edges describe the behavior of A: An edge labelled by $x \in \{0, 1\}$ leads to (i', q') such that A moves to internal configuration (i', q') within one computation step when the input head reads x. The edge is additionally labelled by $+$ if and only if A moves its input head in this step.

The internal configuration graph is completely defined by the transition function of A and the advice for input length $n + f(n)$. Also, the behavior of A on inputs of length $n + f(n)$ is determined by the internal configuration graph. When A is in internal configuration $z = (i, q)$ and reads symbol x, it follows a path in G induced by edges labelled by x which ends with an edge labelled by $x+$, leading to internal configuration z' and we say that x *leads from* z *to* z'. The definition of *leads to* can be naturally extended to words from $\Sigma_2^\star$.

Lemma 1. *Let $z = (i, q) \in G$ be arbitrary internal configuration of A. There exist two words $u, v \in \Sigma_2^\star$ such that*

1. *$u \neq v$,*
2. *neither u is prefix of v nor v is prefix of u,*
3. *$1 \leq |u| \leq 2s + 2, 1 \leq |v| \leq 2s + 2$,*
4. *both u and v lead from z to the same internal configuration z'.*

Proof. Consider all words over Σ_2 of length exactly $2s + 1$. Each of these words leads from z to some $(j, \_)$. Let w be such word where j is minimal. Consider each proper prefix p of w, including the empty word ε; there are $2s + 1$ of them. Thus, we have $w = pxg$ for some $x \in \Sigma_2, g \in \Sigma_2^*$ (see Fig. 1). Since p is prefix of w, it leads from (i, q) into (i', q'), where $i \leq i' \leq j$. Let $x \neq x' \in \Sigma_2$. The word $px'g$ leads from (i, j) via (i', q') to $(j', \_)$, where $j' \geq j$. Thus, some edge outgoing from some $(j, \_)$ is used by A when reading $px'g0$. Let w' be the longest prefix of $px'g0$ such that an edge like this is used when reading the last symbol of w'. It holds that px' is a prefix of w'. This ensures that any two w' constructed for different prefixes p are not a prefix of each other.

In this way, we have constructed $2s + 1$ different words $w'_1, \ldots, w'_{2s+1}$ and any two of them satisfy conditions 1, 2, and 3. Since there are only $2s$ edges outgoing from $(j, \_)$, we can apply the pigeonhole principle to find w'_a and w'_b which use the same outgoing edge. This implies that w'_a and w'_b lead from z to the same internal configuration z'. □

Proof (of Theorem 2). Let $n := 4(s + 1)^2$. We prove that there are two input words of length $n + f(n)$, one in L and one not in L, which are both either accepted or rejected by A.

Automaton A starts in internal configuration $(1, q_0)$. We construct a sequence of internal configurations $c_0 = (1, q_0), c_1, \ldots, c_{2s+2}$ by invoking the claim to get the next configuration from the previous one. In this way, we obtain, for each i, some u_i and v_i satisfying all conditions from the claim that both lead from c_{i-1} to c_i. We now have $2s + 2$ pairs u_i, v_i. For each pair, $||u_i| - |v_i|| \leq 2s + 1$. Our goal is to construct two different words of length at most $4(s + 1)^2$ that lead

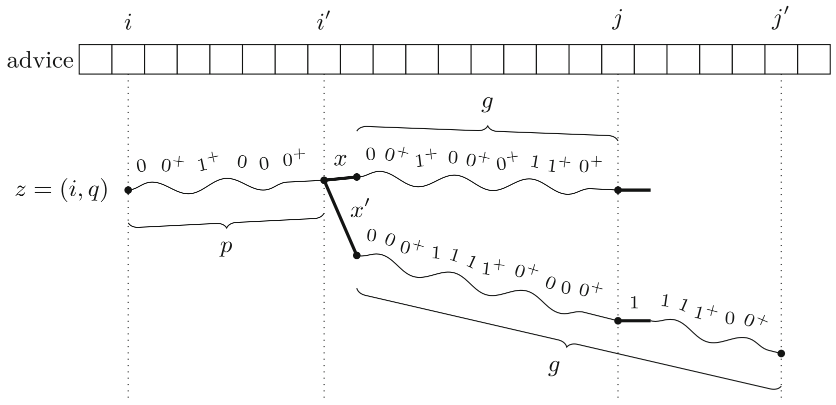


Fig. 1. Situation in the proof of Lemma 1: the word $w = pxg$ leads to some configuration $(j, \_)$. The word $px'g$ leads to some $(j', \_)$ for $j' \geq j$. Hence some prefix of $px'g$ uses an edge outgoing from some $(j, \_)$. In the proof we use the words $px'g0$ to cover the case $j' = j$.

from $(1, q_0)$ to the same internal configuration. We consider two cases. First, if $|u_i| = |v_i|$ for some i, we can take $u_1u_2 \dots u_i$ and $u_1u_2 \dots u_{i-1}v_i$. Both such words have equal length, lead from $(1, q_0)$ to c_i, and their length is at most $4(s+1)^2$. In the second case, $|u_i| \neq |v_i|$ for all $i \in \{1, \dots, 2s+2\}$. By the pigeonhole principle, there are two pairs such that $||u_i| - |v_i|| = ||u_j| - |v_j||$. Without loss of generality, let $|u_i| > |v_i|$ and $|u_j| > |v_j|$. Then the words $u_1 \dots u_{j-1}v_j$ and $u_1 \dots u_{i-1}v_iu_{i+1} \dots u_j$ satisfy our condition.

Thus, we have some two different words of equal length, no longer than $n = 4(s+1)^2$, that lead from $(1, q_0)$ to the same internal configuration. We can arbitrary pad both words to have length exactly n to obtain $u \neq v$ such that $|u| = |v| = n$ that lead from the initial internal configuration to the same internal configuration. Since L is prefix-sensitive, there is a $y \in \Sigma^{f(n)}$ such that $uy \in L$ and $vy \notin L$, or $vy \in L$ and $uy \notin L$. However, uy and vy are either both accepted or both rejected by A. ☐

Using Theorem 2, we can show that several languages cannot be recognized by DFA, regardless of the advice size. In particular:

Corollary 1. *$L_{\mathsf{PAL}} \notin \mathscr{L}(\mathrm{DFA})/\star$.*

An interesting question to ask is what is the maximal advice that a DFA is able to use. We can show that advice above $2^{O(n)}$ cannot be utilized:

Theorem 3. *$\mathscr{L}(\mathrm{DFA})/\star = \mathscr{L}(\mathrm{DFA})/2^{O(n)}$.*

Proof. Let A be an s-state DFA with advice α recognizing some language L with advice using alphabet Σ_k for some k, and let $\ell = s^{ks} + 1$. We show that L can be recognized by A using advice of length less than $\ell(nk^n + 1) = 2^{O(n)}$.

Suppose, for the sake of contradiction, that $|\alpha(n)| \geq \ell(nk^n + 1)$ for some n. We construct a modified advice function α' such that $\alpha'(n') = \alpha(n')$ for $n' \neq n$, and $|\alpha'(n)| < |\alpha(n)|$, and show that A recognizes the same language with advice α, and α'.

Suppose, without loss of generality, that A always moves both its heads to the ends of the respective tapes. Also, suppose that A in each step moves at least one head. Partition the advice string $\alpha(n)$ into blocks of size ℓ. For each of the k^n possible input words w of length n consider the computation of A on w with advice $\alpha(n)$. If the head on the input tape moves at least once while the head on the advice tape scans block B, mark B as *relevant*. Since each word w can mark at most n blocks, there are at most nk^n relevant blocks overall. Since $|\alpha(n)| > \ell nk^n$, there is some block B that was not marked relevant by any word.

For each word w, the automaton A enters block B in state q_w, reading symbol a_w on the input tape. Then for the consecutive ℓ steps it moves only the advice head, going through a sequence of states $q_w = q_w^{(1)}, q_w^{(2)}, \ldots, q_w^{(\ell)}$. Since the input head does not move, this sequence is fully determined by (a_w, q_w), so there are only ks distinct sequences. For an index i, consider the vector $\eta_i = \left(q_{w_1}^{(i)}, q_{w_2}^{(i)}, \ldots, q_{w_{k^n}}^{(i)}\right)$ where $\{w_1, \ldots, w_{k^n}\} = \Sigma_k^n$. Since there are ks distinct sequences, there are at most s^{ks} possible values of η_i, and because $\ell > s^{ks}$, there are two indices i, j, such that for each $w \in \Sigma_k^n$ it holds $q_w^{(i)} = q_w^{(j)}$. This means that if the advice string $\alpha(n)$ is shortened by leaving out the part of block B starting from index $i+1$, and continuing until (and including) index j, the automaton A will behave exactly the same way. □

It remains open whether advice of exponential size can actually be utilized by a DFA. From Theorem 1 we know that for any polynomial n^k, there is a language that needs advice $O(n^k)$, but we are not aware of any example of a language that would require more than polynomial advice.

3.1 Nondeterministic Automata

Next we turn our attention to nondeterministic automata, which have not been studied with respect to advice before. With no restriction on the advice size, it is easy to see that NFA can recognize all languages:

Theorem 4. *$\mathscr{L}(\mathrm{NFA})/f(n) = 2^{\Sigma^\star}$, where $f(n) = (n+1)|\Sigma|^n$. In particular, any language L can be recognized by an* NFA *with advice of size $(n+1)|L \cap \Sigma^n|$.*

Proof. Let $L \subseteq \Sigma^\star$ be any language. The advice function

$$\alpha(n) = \#w_1\#w_2\#\cdots\#w_z$$

where $\{w_1, \ldots, w_z\} = L \cap \Sigma^n$. The 2-tape NFA automaton just scans the advice tape, stops nondeterministically on some symbol #, and checks the input tape with the advice. □

In [6] it has been proven that $\mathscr{L}(\mathrm{DFA})/k(\texttt{2w-input}) = \mathrm{REG}/k$. Again, it is easy to observe that to constant advice, the nondeterminism is not more powerful than determinism:

Theorem 5. *$\mathscr{L}(\mathrm{NFA})/k = \mathrm{REG}/k$.*

Proof. Obviously, $\mathscr{L}(\mathrm{NFA})/k \supseteq \mathrm{REG}/k$. The other inclusion is easy to see, too. Any 2-tape NFA with advice tape of constant size can be transformed into a normal form where it first (deterministically) reads the content of the tape, stores it in the state, and continues to work (nondeterministically) on the input tape. We can use standard subset construction to turn this automaton into a deterministic one that first reads the advice for either the prefix of the input or a separate advice tape. □

Unlike DFA, NFA can recognize all languages with exponential advice, but are not more powerful than DFA's when equipped with constant advice. One may ask where is the threshold when NFA become more powerful. We show that it is just above the constant:

Theorem 6. *Let $f(n) = \omega(1)$. There is a language $L \in \mathscr{L}(\mathrm{NFA})/o(f(n))$ such that $L \notin \mathscr{L}(\mathrm{DFA})/\star$.*

Proof. Choose a function $g(n)$ such that $g(n)2^{g(n)} = o(f(n))$, e.g., $g(n) := \log\log f(n)$. Let $L := \{v \mid v = ww^{\mathsf{R}}\#^i, |w| = g(|v|)\}$. From Theorem 4 it follows that L_{PAL} can be recognized by a NFA with advice α such that $|\alpha(n)| = O(n2^{\frac{n}{2}})$. Moreover, to recognize L, one can utilize the advice $\alpha(2g(n))$, which is of size $o(f(n))$.

On the other hand, L is a prefix-sensitive language in terms of Definition 5, so due to Theorem 2, $L \notin \mathscr{L}(\mathrm{DFA})/\star$. □

Interesting classes of languages are those that can be recognized by NFA (or DFA) with polynomial advice. While we don't know whether DFA can utilize more than polynomial advice, we show that NFA can. In particular, NFA can recognize all languages with exponential advice, but cannot recognize with polynomial advice (even with two-way advice tape) many prefix-sensitive languages. To state the next theorem, we use a subclass of prefix-sensitive languages:

Definition 6. *A prefix-sensitive language L is called* strictly prefix-sensitive, *if there is a function $d\colon \Sigma_2^\star \to \Sigma^\star$, such that for each $x \in \Sigma_2^\star$, it holds $R_{|x|}(x, d(x))$ (i.e., $|d(x)| = f(|x|)$, $xd(x) \in L$), and for any two $x_0, x_1 \in \Sigma_2^n, x_0 \neq x_1$ it holds $R_n(x_i, d(x_i))$, and $\neg R_n(x_{1-i}, d(x_i))$ for some $i \in \{0, 1\}$.*

Note that all the languages from Example 1 are strictly prefix-sensitive: for $\mathit{NUM}_{\geq}$, consider the function $d(x) = \#x$. Similarly, for $\mathit{NUM}_{<}$ the function $d(x) = \#y$ such that $[y]_2 = [x]_2 + 1$ fulfills the previous definition.

Theorem 7. *Let L be a strictly prefix-sensitive language. Then*

$$L \notin \mathscr{L}(\mathrm{NFA})/\mathit{poly}(\texttt{2w-advice}).$$

Proof. Consider a 2-tape NFA A with polynomial advice α. For a fixed n, consider all words $w \in \Sigma^{n+f(n)}$, such that $w = xd(x)$, $x \in \Sigma_2^n$, $w \in L$, and select one accepting computation γ_w of A on w (using the advice $\alpha(n+f(n))$). After reading x in γ_w, let A be in state q_w, and the advice head be on the i_w-th symbol. Since there are only polynomially many pairs (q_w, i_w), there are two words $x_1, x_2 \in \Sigma_2^n$ such that A is in the same state with the same position on the advice tape after reading x_1 and x_2 in the respective accepting computations $\gamma_{w_1}, \gamma_{w_2}$. Since L is strictly prefix-sensitive, without loss of generality $x_1 d(x_1) \in L$, $x_2 d(x_1) \notin L$. Since the automaton is 1-way, and the advice is fixed, there is an accepting computation on $x_2 d(x_1)$ with the same prefix as γ_{w_2}. Thus, $x_2 d(x_1) \in L$ – a contradiction. □

Corollary 2. $L_{\mathsf{PAL}} \notin \mathscr{L}(\text{NFA})/poly$.

On the other hand,

Theorem 8. *co*$L_{\mathsf{PAL}} = \{a,b\}^\star - L_{\mathsf{PAL}} \in \mathscr{L}(\text{NFA})/O(n^2)$.

Proof. We construct a nondeterministic automaton A with quadratic advice α that recognizes *co*L_{PAL}. For odd n, let the advice $\alpha(n) = \$$. A immediately accepts in this case.

For even n, let the advice

$$\alpha(n) = \#w_1\#w_2\#\cdots\#w_{\frac{n}{2}}\#$$

where $w_i = 0^{i-1}10^{n-2i}10^{i-1}$. Note that $|w_i| = n$ so the advice is of length $O(n^2)$. The automaton A nondeterministically selects one word w_i, and uses it to check the input symbols on the positions indicated by ones in w_i. If they differ, A accepts. □

One class of languages that can be accepted by NFA with polynomial advice is the class of bounded languages:

Theorem 9. *Let* $w_1, \ldots, w_k \in \Sigma^\star$. *Let* L *be any bounded language* $L \subseteq w_1^\star \cdots w_k^\star$. *Then* $L \in \mathscr{L}(\text{NFA})/poly$.

Proof. It is easy to see that a bounded language contains at most $(n+1)^k = O(n^k)$ words of length n. The result follows from Theorem 4. □

We don't know whether bounded languages can be recognized with polynomial advice by DFA, but an important subclass of bounded languages can:

Theorem 10. *There is a* DFA A_k *such that for any language*

$$L \subseteq 0^\star 1^\star \cdots (k-1)^\star \subseteq \Sigma_k^\star$$

there is an advice α, *such that* $|\alpha(n)| \le c_k n^{k-1}$ *for some* c_k, *and* A_k *recognizes* L *with advice* α.

Proof. The proof is by induction on k. For $k = 1$, any language over unary alphabet contains at most one word for each n, so advice of size 1 is sufficient. Let $L \subseteq 0^\star 1^\star \cdots (k-1)^\star$. For a word w, and a language L, let $wL := \{wu \mid u \in L\}$. Denote L_i the language $L_i \subseteq 1^\star 2^\star \cdots (k-1)^\star$ such that $0^i L_i = L \cap 0^i\{1, \dots, k-1\}^\star$. Obviously, $L = \cup_{i=0}^{\infty} L_i$, and each L_i is a bounded language over alphabet Σ_{k-1} (under a renaming of the symbols). By induction, each language L_i can be recognized by some DFA A_{k-1} with advices α_i such that $|\alpha_i(n)| \le c_{k-1} n^{k-2}$ for some c_{k-1}. Construct the advice function α such that

$$\alpha(n) = 0\alpha_0(n)0\alpha_1(n-1)0\alpha_2(n-2)0 \cdots 0\alpha_{n-1}(1)0\alpha_n(0).$$

Note that $|\alpha(n)| \le (n+1)c_{k-2}n^{k-2} + n + 1 \le c_k n^{k-1}$ for some c_k. The DFA A_k recognizing L with advice α works as follows: while the input symbol is 0, it scans the advice tape for the next occurrence of 0. If the input symbol is not 0, it simulates the automaton A_{k-1} with the current advice. Note that the transition function of A does not depend on the language L, so it fulfills the statement of the theorem. □

3.2 Hierarchies

In [6] it was shown that for all k, $\mathscr{L}(\text{DFA})/(n^k) \subsetneq \mathscr{L}(\text{DFA})/(n^{k+1})$. We show a similar hierarchy for NFA:

Theorem 11. *Let $f, g \colon \mathbb{N} \to \mathbb{N}$ be such that $f(n)\log(f(n)) = o(g(n))$ and $g(n) \le n2^{\frac{n}{2}}$. Then $\mathscr{L}(\text{NFA})/f(n)(\texttt{2w-advice}) \subsetneq \mathscr{L}(\text{NFA})/g(n)$.*

Proof. We repeat the ideas from Theorems 6 and 7 in a more specific way. Fix two functions f, g from the statement of the theorem. Note that $\lim_{n\to\infty} g(n) = \infty$. Let h be a function such that

$$h(n) := \max\{x \mid (2x+1)2^x \le g(n)\}.$$

Since $(2h(n)+1)2^{h(n)} \le n2^{\frac{n}{2}}$, it holds $2h(n) \le n$. Consider the language

$$L = \{ww^{\mathsf{R}}\#^{n-2h(n)} \mid w \in \Sigma_2^{h(n)}, n \in \mathbb{N}\},$$

where $\#^0$ is defined as empty string. First, we show that

$$L \notin \mathscr{L}(\text{NFA})/f(n)(\texttt{2w-advice}).$$

Let us suppose, for the sake of contradiction, that L is recognized by some NFA A with advice $|\alpha(n)| \le f(n)$. Let s be the number of states of A. We show that for all large enough n it holds

$$sf(n) < 2^{h(n)}. \tag{1}$$

Assume, for the sake of contradiction, that $sf(n) \ge 2^{h(n)}$ for arbitrary large n. Since $sf(n) \ge 2^{h(n)}$, it holds $\log s + \log f(n) \ge h(n)$, so $\log f(n) \ge h(n) - \log s$.

Also recall that $f(n) \geq 2^{h(n)}/s$. We have

$$\begin{aligned} f(n)\log f(n) &\geq f(n)(h(n) - \log s) \geq \frac{2^{h(n)}}{s}h(n) - \frac{2^{h(n)}}{s}\log s \\ &= \frac{2^{h(n)+1}4h(n)}{8s} - \frac{2^{h(n)}\log s}{s} > \frac{g(n)}{8s} - \frac{2^{h(n)}\log s}{s} \end{aligned}$$

where the last inequality comes from the fact that $4h(n) \geq 2(h(n)+1)+1$, which is satisfied for $h(n) \geq 2/3$, and $2^{h(n)+1}(2(h(n)+1)+1)$, which follows directly from the definition of $h(n)$. Thus, we get

$$f(n)\log f(n) > \frac{g(n)}{16s} + \left(\frac{g(n)}{16s} - \frac{2^{h(n)}\log s}{s}\right) \geq \frac{g(n)}{16s} \tag{2}$$

where the last inequality holds since for large enough n we have $2^{h(n)}16\log s \leq 2^{h(n)}(2h(n)+1) \leq g(n)$. However, $f(n)\log(f(n)) = o(g(n))$, so (2) cannot hold arbitrary large n – thus we have proven (1).

Now fix a large enough n. $|L \cap \Sigma_2^n| = 2^{h(n)}$. For each $w \in \Sigma_2^{h(n)}$ choose one accepting computation γ_w of A on $ww^{\mathsf{R}}\#^{n-2h(n)}$. Let A be in state q_w after reading the prefix w in γ_w, and its advice head is on position i_w. Since there are $sf(n)$ pairs (q_w, i_w), and $sf(n) < 2^{h(n)}$ there must be two words $w \neq w'$ with the same pair $(q_w, i_w) = (q_{w'}, i_{w'})$, which means there is also an accepting computation for $ww'^{\mathsf{R}}\#^{n-2h(n)}$.

On the other hand, it is easy to observe that $L \in \mathscr{L}(\text{NFA})/g(n)$: an advice of length $2^{h(n)}(2h(n)+1)$ is enough to describe all palindromes ww^{R}, $w \in \Sigma_2^{h(n)}$. □

For sublinear advice we give a stronger result:

Theorem 12. *Let $f, g\colon \mathbb{N} \to \mathbb{N}$ be such that $f(n) \leq n$, $g(n) = o(f(n))$. Then there is a language $L_f \subseteq \{a, b\}^\star$ that does not depend on g such that $L_f \in \mathscr{L}(\text{DFA})/f(n)$ and $L_f \notin \mathscr{L}(\text{NFA})/g(n)$.*

Proof. We shall present a language $L_f \subseteq \Sigma_2^\star$, such that for each n, L_f contains exactly one word of length n, called w_n. Moreover, this word is of the form $w_n \in \Sigma_2^{f(n)}0^{n-f(n)}$. This immediately implies that $L_f \in \mathscr{L}(\text{DFA})/f(n)$. In the rest of the proof we show how to specify w_n such that $L_f \notin \mathscr{L}(\text{NFA})/g(n)$ for any $g(n) = o(f(n))$.

It is sufficient to prove the claim for NFA's with binary advice alphabet: If there exists an NFA with advice alphabet of size k that accepts L_f with advice $g(n) = o(f(n))$, there also exists a NFA with binary advice alphabet that accepts L_f with advice $g(n)\log k = o(f(n))$.

Consider a fixed enumeration $A_1, A_2, \ldots$ of all 2-tape NFA's with binary advice alphabets. For a given 2-tape NFA A, we call a word $v \in \Sigma_2^\star$ a *n-singleton word* for A, if A when equipped with v as advice, accepts exactly one word u of length n (among all words of length n), and u is of the form $\Sigma_2^{f(n)}0^{n-f(n)}$.

Now we describe how to select w_n for a given n. Let d_n be the maximum integer such that $2d_n + 1 < f(n)$, i.e., $d_n := \lfloor (f(n) - 1)/2 \rfloor$. Consider the first 2^{d_n} automata in the fixed enumeration. Since there are at most 2^{d_n+1} binary words of length at most d_n, there are at most $2^{d_n+d_n+1}$ possible pairs (A, v) such that v is n-singleton for A, $|v| \leq d_n$. Since $d_n + d_n + 1 < f(n)$, there is a word $w_n \in \Sigma_2^{f(n)}$ such that w_n is never accepted as a single n-letter word by any of the first 2^{d_n} automata, with advice of size at most d_n.

Finally, we show that language L_f defined as above is not in $\mathscr{L}(\text{NFA})/g(n)$. Assume, for the sake of contradiction, that there is an automaton $A = A_j$ that recognizes L_f with some advice α of size $|\alpha(n)| = g(n)$. Obviously, $\alpha(n)$ is n-singleton for A, for any n. Since $g(n) = o(f(n))$, there is some large enough $\bar{n}$ such that $g(\bar{n}) \leq d_{\bar{n}}$ and $d_{\bar{n}} \geq \log j$. Hence, A with advice $\alpha(\bar{n})$ accepts $w_{\bar{n}}$ as the single $\bar{n}$-letter word. However $d_{\bar{n}} \geq \log j$ means $w_{\bar{n}}$ is never accepted as a single $\bar{n}$-letter word by any of the first j automata with any advice of size at most $d_{\bar{n}}$, thus it is not accepted with any advice of size $g(\bar{n})$. □

4 Conclusion and Open Questions

We showed that there are languages that cannot be recognized by DFA, regardless of advice size. Moreover, we showed that DFA cannot utilize more than exponential advice. However, we don't know any example of a language, where advice of exponential size is needed. Indeed, it may be the case that any language that can be recognized by DFA with advice, can be recognized also with polynomial advice. In particular, it would be interesting to know if all bounded languages can be recognized by DFA with polynomial advice.

We initiated the study of NFA with advice. We showed that there are languages that cannot be recognized with polynomial advice, but any language can be recognized with exponential advice. It is a natural task to characterize the languages in $\mathscr{L}(\text{NFA})/\mathit{poly}$.

Also, Küçük et al. showed in [6] that the language

$$\mathsf{EQUAL}_3 = \{w \in \Sigma_3 \mid \#_0(w) = \#_1(w) = \#_2(w)\}$$

cannot be recognized by a DFA with linear advice, but it can be recognized by a randomized FA with 1-sided bounded error with linear advice. It would be interesting to know whether randomization can help for larger advices: in particular, what languages can be recognized by randomized FA with polynomial advice.

Finally, one of the features of the model from [3] is that it is concerned only for the size of the advice, which can be split into several tapes. In our model, DFA with two advice tapes with exponential advice can recognize all languages, however, the power of multi-tape DFA's with limited advice is to be considered.

Dedication

The authors are very grateful to Juraj for his great support, many enlightening discussions and mountain adventures. He has always encouraged people to push their limits, and has been finding means to help them grow. His constant optimism has been a stable source of inspiration.

References

1. Damm, C., Holzer, M.: Automata that take advice. In: Wiedermann, J., Hájek, P. (eds.) MFCS 1995. LNCS, vol. 969, pp. 149–158. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60246-1_121
2. Fischer, P.C., Rosenberg, A.L.: Multitape one-way nonwriting automata. J. Comput. Syst. Sci. **2**(1), 88–101 (1968)
3. Freivalds, R.: Amount of nonconstructivity in deterministic finite automata. Theor. Comput. Sci. **411**(38–39), 3436–3443 (2010)
4. Ibarra, O.H., Ravikumar, B.: Sublogarithmic-space turing machines, nonuniform space complexity, and closure properties. Math. Syst. Theory **21**(1), 1–17 (1988)
5. Karp, R.M., Lipton, R.J.: Turing machines that take advice. Enseign. Math. **28**(2), 191–209 (1982)
6. Küçük, U., Cem Say, A.C., Yakaryilmaz, A.: Finite automata with advice tapes. Int. J. Found. Comput. Sci. **25**(8), 987–1000 (2014)
7. Tadaki, K., Yamakami, T., Lin, J.C.H.: Theory of one-tape linear-time turing machines. Theor. Comput. Sci. **411**(1), 22–43 (2010)

A Survey on Fooling Sets as Effective Tools for Lower Bounds on Nondeterministic Complexity

Michal Hospodár[1], Galina Jirásková[1](✉), and Peter Mlynárčik[1,2]

[1] Mathematical Institute, Slovak Academy of Sciences, Grešákova 6, 040 01 Košice, Slovakia
hosmich@gmail.com, jiraskov@saske.sk

[2] Faculty of Electrical Engineering and Informatics, Technical University of Košice, Boženy Němcovej 32, 042 00 Košice, Slovakia
mlynarcik1972@gmail.com

Abstract. A fooling set for a regular language is a special set of pairs of strings whose size provides a lower bound on the number of states in any nondeterministic finite automaton accepting this language. We show that, in spite of the fact that the difference between the size of the largest fooling set and the nondeterministic state complexity may be arbitrarily large, the fooling set lower bound methods work in many cases. We modify the method in the case when multiple initial states may save one state. We also state some useful properties that allow us to avoid describing particular fooling sets which may often be difficult and tedious.

1 Introduction

The nondeterministic state complexity of a regular language is the smallest number of states in any nondeterministic finite automaton (NFA) accepting this language. To get lower bounds on the nondeterministic state complexity, usually a fooling set technique is used. A fooling set is a special set of pairs of strings whose size provides a lower bound on the number of states in any NFA for a given language.

The lower bound method based on fooling sets, as a version of the crossing sequence argument, has been used for proving lower bounds on VLSI computations [4,14,15,19,22]. The fooling set method as a method providing lower bounds on communication complexity has been formulated by Aho, Ullman, and Yannakakis [1]. In the settings of formal languages, the method has been first described by Birget [2,3], and examined by Glaister and Shallit [6].

Research supported by VEGA grant 2/0084/15 and grant APVV-15-0091. This work was conducted during PhD study of Michal Hospodár at the Faculty of Mathematics, Physics and Informatics of the Comenius University in Bratislava, Slovakia.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 17–32, 2018.
https://doi.org/10.1007/978-3-319-98355-4_2

Although the gap between the size of a largest fooling set and the nondeterministic state complexity of a regular language may be arbitrarily large [7], in many cases, the fooling set method provides lower bounds that are tight.

In [12,13,18,20], the nondeterministic state complexity of basic regular operations in the subclasses of convex languages has been investigated. The authors considered operations of union, intersection, concatenation, star, reversal, and complementation in the classes of prefix-, suffix-, factor-, and subword-free, -closed, and -convex languages, and right, left, two-sided, and all-sided ideals. For each operation and each class, except for complementation on factor- and subword-convex languages, tight upper bounds have been obtained, and to get lower bounds, a fooling set lower bound method has been used in each case.

Here we present the fooling set method as an effective tool for getting lower bounds on the nondeterministic state complexity. We state some sufficient properties on NFAs that guarantee the existence of a sufficiently large fooling set for the accepted language. As a result, we can avoid the description of a fooling set which may sometimes be rather difficult and tedious. Moreover, the size of a fooling set provides a lower bound on the size of NFAs even with multiple initial states. This means that, for example, in the case of union or reversal, where NFAs with multiple initial states may save one state in the resulting automaton, the method cannot yield matching bounds. We describe a modification of the method consisting in a possibility to divide a fooling set into two parts such that adding a pair with left component equal to the empty string results again in a fooling set. Since after reading the empty string, the NFA is in its unique initial state, this state must be different from all states given by the fooling set.

We start by restating the result from [7] that the gap between the size of a largest fooling set and the nondeterministic state complexity of a regular language may be arbitrarily large. Then we formulate two lemmas with sufficient conditions on NFAs that guarantee their minimality. We continue with a modification of the fooling set method providing lower bounds on the size of NFAs with a unique initial state. Finally, we give a sufficient condition for getting large lower bounds for the complementation operation.

2 Preliminaries

We assume that the reader is familiar with basic notions in formal languages and automata theory. For details and all the unexplained notions, the reader may refer to [11,23,24].

Let Σ be a finite non-empty alphabet of symbols. Then Σ^* denotes the set of strings over the alphabet Σ including the empty string ε. The length of a string w is denoted by $|w|$, and the number of occurrences of a symbol a in a string w by $|w|_a$. A language is any subset of Σ^*. For a finite set X, the cardinality of X is denoted by $|X|$, and its power set by 2^X.

A *nondeterministic finite automaton* (with a nondeterministic choice of initial states; cf. [24]) (NNFA) is a quintuple $A = (Q, \Sigma, \cdot, I, F)$, where Q is a finite non-empty set of states, Σ is a finite non-empty alphabet, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final (or accepting) states, and the function $\cdot\colon Q \times \Sigma \to 2^Q$ is the transition function which is naturally extended to the domain $2^Q \times \Sigma^*$. The *language accepted by* A is $L(A) = \{w \in \Sigma^* \mid I \cdot w \cap F \neq \emptyset\}$.

We say that (p, a, q) is a *transition* in A if $q \in p \cdot a$. If (p, a, q) is a transition in A, then we say that the state q has an *in-transition*, and the state p has an *out-transition*. We sometimes write $p \xrightarrow{w} q$ if $q \in p \cdot w$.

An NNFA A is a *trim* NNFA if each its state q is reachable and useful, that is, there are strings u and v in Σ^* such that $q \in I \cdot u$ and $q \cdot v \cap F \neq \emptyset$.

If $|I| = 1$, we say that A is a nondeterministic finite automaton (NFA). In an ε-NFA, we also allow transitions on the empty string. It is known that the ε-transitions can be removed without increasing the number of states in the resulting NFA (cf. [11, Theorem 2.2] and [24, Theorem 2.3]).

An NFA A is a *deterministic finite automaton* (DFA) if $|q \cdot a| = 1$ for each q in Q and each a in Σ. Next, A is a *partial deterministic finite automaton* if $|q \cdot a| \leq 1$ for each q in Q and each a in Σ.

Every NNFA $A = (Q, \Sigma, \cdot, I, F)$ can be converted to an equivalent DFA $\mathcal{D}(A) = (2^Q, \Sigma, \cdot, I, \{S \in 2^Q \mid S \cap F \neq \emptyset\})$. We call the DFA $\mathcal{D}(A)$ the *subset automaton* of the NNFA A. The subset automaton might not be minimal since some of its states may be unreachable or equivalent to other states.

The *nondeterministic state complexity of a regular language* L, $\operatorname{nsc}(L)$, is the smallest number of states in any NFA accepting L. The *nondeterministic state complexity of a regular operation* is the number of states that are sufficient and necessary in the worst case for an NFA to accept the language resulting from the operation, considered as a function of the number of states of NFAs for the given operands. Formally, the nondeterministic state complexity of a binary regular operation $\circ$ is a function from $\mathbb{N}^2$ to $\mathbb{N}$ defined as

$$(m, n) \mapsto \max\{\operatorname{nsc}(K \circ L) \mid \operatorname{nsc}(K) \leq m \text{ and } \operatorname{nsc}(L) \leq n\}.$$

For a language L over an alphabet Σ, the *complement* of L is the language $L^c = \Sigma^* \setminus L$. The *intersection* of languages K and L is the language $K \cap L = \{w \mid w \in K \text{ and } w \in L\}$. The *union* of languages K and L is the language $K \cup L = \{w \mid w \in K \text{ or } w \in L\}$. The *concatenation* of languages K and L is the language $KL = \{uv \mid u \in K \text{ and } v \in L\}$. The *(Kleene) star* of a language L is the language $L^* = \bigcup_{i \geq 0} L^i$ where $L^0 = \{\varepsilon\}$ and $L^i = LL^{i-1}$ if $i \geq 1$.

The reversal w^R of a string w is defined as $\varepsilon^R = \varepsilon$ and $(wa)^R = aw^R$ for each symbol a and string w. The reversal of a language L is $L^R = \{w^R \mid w \in L\}$. If a language L is accepted by an NNFA $A = (Q, \Sigma, \cdot, I, F)$, then the language L^R is accepted by the NNFA A^R obtained from A by reversing all the transitions, and by swapping the roles of the initial and final states. Formally, we have $A^R = (Q, \Sigma, \cdot^R, F, I)$ where $q \cdot^R a = \{p \in Q \mid q \in p \cdot a\}$.

Let $A = (Q, \Sigma, \cdot, I, F)$ be an NNFA and $S, T \subseteq Q$. We say that S is *reachable* in A if there is a string w in Σ^* such that $S = I \cdot w$. Next, we say that T is *co-reachable* in A if T is reachable in A^R. Notice that if T is co-reachable in A, then there is a string w in Σ^* such that w is accepted by A from each state in T and rejected from each state in T^c.

If $u, v, w, x \in \Sigma^*$ and $w = uxv$, then u is a *prefix* of w, x is a *factor* of w, and v is a *suffix* of w. If $w = u_0 v_1 u_1 \cdots v_n u_n$, where $u_i, v_i \in \Sigma^*$, then $v_1 v_2 \cdots v_n$ is a *subword* of w. A prefix v (suffix, factor, subword) of w is *proper* if $v \neq w$.

A language L is *prefix-free* if $w \in L$ implies that no proper prefix of w is in L; it is *prefix-closed* if $w \in L$ implies that each prefix of w is in L; and it is *prefix-convex* if $u, w \in L$ and u is a prefix of w imply that each string v such that u is a prefix of v and v is a prefix of w is in L. Suffix-, factor-, and subword-free, -closed, and -convex languages are defined analogously.

A language L is a right (respectively, left, two-sided, all-sided) *ideal* if $L = L\Sigma^*$ (respectively, $L = \Sigma^* L, L = \Sigma^* L \Sigma^*, L = L \sqcup\!\sqcup \Sigma^*$), where $\sqcup\!\sqcup$ denotes the shuffle operation [5]. Notice that the classes of free, closed, and ideal languages are subclasses of convex languages.

3 Fooling Set Lower Bound Method

To get lower bounds on the number of states in an NNFA accepting a regular language, the fooling set technique has been successfully used in the literature. We start with the definition of a fooling set, and with a lemma showing that the size of a fooling set for a regular language provides a lower bound on the nondeterministic state complexity of this language.

Definition 1. *A set of pairs of strings* $\{(x_i, y_i) \mid i = 1, 2, \ldots, n\}$ *is called a* fooling set for a language L *if for each* i, j *in* $\{1, 2, \ldots, n\}$,

(1) $x_i y_i \in L$, *and*
(2) if $i \neq j$, *then* $x_i y_j \notin L$ *or* $x_j y_i \notin L$.

Example 1. Let $L = \{a^n\}$ where $n \geq 0$. Consider the set of pairs of strings $\mathcal{F} = \{(a^i, a^{n-i}) \mid i = 0, 1, \ldots, n\}$. The string a^n is in L, while for each k with $k \neq n$, the string a^k is not in L. It follows that $\mathcal{F}$ is a fooling set for L. □

Lemma 1 (Birget [2, Lemma 1]). *Let* $\mathcal{F}$ *be a fooling set for a regular language* L*. Then every* NNFA *for* L *has at least* $|\mathcal{F}|$ *states.*

Proof. Let $\mathcal{F} = \{(x_i, y_i) \mid i = 1, 2, \ldots, n\}$ be a fooling set for L and A be an NNFA for L. For each i, fix an accepting computation of A on $x_i y_i$. Let p_i be the state on this computation reached after reading x_i, that is, we have

$$q_i \xrightarrow{x_i} p_i \xrightarrow{y_i} f_i$$

for an initial state q_i and a final state f_i. We now show that the states $p_1, p_2, \ldots, p_n$ are pairwise distinct. Let $i \neq j$ and suppose for a contradiction that $p_i = p_j$. However, in such a case, the NNFA A has the following accepting computation on both $x_i y_j$ and $x_j y_i$:

$$q_i \xrightarrow{x_i} p_i = p_j \xrightarrow{y_j} f_j \text{ and } q_j \xrightarrow{x_j} p_j = p_i \xrightarrow{y_i} f_i.$$

This is a contradiction with condition (2) of Definition 1. □

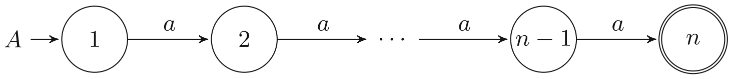


Fig. 1. A partial DFA in which each set $\{i\}$ is reachable and co-reachable.

Our first aim is to show that a gap between the maximal size of the fooling set and the nondeterministic state complexity of a language can be arbitrarily large. To this end, we introduce the notion of a *fooling set for an automaton.*

Definition 2. *A set of pairs of sets of states $\mathcal{S} = \{(X_i, Y_i) \mid i = 1, 2, \ldots, n\}$ is called a* fooling set for an NNFA A *if, for each* i, j *in* $\{1, 2, \ldots, n\}$*,*

(1) X_i is reachable and Y_i is co-reachable in A,
(2) $X_i \cap Y_i \neq \emptyset$, and
(3) if $i \neq j$, then $X_i \cap Y_j = \emptyset$ or $X_j \cap Y_i = \emptyset$.

Example 2. Let A be the partial DFA shown in Fig. 1. The set of pairs of sets

$$\{(\{i\}, \{i\}) \mid i = 1, 2, \ldots, n\}$$

is a fooling set for A since each $\{i\}$ is reachable and co-reachable in A, and conditions (2) and (3) of Definition 2 are satisfied as well. □

Proposition 1. *Let A be an* NNFA*. Then a fooling set of size n for the language $L(A)$ exists if and only if a fooling set of size n for the automaton A exists.*

Proof. Let $A = (Q, \Sigma, \cdot, I, F)$ be an NNFA. Let $\mathcal{F} = \{(x_i, y_i) \mid i = 1, 2, \ldots, n\}$ be a fooling set for $L(A)$. Set $X_i = I \cdot x_i$ and $Y_i = F \cdot^R y_i^R$ for $i = 1, 2, \ldots, n$. Then $\mathcal{S} = \{(X_i, Y_i) \mid i = 1, 2, \ldots, n\}$ is the desired fooling set for A.

Conversely, since X_i is reachable, there is a string x_i such that $X_i = I \cdot x_i$. Since Y_i is co-reachable, there is a string y_i such that $Y_i = F \cdot^R y_i$. Then the set $\{(x_i, y_i^R) \mid i = 1, 2, \ldots, n\}$ is a fooling set for $L(A)$. □

Theorem 1 (cf. Gruber and Holzer [7, Theorem 10]**).** *Let $n \geq 4$. There exists a language L such that every fooling set for L is of size at most 3, while every* NFA *for L has at least* $\log_2 n$ *states.*

Proof. Let L be the unary language accepted by the DFA A with the state set $Q = \{1, 2, \ldots, n\}$ shown in Fig. 2. The DFA A has n states and it is a minimal DFA for L. It follows that every NFA for L must have at least $\log_2 n$ states.

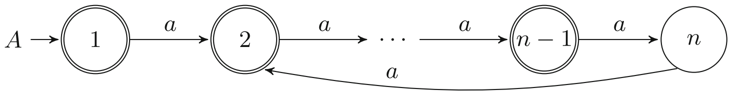


Fig. 2. The minimal DFA of a language with a fooling set of size at most 3 and with nondeterministic state complexity at least $\log_2 n$.

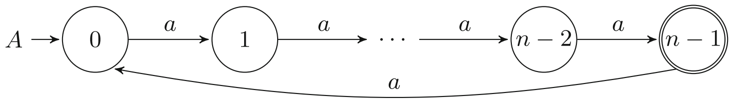


Fig. 3. A unary witness language for star meeting the upper bound $n + 1$.

Let us show that every fooling set for L is of size at most 3. Suppose for a contradiction that there is a fooling set for L of size 4. Then there is a fooling set $\mathcal{S} = \{(X_i, Y_i) \mid i = 1, 2, 3, 4\}$ for A. This means that for each i, the set X_i is reachable and the set Y_i is co-reachable in A with $X_i \cap Y_i \neq \emptyset$, and if $i \neq j$, then at least one of $X_i \cap Y_j$ and $X_j \cap Y_i$ is empty. Consider the bipartite graph $(\mathcal{R}, \mathcal{C}, E)$ where

- $\mathcal{R} = \{S \subseteq Q \mid S \text{ is reachable in } A\} = \{\{i\} \mid i = 1, 2, \ldots, n\}$,
- $\mathcal{C} = \{T \subseteq Q \mid T \text{ is co-reachable in } A\} = \{Q \setminus \{i\} \mid 2 \leq i \leq n\} \cup \{Q \setminus \{1, n\}\}$,
- $(S, T) \in E$ iff $S \cap T \neq \emptyset$.

Notice that each $\{i\}$ in $\mathcal{R}$, except for $\{n\}$, is connected to each set in $\mathcal{C}$, except for the set $Q \setminus \{i\}$. The set $\{n\}$ is connected to each set in $\mathcal{C}$, except for $Q \setminus \{n\}$ and $Q \setminus \{1, n\}$.

Now the set $\mathcal{S} = \{(X_i, Y_i) \mid i = 1, 2, 3, 4\}$ is a fooling set for A by assumption. Consider the subgraph G of $(\mathcal{R}, \mathcal{C}, E)$ induced by the set $\{X_i \mid i = 1, 2, 3, 4\} \cup \{Y_i \mid i = 1, 2, 3, 4\}$. If $X_i \neq \{n\}$, then X_i is not connected to at most one of Y_i, $i = 1, 2, 3, 4$. If $X_i = \{n\}$, then X_i is not connected to at most two of Y_i, $i = 1, 2, 3, 4$. This means that there are at least 11 edges in G. However, since $\mathcal{S}$ is a fooling set, for every two distinct pairs (X_i, Y_i) and (X_j, Y_j), at least one edge must be missing in G. In total, at least 6 edges must be missing in G – one for every two distinct pairs. However, this only gives 10 possible edges in G, a contradiction. □

Although the previous theorem shows that the gap between the size of a fooling set for a regular language and its nondeterministic state complexity may be arbitrarily large, our next aim is to show that in most cases, the fooling set technique provides lower bounds that are tight.

The next example illustrates how both types of fooling sets can be used to obtain the nondeterministic state complexity of the star operation. The upper bound on the nondeterministic state complexity of the star operation is $n + 1$ since we can construct an ε-NFA for the star of a language given by an NFA by adding a new initial and final state connected through ε-transitions to the original initial state, and by adding ε-transitions from every final state to the original initial state. The next example provides a unary witness language.

Example 3 (Star on regular languages; cf. [10, Theorem 9]*).* Let L be the unary language accepted by the n-state DFA A shown in Fig. 3, where $n \geq 2$. Let us show that every NFA for L^* has at least $n+1$ states. We prove this by describing (a) a fooling set for L^*, and (b) a fooling set for an NFA for L^*.

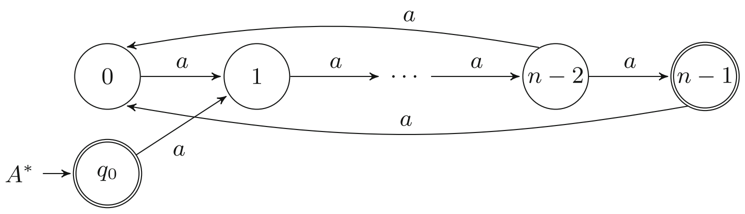


Fig. 4. The NFA A^* for the automaton A from Fig. 3.

(a) Consider the set of pairs of strings

$$\mathcal{F} = \{(\varepsilon, \varepsilon)\} \cup \{(a^i, a^{n-1-i}) \mid 1 \leq i \leq n-2\} \cup \{(a^{n-1}, a^n), (a^n, a^{n-1})\}.$$

We have $\{\varepsilon, a^{n-1}, a^{2n-1}\} \subseteq L^*$. However, if $1 \leq k \leq n-2$, then $a^k \notin L^*$ and $a^{n+k} \notin L^*$. Moreover, $a^n \notin L^*$ and $a^{2n} \notin L^*$. This means that $\mathcal{F}$ is a fooling set for L^*, so every NFA for L^* has at least $n+1$ states.

(b) Construct an NFA A^* for L^* from A by adding the transition $(n-2, a, 0)$, and by adding a new initial state q_0 going to the state 1 on a; see Fig. 4. Set

$$\begin{aligned}
X_0 &= \{q_0\}, & Y_0 &= \{q_0, n-1\}, & \\
X_i &= \{i\}, & Y_i &= \{i\} & \text{for } i = 1, 2, \ldots, n-2, \\
X_{n-1} &= \{0, n-1\}, & Y_{n-1} &= \{n-2, n-1\}, & \\
X_n &= \{0, 1\}, & Y_n &= \{0, q_0\}. &
\end{aligned}$$

Then for each $i = 0, 1, \ldots, n$, the set X_i is reachable and the set Y_i is co-reachable in A^*. Next, if $(i, j) \in \{(0, n), (n-2, n-1), (n-1, n)\}$, then $X_j \cap Y_i = \emptyset$, otherwise, $X_i \cap Y_j = \emptyset$ if $i < j$. It follows that the set $\{(X_i, Y_i) \mid i = 0, 1, \ldots, n\}$ is a fooling set for A^*, so every NNFA for L^* has at least $n+1$ states. □

4 Simplifications of the Fooling Set Method

In this section, we state some sufficient conditions on an NNFA that guarantee its minimality. Having such an NNFA, there is no need to describe a fooling set for the accepted language since we know that every equivalent NNFA has at least as many states as the given NNFA.

Lemma 2. *Let $A = (Q, \Sigma, \cdot, I, F)$ be an* NNFA*. Suppose that, for each state q in Q, the one-element set $\{q\}$ be reachable as well as co-reachable in A. Then every* NNFA *for $L(A)$ has at least $|Q|$ states.*

Proof. Since $\{q\}$ is reachable in A, there is a string x_q such that $I \cdot x_q = \{q\}$. Since $\{q\}$ is co-reachable in A, there is a string y_q accepted by A from and only from the state q. Then $\{(x_q, y_q) \mid q \in Q\}$ is a fooling set for the language $L(A)$. By Lemma 1, every NNFA for $L(A)$ has at least $|Q|$ states. □

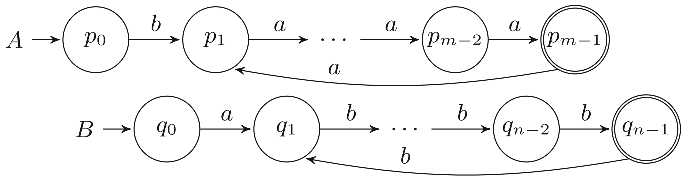


Fig. 5. Binary suffix-free witnesses for union meeting the upper bound $m + n - 1$.

Notice that if A is a trim partial DFA, then for each state q of A, the singleton set $\{q\}$ is reachable. If moreover A^R is a partial DFA, then $\{q\}$ is co-reachable in A. So we get the following result.

Lemma 3. *Let A be an n-state trim* NFA*. If both A and A^R are partial* DFAs*, then every* NNFA *for $L(A)$ has at least n states.* □

Let us show how Lemma 2 can be used to get the nondeterministic state complexity of the union operation on suffix-free languages. Recall that if two NFAs A and B accept suffix-free languages, then we may assume that their initial states do not have any in-transitions [8,21] This means that we can merge the initial states to get an NFA for $L(A) \cup L(B)$. This gives an upper bound of $m + n - 1$. In the next example, we use Lemma 2 to prove the tightness of this upper bound.

Example 4 (Union on suffix-free languages; cf. [13, Theorem 9]*).* Consider the NFAs A and B shown in Fig. 5; notice that the languages $L(A)$ and $L(B)$ are suffix-free. Construct an NFA for $L(A) \cup L(B)$ by merging the initial states of A and B; see Fig. 6. For each state q of the resulting NFA, the set $\{q\}$ is reachable, as well as co-reachable. By Lemma 2, every NNFA for $L(A) \cup L(B)$ has at least $m + n - 1$ states. □

Now, we use Lemma 3 to get the nondeterministic complexity of intersection on regular languages. The upper bound is mn since the product automaton $A \times B = (Q_A \times Q_B, \Sigma, \cdot, (s_A, s_B), F_A \times F_B)$, where $(p, q) \cdot a = (p \cdot_A a) \times (q \cdot_B a)$, recognizes $L(A) \cap L(B)$. The next example provides binary witness languages.

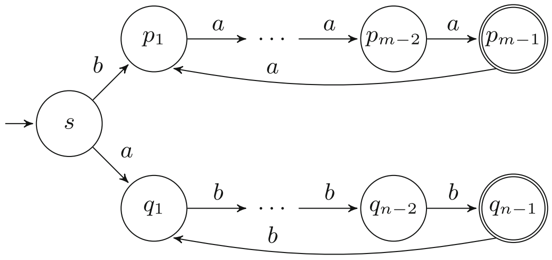


Fig. 6. The NFA for the union of languages from Fig. 5.

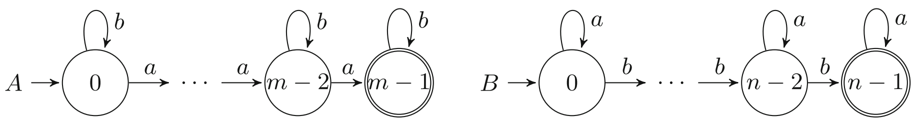


Fig. 7. Binary witness languages for intersection meeting the upper bound mn.

Example 5 (Intersection on regular languages; cf. [10, Theorem 3]*).* Consider the partial DFAs A and B shown in Fig. 7. The product automaton $A \times B$ for $L(A) \cap L(B)$ is a trim partial DFA, and its reverse is a partial DFA as well; see Fig. 8 for $m = 3$ and $n = 4$. By Lemma 3, every NNFA for $L(A) \cap L(B)$ has at least mn states. □

5 Modification of the Fooling Set Method

The fooling set method provides a lower bound on the number of states in any NNFA, that is, in any nondeterministic finite automaton with, possibly, multiple initial states. However, sometimes the NFA with a unique initial state must have one additional state. This is true for the case of union, where for every pair of languages K and L accepted by an m-state and n-state NFA, respectively, there exists an NNFA of size $m + n$ accepting $K \cup L$, hence no fooling set for $K \cup L$ can be of size more than $m + n$. Similarly, no fooling set for L^R can be of size more than n.

The idea for getting lower bounds of $m + n + 1$ for union and of $n + 1$ for reversal, is to divide a fooling set into two parts $\mathcal{A}$ and $\mathcal{B}$ and to find pairs (ε, u) and (ε, v) such that $\mathcal{A} \cup \{(\varepsilon, u)\}$ and $\mathcal{B} \cup \{(\varepsilon, v)\}$ are fooling sets. This implies that a unique initial state, reached after reading the empty string, must be different from all states given by fooling set $\mathcal{A} \cup \mathcal{B}$.

Lemma 4 (Jirásková and Masopust [17, Lemma 4]**).** *Let $\mathcal{A}$ and $\mathcal{B}$ be disjoint sets of pairs of strings and let u and v be two strings such that $\mathcal{A} \cup \mathcal{B}$, $\mathcal{A} \cup \{(\varepsilon, u)\}$, and $\mathcal{B} \cup \{(\varepsilon, v)\}$ are fooling sets for a language L. Then every* NFA *for L has at least $|\mathcal{A}| + |\mathcal{B}| + 1$ states.*

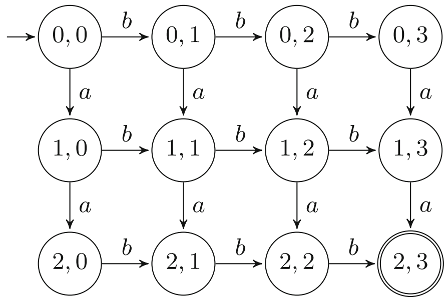


Fig. 8. The product automaton for the intersection of languages from Fig. 7, for $m = 3$ and $n = 4$.

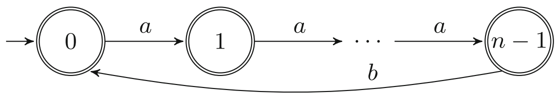


Fig. 9. A binary witness language for reversal meeting the upper bound $n + 1$.

Proof. Let A be an NFA for L with the initial state s. Let

$$\mathcal{A} = \{(x_i, y_i) \mid i = 1, 2, \dots, m\}, \text{ and}$$
$$\mathcal{B} = \{(x_{m+j}, y_{m+j}) \mid j = 1, 2, \dots, n\}.$$

Since each string $x_k y_k$ is in L, we can fix an accepting computation of A on $x_k y_k$, and let p_k be the state on this computation reached after reading x_k. Since $\mathcal{A} \cup \mathcal{B}$ is a fooling set for L, the states $p_1, p_2, \dots, p_{m+n}$ are pairwise distinct, as shown in the proof of Lemma 1. Since $\mathcal{A} \cup \{(\varepsilon, u)\}$ is a fooling set, the initial state s is distinct from all the states $p_1, p_2, \dots, p_m$. Since $\mathcal{B} \cup \{(\varepsilon, v)\}$ is a fooling set, the initial state s is also distinct from all the states $p_{m+1}, p_{m+2}, \dots, p_{m+n}$. Thus the NFA A has at least $m + n + 1$ states. □

It is shown in [16, Theorem 2] that there is a *binary* regular language L accepted by an n-state NFA such that every NFA for L^R has at least $n + 1$ states. An NFA for the language is shown in Fig. 9, and the proof in [16] is by a counting argument. In the next example we use Lemma 4 to get the lower bound.

Example 6 (Reversal on regular languages; cf. [16, Theorem 2]*).* Let L be the binary language accepted by the partial DFA shown in Fig. 9. Set

$$\begin{aligned}
\mathcal{A} &= \{(ba^i, a^{n-1-i}) \mid i = 0, 1, \dots, n-2\},\\
\mathcal{B} &= \{(ba^{n-1}, \varepsilon)\},\\
u &= \varepsilon,\\
v &= a.
\end{aligned}$$

Notice that we have $\{ba^{n-1}, \varepsilon, a\} \subseteq L^R$. On the other hand, if $k \neq n - 1$, then the string ba^k is not in L^R. It follows that $\mathcal{A} \cup \mathcal{B}$, $\mathcal{A} \cup \{(\varepsilon, u)\}$, and $\mathcal{B} \cup \{(\varepsilon, v)\}$ are fooling sets for the language L^R. By Lemma 4, every NFA for L^R has at least $n + 1$ states. Since $n + 1$ is also an upper bound on the nondeterministic state complexity of the reversal operation, we get $\operatorname{nsc}(L^R) = n + 1$. □

The following two examples use Lemma 4 to get the nondeterministic state complexity of the union operation on regular and prefix-free languages.

Example 7 (Union on regular languages; cf. [10, Theorem 1]*).* Let $K = \{a^m\}^*$ and $L = \{b^n\}^*$ where $m, n \geq 1$. The languages K and L are accepted by the m-state and n-state NFA A and B, respectively, shown in Fig. 10. It is shown in [10, Theorem 1] that every NFA for the language $K \cup L$ has at least $m + n + 1$

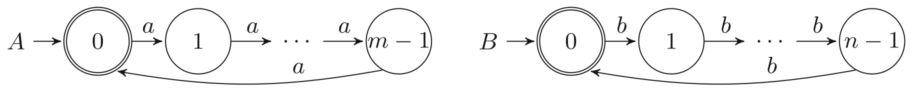


Fig. 10. Binary witness languages for union meeting the upper bound $m+n+1$.

states, and the proof is almost one page long. Let us again use Lemma 4. To this end, let

$$\begin{aligned}\mathcal{A} &= \{(a^i, a^{m-i}) \mid i = 1, 2, \ldots, m-1\} \cup \{(a^m, a^m)\},\\ \mathcal{B} &= \{(b^j, b^{n-j}) \mid j = 1, 2, \ldots, n-1\} \cup \{(b^n, b^n)\},\\ u &= b^n,\\ v &= a^m.\end{aligned}$$

We have $\{a^m, a^{2m}, b^n, b^{2n}\} \subseteq K \cup L$. On the other hand, if $k \not\equiv 0 \bmod m$, then $a^k \notin K \cup L$, and similarly, if $\ell \not\equiv 0 \bmod n$, then $b^\ell \notin K \cup L$. Moreover, no string in $K \cup L$ contains both a and b. It follows that $\mathcal{A} \cup \mathcal{B}$, $\mathcal{A} \cup \{(\varepsilon, u)\}$, and $\mathcal{B} \cup \{(\varepsilon, v)\}$ are fooling sets for the language $K \cup L$. By Lemma 4, every NFA for $K \cup L$ has at least $m+n+1$ states. □

Example 8 (Union on prefix-free languages; cf. [13, Theorem 9]*).* A minimal NFA for a prefix-free language has exactly one final state with no out-transitions. We can construct an NNFA for the union of prefix-free languages given by minimal NFAs A and B (with disjoint states sets) just by merging their final states. This gives $m+n-1$ states in the resulting NNFA. Therefore $m+n$ is an upper bound on the nondeterministic complexity of the union of prefix-free languages.

In [9] it is claimed that the upper bound $m+n$ is met by the union of the prefix-free languages $K = (a^{m-1})^* b$ and $L = (c^{n-1})^* d$, and a set P of pairs of strings of size $m+n$ is described in [9, Proof of Theorem 3.2]. The authors claim that the set P is a fooling set for $K \cup L$. However, as shown above, the language $K \cup L$ is accepted by an NNFA of $m+n-1$ states. Therefore P cannot be a fooling set for $K \cup L$; indeed, the pairs $(\varepsilon, a^{m-1}b)$ and (a^{m-1}, b) do not satisfy the second condition in Definition 1.

Here we prove the tightness of the upper bound $m+n$ for the union of prefix-free languages using a binary alphabet and Lemma 4. Let $m, n \geq 3$ and consider binary languages K and L accepted by the m-state and n-state NFA A and B, respectively, shown in Fig. 11. Notice that K is prefix-free since every string in K ends with b while every proper prefix of every string in K is in a^*. Now, using Lemma 4, we show that every NFA for $K \cup L$ has at least $m+n$ states.

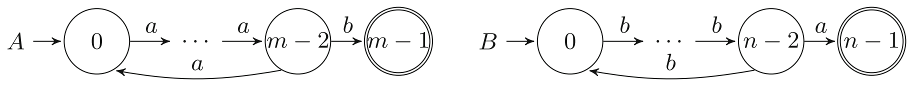


Fig. 11. Binary prefix-free witnesses for union meeting the upper bound $m + n$.

To this end, let

$$\begin{aligned}\mathcal{A} &= \{(a^{m-1+i}, a^{m-2-i}b) \mid 0 \le i \le m-2\} \cup \{(a^{m-2}b, \varepsilon)\},\\ \mathcal{B} &= \{(b^{n-1+j}, b^{n-2-j}a) \mid 0 \le j \le n-2\},\\ u &= b^{n-2}a,\\ v &= a^{m-2}b.\end{aligned}$$

We have $\{a^{2m-3}b, a^{m-2}b, b^{2n-3}a, b^{n-2}a\} \subseteq K \cup L$. Next, every string in the language $K \cup L$ starting with a has $(m-2) \bmod (m-1)$ consecutive a's followed by b, and every string starting with b has $(n-2) \bmod (n-1)$ consecutive b's followed by a. This means that the sets $\mathcal{A} \cup \mathcal{B}$, $\mathcal{A} \cup \{(\varepsilon, u)\}$ and $\mathcal{B} \cup \{(\varepsilon, v)\}$ are fooling sets for $K \cup L$. By Lemma 4, every NFA for $K \cup L$ has at least $m + n$ states. □

Finally, we use Lemma 4 to show that the upper bound $m + n + 1$ for the union of regular languages can be met by binary subword-closed languages.

Example 9 (Union on subword-closed languages; [12, Theorem 4]*).* Let $m, n \ge 1$ and $K = \{w \in \{a,b\}^* \mid |w|_a \le m-1\}$ and $L = \{w \in \{a,b\}^* \mid |w|_b \le n-1\}$ be the binary subword-closed languages accepted by the m-state and n-state partial DFA A and B, respectively, shown in Fig. 12. Let

$$\begin{aligned}\mathcal{A} &= \{(b^n a^i, a^{m-1-i}) \mid 0 \le i \le m-1\},\\ \mathcal{B} &= \{(b^{n-1-j}, b^j a^m) \mid 0 \le j \le n-1\},\\ u &= a^m b^{n-1},\\ v &= a^{m-1} b^n.\end{aligned}$$

Each string w with $|w|_a = m-1$ or $|w|_b = n-1$ is in $K \cup L$, while no string with $|w|_a \ge m$ and $|w|_b \ge n$ is in $K \cup L$. It follows that $\mathcal{A} \cup \mathcal{B}$, $\mathcal{A} \cup \{(\varepsilon, u)\}$, and $\mathcal{B} \cup \{(\varepsilon, v)\}$ are fooling sets for $K \cup L$. By Lemma 4, every NFA for $K \cup L$ has at least $m + n + 1$ states. □

6 Fooling Sets for Complementation

The next very useful observation allows us to significantly simplify the proofs of lower bounds on the nondeterministic state complexity of complementation.

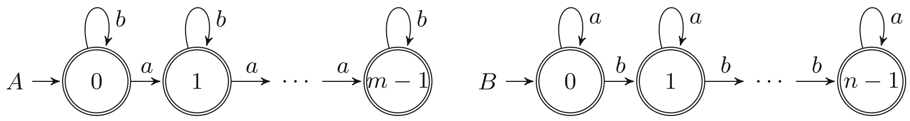


Fig. 12. Binary subword-closed witnesses for union meeting the bound $m + n + 1$.

Lemma 5. *Let A be an n-state* NNFA *in which each subset of the state set is reachable as well as co-reachable. Then every* NNFA *for the complement of the language $L(A)$ has at least 2^n states.*

Proof. Let $A = (Q, \Sigma, \cdot, I, F)$ be an NNFA. Let $S \subseteq Q$. Since S is reachable, there exists a string x_S in Σ^* such that $S = I \cdot x_S$. Since the set S^c is co-reachable, there is a string y_S which is accepted by A from each state in S^c, but rejected from each state in S. It follows that the set $\{(x_S, y_S) \mid S \subseteq Q\}$ is a fooling set for $(L(A))^c$. Hence every NNFA for $(L(A))^c$ has at least 2^n states. $\square$

Recall that to get an NFA (even DFA) for the complement of a language represented by an NFA A, we first apply the subset construction to the NFA A, and in the resulting DFA, we interchange the accepting and rejecting states. This gives an upper bound of 2^n on the nondeterministic state complexity of complementation. In the next example we use Lemma 5 to show that the complement of the binary language from [16, Proof of Theorem 5] meets this upper bound. Notice that a rather complicated fooling set is described in [16], and the proof is almost three pages long. Moreover, the NFA for this binary witness and its reverse are isomorphic, so we only need to show reachability of all subsets.

Example 10 (Complementation on regular languages [16, Theorem 5]*).* Consider the n-state NFA A shown in Fig. 13, where $i \cdot a = \{i+1\}$ and $i \cdot b = \{0, i+1\}$ if $0 \le i \le n-2$, $(n-1) \cdot b = \{1, 2, \ldots, n-1\}$, and all the remaining transitions go to the empty set. First, notice that the automata A and A^R are isomorphic. Hence to prove that every NFA for $(L(A))^c$ has at least 2^n states, we only need to show that every subset of $\{0, 1, \ldots, n-1\}$ is reachable in A. Since every subset S with $0 \notin S$ can be reached by $a^{\min S}$ from the set $\{s - \min S \mid s \in S\}$ which contains state 0, we only need to show the reachability of subsets containing 0. The proof is by induction on the size of subsets. The set $\{0\}$ is the initial subset, and every

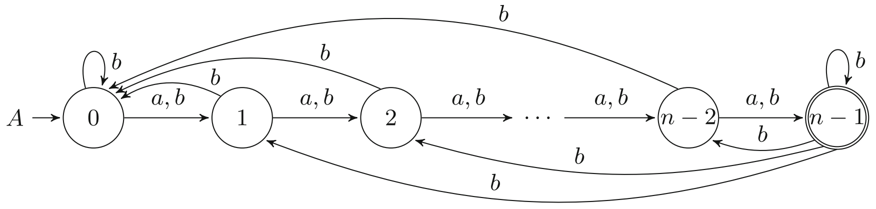


Fig. 13. A binary witness for complementation meeting the upper bound 2^n.

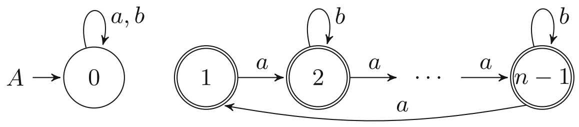


Fig. 14. Transitions on a and b in a suffix-convex witness for complementation.

subset $\{0, i_2, \ldots, i_k\}$ of size k, where $1 \le i_2 < \cdots < i_k \le n-1$, is reachable from the set $\{0, i_3 - i_2, \ldots, i_k - i_2\}$ of size $k-1$ by the string ab^{i_2-1}. □

Finally, we prove that the upper bound 2^n can be met by the complement of a suffix-convex language. Let us emphasize that such a language must be so-called *proper suffix-convex*, that is, it can be neither suffix-free, nor suffix-closed, nor left ideal since it is proved in [18, Lemma 4, Theorem 2], [12, Theorem 10], and [20, Theorem 26], respectively, that the nondeterministic complexity of complementation is less than 2^n in these three subclasses of convex languages.

Example 11 (Complementation on suffix-convex languages; [13, Theorem 13]*).* Let $n \ge 3$. Let L be the regular language accepted by the nondeterministic finite automaton $A = (\{0, 1, \ldots, n-1\}, \{a, b, c, d, e\}, 0, \cdot, \{1, 2, \ldots, n-1\})$ where the transitions on a and b are shown in Fig. 14, the transitions on c, d, e are as follows:

$$\begin{aligned} 0 \cdot c &= \{0, 1, \ldots, n-1\}, \\ 0 \cdot d &= \{1, 2, \ldots, n-1\}, \\ q \cdot e &= \{n-1\}, \text{ for each state } q \text{ of } A, \end{aligned}$$

and all the remaining transitions go to the empty set. In the NFA A^R, the final state 0 loops on a, b, c and goes to the empty set on d and e. Next, every other state of A^R goes to 0 on d, and the state $n-1$ goes to $\{0, 1, \ldots, n-1\}$ on e. Thus in the subset automaton of A^R, each final subset, that is, a subset containing the state 0, goes either to a final subset containing 0 or to the empty set on each input symbol. It follows that the language L^R is prefix-convex, so L is suffix-convex. Now we show that each subset of the state set of A is reachable and co-reachable in A. Notice that $\{0\} \cdot a = \{0\}, \{0\} \cdot b = \{0\}$, $0 \cdot c = \{0, 1, \ldots, n-1\}$, and $0 \cdot d = \{1, 2, \ldots, n-1\}$.

Moreover, we can shift each subset of $\{1, 2, \ldots, n-1\}$ cyclically by one using the symbol a, that is, we have $S \cdot a = \{(s+1) \bmod n \mid s \in S\}$. Next, we can eliminate the state 1 from each subset containing 1 by reading the symbol b. It follows that each subset is reachable. To prove co-reachability, notice that the initial subset of A^R is $\{1, 2, \ldots, n-1\}$ and it goes to $\{0, 1, \ldots, n-1\}$ on e. We again use symbol a to shift the subsets of $\{1, 2, \ldots, n-1\}$ and symbol b to eliminate the state 1. It follows that every subset is co-reachable. By Lemma 5, every NNFA for L^c has at least 2^n states. □

7 Conclusions

The fooling set method provides lower bounds on the number of states in nondeterministic finite automata that are tight in many cases despite the fact that the gap between the size of a fooling set and the nondeterministic state complexity may be arbitrarily large. We illustrated this on a number of examples.

We also provided sufficient conditions on nondeterministic finite automata that guarantee the existence of appropriate fooling sets. This allowed us to avoid the tedious description of such fooling sets.

Since fooling sets provide lower bounds on the number of states in nondeterministic finite automata with multiple initial states, in the case of union or reversal, where such automata may save one state, the fooling set method cannot be used. However, if a fooling set can be divided into two parts such that adding a pair with its left component equal to the empty string results in another fooling set, we get tight lower bounds also for automata with a unique initial state.

Finally, we provided a very useful observation from [13, Proof of Theorem 13] claiming that if all subsets of the state set of a nondeterministic finite automaton are reachable and co-reachable, then the accepted language is a witness for the complementation operation

References

1. Aho, A.V., Ullman, J.D., Yannakakis, M.: On notions of information transfer in VLSI circuits. In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, STOC 1983, pp. 133–139. ACM, New York (1983)
2. Birget, J.: Intersection and union of regular languages and state complexity. Inf. Process. Lett. **43**(4), 185–190 (1992)
3. Birget, J.: Partial orders on words, minimal elements of regular languages and state complexity. Theor. Comput. Sci. **119**(2), 267–291 (1993)
4. Brent, R.P., Kung, H.T.: The chip complexity of binary arithmetic. In: Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, STOC 1980, pp. 190–200. ACM, New York (1980)
5. Brzozowski, J., Jirásková, G., Liu, B., Rajasekaran, A., Szykuła, M.: On the state complexity of the shuffle of regular languages. In: Câmpeanu, C., Manea, F., Shallit, J. (eds.) DCFS 2016. LNCS, vol. 9777, pp. 73–86. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41114-9_6
6. Glaister, I., Shallit, J.: A lower bound technique for the size of nondeterministic finite automata. Inf. Process. Lett. **59**(2), 75–77 (1996)
7. Gruber, H., Holzer, M.: Finding lower bounds for nondeterministic state complexity is hard. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 363–374. Springer, Heidelberg (2006). https://doi.org/10.1007/11779148_33
8. Han, Y., Salomaa, K.: Nondeterministic state complexity for suffix-free regular languages. In: McQuillan, I., Pighizzini, G. (eds.) Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS 2010. EPTCS, vol. 31, pp. 189–196 (2010)
9. Han, Y., Salomaa, K., Wood, D.: Nondeterministic state complexity of basic operations for prefix-free regular languages. Fundam. Inform. **90**(1–2), 93–106 (2009)

10. Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. Int. J. Found. Comput. Sci. **14**(6), 1087–1102 (2003)
11. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Boston (1979)
12. Hospodár, M., Jirásková, G., Mlynárčik, P.: Nondeterministic complexity of operations on closed and ideal languages. In: Han, Y.-S., Salomaa, K. (eds.) CIAA 2016. LNCS, vol. 9705, pp. 125–137. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40946-7_11
13. Hospodár, M., Jirásková, G., Mlynárčik, P.: Nondeterministic complexity of operations on free and convex languages. In: Carayol, A., Nicaud, C. (eds.) CIAA 2017. LNCS, vol. 10329, pp. 138–150. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60134-2_12
14. Hromkovič, J.: Some complexity aspects of VLSI computations, part 1. Comput. Artif. Intell. **7**(3), 229–252 (1988)
15. Jirásková, G.: Comparison of two VLSI models. Comput. Artif. Intell. **10**, 121–232 (1991)
16. Jirásková, G.: State complexity of some operations on binary regular languages. Theor. Comput. Sci. **330**(2), 287–298 (2005)
17. Jirásková, G., Masopust, T.: Complexity in union-free regular languages. Int. J. Found. Comput. Sci. **22**(7), 1639–1653 (2011)
18. Jirásková, G., Mlynárčik, P.: Complement on prefix-free, suffix-free, and non-returning NFA languages. In: Jürgensen, H., Karhumäki, J., Okhotin, A. (eds.) DCFS 2014. LNCS, vol. 8614, pp. 222–233. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09704-6_20
19. Lipton, R.J., Sedgewick, R.: Lower bounds for VLSI. In: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, STOC 1981, pp. 300–307. ACM, New York, NY, USA (1981)
20. Mlynárčik, P.: Complement on free and ideal languages. In: Shallit, J., Okhotin, A. (eds.) DCFS 2015. LNCS, vol. 9118, pp. 185–196. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19225-3_16
21. Mlynárčik, P.: Nondeterministic state complexity in subregular classes. Dissertation thesis. FMFI UK, Bratislava (2017). http://im.saske.sk/~jiraskov/students/phd_thesis_mlynarcik.pdf
22. Savage, J.E.: Planar circuit complexity and the performance of VLSI algorithms +. In: Kung, H.T., Sproull, B., Steele, G. (eds.) VLSI Systems and Computations, pp. 61–68. Springer, Heidelberg (1981). https://doi.org/10.1007/978-3-642-68402-9_8
23. Sipser, M.: Introduction to the theory of computation. Cengage Learning (2012)
24. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. I, 1st edn. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59136-5_2

Optimal 2DFA Algorithms for One-Way Liveness on Two and Three Symbols

Christos A. Kapoutsis(✉)

Carnegie Mellon University in Qatar, Doha, Qatar
cak@cmu.edu

Abstract. For all $h, n \geq 2$, the problem of *one-way liveness* for height h and length n captures the task of simulating *one-way nondeterministic finite automata* with h states on inputs with $n-2$ symbols. We prove that the number of states in a smallest *two-way deterministic finite automaton* which decides this problem is $\Theta(h)$, if $n = 2$; and $\Theta(h^2/\log h)$, if $n = 3$.

1 Introduction

A long-standing open question at the intersection of automata theory and computational complexity is whether every *two-way nondeterministic finite automaton* (2NFA) with s states can be simulated by a *two-way deterministic finite automaton* (2DFA) with a number of states which is only polynomial in s. The well-known *Sakoda-Sipser conjecture* proposes that the answer is negative [5].

A stronger variant of this conjecture claims that, indeed, a 2DFA needs super-polynomially many states even when the input head of the 2NFA can only move forward, namely even when the 2NFA is really a *one-way nondeterministic finite automaton*. By [5], we know that this stronger claim holds iff a certain graph-theoretic problem called *one-way liveness* for height h (OWL_h) cannot be solved by any 2DFA whose number of states is only polynomial in h.

In one of the many approaches to this question, Hromkovič et al. [3] proposed the concept of a *reasonable automaton* (RA). Intuitively, this is a 2DFA with the following modifications: it works only on inputs of a fixed length n; it is *random-access*, in the sense that its head can move from any cell of its n-long input to any other in a single step; and, most importantly, it has every state associated with a propositional formula which indicates what the automaton 'knows' about its input when it is in that state. In this model, one uses the additional information provided by the formulas to study the different types of propositional reasoning that a 2DFA could possibly employ in trying to solve OWL_h on inputs of length n. Clearly, the standard 2DFA model would be too lean to support such a study.

Not surprisingly, the power of a RA varies as we vary (i) the set of propositional variables that can appear in the formulas and (ii) the rules for how these variables can be combined to create the formulas. By [3, Theorem 2], we know that a sufficiently expressive set of variables allows a RA to simulate any s-state 2DFA on

n-long inputs with only $O(sn)$ states; so, the model is powerful enough to represent any 2DFA algorithm for OWL$_h$ on inputs of fixed length, and remains small if so do both the 2DFA and the length. We also know that certain options for (i) and (ii) make it necessary for a RA solving OWL$_h$ to use $2^{\Omega(h)}$ states, even when the length is restricted to just $n = 2$ symbols [3, Theorems 3 and 4]; while other options make $O(h)$ states already sufficient for $n = 2$ symbols [6], and $O(h^2)$ states already sufficient for $n = 3$ and $n = 4$ symbols [3, Theorems 6 and 7].

For example, the linear upper bound for two symbols from [6] was proved for the case where (i) there is one propositional variable $e_{a,b}$ for every two vertices a, b of the input graph, indicating whether an edge between a and b exists; and (ii) all rules of propositional-formula formation are allowed. That is, if 2OWL$_h$ denotes the restriction of OWL$_h$ to instances of length exactly 2, then $O(h)$ states are enough for solving 2OWL$_h$ by a RA which builds its formulas by arbitrarily applying the logical connectives $\wedge, \vee, \neg$ to the variables $e_{a,b}$. In fact, this upper bound is also known to be optimal in the specific case, as Bianchi, Hromkovič, and Kováč [1, Theorem 1] recently proved that such RAs for 2OWL$_h$ also need $\Omega(h)$ states. Overall, we arrive at the nice conclusion that, *with such variables and rules, every smallest* RA for 2OWL$_h$ *has* $\Theta(h)$ *states.*

By inspecting the proof of [1, Theorem 1] for the above linear lower bound, one can observe that it is actually valid not only for RAs with the particular variables and rules, but also for all possible RAs. In fact, minor technical changes make that proof valid even for arbitrary 2DFAs. That is, every 2DFA that solves 2OWL$_h$ needs $\Omega(h)$ states, even if it is not reasonable. At the same time, one easily realizes that the RA that proves the matching upper bound in [6] implies directly that a 2DFA, too, can solve 2OWL$_h$ with $O(h)$ states. Overall, we actually know the much stronger fact that *every smallest* 2DFA *for* 2OWL$_h$ *has* $\Theta(h)$ *states.*

At this point, it is interesting to ask what the corresponding fact is for OWL$_h$ on three symbols, or four, or five, and so on. In general, for $n \geq 2$, we let nOWL$_h$ denote the restriction of OWL$_h$ to instances of exactly n symbols, and ask:

How many states are there in a smallest 2DFA *for* nOWL$_h$*?* (1)

For $n = 2$, the *asymptotic* answer to this question is, of course, provided by our discussion above. Note, however, that we are still missing the *exact* answer, namely a function $s(h)$ such that some 2DFA for 2OWL$_h$ has at most $s(h)$ states and no 2DFA for 2OWL$_h$ has strictly fewer than $s(h)$ states.

For $n \geq 3$, we know neither the asymptotic nor the exact answer to (1). We only have the asympotic upper bounds implied by the RAs of [3, Theorems 6 and 7] which implement Savitch's algorithm on n symbols and are easily converted into 2DFAs with (n times more states, and thus with) the same asymptotic size (if n is constant). For the cases $n = 3$ and $n = 4$, those bounds are both quadratic, so we know that *every smallest* 2DFA *for* 3OWL$_h$ *or for* 4OWL$_h$ *has* $O(h^2)$ *states.*

In this paper, we study (1) for the cases $n = 2$ and $n = 3$. Our main contribution is the asymptotic answer for $n = 3$ (Sect. 4):

Theorem 1. *Every smallest* 2DFA *for* 3OWL *has* $\Theta(h^2/\log h)$ *states.*

This involves a new algorithm for the upper bound (Lemma 8) and a standard argument for the lower bound (Lemma 7). Before that, we also take the time to carefully examine the case $n = 2$ and the known asymptotic answer (Sect. 3):

Theorem 2. *Every smallest* 2DFA *for* 2OWL *has $\Theta(h)$ states.*

We give a detailed argument for the lower bound (Lemma 6), which mimics that of [1, Theorem 1] but applies to any 2DFA and results in a higher exact value. For completeness, we also give the known algorithm for the upper bound (Lemma 5).

In both cases, we put the extra effort to find exact values for our bounds, so that one can appreciate the gap, between lower and upper bound, where the actual size of the smallest 2DFA lies. For example, in the case $n = 2$, one sees that the $\Theta(h)$ size of the best 2DFA is actually somewhere above $\frac{1}{2}h + \frac{1}{4}\lg h - \frac{1}{2}$ and below $2h$. We also put the effort to make our intermediate lemmata as general as possible, even if this full generality is not strictly needed in our proofs. For example, the Hybrid Rule (Lemma 2) is proved for all cases, even for when the computation on the hybrid string is looping, although we only use that rule once (Lemma 6), in a case where that computation is (accepting, and thus) halting.

2 Preparation

If S is a set, then $|S|$, $\overline{S}$, and $\mathbb{P}(S)$ are respectively its size, complement, and powerset. If Σ is an alphabet, then Σ^* is the set of all strings over it and Σ^n is its subset containing only the strings of length exactly n. If $z \in \Sigma^*$ is a string, then $|z|$ and z_j are respectively its length and its j-th symbol (if $1 \le j \le |z|$). If $n \ge 0$, then $[n] := \{1, 2, \ldots, n\}$ is the set of the n smallest positive integers.

Problems. A (*promise*) *problem* over Σ is any pair $\mathfrak{L} = (L, \tilde{L})$ of disjoint subsets of Σ^*. Its *positive instances* are all $w \in L$, whereas its *negative instances* are all $w \in \tilde{L}$. A machine *solves* $\mathfrak{L}$ if it accepts every positive instance but no negative one. If $\tilde{L} = \overline{L}$, then we call $\mathfrak{L}$ a *language* and represent it only by L.

Let $h \ge 2$.[1] The alphabet $\Sigma_h := \mathbb{P}([h] \times [h])$ consists of all two-column directed graphs with h nodes per column and only rightward arrows (Fig. 1a). A string $w \in \Sigma_h^n$ is naturally viewed as an $(n+1)$-column graph, where every arrow connects successive columns (Fig. 1b, c); we usually index the columns from 0 to n and, for simplicity, drop the directions of the arrows. If w contains a path from the leftmost to the rightmost column (called a *live* path), then we say that w is *live*; otherwise we say that w is *dead*. The language

$$\text{OWL}_h := \{\ w \in \Sigma_h^* \ \mid\ w \text{ is live}\ \}$$

represents the computational task of checking that a given string in Σ_h^* contains a live path [5]. If the string is guaranteed to be of a fixed length n, then the task is best represented by the promise problem

$$n\text{OWL}_h := \big(\ \{w \in \Sigma_h^n \mid w \text{ is live}\},\ \{w \in \Sigma_h^n \mid w \text{ is dead}\}\ \big)\,.$$

[1] Here, we exclude the trivial case of height $h = 1$, so that we can divide by $\lg h \neq 0$.

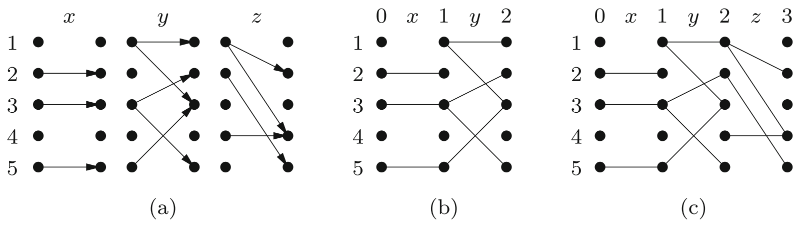


Fig. 1. (a) Three symbols $x, y, z \in \Sigma_5$; e.g., $z = \{(1,2),(1,4),(2,5),(4,4)\}$. (b) The string xy of the first two symbols, simplified and indexed, as an instance of 2OWL$_5$. (c) The string xyz of all symbols, simplified and indexed, as an instance of 3OWL$_5$.

Then nOWL := $(n\text{OWL}_h)_{h\geq 2}$ is the family of all such promise problems for $h \geq 2$.

Machines. A *two-way deterministic finite automaton* (2DFA) is any tuple of the form $M = (S, \Sigma, \delta, q_s, q_a)$, where S is a set of *states*, Σ is an *alphabet*, $q_s, q_a \in S$ are respectively the *start* and *accept* states, and $\delta : S \times (\Sigma \cup \{\vdash,\dashv\}) \rightharpoonup S \times \{\text{L},\text{R}\}$ is a (partial) *transition function*, for $\vdash,\dashv \notin \Sigma$ the left and right endmarkers, and L,R the left and right directions.

An input $w \in \Sigma^*$ is presented to M surrounded by the endmarkers, as $\vdash w \dashv$. The computation starts at q_s and on $\vdash$. In each step, the next state and head move (if any) are derived from δ and the current state and symbol. Endmarkers are never violated, except if the next state is q_a; that is, $\delta(\,.\,,\vdash)$ is always (q_a, L) or of the form $(\,.\,,\text{R})$; and $\delta(\,.\,,\dashv)$ is always (q_a, R) or of the form $(\,.\,,\text{L})$. Hence, the computation either *loops*, if it ever repeats a state on the same input cell; or *hangs*, if it ever reaches a state and symbol for which δ is undefined; or falls off $\vdash$ or $\dashv$ into q_a, in which case we say that M *accepts* w.

Formally, for any string z, position i, and state q, the *computation of M when started at q on the i-th symbol of z* is the unique sequence

$$\text{COMP}_{M,q,i}(z) = \big((q_t, i_t)\big)_{0\leq t<m}$$

where $(q_0, i_0) = (q, i)$, $1 \leq m \leq \infty$, every pair is derived from its predecessor via δ and z, every pair is within z ($1 \leq i_t \leq |z|$) except possibly for the last one, and the last pair is within z iff δ is undefined on the corresponding state and symbol. We say m is the *length* of this computation. If $m = \infty$, then the computation *loops*. Otherwise, it *hits left into* q_{m-1}, if $i_{m-1} = 0$; or *hangs*, if $1 \leq i_{m-1} \leq |z|$; or *hits right into* q_{m-1}, if $i_{m-1} = |z|+1$ (Fig. 2). When $i = 1$ (respectively, $i = |z|$) we get the *left* (right) *computation of M from q* on z:

$$\text{LCOMP}_{M,q}(z) := \text{COMP}_{M,q,1}(z) \quad \text{and} \quad \text{RCOMP}_{M,q}(z) := \text{COMP}_{M,q,|z|}(z)\,.$$

The (*full*) *computation of M on z* is the typical $\text{COMP}_M(z) := \text{LCOMP}_{M,q_s}(\vdash z \dashv)$, so that M accepts z iff $\text{COMP}_M(z)$ hits right or left (into q_a).

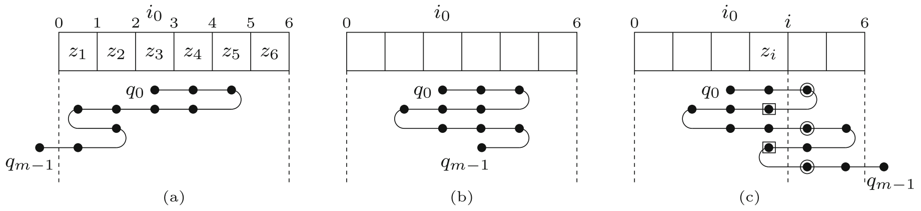


Fig. 2. (a) Cells and boundaries on a 6-long z; a computation that hits left. (b) One that hangs. (c) One that hits right, and its i-th frontier: R_i^c in circles and L_i^c in boxes.

Frontiers. Consider a computation $c = ((q_t, i_t))_{0 \le t < m}$ over some input z, and the index $0 \le i \le |z|$ of some boundary of z (Fig. 2c). How does c behave over that boundary? One answer to this question is the standard notion of the *i-th crossing sequence of* c [2], namely the sequence $q_1, q_2, q_3, \ldots$ where q_j is the state entered by c right after the j-th time that c crosses that boundary.

Another answer, with much less information, is the *i-th frontier of* c [4], which records only which states are used by the crossings, completely ignoring the order in which they are used. Formally, this is the pair of sets of states (L_i^c, R_i^c), where R_i^c (respectively, L_i^c) consists of every state which is entered by c right after some left-to-right (right-to-left) crossing of the i-th boundary of z:

$$R_i^c := \{q_t \mid 0 \le t < m \ \&\ i_{t-1} = i \ \&\ i_t = i+1\},$$
$$L_i^c := \{q_t \mid 0 \le t < m \ \&\ i_{t-1} = i + 1 \ \&\ i_t = i\}.$$

Here we also assume $i_{-1} = i_0 - 1$, so that, if c starts on the cell right after the boundary ($i_0 = i + 1$), then R_i^c also contains q_0. (This reflects the convention that the initial state q_0 is always the result of an 'invisible' left-to-right step.)

If c is a full computation, then it starts (on $\vdash$, and thus) on the left side of the i-th boundary of z (since $i \ge 0$), and then eventually hangs or loops or accepts. If it hangs or accepts also on the left side of the boundary, then it crosses it from left to right exactly as many times as it crosses it from right to left; hence, R_i^c and L_i^c contain the same number of states. By similar reasoning, if c hangs or accepts on the right side of the boundary, then R_i^c contains one more state than L_i^c. Finally, if c loops, then R_i^c contains either exactly as many states as L_i^c, if c never crosses the boundary or the latest crossing which does not result in a repeated pair (q_t, i_t) is from right to left; or one more state, otherwise. Overall, we conclude that, if c is full, then $|R_i^c|$ is always either $|L_i^c|$ or $|L_i^c|+1$.

With this motivation, we define *a frontier of M* to be any pair (L, R) such that $L, R \subseteq S$ and either $|L| = |R|$ or $|L|+1 = |R|$. In the former case, the frontier is called *balanced*; otherwise, it is called *unbalanced*. Standard counting arguments show that, if M has s states, then it has $\binom{2s}{s}$ balanced and $\binom{2s}{s+1}$ unbalanced frontiers, for a total of $\binom{2s+1}{s+1}$ frontiers overall.

The next lemma (Lemma 1) and rule (Lemma 2) are of independent interest. In this paper, we will use them to prove a lower bound for 2OWL$_h$ (Lemma 6).

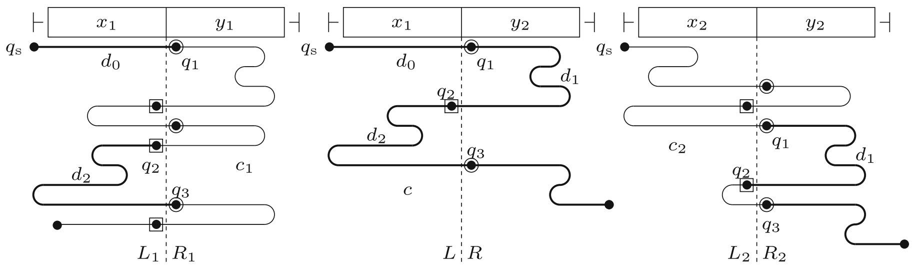


Fig. 3. Computations in the proof of the Hybrid Lemma.

Lemma 1 (Hybrid Lemma). *Let c_1, c_2, c be respectively the full computations of M on strings x_1y_1, x_2y_2 and their hybrid x_1y_2. Let $(L_1, R_1), (L_2, R_2), (L, R)$ be the respective frontiers on the boundaries x_1-y_1, x_2-y_2, and x_1-y_2. Then*

$$L_1 \supseteq L_2 \;\&\; R_1 \subseteq R_2 \implies L_1 \supseteq L_2 \supseteq L \;\&\; R \subseteq R_1 \subseteq R_2 \,.$$

Proof. Suppose $L_1 \supseteq L_2$ and $R_1 \subseteq R_2$. We must prove four inclusions. The first and last of them are just our assumption (included in the statement only for aesthetics), so we just need to prove that $L_2 \supseteq L \,\&\, R \subseteq R_1$.

Let $0 \le k \le \infty$ be the number of times c crosses the x_1-y_2 boundary. Let q_j be the state entered by c right after the j-th crossing, for all finite j with $1 \le j \le k$ (Fig. 3). It suffices to prove the following.

Claim. *All q_j for odd j are in R_1, and all q_j for even j are in L_2.*

Indeed, if the claim holds, then every $q \in R$ is also in R_1, because it is a q_j for some odd j; and every $q \in L$ is also in L_2, because it is a q_j for some even j.

To prove the claim, we first note that it is vacuously true when $k = 0$. So, we assume $1 \le k \le \infty$ and apply induction on j.

In the base case, $j = 1$ and we must prove $q_1 \in R_1$. But q_1 is the result of the first crossing of the x_1-y_2 boundary, so $d_0 := \textsc{lcomp}_{M,q_s}(\vdash x_1)$ hits right into q_1. But d_0 is clearly a prefix of c_1, therefore the first crossing of the x_1-y_1 boundary along c_1 results into q_1, as well. Hence, $q_1 \in R_1$.

In the inductive step, we assume the claim for $j \ge 1$ and prove it for $j{+}1 \le k$.

If j is odd, then $j{+}1$ is even, and thus q_{j+1} is the result of crossing the x_1-y_2 boundary from right to left. In particular, $d_j := \textsc{lcomp}_{M,q_j}(y_2\dashv)$ is an infix of c and hits left into q_{j+1}. We know $q_j \in R_1$ (by the inductive hypothesis), and thus $q_j \in R_2$ (since $R_1 \subseteq R_2$). Therefore, c_2 produces q_j in one of its left-to-right crossings of the x_2-y_2 boundary. Hence, after that crossing, c_2 continues as in $\textsc{lcomp}_{M,q_j}(y_2\dashv)$, namely as in d_j, and thus crosses the boundary again, from right to left and into q_{j+1}. Consequently, $q_{j+1} \in L_2$.

If j is even, we work symmetrically. Since $j{+}1$ is odd and ≥ 3, q_{j+1} results from a left-to-right crossing of the x_1-y_2 boundary and c contains the infix $d_j := \textsc{rcomp}_{M,q_j}(\vdash x_1)$ which hits right into q_{j+1}. But q_j is in L_2 (by the inductive hypothesis), and thus in L_1 (since $L_1 \supseteq L_2$), so c_1 produces it in some

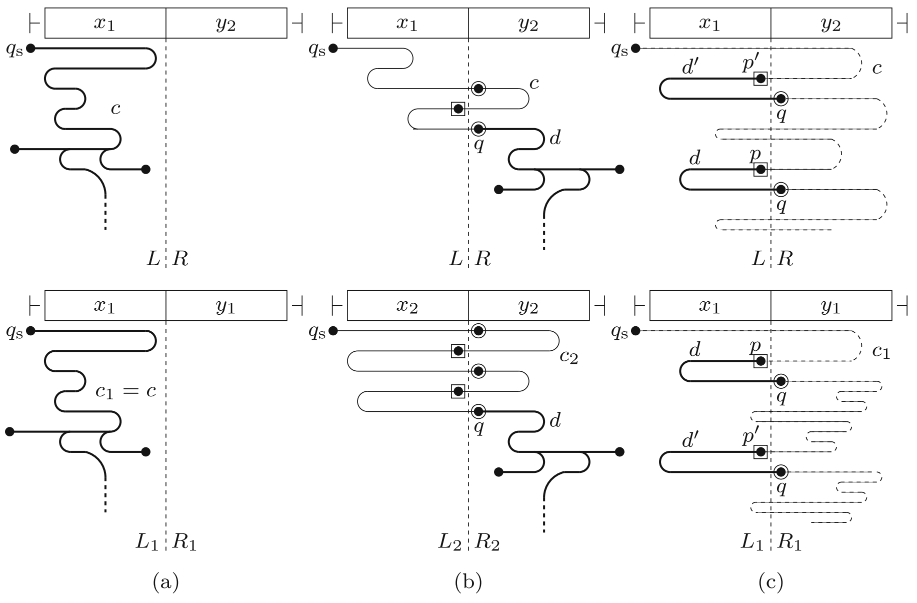


Fig. 4. Computations in the proof of the Hybrid Rule. (a) If c never crosses the critical boundary: then c, c_1 decide the same. (b) If c crosses the critical boundary finitely often, and the last crossing is from left to right: then c, c_2 decide the same. (c) If c crosses the critical boundary infinitely often, and the earliest same-side repetition is after a left-to-right crossing: then c, c_1 decide the same.

right-to-left crossing of the x_1-y_1 boundary. Hence, after that, c_1 continues as in $\text{RCOMP}_{M,q_j}(\vdash x_1) = d_j$, which means that it left-to-right crosses the boundary into q_{j+1}, causing $q_{j+1} \in R_1$. □

Lemma 2 (Hybrid Rule). *Suppose M decides identically on strings x_1y_1, x_2y_2 but differently on their hybrid x_1y_2. Then the frontiers $(L_1, R_1), (L_2, R_2)$ of the full computations of M on x_1y_1, x_2y_2 on the boundaries x_1-y_1 and x_2-y_2 satisfy:*

$$L_1 \not\supseteq L_2 \quad \vee \quad R_1 \not\subseteq R_2\,.$$

Proof. Let c_1, c_2, c be the full computations of M on strings x_1y_1, x_2y_2, and their hybrid x_1y_2. Let $(L_1, R_1), (L_2, R_2), (L, R)$ be the frontiers of these computations on the boundaries x_1-y_1, x_2-y_2, and x_1-y_2, respectively. Towards a contradiction, assume $L_1 \supseteq L_2 \,\&\, R_1 \subseteq R_2$. Then, by the Hybrid Lemma,

$$L_1 \supseteq L_2 \supseteq L \;\;\&\;\; R \subseteq R_1 \subseteq R_2\,.$$

Using this, we will prove that on at least one of x_1y_1 and x_2y_2, the decision of M must be identical to its decision on the hybrid x_1y_2—a contradiction.

We take cases on how often c crosses the critical boundary x_1-y_2 (Fig. 4).

If c never crosses the critical boundary, then c lies fully within $\vdash x_1$ (Fig. 4a). So, M notices no difference between x_1y_2, x_1y_1, and decides identically on both.

If c crosses the critical boundary finitely often, then there is a last crossing.

Suppose this last crossing is from left to right (Fig. 4b). Let q be the state resulting from it. Then $d := \text{LCOMP}_{M,q}(y_2\dashv)$ does not hit left (it hangs, or loops, or falls off $\dashv$), and is thus a suffix of c. At the same time, $q \in R$ (by its selection as the result of a left-to-right crossing), and thus $q \in R_2$ (since $R \subseteq R_2$). Hence, c_2 also contains a left-to-right crossing of the boundary x_2-y_2 that results in q. Clearly, from that point on, c_2 behaves as in $\text{LCOMP}_{M,q}(y_2\dashv)$, namely as in d, which does not hit left. Therefore c_2 also finishes with d. Overall, c and c_2 have the same suffix d, which implies that M decides identically on x_1y_2 and x_2y_2.

If the last crossing of the x_1-y_2 boundary along c is from right to left, then a symmetric argument applies: $q \in L \subseteq L_1$ and $d := \text{RCOMP}_{M,q}(\vdash x_1)$ is a suffix of both c and c_1, causing M to decide identically on x_1y_2 and x_1y_1.

If c crosses the critical boundary infinitely often, then consider the infinite list $q_1, q_2, \ldots$ where q_j is the state produced by the j-th crossing (Fig. 4c). Of course, this list contains repetitions: there exist $1 \leq j_1 < j_2$ such that $q_{j_1} = q_{j_2}$. Equally clearly, it also contains *same-side* repetitions: namely, repetitions where j_1, j_2 are either both odd or both even (so that q_{j_1}, q_{j_2} are produced both by left-to-right crossings or both by right-to-left crossings, respectively). Let q be the state in the earliest such repetition (namely $q = q_{j_1} = q_{j_2}$ in the same-side repetition with the smallest j_2) and p the state produced by the crossing just before that repetition happened (namely $p = q_{j_2-1}$).

Suppose the j_1-th and j_2-th crossings are from left to right. Then the j_2-1st crossing is from right to left and is followed by $d := \text{RCOMP}_{M,p}(\vdash x_1)$, which hits right into q. At the same time, $p \in L$ and $q \in R$ (by their selection as the results of a right-to-left and a left-to-right crossing, respectively) and thus $p \in L_1$ and $q \in R_1$ (as $L_1 \supseteq L$ and $R \subseteq R_1$). So, c_1 also contains right-to-left crossings that produce p (to be called "p-crossings") and left-to-right crossings that produce q (to be called "q-crossings"). The next claim implies that at least one of these two types of crossings repeats, and thus c_1 loops, exactly as c does. Therefore, M decides identically on x_1y_2 and x_1y_1.

Claim. *There are at least two p-crossings or at least two q-crossings in c_1.*

Proof. Towards a contradiction, assume c_1 contains exactly one p-crossing and exactly one q-crossing. We distinguish cases based on their order inside c_1.

If the q-crossing appears before the p-crossing: We know the p-crossing is followed by $\text{RCOMP}_{M,p}(\vdash x_1)$, namely by d, which we already know hits right into q. So, the p-crossing is followed by a q-crossing. Hence, c_1 contains at least two q-crossings (one before and one after the p-crossing), a contradiction.

If the q-crossing appears after the p-crossing: We first return to c to observe that $j_1 \neq 1$, namely the first of the two crossings that produce q cannot be the very first of all crossings. (Because then $\text{LCOMP}_{M,q_\text{s}}(\vdash x_1)$ hits right into q; so the one q-crossing in c_1 is also the very first of all crossings, and thus appears before the p-crossing, a contradiction.)

Hence, we can talk about the $(j_1{-}1)$-st crossing of the critical boundary in c (from right to left). Let p' be the state it produces. Then $d' := \textsc{rcomp}_{M,p'}(\vdash x_1)$ hits right into q. Note that $p' \neq p$, or else $q_{j_1-1} = q_{j_2-1}$, contrary to our selection of $q = q_{j_1} = q_{j_2}$ as the earliest same-side repetition. Also note that $p' \in L$ (as the result of a right-to-left crossing), hence $p' \in L_1$ (since $L_1 \supseteq L$), hence c_1 also contains a right-to-left crossing that produces p' (to be called "p'-crossing").

It now follows that c_1 contains at least two q-crossings: the one right after the p-crossing (caused by d) and the one right after the p'-crossing (caused by d'), which we know are distinct (because $p' \neq p$). This is again a contradiction. $\boxdot$

If the j_1-th and j_2-th crossings are from right to left, then we argue symmetrically. We know that $d := \textsc{lcomp}_{M,p}(y_2\dashv)$ hits left into q, and that $p \subseteq R \subseteq R_2$ and $L_2 \supseteq L \ni q$, so c_2 also contains left-to-right crossings that produce p and right-to-left crossings that produce q. As before, at least one of these two types of crossings repeats, hence c_2 loops, causing M to decide on x_2y_2 just as on x_1y_2.

This concludes the third case of our argument and, with it, the full proof. □

Behaviors. Let z be any string. The *behavior of M on z* is the (partial) function which returns the results of all left and right computations of M on z. Specifically, it is the function $\gamma_{M,z} : S \times \{\textsc{l},\textsc{r}\} \rightharpoonup S \times \{\textsc{l},\textsc{r}\}$ such that, for all $p \in S$,

$$\gamma_{M,z}(p,\textsc{l}) := \begin{cases} (q,\textsc{l}) & \text{if } \textsc{lcomp}_{M,p}(z) \text{ hits left into } q, \\ \text{undefined} & \text{if } \textsc{lcomp}_{M,p}(z) \text{ hangs or loops}, \\ (q,\textsc{r}) & \text{if } \textsc{lcomp}_{M,p}(z) \text{ hits right into } q; \end{cases}$$

and similarly for $\gamma_{M,z}(p,\textsc{r})$, using $\textsc{rcomp}_{M,p}(z)$ instead of $\textsc{lcomp}_{M,p}(z)$.

With this motivation, we define a *behavior of M* to be any partial function from $S\times\{\textsc{l},\textsc{r}\}$ to $S\times\{\textsc{l},\textsc{r}\}$. Easily, if M has s states, then it has $(2s{+}1)^{2s}$ behaviors. However, if $|z| = 1$, then every left computation is also a right computation (since the first and last cells of z coincide), causing $\gamma_{M,z}(p,\textsc{l}) = \gamma_{M,z}(p,\textsc{r})$ for all p; hence, the number of *single-symbol behaviors* of M is only $(2s{+}1)^s$.

A standard fact in the analysis of 2DFA computations is that, if M exhibits the same behavior on two strings, then each of them can be replaced by the other in any context, without M noticing the difference. Formally, this is captured by the next lemma, which we state without proof.

Lemma 3 (Infix Lemma). *If $\gamma_{M,y_1} = \gamma_{M,y_2}$, then $\gamma_{M,xy_1z} = \gamma_{M,xy_2z}$.*

As a direct consequence of this, M cannot decide differently on two strings which differ only at two infixes that do not change its behavior.

Lemma 4 (Infix Rule). *If M decides differently on strings xy_1z, xy_2z, then:*

$$\gamma_{M,y_1} \neq \gamma_{M,y_2}\,.$$

Proof. Towards the contrapositive, assume $\gamma_{M,y_1} = \gamma_{M,y_2}$. Then, by the Infix Lemma, $\gamma_{M,\vdash xy_1z\dashv} = \gamma_{M,\vdash xy_2z\dashv}$. In particular, the two functions return the same value on $(q_s,\textsc{l})$, which is either $(q_{\mathrm{a}},\textsc{l})$, or $(q_{\mathrm{a}},\textsc{r})$, or undefined. In all three cases, it follows that M decides identically on xy_1z and xy_2z. □

3 The Case of Two Symbols

We now prove Theorem 2, that *all smallest* 2DFAs for 2OWL_h have $\Theta(h)$ *states*. This follows directly from the next two lemmata. The upper bound (Lemma 5) is well-known and also implied by [6]; here, we give a careful construction. The lower bound (Lemma 6) is a tighter variant of [1, Theorem 1] that uses frontiers, as opposed to arbitrary pairs of sets of states. (Without this modification, the lower bound for 2DFAs by the argument of [1, Theorem 1] is only $\frac{1}{2}h$.)

Lemma 5. *Some* 2DFA *solves* 2OWL_h *with* $\leq 2h$ *states.*

Proof. Fix $h \geq 2$ and consider an instance xy of 2OWL_h, for $x, y \in \Sigma_h$ (Fig. 1b). Let $u_1, u_2, \ldots, u_h$ be the nodes of column 1, from top to bottom. We say u_i is L-*live*, if it has non-zero degree in x; R-*live*, if it has non-zero degree in y; *live*, if it is both L-live and R-live; and *dead* if it is not live. Clearly, xy is live iff some u_i is live. So, our 2DFA M simply searches for a live u_i sequentially, from u_1 to u_h.

The set of states is $S := [h] \times \{\text{L},\text{R}\}$ and state $(1, \text{L})$ serves as both the start and the accept state. Each other state (i, L) is used only on $\vdash x$; it assumes that all u_j above u_i are dead and that u_i is R-live, and tries to check if u_i is also L-live. Symmetrically, every state (i, R) is used only on $y\dashv$; it assumes that all u_j above u_i are dead and that u_i is L-live, and tries to check if u_i is also R-live.

The transitions between these states are now not hard to see:

From $(1, \text{L})$ on $\vdash$, M moves to $(1, \text{L})$ on x. If x contains no edges, then xy is obviously dead, so M just hangs. Otherwise, there exists at least one L-live u_i, so M finds the topmost such u_i and moves to the corresponding state (i, R) on y.

From a state (i, R) on y, M checks if u_i is R-live. If so, then xy is live, so M moves to (i, R) on $\dashv$, and then off $\dashv$ into $(1, \text{L})$ to accept. Otherwise, it checks if any u_j below u_i is R-live. If not, then xy is dead, so M just hangs. Otherwise, M finds the topmost R-live u_j below u_i, and moves to the corresponding state (j, L) on x, to check whether that u_j is also L-live.

From a state (i, L) on x with $i \geq 2$, M behaves symmetrically as above: if u_i is L-live, then M moves to (i, L) on $\vdash$, and then off $\vdash$ into $(1, \text{L})$ to accept. Otherwise, it either hangs, if no u_j below u_i is L-live; or moves to y and into the state (j, R) corresponding to the topmost such u_j.

It should be clear that M works correctly and uses exactly $2h$ states. □

Lemma 6. *Every* 2DFA *solving* 2OWL_h *has* $> \frac{1}{2}h + \frac{1}{4}\lg h - \frac{1}{2}$ *states.*

Proof. Let M be any 2DFA solving 2OWL_h. Let S be its set of states and $s := |S|$.

For every $\alpha \subseteq [h]$, let $x_\alpha := \{(u, u) \mid u \in \alpha\}$ be the symbol consisting of the "horizontal" edges which correspond to indices in α (e.g., see the leftmost symbol in Fig. 1a). Clearly, for every $\alpha, \beta \subseteq [h]$, the string $x_\alpha x_\beta$ is live iff $\alpha \cap \beta \neq \emptyset$. We also easily verify the following.

Claim 1. *For all distinct* $\alpha, \beta \subseteq [h]$:

(i) the strings $x_\alpha x_{\overline{\alpha}}$ *and* $x_\beta x_{\overline{\beta}}$ *are dead, but*
(ii) at least one of their two hybrids $x_\alpha x_{\overline{\beta}}$ *and* $x_\beta x_{\overline{\alpha}}$ *is live.*

Proof. Let $\alpha, \beta \subseteq [h]$ with $\alpha \neq \beta$. Then (i) is obvious, since $\alpha \cap \overline{\alpha} = \beta \cap \overline{\beta} = \emptyset$. For (ii), suppose both hybrids $x_\alpha x_{\overline{\beta}}$ and $x_\beta x_{\overline{\alpha}}$ are dead. Then $\alpha \cap \overline{\beta} = \emptyset$ and $\beta \cap \overline{\alpha} = \emptyset$; equivalently, $\alpha \subseteq \beta$ and $\beta \subseteq \alpha$; hence $\alpha = \beta$, a contradiction. $\boxdot$

Now, for each $\alpha \subseteq [h]$, consider the the full computation of M on $x_\alpha x_{\overline{\alpha}}$ and let (L_α, R_α) be its frontier over the middle boundary. This effectively defines 2^h frontiers of M, one for each α. We claim that these are all distinct.

Claim 2. *For all distinct* $\alpha, \beta \subseteq [h]$: $\quad (L_\alpha, R_\alpha) \neq (L_\beta, R_\beta)$.

Proof. Let $\alpha, \beta \subseteq [h]$ with $\alpha \neq \beta$. By Claim 1 and since M solves 2OWL$_h$, we know M decides identically on $x_\alpha x_{\overline{\alpha}}$ and $x_\beta x_{\overline{\beta}}$ (it does not accept) but differently on at least one of their hybrids $x_\alpha x_{\overline{\beta}}$ and $x_\beta x_{\overline{\alpha}}$ (it accepts). Without loss of generality, assume the interesting hybrid is $x_\alpha x_{\overline{\beta}}$ (otherwise, swap the roles of the two strings). Then, by the Hybrid Rule (Lemma 2), we know $L_\alpha \not\supseteq L_\beta$ or $R_\alpha \not\subseteq R_\beta$. Therefore $L_\alpha \neq L_\beta$ or $R_\alpha \neq R_\beta$, namely $(L_\alpha, R_\alpha) \neq (L_\beta, R_\beta)$. $\boxdot$

Overall, we conclude that M uses distinct frontiers on the 2^h distinct dead strings of the form $x_\alpha x_{\overline{\alpha}}$. Since it has only $\binom{2s+1}{s+1}$ frontiers total, it follows that

$$\binom{2s+1}{s+1} \geq 2^h \, , \tag{2}$$

and we need to solve for s. Using the well-known Stirling's bounds

$$\sqrt{2\pi}\sqrt{n} \cdot \left(\tfrac{n}{e}\right)^n \;\leq\; n! \;\leq\; e\sqrt{n} \cdot \left(\tfrac{n}{e}\right)^n$$

for the factorial function, we calculate:

$$\binom{2s+1}{s+1} = \frac{(2s+1)!}{(s+1)!s!} = \frac{2s+1}{s+1} \cdot \frac{(2s)!}{(s!)^2} < 2 \cdot \frac{e\sqrt{2s} \cdot (2s/e)^{2s}}{(\sqrt{2\pi}\sqrt{s} \cdot (s/e)^s)^2} = \frac{e\sqrt{2}}{\pi} \cdot \frac{2^{2s}}{\sqrt{s}}$$

so that (2) implies $(e\sqrt{2}/\pi)(2^{2s}/\sqrt{s}) > 2^h$, and thus

$$s > \tfrac{1}{2}h + \tfrac{1}{4}\lg s - \tfrac{1}{2}\lg(e\sqrt{2}/\pi) \, . \tag{3}$$

Since $s \geq 1$ and $\frac{1}{2}\lg(e\sqrt{2}/\pi) \leq 0.15$, this implies $s > \frac{1}{2}h - 0.15$, and thus $s \geq \frac{1}{2}h$ (since s and h are both integers). So, $\lg s \geq \lg h - 1$. Substituting in (3), we get:

$$s > \tfrac{1}{2}h + \tfrac{1}{4}\lg h - \tfrac{1}{2}\lg(2e/\pi) \, .$$

Finally, we note that $\lg(2e/\pi) \approx 0.8 < 1$. $\square$

4 The Case of Three Symbols

To prove Theorem 1, that *all smallest* 2DFAs for 3OWL$_h$ *have* $\Theta(h^2/\log h)$ *states*, we start with the lower bound (Lemma 7), which is simpler; then continue with the upper bound (Lemma 8), which is a bit more involved.

Lemma 7. *Every* 2DFA *solving* 3OWL$_h$ *has* $\geq \frac{1}{2}\frac{h^2}{\lg h}$ *states.*

Proof. Let M be any 2DFA solving 3OWL$_h$. Let S be its set of states and $s := |S|$.

Let $y_1, y_2, \dots, y_N$ be a list of all symbols in the input alphabet Σ_h. Since every symbol may contain up to h^2 edges, we know $N = 2^{h^2}$. For each $i = 1, 2, \dots, N$, let $\gamma_i := \gamma_{M,y_i}$ be the behavior of M on y_i.

Claim. *For all distinct* $i, j \in [N]$: $\quad \gamma_i \neq \gamma_j$.

Proof. Let $i, j \in [N]$ with $i \neq j$. Then $y_i \neq y_j$. Hence, there exists at least one edge (u, v) which appears in one of y_i, y_j, but not the other. Without loss of generality, assume (u, v) appears in y_i but not in y_j. Let $x := \{(u, u)\}$ and $z := \{(v, v)\}$ be the symbols containing only the "horizontal" edges corresponding to u and v. Then the strings xy_iz and xy_jz are respectively a live and a dead instance of 3OWL$_h$. Hence, M decides differently on them. By the Infix Rule (Lemma 4), it follows that $\gamma_i \neq \gamma_j$. ⊡

Hence, M uses distinct behaviors on the 2^{h^2} possible middle symbols. Since it has only $(2s + 1)^s$ single-symbol behaviors in total (cf. p. 9), it follows that

$$(2s+1)^s \geq 2^{h^2} , \tag{4}$$

which implies the bound in the statement, as follows.

If $h = 2$, then (4) asks that $(2s+1)^s \geq 16$. Easily, this holds only if $s \geq 2$, which matches the bound in the statement: $\frac{1}{2}(h^2/\lg h) = \frac{1}{2}(4/1) = 2$.

If $h = 3$, then similarly we need $(2s+1)^s \geq 512$, which holds only if $s \geq 4$, which exceeds the bound in the statement: $\frac{1}{2}(h^2/\lg h) = \frac{1}{2}(9/\lg 3) \approx 2.84$.

If $h \geq 4$, then we first take logarithms to rewrite (4) as $s \lg(2s+1) \geq h^2$. Towards a contradiction, we assume $s < \frac{1}{2}\frac{h^2}{\lg h}$ and calculate:

$$\lg(2s+1) < \lg\big(2 \cdot 2s\big) < \lg\big(2\tfrac{h^2}{\lg h}\big) = 2 \lg h - (\lg \lg h - 1) \leq 2 \lg h ,$$

where the first step uses the fact that $2s+1 < 4s$ (since $s \geq 1$); the second step uses the assumption that $s < \frac{1}{2}\frac{h^2}{\lg h}$; and the last step uses the fact that $\lg \lg h \geq 1$ (since $h \geq 4$). Therefore, we can conclude that

$$s \lg(2s+1) < \big(\tfrac{1}{2}\tfrac{h^2}{\lg h}\big)\big(2 \lg h\big) = h^2 ,$$

contrary to the rewriting of (4) above. □

Lemma 8. *Some* 2DFA *solves* 3OWL$_h$ *with* $\leq 4h\lceil \frac{h}{\lfloor \lg h \rfloor} \rceil$ *states.*

Proof. Fix $h \geq 2$ and consider an instance xyz of 3OWL$_h$ for $x, y, z \in \Sigma_h$ (Fig. 1c). Let $u_1, u_2, \dots, u_h$ and $v_1, v_2, \dots, v_h$ be the nodes of columns 1 and 2, respectively, from top to bottom. Similarly to the proof of Lemma 5, we say u_i is L-*live* if it has non-zero degree in x; and v_i is R-*live* if it has non-zero degree in z.

We first partition column 1 into $h/\lg h$ blocks of length $\lg h$ each. More carefully, we let $l := \lfloor \lg h \rfloor$ be the desired length, and assign every u_i to block $b_{\lceil i/l \rceil}$.

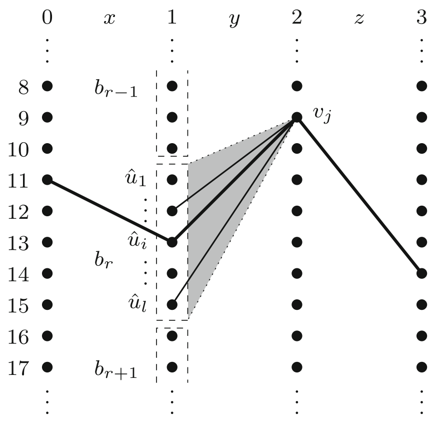


Fig. 5. The proof of Lemma 8 for the example $h = 32$, $l = 5$, $m = 7$, and $r = 3$. Relative to b_r, the shown vertex v_j has connectivity $\{2, 3, 5\}$.

Easily, this produces $m := \lceil h/l \rceil$ disjoint blocks, $b_1, b_2, \ldots, b_m \subseteq \{u_1, u_2, \ldots, u_h\}$. For example, for $h = 5$, the desired length is $l = 2$ and we get the $m = 3$ blocks $b_1 = \{u_1, u_2\}$, $b_2 = \{u_3, u_4\}$, and $b_3 = \{u_5\}$. Note that, if l does not divide h, then the length of the last block b_m is not l, but only $l' := h \bmod l$.

We say a block b_r is *live* if there exists a live path that passes through a node belonging to b_r (Fig. 5). Clearly, xyz is live iff some b_r is live. So, our 2DFA M simply searches for a live b_r sequentially, from b_1 to b_m.

Specifically, M consists of m disjoint sub-automata $M_1, M_2, \ldots, M_m$, where every M_r uses states of the form $(r, \ldots)$ and is responsible for checking whether the corresponding b_r is live. The start state of M is the start state of M_1. From then on, the iteration works as follows. If the current M_r confirms that b_r is live, then it falls off $\dashv$ into the designated accept state q_a of M, causing M to accept, too. Otherwise, M_r arrives at a state and cell where it has confirmed that b_r is dead; from there, it either advances the iteration by moving to x and to the start state of M_{r+1}, if $r < m$; or just hangs, causing M to hang too, if $r = m$.

Every M_r works in the same way as every other, except that it focuses on its own b_r. So, all sub-automata are of the same size $\tilde{s}$, causing M's total size to be $s = m\tilde{s} = O(h\tilde{s}/\log h)$. Hence, to achieve the $O(h^2/\log h)$ size in the statement, we ensure that each M_r has size $O(h)$. Here is how.

Checking a block. Fix any b_r and let $\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_l$ be its vertices, from top to bottom. In the case where b_r is the last block b_m and has only $l' < l$ nodes, we pretend that the nodes $\hat{u}_{l'+1}, \ldots, \hat{u}_l$ exist but have degree zero in both x and y.

Now consider any v_j in column 2. (See Fig. 5.) The *connectivity of* v_j (*relative to* b_r) is the set $\xi \subseteq [l]$ of the indices of all $\hat{u}_i$ which connect to v_j:

$$i \in \xi \iff y \text{ contains the edge } (\hat{u}_i, v_j)\,.$$

With this motivation, we call *connectivity* any subset of $[l]$. Clearly, the number of different connectivities is $k := 2^l = 2^{\lfloor \lg h \rfloor} \leq 2^{\lg h} = h$. Let $\xi_1, \xi_2, \dots, \xi_k$ be any fixed listing of them, so that each v_j has its connectivity in this list.

We say that, relative to b_r, a connectivity ξ_t is L-*live*, if for at least one $i \in \xi_t$ the corresponding $\hat{u}_i$ is L-*live*; R-*live*, if at least one of the v_j with connectivity ξ_t is R-*live*; *live*, if it is both L-live and R-live; and *dead*, if it is not live. Easily:

Claim. *Block b_r is live iff at least one connectivity ξ_t is live relative to it.*

Proof. Suppose b_r is live. Then some live path passes through it. Let $\hat{u}_i$ and v_j be the nodes used by this path in b_r and in column 2, respectively, and ξ_t the connectivity of v_j relative to b_r. Then $\hat{u}_i$ is L-live and v_j is R-live (clearly); and $i \in \xi_t$ (because of the edge from $\hat{u}_i$ to v_j). Hence ξ_t is both L-live (because of $\hat{u}_i$) and R-live (because of v_j). So, ξ_t is live.

Conversely, suppose some ξ_t is live relative to b_r. Let $\hat{u}_i$ and v_j be witnesses for why it is L-live and R-live, respectively. Then (i) $\hat{u}_i$ is L-live and (ii) $i \in \xi_t$; and (iii) v_j is R-live and (iv) v_j has connectivity ξ_t relative to b_r. By (ii) and (iv), we know the edge $(\hat{u}_i, v_j)$ exists. By (i) and (iii), we know this edge is actually on a live path through $\hat{u}_i$ and v_j. Hence, b_r is live. $\boxdot$

Therefore, our sub-automaton M_r simply searches for a live ξ_t sequentially, from ξ_1 to ξ_k. Intuitively, M_r consists of k sub-automata $M_{r,1}, M_{r,2}, \dots, M_{r,k}$, where every $M_{r,t}$ is responsible for checking whether the corresponding ξ_t is live. The machine starts on x and in the start state of $M_{r,1}$. From then on, the iteration works as follows. If the current $M_{r,t}$ confirms that ξ_t is live, then it falls off $\dashv$ into q_a, causing M_r to behave exactly as promised above. Otherwise, $M_{r,t}$ "*halts*" on x or y, in the sense that it enters a state where it has confirmed that ξ_t is dead. From there: if $t < k$, then it advances the iteration over all connectivities by moving to x and to the start state of $M_{r,t+1}$; otherwise ($t = k$), it either moves to x and to the start state of M_{r+1}, if $r < m$, or just hangs, if $r = m$, causing M_r to behave exactly as promised above.

Checking a connectivity. Every $M_{r,t}$ starts on x in state (r, t, L), trying to decide whether ξ_t is L-live. For this, it checks if any of the $\hat{u}_i$ in b_r with $i \in \xi_t$ have non-zero degree in x. If not, then ξ_t is dead, so $M_{r,t}$ *halts* (on x). Else, it moves to y in state (r, t, R), to check if ξ_t is also R-live. Reading y, it computes the connectivities of all v_j relative to b_r, and focuses only on those v_j with connectivity ξ_t. If no such v_j exist, then ξ_t is dead, so $M_{r,t}$ again *halts* (on y). Else, $M_{r,t}$ iterates over all such v_j, from top to bottom, moving to z to check if any of them is R-live.

Let us first see a naive way to implement this last iteration: From y and in state (r, t, R), $M_{r,t}$ finds the topmost v_j with connectivity ξ_t and moves to z in state $(r, t, j, \top)$. Reading z, it checks whether v_j is R-live. If so, then it moves to $\dashv$ into q_a and then off $\dashv$ into q_a. Otherwise, it moves back to y in a state $(r, t, j, \perp)$. Reading y, it finds the topmost $v_{j'}$ with $j' > j$ and connectivity ξ_t. If none exists, then it *halts* (on y). Else, it moves to z in state $(r, t, j', \top)$, and so on. Of course, this implementation needs $2h$ states in each $M_{r,t}$ (two for each $j = 1, \dots, h$), causing the total size of M_r to rise to $\Theta(h^2)$—when our goal is only $O(h)$.

The smart implementation works similarly to the naive one, except for the following modification. Every time $M_{r,t}$ moves to z to examine the R-liveness of some v_j, its state is not $(r, t, j, \top)$, but just $(r, j, \top)$. In other words, whenever on z, the automaton "forgets" the connectivity ξ_t that it is responsible for. Of course, this creates no problem in checking whether v_j is R-live (since this needs only j). Still, if v_j is not R-live and the automaton returns to y, then it is unclear how it will continue its iteration over the remaining nodes of connectivity ξ_t (below v_j), let alone the iteration over the remaining connectivities (after ξ_t). How will it manage to do all this, if it has forgotten ξ_t?

The answer is that the automaton can recover the forgotten ξ_t from (i) what it still remembers and (ii) the edges in y. Specifically, whenever v_j is not R-live, the automaton moves from z and state $(r, j, \top)$ back on y and in state $(r, j, \bot)$. Reading y, it finds all edges between the nodes of b_r and v_j; from these, it computes the connectivity of v_j relative to b_r, which is exactly ξ_t. Hence, having remembered ξ_t, it continues its iteration as if it had never forgotten it.

Put another way, the trick is that $M_{r,t}$ is not disjoint from the rest of the sub-automata $M_{r,\cdot}$. Although it uses its own states (r, t, L) and (r, t, R) on xy, it shares with the rest of the $M_{r,\cdot}$ the states $(r, j, \bot)$ and $(r, j, \top)$ on yz.

Overview. Returning to M, we see that it has states of the form (r, t, L), (r, t, R), $(r, j, \bot)$, and $(r, j, \top)$, where $r \in [m]$, $t \in [k]$, $j \in [h]$. Namely, its set of states is

$$S := ([m] \times [k] \times \{\text{L,R}\}) \;\cup\; ([m] \times [h] \times \{\bot,\top\})\,,$$

for a total size of $2mk + 2mh \leq 4mh$, as promised in the statement.

State $(1, 1, \text{L})$ is simultaneously the start and accept state: $q_\text{s} = q_\text{a} := (1, 1, \text{L})$. When in it and on $\vdash$ or $\dashv$, the machine stays in the state and moves right (either to x, to start the search for a live block; or off $\dashv$, to accept). This is the only state used on $\vdash$ and $\dashv$. All other states are used only on x, if of the form (r, t, L); or only on y, if of the form (r, t, R) or $(r, j, \bot)$; or only on z, if of the form $(r, j, \top)$.

When in (r, t, L) reading x, the machine checks if some node in b_r with non-zero degree in x has its index in ξ_t (i.e., ξ_t is L-live relative to b_r). If so, then M moves to y and in (r, t, R) to check if ξ_t is also R-live relative to b_r. Otherwise, M "*continues the search from x*", meaning that: it moves to (stays on) x and enters $(r, t{+}1, \text{L})$ to advance the iteration over connectivities, if $t < k$; or moves to (stays on) x and enters $(r{+}1, 1, \text{L})$ to advance the iteration over blocks, if $t = k$ and $r < m$; or hangs immediately to signify rejection, if $t = k$ and $r = m$.

When in (r, t, R) reading y, the machine checks if any node of column 2 has connectivity ξ_t relative to b_r. If so, then M finds the topmost such node v_j and moves to z and in $(r, j, \top)$ to check if v_j is R-live. Otherwise (ξ_t is not R-live), *M continues the search from x* (as above).

When in $(r, j, \top)$ reading z, the machine checks if node v_j has non-zero degree in z. If so (b_r is live), then M moves to $\dashv$ and in q_a to signify acceptance. Else, it moves back to y in $(r, j, \bot)$ to advance the iteration over nodes which share with v_j the same connectivity relative to b_r.

When in $(r, j, \bot)$ reading y, the machine finds the connectivity ξ_t of node v_j relative to b_r and searches for the topmost $v_{j'}$ with $j' > j$ and the same connectivity relative to b_r. If such $v_{j'}$ exists, then M moves to z and in $(r, j', \top)$ to check if $v_{j'}$ is R-live. Otherwise, M *continues the search from* x (as above).

This concludes the definition of the transition function of M. Note that our description uses transitions (out of the states (r, t, L)) which do not move the input head, contrary to our definition of 2DFAs as machines which always move (Sect. 2); but this violation can be easily removed, with a standard, well-known technique that does not change the number of states.

Other than that, it should be clear that M decides 3OWL_h, as promised. □

5 Conclusion

Motivated by recent work on *reasonable automata* [1,3], we initiated a study of the size of arbitrary 2DFAs solving *one-way liveness* on inputs of constant length. We gave (i) a more detailed proof of the known fact (from [1]) that $\Theta(h)$ states are necessary and sufficient for length 2; and (ii) a proof of the new fact that $\Theta(h^2/\log h)$ states are necessary and sufficient for length 3. This concludes the discussion for these two lengths and for the asymptotic size.

For longer lengths, the question remains open. For example, we can still ask: *What is the size of a smallest* 2DFA *solving one-way liveness on four symbols*? The answer is known to be both $O(h^2)$ and $\Omega(h^2/\log h)$ (by the application of Savitch's method in [3, Theorems 6 and 7]; and our Lemma 7, which clearly also extends to four symbols). However, the tight asymptotic growth remains elusive.

At the same time, we still do not know the exact sizes of the smallest 2DFAs for two and three symbols. Finding those sizes would also be very interesting.

References

1. Bianchi, M.P., Hromkovič, J., Kováč, I.: On the size of two-way reasonable automata for the liveness problem. In: International Conference on Developments in Language Theory, pp. 120–131 (2015)
2. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Boston (1979)
3. Hromkovič, J., Královič, R., Královič, R., Štefanec, R.: Determinism vs. nondeterminism for two-way automata: representing the meaning of states by logical formulæ. Int. J. Found. Comput. Sci. **24**(7), 955–978 (2013)
4. Kapoutsis, C.: Removing bidirectionality from nondeterministic finite automata. In: Jędrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 544–555. Springer, Heidelberg (2005). https://doi.org/10.1007/11549345_47
5. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Symposium on the Theory of Computing, pp. 275–286 (1978)
6. Štefanec, R.: Semantically restricted two-way automata. Ph.D. thesis, Comenius University, Bratislava (2014)

Regularity of k-Abelian Equivalence Classes of Fixed Cardinality

Juhani Karhumäki and Markus A. Whiteland(✉)

Department of Mathematics and Statistics, University of Turku, 20014 Turku, Finland
{karhumak,mawhit}@utu.fi

Abstract. Two words u and v are said to be k-Abelian equivalent if, for each word x of length at most k, the number of occurrences of x as a factor of u is the same as for v. In this note we continue the analysis of k-Abelian equivalence classes. In particular, we show that, for any fixed integer $r \geq 1$, the language of words representing equivalence classes of cardinality r is regular.

Keywords: k-Abelian equivalence · Regular languages

1 Introduction

The k-Abelian equivalence, originally introduced in [9], is an equivalence relation in between equality and Abelian equivalence of words. It identifies words u and v which contain all words of length at most k equally many times as factors. For $k = 1$ it coincides with the Abelian equivalence. Obviously, if two words u and v are k-Abelian equivalent, in symbols $u \sim_k v$, they are of equal length and, moreover, if they are so for each $k \geq 1$, then they are equal as words.

The k-Abelian equivalence defines in a natural way a complexity measure for languages, as well as for infinite words. Such a research was initiated in [12], and later continued, e.g., in [4]. Other topics of the k-Abelian equivalence such as k-Abelian repetitions, k-Abelian palindromicity and k-Abelian singletons were studied in [8,10,11,14], respectively.

A characterization of the k-Abelian equivalence in terms of rewriting was obtained in [11]. This was based on the so-called k-switching lemma. This turned out to be quite an interesting approach. It allowed, see [2,3], to show that the union of singleton classes is a regular language. Similarly, the set of all lexicographically least (or, equivalently, greatest) representatives constitutes a regular set. All these proofs are constructive, that is, given $k \in \mathbb{N}$, the above regular sets can be algorithmically found, in principle. In practice, this is not true since the size of the automaton grows at least exponentially in k. It follows that the sequence of the number of singletons of length n is a rational, or in fact, an $\mathbb{N}$-rational sequence. Similarly, the sequence of numbers of minimal elements in equivalence classes of words of length n is $\mathbb{N}$-rational, in other words, the number of equivalence classes of words of length n defines an $\mathbb{N}$-rational sequence.

This latter sequence can be viewed as the complexity function of the k-Abelian equivalence. For a few small values of k and alphabet size m, the sequences are computed in [2], and are included in the On-Line Encyclopedia of Integer Sequences [17] as the sequences A289657 and A289658.

From the above it was concluded in [2] that also the union of two-element equivalence classes constitutes a regular set. This is based on the closure properties of regular sets and the above-mentioned three regular sets: the singletons, the set of minimal, and maximal elements in the equivalence classes. We recall this proof in Sect. 3. Interestingly, this approach does not extend to larger equivalence classes. The reasons for that are elaborated at the beginning of Sect. 3.

In [2], the main usage of the tool of the regular languages was to show that the number of equivalence classes of length n is asymptotic to a certain polynomial. However, by using the ideas presented in [2], we are able to prove our main theorem. For each $r \geq 1$, the union of the k-Abelian equivalence classes of cardinality r is a regular set. This is the main context of this note.

This note is arranged as follows. In Sect. 2 we lay down the basic terminology of combinatorics on words, automata theory, and graph theory needed in the remainder of the text. We also recall relevant results from the literature. In Sect. 3 we prove our main result. We also discuss an approach suggested in [2] and show that it fails. We then conclude with straightforward implications of the main result and further discussion in Sect. 4.

2 Preliminaries and Notation

We set some basic terminology and notions from the literature of combinatorics on words, for more on this topic we refer the reader to [13]. A finite set Σ of symbols is called an *alphabet*. The set of finite sequences, or *words*, over Σ is denoted by Σ^* and the set of non-empty words is denoted by Σ^+. The empty word is denoted by ε. We let $|w|$ denote the length of a word (as a sequence) $w \in \Sigma^*$. By convention, we set $|\varepsilon| = 0$. The set of words of length n over the alphabet Σ is denoted by Σ^n.

We index the letters of a given word starting from 1. For a word $w = a_1 a_2 \cdots a_n \in \Sigma^*$ and indices $1 \leqslant i \leqslant j \leqslant n$, we call the word $a_i \cdots a_j$, denoted by $w[i,j]$, a *factor* of w. For $i > j$ we set $w[i,j] = \varepsilon$. Similarly, for $i < j$ we let $w[i,j)$ denote the factor $a_i \cdots a_{j-1}$, and we set $w[i,j) = \varepsilon$ when $i \geqslant j$. We let $w[i..]$ (resp., $w[..i]$) denote the factor $w[i,n]$ (resp., $w[1,i]$) for brevity. For any i, the factor $w[..i]$ (resp., $w[i..]$) is called a *prefix* (resp., *suffix*) of w. We say that a word $x \in \Sigma^*$ *occurs at position i in w* if the word $w[i..]$ has x as a prefix. For $u \in \Sigma^+$ we let $|w|_u$ denote the number of occurrences of u as a factor of w. The set of factors of a word w is denoted by $F(w)$, and we set $F_n(w) = F(w) \cap \Sigma^n$. We call u a *complete return to x in w* if $u \in F(w)$ such that $|u|_x = 2$ and x occurs as both a prefix and a suffix of u. The set of complete first returns to x in u is denoted by $\Re_w(x)$.

We also need a few basic properties of *regular languages*. *Regular expressions* over an alphabet Σ are the finite expressions constructed recursively as follows.

The symbol $\emptyset$, and each $a \in \Sigma \cup \{\varepsilon\}$ are expressions. If E and E' are expressions then so are $E \cdot E'$, $E + E'$, and E^*. Each expression E defines a language, denoted by $L(E)$ as follows: Each $a \in \Sigma \cup \{\varepsilon\}$ defines the singleton language $L(a) = \{a\}$ and $\emptyset$ defines the empty language. For expressions E and E', the expressions $E \cdot E'$, $E + E'$ and E^* define the languages $L(E) \cdot L(E')$, $L(E) \cup L(E')$, and $\bigcup_{n \geq 0} L(E)^n$, respectively.

A *deterministic finite automaton* (DFA) $\mathcal{A}$ over Σ is a tuple (Q, q_0, δ, F), where Q is a finite set of states, q_0 is the initial state, δ is a partial function $\delta\colon Q \times \Sigma \to Q$ called the *transition function*, and $F \subseteq Q$ is the set of final states. Given a word $w = a_1 \cdots a_n$, the automaton operates on w using δ starting from q_0 by the rule $\delta(q, au) = \delta(\delta(q, a), u)$ for all $u \in \Sigma^+$. If $\delta(q_0, w) \in F$ we say that $\mathcal{A}$ *accepts* w. We let $L(\mathcal{A})$ denote the language *recognized* by $\mathcal{A}$; $L(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ accepts } w\}$.

The languages defined by regular expressions (or recognized by finite automata) are exactly the regular languages, in fact, these two models are equivalent. Another equivalent model for regular languages considered here are *nondeterministic finite automata* (NFA), in which case the transition function may be multi-valued. We refer to [7] for this knowledge, and on more of equivalent models and closure properties of regular languages (closure under complementation, taking a morphic pre-image, etc.). In addition to classical language-theoretic notions, we use the theory of *languages with multiplicities*. This counts how many times a word occurs in a language. This leads to the theory of $\mathbb{N}$-*rational sets*. Using the terminology of [5,16], a multiset over Σ^* is called $\mathbb{N}$-*rational* if it is obtained from finite multisets by applying finitely many times the rational operations *product*, *union*, and taking *quasi-inverses*, i.e., *iteration* restricted to ε-free languages. Equivalently, an $\mathbb{N}$-rational set equals the set of multiplicities of distinct ways an NFA accepts each word. Further, a unary $\mathbb{N}$-rational set is referred to as an $\mathbb{N}$-*rational sequence*. We refer to [5,16] for more on this topic. For a language $L \subseteq \Sigma^*$, the *generating function* $G_L(x)$ of L is defined as the formal power series $G_L(x) = \sum_{n \geq 0} \#(L \cap \Sigma^n) x^n$. The basic result we need is (see [5,16]):

Proposition 1. *Let L be a regular language. The sequence $(\#(L \cap \Sigma^n))_{n \geq 0}$, is an $\mathbb{N}$-rational sequence. Consequently, the* generating function G_L *is a rational function.*

When speaking of the generating function for a language $L \subseteq \Sigma^*$, we mean the generating function for the function ℓ_L defined by $\ell_L(n) = \#(L \cap \Sigma^n)$.

We now turn to the main notion of this paper, k-Abelian equivalence. We recall some results from the literature needed in the remainder of the paper.

Definition 1. *The words $u, v \in \Sigma^*$ are k-*Abelian equivalent*, $u \sim_k v$ in symbols, if $|u|_x = |v|_x$ for all $x \in \Sigma^+$ with $|x| \leqslant k$.*

The relation $\sim_k$ is clearly an equivalence relation; we let $[u]_k$ denote the k-Abelian equivalence class represented by u.

In [11], k-Abelian equivalence is characterized in terms of rewriting, namely by *k-switching*. For this we define the following. Let $k \geqslant 1$ and let $u \in \Sigma^*$.

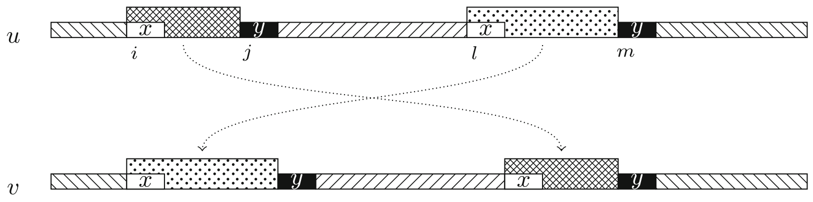


Fig. 1. Illustration of a k-switching. Here $v = S_{k,u}(i,j,l,m)$; the white rectangles symbolize $x \in \Sigma^{k-1}$ and the black rectangles symbolize $y \in \Sigma^{k-1}$.

Suppose that there exist $x, y \in \Sigma^{k-1}$, not necessarily distinct, and indices i, j, l and m, with $1 \leq i < j \leqslant l < m \leq n$, such that x has positions i and l in u, and y has positions j and m in u. In other words, we have

$$u = u[1,i) \cdot u[i,j) \cdot u[j,l) \cdot u[l,m) \cdot u[m..],$$

where the words $u[i..]$ and $u[l..]$ both begin with x, and the words $u[j..]$ and $u[m..]$ both begin with y. Furthermore, $u[i,j)$, $u[l,m) \neq \varepsilon$, but we allow $l = j$, in which case $y = x$ and $u[j,l) = \varepsilon$. We define a *k-switching* on u, denoted by $S_{u,k}(i,j,l,m)$, as

$$S_{u,k}(i,j,l,m) = u[1,i) \cdot u[l,m) \cdot u[j,l) \cdot u[i,j) \cdot u[m..]. \tag{1}$$

An illustration of a k-switching can be found in Fig. 1.

Let us define a relation R_k of Σ^* by uR_kv if and only if v is obtained from u by a k-switching. Now R_k is clearly symmetric, so that the reflexive and transitive closure R_k^* of R_k is an equivalence relation on Σ^*. This gives quite a different characterization of the relation $\sim_k$ in terms of rewriting, see [11] for more details:

Proposition 2. *For $u, v \in \Sigma^*$, we have $u \sim_k v$ if and only if uR_k^*v.*

In order to recall the following result, we need some notation. Let $\lhd$ denote a total order on Σ and the corresponding lexicographic order on Σ^*. We define the language operation $\lhd$-Min$_k$ by $\lhd\text{-Min}_k(L) = \{u \in L \mid w \in L \cap [u]_k \Rightarrow u \trianglelefteq w\}$. We shall often omit $\lhd$ from the prefix to avoid cluttering the text, as this does not usually concern us. Whenever $\lhd$ is omitted, the reader should consider some fixed (but arbitrary) total order $\lhd$.

The characterization in Proposition 2 allows to conclude, see [2]:

Proposition 3. *The language $\lhd$-Min$_k(\Sigma^*)$ is regular for any lexicographic order $\lhd$. In other words, the language of lexicographically least representatives with respect to $\lhd$ of the k-Abelian equivalence classes over Σ is regular.*

Example 1. In [2], minimal DFAs are constructed for the languages Min$_k(\Sigma^*)$ for small values of k and small alphabet Σ. The motivation for this is to compute explicit formulae for the number of k-Abelian equivalence classes for a given length. We recall the number of states in the constructed automata. In the case of a binary alphabet, we have minimal DFAs with 10, 49, and 936 states (including the cases for $k = 2$, 3, and 4, respectively). For the ternary alphabet and $k = 2$, the minimal DFA has 66 states. The automata grow quite fast with respect to k, so in practice such automata seem to be intractable to compute.

Another result, more in the spirit of classical formal language theory, is shown in [2] (for a detailed proof, see [3]):

Proposition 4. *Regular languages are closed under the language operation R_k defined by $R_k(L) = \{v \in \Sigma^* \mid \exists u \in L\colon uR_kv\}$.*

Note that we overload the symbol R_k; we use R_k either as a relation or as a many-valued function, and it will always be clear from context.

We need some terminology and notation of directed graphs with loops and multiple labeled edges. For a graph $G = (V, E)$, we let $V(G)$ denote the set of vertices, and $E(G) \subseteq V \times V$ the set of edges of G. For an edge $e \in E(G)$ from x to y, we call the vertex x the *tail* of e, denoted by $\mathrm{tail}(e)$, and y the *head* of e, denoted by $\mathrm{head}(e)$. The number of edges from x to y is denoted by $m_G(x, y)$. A sequence $W = (e_i)_{i=1}^t$ of edges satisfying $\mathrm{head}(e_i) = \mathrm{tail}(e_{i+1})$ for each $i \in [1, t-1]$ is called a *walk* (in G). We set $\mathrm{tail}(W) = \mathrm{tail}(e_1)$ and $\mathrm{head}(W) = \mathrm{head}(e_t)$. Further, we let $|W|$ denote the length t of W. We call W a *path* if $\mathrm{tail}(e_i) \neq \mathrm{tail}(e_j)$ when $i \neq j$, and $\mathrm{head}(e_t) \neq \mathrm{tail}(e_i)$ for all $i \in [1, t]$. In other words, a path P does not visit any vertex twice. If the walk $(e_i)_{i=1}^{t-1}$ is a path and e_0 is an edge such that $\mathrm{tail}(e_0) = \mathrm{head}(e_{t-1})$ and $\mathrm{head}(e_0) = \mathrm{tail}(e_1)$, then we call the walk $(e_i)_{i=0}^{t-1}$ a *cycle*. We index the edges of a cycle starting from 0 for notational reasons. We consider also loops as cycles. Finally, W is called an *Eulerian walk* if W traverses each edge of G exactly once.

The concatenation $W \cdot W'$ of walks W and W' satisfying $\mathrm{head}(W) = \mathrm{tail}(W')$ is defined in a natural way. For an empty walk W, we define $W \cdot W' = W' \cdot W = W'$. Note here that a cycle C can be concatenated with itself arbitrarily many times. We say that a walk W is a *repetition of a cycle* if we may write $W = C^r$ (C concatenated r times) for some $r \geq 1$.

In this note we make use of *de Bruijn graphs* (see [1] and references therein) defined as follows. For any $k \geqslant 1$ and alphabet Σ, the *de Bruijn graph $dB_\Sigma(k)$ of order k over Σ* is defined as a directed graph for which the set of vertices equals Σ^k. For each word $z \in \Sigma^{k+1}$ we have an edge $(z[..k], z[2..])) \in dB(k)$. In other words, we have $(x, y) \in E(dB_\Sigma(k))$ if and only if there exists a letter $a \in \Sigma$ such that the word $xa \in \Sigma^{k+1}$ ends with y. In this case (x, y) is denoted by (x, a), a being the *label* $\mathrm{label}((x, y))$ of the edge. We shall often omit Σ from the subscript, as it is usually clear from the context. For a walk $W = (e_i)_{i=1}^t$ in $dB(k)$, we set $\mathrm{label}W = \mathrm{label}(e_1) \cdots \mathrm{label}(e_t)$.

We note that any word $u = a_1 \cdots a_n$, where $n \geq k$ and $a_i \in \Sigma$ for each $i \in [1, n]$, defines the walk $W_u = (e_i)_{i=1}^{n-k}$ in $dB(k)$. (Here $e_i = (u[i, i+k), a_{i+k})$, $i \in [1, n-k]$.) Conversely, any walk $W_u = ((x_i, a_i))_{i=1}^t$ in $dB(k)$ defines the word $\mathrm{tail}(W_0) \cdot \mathrm{label}(W) = x_1 \cdot a_1 a_2 \cdots a_t \in \Sigma^{k+t}$. Thus a (long enough) word $u \in \Sigma^*$ should be considered as a walk in $dB(k)$ and vice versa.

The authors of [12] observed a connection between k-Abelian equivalence and Eulerian paths in multigraph versions of *de Bruijn graphs* and we overview it here. Let $f \in \mathbb{N}^{\Sigma^k}$ be an arbitrary vector. We define $G_f = (V, E)$ as follows. We set V as the set of words $x \in \Sigma^{k-1}$ such that x is a prefix or a suffix of a word $z \in \Sigma^k$ for which $f[z] > 0$. For each $z \in \Sigma^k$ with $f[z] > 0$, we have the edge $(z[..k-1], z[2..])$ with multiplicity $f[z]$. Now G_f is a subgraph of $dB(k-1)$ equipped with weights on edges. Note that, if $f[z] = |w|_z$ for all $z \in \Sigma^*$ for some $w \in \Sigma^*$ (this being the case we set $f = f_w$), the graph G_f is the *Rauzy graph* of w of order $k-1$ (see [15]) equipped with weights on edges.

In the following, for $u, v \in \Sigma^{k-1}$, we let $\Sigma(u, v) = u\Sigma^* \cap \Sigma^* v$ and $\Sigma(u, v, n) = \Sigma(u, v) \cap \Sigma^n$.

Lemma 1 (Karhumäki et al. [12, Lemma 2.12]). *For a vector $f \in \mathbb{N}^{\Sigma^k}$ and words $u, v \in \Sigma^{k-1}$, the following are equivalent:*

1. *There exists a word $w \in \Sigma(u, v)$ such that $f = f_w$;*
2. *G_f has an Eulerian path starting from u and ending at v;*
3. *The underlying graph of G_f is connected, and $d^-(s) = d^+(s)$ for every vertex s, except that if $u \neq v$, then $d^-(u) = d^+(u) - 1$ and $d^-(v) = d^+(v) + 1$.*

The following corollary is immediate, as noted in [11].

Corollary 1. *For a word $w \in \Sigma(u, v)$ and $k \geq 1$, we have that $w' \sim_k w$ if and only if the walk $W_{w'}$ is an Eulerian path from u to v in G_w.*

Example 2. Let $u \in \Sigma^*$ and $x \in F_{k-1}(u)$ such that $|u|_x \geq 3$. We may then write $W_u = W_1 W_2 W_3 W_4$ for some walks W_i with $\text{head}(W_i) = x = \text{tail}(W_{i+1})$ for each $i = 1, \ldots, 3$. Then, by the above corollary, we have $u \sim_k v$, where v is defined by the walk $W_v = W_1 W_3 W_2 W_4$. Indeed, W_v is well-defined due to the choice of the extremal vertices of the walks W_i and the same edges are traversed equally many times as in W_u.

Continuing this line of thought, a formula for computing the size of a k-Abelian equivalence class represented by a given word is obtained in [11]. In the following, a *rooted spanning tree with root* v of a graph G is a spanning tree of G for which all edges are directed towards the root vertex v.

Proposition 5. *Let $k \geq 1$ and $w \in \Sigma(u, v)$ for some $u, v \in \Sigma^{k-1}$. Then*

$$\#[w]_k = \kappa_v \prod_{x \in V(G_w)} \frac{(|w|_x - 1)!}{\prod_{a \in \Sigma} |w|_{xa}!}, \tag{2}$$

where κ_v is the number of rooted spanning trees with root v *in G_w.*

3 The Regularity of Classes of Constant Cardinality

Our main goal is to analyze the language $L_{r,k,\Sigma}$ of words w over Σ satisfying $\#[w]_k = r$, or more formally, $L_{r,k,\Sigma} = \{w \in \Sigma^* \mid \#[w]_k = r\}$. We shall often omit Σ from the subscript when there is no danger of confusion.

The language $L_{1,k}$ consists of words representing singleton classes. They are thus uniquely defined by the frequencies of the factors of length k together with the suffix of length $k-1$. The number of such words of length n is considered extensively in [11]. In [2], $L_{1,k}$ is shown to be regular. Indeed, the complement of $L_{1,k,\Sigma}$ is defined by the regular expression

$$\Sigma^* \left(\sum_{\substack{x,y \in \Sigma^{k-1} \\ a,b\in\Sigma,\ a\neq b}} \Big(\big((xb\Sigma^* \cap \Sigma^* y)\, \Sigma^* \cap \Sigma^* x \big) a\Sigma^* \cap \Sigma^* y \Big) \right) \Sigma^*. \tag{3}$$

We remark that a crude upper bound on the number of states in the minimal DFA recognizing $L_{1,k}$ can be obtained from the above regular expression using well-known conversions between various models of regular languages (see, [6,7]). Indeed, e.g., *Glushkov's algorithm* outputs an equivalent NFA of $n+1$ states, given a regular expression of n occurrences of alphabet symbols. The determinization of an n-state NFA can, in the worst case, give a DFA with 2^n states. Observe that the minimal DFA of a language and its complement have the same number of states.

The first few exact values for $\Sigma = \{a, b\}$ are as follows. The minimal DFAs recognizing $L_{1,k}$ for $\Sigma = \{a, b\}$ contain 15, 87, and 1011 states for $k = 2$, 3, and 4, respectively. These values include the garbage state. The automata for $k = 2$ and 3 are presented in [2] (garbage state omitted), and we propose the value for $k = 4$ without proof. For a ternary Σ, we have 84 states for $k = 2$. We recall the minimal DFA of $L_{1,2,\{a,b\}}$ in Fig. 2. For more on this automaton and for automata for other values of k and alphabets, we refer the reader to [2].

Not only the languages $L_{1,k}$ are regular, but so are the languages $L_{2,k}$ as shown in [2]. The proof goes as follows. Recall that $\lhd$-$\mathrm{Min}_k(\Sigma^*)$ is regular for any lexicographic order $\lhd$. Let $\lhd^R$ be the reversal of $\lhd$, that is, $b \lhd^R a$ if and only if $a \lhd b$. After a brief consideration, it becomes clear that

$$\Sigma^* \setminus R_k^2(\Sigma^* \setminus (\lhd\text{-}\mathrm{Min}_k(\Sigma^*) \cup \lhd^R\text{-}\mathrm{Min}_k(\Sigma^*))) = L_{1,k,\Sigma} \cup L_{2,k,\Sigma}$$

is regular since all the language operations, including R_k, preserve regularity. It follows that $L_{2,k,\Sigma} = \Sigma^* \setminus \big(R_k^2(\Sigma^* \setminus (\lhd\text{-}\mathrm{Min}_k(\Sigma^*) \cup \lhd^R\text{-}\mathrm{Min}_k(\Sigma^*)))\big) \setminus L_{1,k,\Sigma}$ is regular since $L_{1,k,\Sigma}$ is regular.

The main result of this paper is the generalization of the above to all $r \in \mathbb{N}$:

Theorem 1. *For any $k, r \geq 1$ and alphabet Σ, the language $L_{r,k}$ is regular.*

The approach of removing (in a regular way) one element of each class at a time does not seem to extend to the languages $L_{r,k}$, $r \geq 3$. The approach of

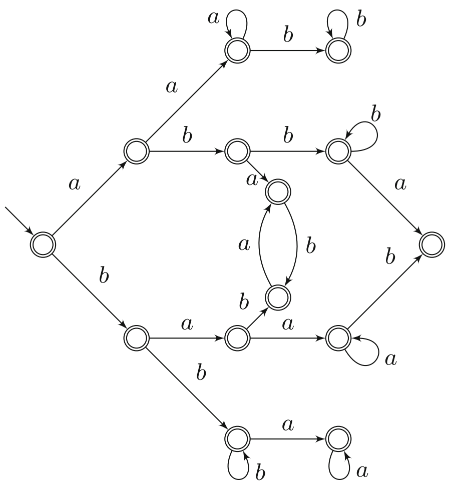


Fig. 2. The minimal DFA recognizing $L_{1,2,\{a,b\}}$. The garbage state is not illustrated. All other states are accepting.

proving the above theorem as suggested in [2, end of Sect. 6], fails as we will shortly show. The method described is of similar flavour as in the proof of the regularity of $L_{2,k,\Sigma}$, namely, by setting $K_{i+1} = K_i \setminus \mathrm{Min}_k(K_i)$, $K_0 = \Sigma^*$, we obtain that $\Sigma^* \setminus R_k^r(K_r) = \cup_{i \leq r} L_{r,k}$ for each $r \in \mathbb{N}$. If Min_k preserved regularity, the above theorem would follow since a finite sequence of regularity-preserving operations would be used.

The following example shows that, unfortunately, this approach does not work, as Min_k does not preserve regularity.

Example 3. Let $k \geq 1$ and $L = (ab^k)^* \cup ab^{k-1}b^*(ab^{k-1})^*$. It is straightforward to check, e.g., using Corollary 1, that

$$\mathrm{Min}_k(L) = (ab^k)^* \cup \{ab^{k-1}b^r(ab^{k-1})^s \mid r \neq s+1\}.$$

It follows that $L \setminus \mathrm{Min}_k(L) = \{(ab^{k-1})b^{r+1}(ab^{k-1})^r \mid r \geq 1\}$. The language $h^{-1}(L \setminus \mathrm{Min}_k(L))$, where $a \mapsto ab^{k-1}$, $b \mapsto b$, equals $\{abb^r a^r \mid r \geq 1\}$ which is not regular. Since all other operations preserve regularity, we conclude that Min_k does not preserve regularity.

As a conclusion, to prove Theorem 1 we need a new approach. We do this via a characterization of the lexicographically least representatives of k-Abelian equivalence classes given in [2].

3.1 The Proof of Theorem 1

Before turning to the formal proof, we sketch the main ingredients. Our first observation (Lemma 3) states that there exists a constant $\mathcal{B}_{k,r}$ (depending on k

and r) such that, for any factor x having at least two distinct returns in u, the total number of occurrences of x in $u \in L_{r,k}$ is bounded by $\mathcal{B}_{k,r}$. We then turn to the language of minimal representatives and show that $\mathrm{Min}_k(L_{r,k})$ is regular (Theorem 2). For the proof, we make an observation concerning factors x occurring more than $\mathcal{B}_{k,r}$ times in $u \in \mathrm{Min}_k(L_{r,k})$ (Lemma 4). The main implication is that corresponding to u and a factor x occurring more than $\mathcal{B}_{k,r}$ times, we may construct a regular language $L_x = zy^*z'$ such that $u \in L_x \subseteq \mathrm{Min}_k(L_{r,k})$. In the proof of Theorem 2, this observation is applied to all such factors x to obtain, for each $u \in \mathrm{Min}_k(L_{r,k})$, a regular expression $L_u = z_0 y_1^* z_1 \cdots y_t^* z_t \subseteq \mathrm{Min}_k(L_{r,k})$. This implies that $\mathrm{Min}_k(L_{r,k})$ is a (possibly infinite) union of regular expressions. To conclude the proof we show that there are actually only finitely many distinct regular expressions in the union with the help of Lemma 5. Finally, Theorem 1 follows from Theorem 2 by applying the regularity-preserving language operation R_k on $\mathrm{Min}_k(L_{r,k})$ finitely many times.

We express the above lemmas via de Bruijn graphs and walks within. To this end we first recall some terminology from [2]. We say that a cycle $C = (d_j)_{j=0}^{s-1}$ *occurs along the walk* W if W can be written as the concatenation $W = W_1 \cdot (d_{r+j \pmod s})_{j=0}^{s-1} \cdot W_2$ for some $r \in [0, s-1]$ and some (possibly empty) walks W_1, W_2. We say that W *enters* C *via the vertex* $\mathrm{tail}(d_r)$ if W_1 is either empty or $d_{r-1 \pmod s}$ is not the last edge of W_1. In this case we say that W enters C at position $|W_1| + 1$. We say that W leaves C via the vertex $\mathrm{head}(d_{r+s-1 \pmod s})$ if W_2 is empty or d_r is not the first edge of W_2. In this case we say that W leaves C at position $|W_1 C|$.

Example 4. Consider the de Bruijn graph $dB_{\{a,b\}}(2)$. The walk $W_u = (e_i)_{i=1}^{10}$, defined by the word $u = aaaabaabaaba$, has two distinct cycles occurring along it, namely the loop $C_1 = (aa, a)$ and the cycle $C_2 = ((aa, b), (ab, a), (ba, a))$. The walk W_u enters C_1 at position 1 and leaves C_1 at position 3 (both via the vertex aa), and does not enter C_1 later on. Further, W_u enters the cycle C_2 at position 2 (via the vertex aa) and W leaves the cycle at position 10 (via the vertex ba). We may write $W = C_1^2 \cdot C_2^2 \cdot ((aa, b), (ab, a))$.

On the other hand, the walk W_{uaa} in $dB(2)$ defined by the word uaa is not cycle-deterministic, as we may write $W_{uaa} = W_u \cdot ((ba, a), (aa, a)) = C_1^2 \cdot C_2^3 \cdot C_1$, whence W_{uaa} enters the cycle C_1 at positions 1 and 12. The cycle C_2 is now left at position 11.

We recall a lemma we need in future considerations.

Lemma 2 (Cassaigne et al. [2, Lemma 5.17]**).** *Let W_u be a walk in $dB(k-1)$ defined by $u \in Min_k(\Sigma^*)$. Let $C = (d_j)_{j=0}^{s-1}$ be a cycle occurring along W_u. Suppose further that we may write $W_u = W_0 \cdot C^r \cdot W_1$ for some walks W_1, W_2, $r \geq 2$. Then, for any $t \geqslant 0$, the word u_t corresponding to the walk $W_0 \cdot C^t \cdot W_1$ is in $Min_k(\Sigma^*)$.*

We first need a couple of observations about the properties of lexicographically least representatives of equivalence classes.

Lemma 3. *Let $k, r \geq 1$. There exists an integer $\mathcal{B}_{k,r}$ such that, for each $u \in L_{r,k}$, if $\#\Re_u(x) \geq 2$ for some $x \in F_{k-1}(u)$, then $|u|_x \leq \mathcal{B}_{k,r}$.*

Proof. Let $u \in L_{r,k}$ and assume $\#\Re_u(x) \geq 2$ for some $x \in F_{k-1}(u)$. Let us write W_u in terms of complete first returns of x in u;

$$W_u = W_0 W_1 \cdots W_{|u|_x - 1} W_{|u|_x},$$

where $\text{tail}(W_i) = \text{head}(W_i) = x$ for all $i = 1, \ldots, |u|_x - 1$. Observe now that each walk W_i, $i = 1, \ldots, |u|_x - 1$, corresponds to the complete first return to x in u. Furthermore, W_0 and $W_{|u|_x}$ do not contain the vertex x anywhere else other than what is implied above. Now, for any permutation σ of $[1, |u|_x)$, we have that $u \sim_k v_\sigma$, where v_σ is defined by the walk $W_\sigma = W_0 W_{\sigma(1)} \cdots W_{\sigma(|u|_x - 1)} W_{|u|_x}$ (compare to Example 2). The number of distinct words obtained by this method is $\binom{|u|_x - 1}{m_1, \ldots, m_k} = \frac{(|u|_x - 1)!}{m_1! \cdots m_k!}$, where $k = \#\Re_x(u)$ and $(m_i)_i = (|u|_y)_{y \in \Re_u(x)}$. This is the number of distinct permutations of words $y \in \Re_u(x)$ with multiplicities $|u|_y$, $y \in \Re(x)$. (To see that two words obtained from distinct permutations are distinct, consider their prefixes and recall the definition of a complete first return word.) We now have, by assumption, $k \geq 2$ whence $\binom{|u|_x - 1}{m_1, \ldots, m_k} \geq |u|_x - 1$. This implies $r = \#[u]_k \geq |u|_x - 1$, or in other words, $|u|_x \leq r + 1$. $\square$

Remark 1. For $r \geq 2$, we have $\mathcal{B}_{k,r} \geq 2$. Indeed, we have $u = a^{k+r-2} b a^{k-1} \in L_{r,k}$ and $\#\Re_u(a^{k-1}) = 2$.

In the case of $r = 1$, similar ideas were considered in [11]. Indeed, there it is shown that, for any $u \in L_{1,k,\Sigma}$, we have $\#\Re_u(x) \leq 1$ for all $x \in \Sigma^{k-1}$. In fact a characterization of words in $L_{1,k,\Sigma}$ is obtained in terms of a slight generalization of our notion of return words.

Lemma 4. *Let $u \in Min_k(L_{r,k})$ and assume $|u|_x > \mathcal{B}_{k,r}$. Then we may write $W_u = W \cdot C^{|u|_x - 1} \cdot W'$ for some cycle C with* $\text{tail}(C) = x$.

Proof. Assume that $|u|_x > \mathcal{B}_{k,r}$ for some $x \in \Sigma^{k-1}$. Since $|u|_x > \mathcal{B}_{k,r}$, we have $\Re_u(x) = 1$ by the above lemma. We may write W_u in terms of y, the unique complete first return to x in u; $W_u = W \cdot W_y^{|u|_x - 1} \cdot W'$, where $\text{head}(W) = \text{tail}(W_y) = \text{head}(W_y) = \text{tail}(W') = x$. We claim that W_y is a cycle. Assume the converse; there then exists a vertex z in W_y such that there are two distinct edges both with tail z along W_u. It now follows that $\#\Re_u(z) \geq 2$ and $|u|_z > \mathcal{B}_{k,r}$, a contradiction. $\square$

We now prove another lemma which already hints towards regular properties of our language.

Lemma 5. *Let $r \geq 2$. Let then u_s, for each $s \geq 0$, denote the word defined by the walk $W(s) = W \cdot C^s \cdot W'$ (in $dB(k-1)$) for some cycle C. Then $u_s \in Min_k(L_{r,k})$ for some $s \geq \mathcal{B}_{k,r}$ if and only if $u_s \in Min_k(L_{r,k})$ for all $s \in \mathbb{N}$.*

Proof. The other implication is immediate, so assume $u_s \in \text{Min}_k(L_{r,k})$ for some $s \geq \mathcal{B}_{k,r}$. The fact that $u_s \in \text{Min}_k(\Sigma^*)$ for all $s \in \mathbb{N}$ follows by Lemma 2, so it is enough to show that $u_s \in L_{r,k}$ for all $s \in \mathbb{N}$ (recall that for $r \geq 2$, we have $\mathcal{B}_{k,r} \geq 2$ by Remark 1). Without loss of generality, we may assume that $W(s)$ enters $C = (e_i)_{i=0}^{l-1}$ ($|C| = l \geq 1$) at position $|W|+1$ and that s is maximal, i.e., $W(s)$ leaves C before position $|W| + (s+1)|C|$ and, further, that $W(s)$ leaves C via vertex $y = \text{tail}(e_o)$, $0 \leq o \leq l-1$.

Observe now that $|u_s|_x \geq s$ for all $x \in V(C)$. It follows by the above lemma that for each vertex $x \in V(C)$ we have $\#\Re_{u_s}(x) = 1$ (if there were another complete first return to x in u for some $x \in V(C)$, we would have $|u_s|_x > \mathcal{B}_{k,r}$, a contradiction). Consequently, by the maximality of s, $|u_s|_{\text{tail}(e_i)} = s+1$ for all $i \in [0,o]$, and $|u_s|_{\text{tail}(e_i)} = s$ for all $i = (o,l)$. Further, each (except possibly the last) occurrence of x is followed by the same letter a_x in u_s. Moreover, the only vertex $y \in V(C)$ followed by a letter $b \neq a_y$ in u is the vertex $y = \text{tail}(e_o)$ via which $W(s)$ leaves C (the only exception is that W' is a subpath of C).

Consider the graph $G_s = G_{u_s}$ in light of Proposition 5. Let κ_s denote $\kappa_{\text{head}(W(s))}$ for each $s \geq 0$. By the above observations we conclude that any rooted spanning tree with root head$(W(s))$ of G_s contains one of the (multiple copies of the) edge e_i for each $i = [0,l) \setminus \{o\}$, and the edge $(y,b) \notin E(C)$ (unless head$(W(s)) = y$ whence no edge from y exists in such a tree). Let us compute κ_s in terms of κ_0 and s. Adding s copies to an edge e_i, $0 \leq i < o$, to G_0 increases the number of trees $(s+1)$-fold, as each tree must contain exactly one copy of this edge and there are $s+1$ to choose from. For the remainder of the vertices $z \in V(C) \setminus y$, any tree in G_s must contain some copy of the path $(e_j)_{j=o+1}^{l-1}$ which connects to a copy of a tree defined by G_0. Given s copies of each edge along this path, there are altogether s^{l-o-1} choices for the path. We conclude that $\kappa_s = \kappa_0 \cdot (s+1)^o s^{l-o-1}$. This may be expressed as $\kappa_s = \kappa_0 \cdot \prod_{x \in V(C)\setminus y}(s + |u_0|_x)$. Further, we observe that, in the product $\prod_{x \in F_{k-1}(u_s)} \frac{(|u_s|_x - 1)!}{\prod_{a \in \Sigma} |u_s|_{|xa|}!}$, the only values that vary according to s are certain values corresponding to the vertices of C. In particular, $|u_s|_x = |u_s|_{xa_x} = s + |u_s|_{x_0} \in \{s, s+1\}$ for each $x \in V(C) \setminus y$, $|u_s|_y = s+1$, and $|u_s|_{ya_y} = s$. Recall that $|u_s|_{yb} = 0$ or 1 depending on whether head$(W(s)) = y$ or not. Plugging these values in (2), we find that the following ratio equals 1 for any $s \geq 1$:

$$\frac{\#[u_s]_k}{\#[u_0]_k} = \prod_{x \in V(C)\setminus y} (s + |u_0|_x) \cdot \prod_{x \in V(C)\setminus y} \frac{1}{(s + |u_0|_x)} = 1.$$

Thus, for any choice of s, the obtained word u_s has $\#[u_s] = r$. The claim follows. □

We are now in the position to prove the key result, from which our main result follows.

Theorem 2. *The language $Min_k(L_{r,k})$ is regular.*

Proof. We claim that $\text{Min}_k(L_{r,k})$ is a finite union of languages defined by regular expressions of the form $z_0 y_1^* z_1 \cdots y_t^* z_t$.

Consider now a word $u \in \mathrm{Min}_k(L_{r,k})$ and write

$$W_u = W_0 C_1^{s_1} W_1 \cdots C_t^{s_t} W_t$$

for some paths W_i, $i = 0, \dots, t$, and some repetitions $C_i^{s_i}$ of cycles C_i, $i = 1, \dots, t$, such that W_u enters cycle C_i at position $|W_0| + \sum_{j=1}^{i-1} |C_j^{s_j} W_j| + 1$ for all $1 \le i \le t$ and leaves C_i before entering C_{i+1}. Now u may be written as

$$u = \mathrm{tail}(W_0) \cdot \mathrm{label}(W_0 C_1^{s_1} W_1 \cdots C_t^{s_t} W_t) = z_0 y_1^{s_1} z_1 \cdots y_t^{s_t} z_t,$$

where $z_0 = \mathrm{tail}(W_0) \cdot \mathrm{label}(W_0)$, $y_i = \mathrm{label}(C_i)$, and $z_i = \mathrm{label}(W_i)$ for $1 \le i \le t$. The above lemma asserts that, if $s_i \ge \mathcal{B}_{k,r}$, then

$$L(z_0 y_1^{s_1} z_1 \cdots y_i^{*} z_i \cdots y_t^{s_t} z_t) \subseteq L_{r,k}. \tag{4}$$

By repeating the above, we may replace all exponents s_j satisfying $s_j \ge \mathcal{B}_{k,r}$ with $*$ in (4).

Let L be the union of all the languages obtained as above from words $u \in \mathrm{Min}_k(L_{r,k})$ satisfying $|u|_x \le \mathcal{B}_{k,r} + 1$ for all $x \in \Sigma^{k-1}$. These words are bounded in length, so that the union is finite. Clearly $L \subseteq \mathrm{Min}_k(L_{r,k})$ by the above observation. We claim that $\mathrm{Min}_k(L_{r,k}) \subseteq L$.

Indeed, let $u \in \mathrm{Min}_k(L_{r,k})$. If $|u|_x > B_{k,r} + 1$, Lemma 4 ensures that we have $W_u = W_0 W_y^{|u|_x - 1} W_1$ for y being the unique complete first return to x in u. Further W_y is a cycle. If W_u does not enter W_y at position $|W_0| + 1$, we may extend the cycle to the left and right to obtain $W_0' W_{y'}^t W_1'$, where $t \in \{|u|_x - 1, |u|_x\}$. By the above lemma we may reduce the number of repetitions of W_y' to obtain a word u' for which $|u'|_x \le \mathcal{B}_{k,r} + 1$ and u is in the language defined by u' as in (4). If $|u'|_{x'} > \mathcal{B}_{k,r} + 1$ for some $x' \in \Sigma^{k-1}$, we may repeat the above for u' to obtain a word u'' having $|u''|_{x'} \le \mathcal{B}_{k,r} + 1$ and such that u and u' are in the language defined by u'' as in (4). This can be continued until we obtain a word v such that $|v|_x \le \mathcal{B}_{k,r} + 1$ for all $x \in \Sigma^{k-1}$ and u is contained in the language defined by v as in (4). We thus have $u \in L$, which concludes the proof. □

Proof (of Theorem 1). The language $\mathrm{Min}_k(L_{r,k})$ is regular. Since the operation R_k preserves regularity by Proposition 4, by applying finitely many iterations of R_k, we have that $L_{r,k} = R_k^r(\mathrm{Min}_k(L_{r,k}))$ is regular. □

4 Conclusions

In this note, we continued to analyze the structure of the k-Abelian equivalence classes, in particular in the framework of regularity. In [2] we concluded, as a consequence of the k-switching lemma of [11], that the set of singleton classes is a regular language. Therein, this was extended to the union of all two-element classes as well. This was based, on one hand, on the regularity of the union of lexicographically least representatives of the equivalence classes and, on the other hand, on strong closure properties of the family of regular languages.

The approach does not extend, at least immediately, and indeed fails for the first attempt, to the union of larger (fixed-size) classes. To show that also these classes are regular, we developed new techniques. This is the content of this note.

All these regular languages are algorithmically constructable. However, in practice, this can be done only in very restricted cases. Indeed, the analysis of the size of the automata of the mentioned regular languages remains for future research.

Once the regularity of the above languages is established, the well-known techniques in rational power series allow to determine the corresponding enumeration functions. This was discussed in [2]. In the current work, Theorem 2 implies the following result:

Corollary 2. *For each $k \geq 1$, the sequence of numbers of k-Abelian equivalence classes with cardinality r of length n is a rational sequence.*

In this spirit, the first few sequences of minimal elements were shown in the On-Line Encyclopedia of Integer Sequences [17]. In more detail, the sequences

$$\mathcal{P}_{k,m}(n) = \#\{[w]_k \mid |w| = n\}$$

were determined for $k = 2, 3$ in the binary alphabet. Similarly, a first few sequences considered in this note, that is, of the sequences

$$\mathcal{S}_{r,k,m}(n) = \#\{[w]_k \mid |w| = n, \#[w]_k = r\}$$

might be worth including in the encyclopedia.

Acknowledgements. We would like to thank Julien Cassaigne and Svetlana Puzynina for fruitful and insightful discussions on the topic.

References

1. de Bruijn, N.G.: Acknowledgement of priority to C. Flye Sainte-Marie on the counting of circular arrangements of 2^n zeros and ones that show each n-letter word exactly once. Technical report. (EUT report. WSK, Department of Mathematics and Computing Science), vol. 75-WSK-06, Technische Hogeschool Eindhoven, Netherlands (1975)
2. Cassaigne, J., Karhumäki, J., Puzynina, S., Whiteland, M.A.: k-abelian equivalence and rationality. Fundamenta Informaticae **154**(1–4), 65–94 (2017)
3. Cassaigne, J., Karhumäki, J., Puzynina, S., Whiteland, M.A.: k-abelian equivalence and rationality. In: Brlek, S., Reutenauer, C. (eds.) DLT 2016. LNCS, vol. 9840, pp. 77–88. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53132-7_7
4. Cassaigne, J., Karhumäki, J., Saarela, A.: On growth and fluctuation of k-abelian complexity. Eur. J. Comb. **65**(Suppl C), 92–105 (2017)
5. Eilenberg, S.: Automata, Languages, and Machines, vol. A. Academic Press Inc., New York (1974)

6. Gruber, H., Holzer, M.: From finite automata to regular expressions and back - a summary on descriptional complexity. Int. J. Found. Comput. Sci. **26**(08), 1009–1040 (2015)
7. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 1st edn. Addison-Wesley Publishing Co., Inc., Boston (1979)
8. Huova, M., Saarela, A.: Strongly k-abelian repetitions. In: Karhumäki, J., Lepistö, A., Zamboni, L. (eds.) WORDS 2013. LNCS, vol. 8079, pp. 161–168. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40579-2_18
9. Karhumäki, J.: Generalized Parikh mappings and homomorphisms. Inf. Control **47**(3), 155–165 (1980)
10. Karhumäki, J., Puzynina, S.: On k-abelian palindromic rich and poor words. In: Shur, A.M., Volkov, M.V. (eds.) DLT 2014. LNCS, vol. 8633, pp. 191–202. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09698-8_17
11. Karhumäki, J., Puzynina, S., Rao, M., Whiteland, M.A.: On cardinalities of k-abelian equivalence classes. Theoret. Comput. Sci. **658**(Part A), 190–204 (2017). Formal Languages and Automata: Models, Methods and Application in honour of the 70th Birthday of Antonio Restivo
12. Karhumäki, J., Saarela, A., Zamboni, L.Q.: On a generalization of abelian equivalence and complexity of infinite words. J. Comb. Theory, Ser. A **120**(8), 2189–2206 (2013)
13. Lothaire, M.: Combinatorics on Words, Encyclopedia of Mathematics and Its Applications. Advanced Book Program, World Science Division, vol. 17. Addison-Wesley, Boston (1983)
14. Rao, M., Rosenfeld, M.: Avoidability of long k-abelian repetitions. Math. Comput. **85**(302), 3051–3060 (2016)
15. Rauzy, G.: Suites á termes dans un alphabet fini. Seminaire de Théorie des Nombres de Bordeaux **12**, 1–16 (1982–1983)
16. Salomaa, A., Soittola, M.: Automata-Theoretic Aspects of Formal Power Series. Texts and Monographs in Computer Science. Springer, New York (1978). https://doi.org/10.1007/978-1-4612-6264-0
17. Sloane, N.J.A.: The on-line encyclopedia of integer sequences. Published electronically at https://oeis.org. Accessed 1 Feb 2018

Reaction Systems, Transition Systems, and Equivalences

Jetty Kleijn[1], Maciej Koutny[2], Łukasz Mikulski[3], and Grzegorz Rozenberg[1,4](✉)

[1] LIACS, Leiden University, P.O. Box 9512, 2300 Leiden, RA, The Netherlands
{h.c.m.kleijn,g.rozenberg}@liacs.leidenuniv.nl

[2] School of Computing, Newcastle University, Newcastle upon Tyne NE1 7RU, UK
maciej.koutny@ncl.ac.uk

[3] Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, 87-100 Toruń, Poland
lukasz.mikulski@mat.umk.pl

[4] Department of Computer Science, University of Colorado Boulder, 430 UCB, Boulder, CO 80309-0430, USA

Abstract. Reaction systems originated as a formal model for processes inspired by the functioning of the living cell. The underlying idea of this model is that the functioning of the living cell is determined by the interactions of biochemical reactions and these interactions are based on the mechanisms of facilitation and inhibition. Since their inception, reaction systems became a well-investigated novel model of computation. Following this line of research, in this paper we discuss a systematic framework for investigating a whole range of equivalence notions for reaction systems. Some of the equivalences are defined directly on reaction systems while some are defined through transition systems associated with reaction systems. In this way we establish a new bridge between reaction systems and transition systems. In order to define equivalences which capture various ways of interacting with an environment, we also introduce models of the environment which evolve in a finite-state fashion.

Keywords: Reaction system · Living cell · Natural computing
Interactive computation · Transition system
Behavioural equivalences · Bisimulation · Context controller

1 Introduction

Natural computing is an interdisciplinary research area concerned with human-designed computing inspired by nature and with computing taking place in nature (see, e.g., [22,29]). The former strand investigates models and computational techniques inspired by nature, while the latter investigates, in terms of information processing, phenomena taking place in nature. Clearly, the two research strands are not disjoint.

© Springer Nature Switzerland AG 2018
H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 63–84, 2018.
https://doi.org/10.1007/978-3-319-98355-4_5

Biomolecular computation is a topic of intense research in natural computing. Within the former strand this research focuses on constructing, either in vitro or in vivo, various building blocks of computing devices, as well as on designing novel, nature inspired algorithms. Within the latter strand this research is concerned with establishing how biocomputations drive natural processes.

The original motivation behind reaction systems (see, e.g., [8,12,17]), the subject of this paper, falls into this second strand of research. Reaction systems were proposed as a formal model for the investigation of the functioning of the living cell, where this functioning is viewed in terms of formal processes resulting from interactions between biochemical reactions taking place in the living cell. The key feature of these interactions is that they are driven by two mechanisms, facilitation and inhibition: the (products of the) reactions may facilitate or inhibit each other. The basic model of reaction systems abstracts from various (technical) features of biochemical reactions to such extent that it becomes a qualitative rather than a quantitative model. However, it takes into account the basic bioenergetics (flow of energy) of the living cell, see [24], and it also takes into account that the living cell is an open system and its behaviour is influenced by its environment.

Although the model of reaction systems was inspired by biology, the research on reaction systems is driven by both biological considerations (see, e.g., [2,3,6,7,10,18,19]) and considerations concerned with the understanding of computations taking place in reaction systems which formalise dynamic processes instigated by the interactions of biochemical reactions in the living cell (see, e.g., [8,11,13,14,16,20,30–32]). As a matter of fact, reaction systems turned out to be an interesting novel model of interactive computation.

This paper follows this computational line of research and its goal is to present a systematic framework for considering various notions of behavioural equivalence for reaction systems. Within this framework we consider equivalences defined directly on reaction systems as well as equivalences defined through transition systems associated with them. In this way we establish a new bridge to transition systems which are central constructs of theoretical computer science used extensively to investigate behaviours of dynamic processes (see, e.g., [1,4,23,26]).

By associating various sorts of transition systems with reaction systems and then by considering various notions of equivalence for these transition systems, we capture a wide range of properties of dynamic processes taking place in reaction systems.

Reaction systems are models of interactive computations, the interaction of a reaction system with its environment constitutes a key feature of this model. In order to account for these interactions in our framework, we introduce models of environment that evolve in a finite-state fashion.

The paper is self-contained in the sense that (1) it introduces basic notions concerning reaction systems and their dynamic processes (together with the original biological motivation), and (2) it introduces basic notions concerning transition systems.

The paper is organised as follows.

In Sects. 2 and 3 we recall basic notions of reaction systems to be used in this paper, also recalling the original biological motivation.

In Sect. 4 we consider two basic equivalences defined directly on reaction systems:

- the standard functional equivalence which is local in the sense that it compares the effect of two systems on individual states (there are finitely many of them), and
- the new process equivalence which is global in the sense that it compares the sets of processes of two systems (there are infinitely many of them).

Both equivalences considered in Sect. 4 are quite restrictive, and so in Sect. 5 we consider two ways of making these equivalences less restrictive:

- by comparing two systems only w.r.t. processes that begin in a designated set of possible initial states, and
- by making only some entities visible to an outside observer.

In Sect. 6 we recall basic notions of transition systems to be used in this paper, and then in Sect. 7 we establish a method of associating transition systems with reaction systems.

Reaction systems are models of interactive computing, the interaction of a system with its environment is the central feature of processes of reaction systems. As a preparatory step for incorporating this interaction in equivalence notions, in Sect. 8 we formalise an intuitive concept of a 'finite-state' environment, i.e., an environment which evolves in a finite-state fashion. It is an intermediate between the two sorts of environments mostly considered in the literature: all possible contexts are allowed and no context is allowed (i.e., a reaction system is considered to be a 'closed' system). The resulting formal constructs are called *context controllers* and they can control the environment either totally independently of reaction systems (operating in it) or in a fashion that is dependent on (aware of) the current states of reaction systems. Incorporating context controllers into the behaviour of reaction systems leads to new definitions of their processes.

In Sect. 9 we consider equivalences defined not directly on reaction systems, but rather through transition systems associated with them.

The notion of bisimulation is a central notion of the theory of concurrent systems [26,27,35] for comparing their dynamic behaviours. In Sect. 10 we transfer this notion to reaction systems using their representations by transition systems.

The discussion in Sect. 11 concludes this paper.

2 Reactions and Reaction Systems

In this section and in Sect. 3, we recall the basic notions of reaction systems needed in this paper. Most of them are taken from [5,12,16].

The formal notion of a reaction reflects the basic intuition of a biochemical reaction – it will take place if all its reactants are present and none of its inhibitors is present; when a reaction takes place it creates its products.

Definition 1. *A* reaction *is a triple $b = (R, I, P)$ such that R, I, P are finite nonempty sets with $R \cap I = \varnothing$.*

The sets R, I, P are called the *reactant set of* b, the *inhibitor set of* b, and the *product set of* b, respectively – they are also denoted as R_b, I_b, and P_b, respectively. If $R, I, P \subseteq Z$ for a finite set Z, then we say that b is a *reaction in* Z and we use $rac(Z)$ to denote the set of all reactions in Z – note that $rac(Z)$ is finite.

Since R and I are nonempty and disjoint, a finite set Z as above must have at least 2 elements – we refer to such finite sets as *background sets.*

To describe the effect of a set of reactions on a state (of a biochemical system), we first define the effect of a single reaction.

Definition 2. *Let Z be a background set, let $X \subseteq Z$, and let $b \in rac(Z)$. Then b is* enabled by X*, denoted by $en_b(X)$, if $R_b \subseteq X$ and $I_b \cap X = \varnothing$. The* result of b on X*, denoted by $res_b(X)$, is defined by $res_b(X) = P_b$ if $en_b(X)$, and $res_b(X) = \varnothing$ otherwise.*

Here the finite set X is a formal representation of a state (i.e., the set of biochemical entities currently present in the given biochemical system). Then b is enabled by X if X separates R_b from I_b, meaning that all reactants from R_b are present in X and none of the inhibitors from I_b is present in X. When b is enabled by X, it contributes its product P_b to the successor state; otherwise it does not contribute anything to the successor state.

The effect of a set of reactions on a state is cumulative, which is formally defined as follows.

Definition 3. *Let Z be a background set, let $X \subseteq Z$, and let $B \subseteq rac(Z)$. The* result of B on X*, denoted by $res_B(X)$, is defined by $res_B(X) = \bigcup\{res_b(X) \mid b \in B\}$.*

Note that if the transition from a current state X to its successor is determined only by the results of reactions (i.e., there is no influence of the environment, which we will consider later on), then the successor state consists only of the entities produced by the reactions enabled in the current state. This implies that in the transition from the current state to its successor state an *entity from X vanishes unless it is sustained/produced by a reaction.* This is the *non-permanency property* which reflects the basic bioenergetics of the living cell: *without a flow/supply of energy the living cell disintegrates, but the use/absorption of energy by the living cell is realised through biochemical reactions* (see, e.g., [24]). Thus an entity vanishes within *one* state transition unless it is produced by a reaction. This non-permanency property is a major difference between reaction systems and models considered in the theory of computation (see, e.g., [1] and

[21]). Also finite duration of entities in reaction systems – corresponding to their presence in several consecutive states – which takes into account decay time, is considered in the literature (see, e.g., [6]).

There is another notable aspect of Definition 3. If a, b are two reactions from B enabled by X, then both of them will take place even if $R_a \cap R_b \neq \varnothing$. Hence there is no notion of conflict between reactions even if they need to share reactants. This is the property of the *threshold nature of resources: either an entity is available and then there is enough of it, or it is not available.* The threshold nature of resources (no conflict) property is a major difference with structural models of concurrency, such as, e.g., Petri nets [28]. This property reflects the *level of abstraction* adopted for the formulation of the basic model of reaction systems: one does not count concentrations of entities/molecules to infer from these which reactions can/will be applied. One operates on a higher level of abstraction: assuming that the cell is running/functioning, the aim is to understand the ongoing processes. This level of abstraction can be compared with the level of abstraction of the standard models of computation in computer science, such as Turing machines and finite automata. These standard models turned out to be very successful in understanding computational processes running on electronic computers, and yet nothing in these models takes into account the electronic/quantitative properties of the underlying hardware. It is simply assumed that the underlying electronics/hardware functions 'well' and then the goal is to understand processes running on (implemented by) this hardware. Similarly, we want to understand the processes resulting from interactions in the living cell and at this stage we abstract from the underlying 'hardware properties' of the living cell. Thus: *the basic model of reaction systems is qualitative rather than quantitative – in particular, there is no counting.*

With the formal notion of a reaction and its effect on states established, we can now proceed to formally define reaction systems as an abstract model of the interactions of biochemical reactions in the living cell.

Definition 4. *A* reaction system *is an ordered pair $\mathcal{A} = (S, A)$, where S is a background set and A is a nonempty finite subset of $rac(S)$.*

The set S is called the *background set of* $\mathcal{A}$, and its elements are called the *entities of* $\mathcal{A}$ – they represent molecular entities (e.g., atoms, ions, molecules) that may be present in the states of the biochemical system (e.g., the living cell). The set A is called the *set of reactions of* $\mathcal{A}$; clearly A is finite (as S is finite).

The subsets of S are called the *states of* $\mathcal{A}$. Given a state $X \subseteq S$, the *result of* $\mathcal{A}$ on X, denoted by $res_{\mathcal{A}}(X)$, is defined by $res_{\mathcal{A}}(X) = res_{A}(X)$.

Thus a reaction system is essentially a set of reactions. We also specify the background set which consists of entities needed for defining the reactions and for reasoning about the reaction system (see the definition of an interactive process below). There are no 'structures' involved in reaction systems (such as, e.g., the tape of a Turing machine). Finally, note that this is a *strictly finite model* – the size of any state is a priori restricted (by the size of the background set).

3 Processes of Reaction Systems

The model of reaction systems formalises the 'static structure' of the living cell as the set of all reactions that can take place in the cell (together with the set of underlying entities). Since the original motivation behind this model was to understand the functioning of living cells, one is really interested in the processes instigated by the interactions of these reactions. They are formalised as follows.

Definition 5. *Let $\mathcal{A} = (S, A)$ be a reaction system.*

An interactive process *in $\mathcal{A}$ is an ordered pair $\pi = (\gamma, \delta)$ of finite sequences such that $\gamma = C_0, \ldots, C_n$ and $\delta = D_0, \ldots, D_n$, for some $n \geq 1$, where $C_0, \ldots, C_n \subseteq S$, $D_0, \ldots, D_n \subseteq S$, and $D_i = res_{\mathcal{A}}(D_{i-1} \cup C_{i-1})$, for all $i \in \{1, \ldots, n\}$.*

The sequence γ is the *context sequence of* π, the sequence δ is the *result sequence of* π, and the sequence $\tau = W_0, \ldots, W_n$, where, for all $i \in \{0, \ldots, n\}$, $W_i = C_i \cup D_i$, is the *state sequence of* π, with $W_0 = C_0 \cup D_0$ the *initial state* of π. We let $STS(\pi)$ denote the state sequence of π. By $PROC(\mathcal{A})$ we denote the set of interactive processes of $\mathcal{A}$ and, for $X \subseteq S$, $PROC_X(\mathcal{A})$ is the set of interactive processes of $\mathcal{A}$ with initial state X.

The dynamic process formalised by an interactive process π begins in its initial state. The reactions of $\mathcal{A}$ enabled by this initial state W_0 produce the result set D_1, which together with the context set C_1 forms the successor state $W_1 = res_{\mathcal{A}}(W_0) \cup C_1$. This formation of successor states is iterated: $W_i = res_{\mathcal{A}}(W_{i-1}) \cup C_i$, resulting in the state sequence $STS(\pi) = W_0, \ldots, W_n$.

An interactive process may be visualised by a three-row representation, where the first row represents the context sets and is labelled by 'γ', the second row represents result sets and is labelled by 'δ', and the third row represents states and is labelled by 'τ'. Such a representation looks as follows:

$$\begin{array}{lccccc} \gamma : & C_0 & C_1 & \ldots & C_{n-1} & C_n \\ \delta : & D_0 & D_1 & \ldots & D_{n-1} & D_n \\ \tau : & W_0 & W_1 & \ldots & W_{n-1} & W_n \end{array} \tag{1}$$

Note that the context sequence γ together with the initial state W_0 uniquely determine π, because γ and D_0 uniquely determine π (through the result function $res_{\mathcal{A}}$). The context sequence formalises the fact that the *living cell is an open system* in the sense that its behaviour is influenced by its environment (the 'rest' of a bigger system).

If, for all $i \in \{0, \ldots, n\}$, $C_i \subseteq D_i$, then we say that π is *context-independent*: whatever C_i adds to D_i, has already been produced by the system (is included in the result D_i) or perhaps C_i adds nothing at all ($C_i = \varnothing$). If π is context-independent, then (in its analysis) we may as well assume that each C_i, for $i \in \{0, \ldots, n\}$, adds nothing, hence the context sequence consists of empty sets C_i only. Clearly, if π is context-independent, then the initial state $W_0 = D_0$ determines π by the repeated application of $res_{\mathcal{A}}$.

When reaction systems were introduced, the definition of an interactive process assumed that $D_0 = \varnothing$. This is not the case anymore and also in this paper this condition is dropped, in this way a suffix (of length at least 2) of an interactive process is also an interactive process. As a consequence we assume that in a context-independent interactive process also $C_0 = \varnothing$.

We will use $\mathit{CIPROC}(\mathcal{A})$ to denote the set of context-independent interactive processes of $\mathcal{A}$ and $\mathit{CIPROC}_{W_0}(\mathcal{A})$ to denote the set of context-independent interactive processes of $\mathcal{A}$ with initial state W_0.

4 Direct Equivalences

In this section we consider equivalences for reaction systems defined directly through their reactions and interactive processes. First we recall the standard functional equivalence that was introduced right from the start in [17] for reaction systems.

Definition 6. *Reaction systems $\mathcal{A}$ and $\mathcal{B}$ with the same background set S are* (functionally) equivalent*, denoted by $\mathcal{A} \sim \mathcal{B}$, if $\mathit{res}_{\mathcal{A}}(X) = \mathit{res}_{\mathcal{B}}(X)$ for all $X \subseteq S$.*

Note that, strictly speaking, we should have written $\sim_S$ in the above definition. However to simplify our notation we will use $\sim$ whenever this will not lead to confusion.

The concept of interactive processes induces in a natural way the following notion of equivalence for reaction systems.

Definition 7. *Reaction systems $\mathcal{A}$ and $\mathcal{B}$ with the same background set are* process equivalent*, denoted by $\mathcal{A} \sim_{PROC} \mathcal{B}$, if $\mathit{PROC}(\mathcal{A}) = \mathit{PROC}(\mathcal{B})$.*

The relation $\sim_{PROC}$ is clearly an equivalence relation. Moreover, it turns out that $\sim$ which is defined locally (just on all subsets of the background set) and $\sim_{PROC}$ which is defined globally (over all, infinitely many, processes) coincide.

Proposition 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S. Then $\mathcal{A} \sim_{PROC} \mathcal{B}$ if and only if $\mathcal{A} \sim \mathcal{B}$.*

Proof. Suppose that $\mathcal{A} \sim_{PROC} \mathcal{B}$. Thus $\mathit{PROC}(\mathcal{A}) = \mathit{PROC}(\mathcal{B})$. Let $X \subseteq S$ and consider a process $\pi \in \mathit{PROC}(\mathcal{A})$ with initial state X and context sequence $\varnothing, \varnothing$. Thus the result sequence of π is $X, \mathit{res}_{\mathcal{A}}(X)$. Since π is also an interactive process in $\mathcal{B}$, it follows that $\mathit{res}_{\mathcal{B}}(X) = \mathit{res}_{\mathcal{A}}(X)$. Hence, for each $X \subseteq S$, $\mathit{res}_{\mathcal{B}}(X) = \mathit{res}_{\mathcal{A}}(X)$ and consequently $\mathcal{A} \sim \mathcal{B}$.

Now suppose that $\mathcal{A} \sim \mathcal{B}$. Hence, $\mathit{res}_{\mathcal{A}}(X) = \mathit{res}_{\mathcal{B}}(X)$ for all $X \subseteq S$, which directly implies that $\mathcal{A} \sim_{PROC} \mathcal{B}$. □

Similarly, we can also show that one does not need the full information from the interactive processes of reaction systems to establish their (process) equivalence – it suffices to consider only their state sequences.

Proposition 2. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S. Then $\mathcal{A} \sim \mathcal{B}$ if and only if $STS(PROC(\mathcal{A})) = STS(PROC(\mathcal{B}))$.*

Proof. The only-if-direction follows immediately from Proposition 1.

So let us assume that $STS(PROC(\mathcal{A})) = STS(PROC(\mathcal{B}))$. Let $X \subseteq S$ and consider $\pi \in PROC(\mathcal{A})$ with some initial state W and context sequence $\varnothing, S, X, \varnothing$. Then $STS(\pi) = W, S, X, res_{\mathcal{A}}(X)$. Hence there exists an interactive process $\pi' \in PROC(\mathcal{B})$ such that $STS(\pi') = W, S, X, res_{\mathcal{A}}(X)$. This implies that $res_{\mathcal{B}}(X) \subseteq res_{\mathcal{A}}(X)$. There also exists an interactive process in $\mathcal{B}$ with state sequence $W, S, X, res_{\mathcal{B}}(X)$. Consequently $W, S, X, res_{\mathcal{B}}(X)$ is the state sequence of an interactive process in $\mathcal{A}$, which implies that $res_{\mathcal{A}}(X) \subseteq res_{\mathcal{B}}(X)$. It follows that $res_{\mathcal{B}}(X) = res_{\mathcal{A}}(X)$. Hence, for each $X \subseteq S$, $res_{\mathcal{B}}(X) = res_{\mathcal{A}}(X)$ and consequently $\mathcal{A} \sim \mathcal{B}$. □

Corollary 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set. Then $\mathcal{A} \sim \mathcal{B}$ if and only if $\mathcal{A} \sim_{PROC} \mathcal{B}$ if and only if $STS(PROC(\mathcal{A})) = STS(PROC(\mathcal{B}))$.*

5 Relaxing Direct Equivalences

The conditions for equivalence as captured by $\sim$ and $\sim_{PROC}$ may be rightly regarded as too restrictive, because this equivalence entails 'always the same effect' of the reactions of the two reaction systems under consideration. In this section we formulate several different ways of relaxing this constraint.

Initial States

The first idea to relax $\sim_{PROC}$ is to compare reaction systems only with respect to interactive processes that begin in certain designated states.

Definition 8. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S. Let Z be a nonempty subset of 2^S. Then $\mathcal{A}$ and $\mathcal{B}$ are* process equivalent w.r.t. *Z, denoted by $\mathcal{A} \sim_Z \mathcal{B}$, if $PROC_X(\mathcal{A}) = PROC_X(\mathcal{B})$ for every $X \in Z$.*

The relation $\sim_Z$ is clearly an equivalence relation. Moreover, $\sim_{2^S}$ is nothing but $\sim_{PROC}$. What is perhaps rather unexpected, is that process equivalence is guaranteed for two reaction systems whenever they are equivalent with respect to some nonempty set of initial states. Any subset $Z \subseteq 2^S$ such that $\mathcal{A} \sim_Z \mathcal{B}$ is thus a 'test subset' for process equivalence.

Proposition 3. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S. Then $\mathcal{A} \sim_{PROC} \mathcal{B}$ if and only if there exists a nonempty $Z \subseteq 2^S$ such that $\mathcal{A} \sim_Z \mathcal{B}$.*

Proof. Since $\sim_{PROC} = \sim_{2^S}$, we only need to prove the if-direction of the statement. Assume that Z is a nonempty subset of 2^S such that $\mathcal{A} \sim_Z \mathcal{B}$. Let $X \in Z$ and let Y be an arbitrary subset of S. Consider the interactive process π in $\mathcal{A}$

with context sequence $\varnothing, S, Y, \varnothing$ and result sequence $X, res_{\mathcal{A}}(X), \varnothing, res_{\mathcal{A}}(Y)$. From $\mathcal{A} \sim_Z \mathcal{B}$ and $X \in Z$, it follows that π is also an interactive process in $\mathcal{B}$ and so $res_{\mathcal{A}}(Y) = res_{\mathcal{B}}(Y)$. Hence, $res_{\mathcal{B}}(Y) = res_{\mathcal{A}}(Y)$, for each $Y \subseteq S$, and consequently $\mathcal{A} \sim \mathcal{B}$. Thus, by Proposition 1, $\mathcal{A} \sim_{PROC} \mathcal{B}$. $\square$

Thus we may conclude that restricting the set of potential initial states has no effect on the equivalences considered so far.

Corollary 2. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S. Then $\mathcal{A} \sim \mathcal{B}$ if and only if $\mathcal{A} \sim_{PROC} \mathcal{B}$ if and only if $\mathcal{A} \sim_Z \mathcal{B}$ for some nonempty $Z \subseteq 2^S$.*

At this point it is worthwhile to observe that also in the case of a restricted set of potential initial states, the state sequences of those interactive processes that are taken into consideration are sufficient to establish (process) equivalence. In fact, even one initial state is sufficient.

Proposition 4. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S. Let $X \subseteq S$ be such that $STS(PROC_X(\mathcal{A})) = STS(PROC_X(\mathcal{B}))$. Then $\mathcal{A} \sim \mathcal{B}$.*

Proof. The statement can be proved along the lines of the proof of the if-direction of Proposition 2 where the choice of an initial state was an arbitrary one. $\square$

In combination with the definition of state sequences of processes, Proposition 3 and Corollary 2, this leads to the following result.

Corollary 3. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S. Then $\mathcal{A} \sim \mathcal{B}$ if and only if $\mathcal{A} \sim_{PROC} \mathcal{B}$ if and only if $STS(PROC_X(\mathcal{A})) = STS(PROC_X(\mathcal{B}))$ for some $X \subseteq S$.*

Observable Entities

Another possibility to relax $\sim_{PROC}$ is to relate it to a subset of the background set of the reaction systems under consideration. Intuitively, such subset is the part of the background set that is 'visible' to an observer.

First we need some notation.

As usual, for any set U, we use U^* to denote the set of finite sequences of elements from U; the empty sequence is denoted by λ.

Let U and W be finite sets such that $U \subseteq W$. The projection function $proj_{W,U}$ from $(2^W)^*$ onto $(2^U)^*$ is defined as follows.

- For $W' \subseteq W$, $proj_{W,U}(W') = W' \cap U$.
- This is lifted to sequences of sets in a homomorphic way: $proj_{W,U}(W_1 W_2 \dots W_m) = proj_{W,U}(W_1) proj_{W,U}(W_2) \dots proj_{W,U}(W_m)$, for every sequence $W_1 \dots W_m$ with $m \geq 1$ and $W_i \subseteq W$, for $i \in \{1, \dots, m\}$, and $proj_{W,U}(\lambda) = \lambda$.

In what follows, we may omit the subscript W whenever it is clear from the context – then we simply write $proj_U$.

Definition 9. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S and let $Y \subseteq S$. Then $\mathcal{A}$ and $\mathcal{B}$ are Y-*projection equivalent*, denoted by $\mathcal{A} \sim^Y B$, if $proj_Y(STS(PROC(\mathcal{A}))) = proj_Y(STS(PROC(\mathcal{B})))$.*

The relation $\sim^Y$ is clearly an equivalence relation. Note that all reaction systems over the background set S are $\varnothing$-projection equivalent. Moreover, by Corollary 1, $\sim^S$ is nothing but $\sim_{PROC}$.

Proposition 5. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S, and let $X \subseteq Y \subseteq S$. Then $\mathcal{A} \sim^Y \mathcal{B}$ implies $\mathcal{A} \sim^X \mathcal{B}$.*

Proof. Follows directly from the definitions. □

It should be noted here, that in contrast to process equivalence (with respect to some set of designated initial states), projection equivalence is phrased in terms of state sequences rather than the interactive processes themselves. A main reason for doing this is that projection of interactive processes would also involve a projection of the context (thrown in by the environment).

Initial States and Observable Entities

To conclude this section, we define an equivalence relation for reaction systems by combining the restriction to designated initial states and the projection onto a subset of the background set.

Definition 10. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S. Let Z be a nonempty subset of 2^S and let $Y \subseteq S$. Then $\mathcal{A}$ and $\mathcal{B}$ are Y-*projection equivalent w.r.t. *Z, denoted $\mathcal{A} \sim^Y_Z \mathcal{B}$, if $proj_Y(STS(PROC_X(\mathcal{A}))) = proj_Y(STS(PROC_X(\mathcal{B})))$, for every $X \in Z$.*

The relation $\sim^Y_Z$ is an equivalence relation and $\sim^S_{2^S} = \sim_{PROC}$.

Proposition 6. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S. Let $X \subseteq Y \subseteq S$ and let $\varnothing \neq W \subseteq Z \subseteq 2^S$. Then $\mathcal{A} \sim^Y_Z \mathcal{B}$ implies $\mathcal{A} \sim^X_W \mathcal{B}$.*

Proof. Follows directly from the definitions. □

6 Transition Systems

Transition systems are a central and important model in the theory of computation. They form an established way [1,4,23] of capturing the behaviour of a dynamic system. Often infinite behaviour (e.g., given in the form of a language consisting of sequences of action symbols) can be represented by a finite transition system. Transition systems have been used as a semantical model for both sequential and concurrent systems.

Transition systems provide a means of defining and checking behavioural equivalence of different systems. There are several different ways in which behavioural equivalence of two transition systems can be expressed, depending on the application area and the desired degree of behavioural closeness. One possibility is to consider two transition systems equivalent if the languages they generate are the same [1,23]. Another, much more demanding, possibility is to consider two transition systems equivalent if there is a bisimulation relation between their states which is preserved by simulating transitions [26,27,35].

After introducing transition systems (in this section) and relating reaction systems and transition systems (in the next section), we will consider bisimulation-based notions of equivalence for reaction systems.

It should be mentioned here that there is a *direct* relationship between reaction systems and transition systems that has been studied in the literature, see e.g., [16,20]. One can simulate finite transition systems by reaction systems and one can simulate reaction systems by finite transition systems, see e.g., [5,16]. Also the state space of a rs $\mathcal{A} = (S, A)$ is a finite directed graph (transition system) with states of $\mathcal{A}$ and its nodes and edges determined either by the result function $res_{\mathcal{A}}$ (in case of context-independent interactive processes) or by the joint effect of $res_{\mathcal{A}}$ and the context sets (in case of unrestricted interactive processes). The isomorphism problem for these graphs and its relationship to the functional equivalence was studied in [20].

In a nutshell, a transition system is a directed graph, where the nodes represent global (system) states and the arcs (transitions) can be labelled with information about the actions/conditions associated with the changes of the state of the system. An evolution of the system (from a certain state) corresponds to a directed path through the graph from the node corresponding to the given state. Thus evolutions can be understood and analysed by looking at ordered sequences of states and (labels of) transitions. Transition systems may be 'initialised', which means that they have a distinguished initial state from which all possible evolutions of the system start.

Definition 11. *1. A* transition system *is an ordered pair $\mathcal{T} = (Q, V)$, where Q is a finite set and $V \subseteq Q \times Q$.*
2. An initialised transition system *is a triple $\mathcal{T} = (Q, V, q_0)$ such that (Q, V) is a transition system and $q_0 \in Q$.*

Definition 12. *Let L be a finite set.*

1. *A* labelled transition system *over L is an ordered pair $\mathcal{T} = (Q, V)$, where Q is a finite set and $V \subseteq Q \times L \times Q$.*
2. *An* initialised labelled transition system *over L is a triple $\mathcal{T} = (Q, V, q_0)$ such that (Q, V) is a labelled transition system over L and $q_0 \in Q$.*

The set of state sequences of a transition system $\mathcal{T} = (Q, V)$, denoted by $sts(\mathcal{T})$, is the set of all finite sequences $\theta = q_0, q_1, \ldots, q_n$ $(n \geq 0)$ of elements of Q such that $(q_i, q_{i+1}) \in V$ for all $0 \leq i < n$. Moreover, we denote by $sts_q(\mathcal{T})$ the set of all state sequences of $\mathcal{T}$ which start from a given state q.

7 Associating Transition Systems with Reaction Systems

In order to capture a range of behavioural equivalences for reaction systems, we first introduce a representation of their interactive processes as well as their context-independent interactive processes in terms of a suitable class of transition systems. Then, we will also associate other transition systems with a given reaction system, reflecting different views one might adopt when observing its behaviour in context.

The first part of the next definition assumes – similarly to the assumption underlying the definition of interactive processes – that the environment can provide any subset of the background set at any time of the execution of a reaction system. The transitions of the transition system associated with a reaction system capture the change of a state of the reaction system resulting from the reactions taking place at that state and the context thrown in by the environment. The second part of the definition assumes that the environment never provides any entities.

Definition 13. *Let $\mathcal{A} = (S, A)$ be a reaction system.*

1. *The* (full) transition system *of $\mathcal{A}$ is an ordered pair $TS(\mathcal{A}) = (Q, V)$ such that $Q = 2^S$ and $V = \{(X, res_{\mathcal{A}}(X) \cup C) \mid X, C \in 2^S\}$.*
2. *The* context-independent transition system *of $\mathcal{A}$ is an ordered pair $CITS(\mathcal{A}) = (Q, V)$ such that $Q = 2^S$ and $V = \{(X, res_{\mathcal{A}}(X)) \mid X \in 2^S\}$.*

The two transition systems from Definition 13 share their nodes, and $CITS(\mathcal{A})$ is a subgraph of $TS(\mathcal{A})$, but they are never equal (e.g., (S, S) is always a transition in $TS(\mathcal{A})$, but not in $CITS(\mathcal{A})$). Neither of the two transition systems is initialised, as any subset of the background set S can be the first state of an interactive process or a context-independent process of $\mathcal{A}$.

Following the well-established approach that the directed paths of a transition system provide the semantics of the dynamic system it is associated with, we can establish the soundness of the definitions of $TS(\mathcal{A})$ and $CITS(\mathcal{A})$ by relating the state sequences of the interactive processes of $\mathcal{A}$ to the state sequences of the two kinds of transition systems.

Proposition 7. *Let $\mathcal{A} = (S, A)$ be a reaction system and $X \subseteq S$.*

1. $sts(TS(\mathcal{A})) = STS(PROC(\mathcal{A}))$.
2. $sts(CITS(\mathcal{A})) = STS(CIPROC(\mathcal{A}))$.
3. $sts_X(TS(\mathcal{A})) = STS(PROC_X(\mathcal{A}))$.
4. $sts_X(CITS(\mathcal{A})) = STS(CIPROC_X(\mathcal{A}))$.

Proof. All statements follow directly from the definitions. □

Developing a satisfactory notion of a transition system capturing the behaviour of a reaction system is not a straightforward task. The reason is that in the standard treatment, the states of a transition system do not have any internal structure and the interactions between a system and its environment are represented as actions, such as messages or signals. In the case of reaction systems, however, the states do have structure (they are sets of entities) and the interaction with the environment (context) takes the form of sets of entities which are added to the current state of the system.

It is instructive to note that the state sequences defined by the directed paths of the (unlabelled) transition system $TS(\mathcal{A})$ record vertex based attributes rather than arc based attributes (as is usually the case in process algebras [26,35]). This is natural for models of biological systems where the behaviours are perceived through changes of states and in general it is not possible to record the contexts thrown in by the environment.

8 Structuring Context

In the literature on reaction systems, one mostly considers two extreme variants of context sequences, viz., there are either no restrictions imposed (as captured by the full transition systems of reaction systems), or no context is provided (as captured by the context-independent transition systems of reaction systems). A fundamental research topic is how to find interesting cases in-between these two extremes – in other words, the question is how to structure contexts? An example of an attempt to answer this question is provided by research on reaction systems with durations [6] where, for a given reaction system, the relevant context sequences are determined by a 'bigger' reaction system in which the original reaction system is embedded.

In this paper, we structure the environment and contexts it provides by assuming that it exhibits a finite-state behaviour. This will be captured by *context controllers* of two kinds: (1) controllers which throw in contexts irrespective of the current state of the reaction system (as proposed in [25] for model checking, where they were called context automata), considered in Subsect. 8.1; and (2) more general controllers which throw in contexts depending also on the current system state of the reaction system (introduced for the first time in this paper), considered in Subsect. 8.2.

8.1 State-Oblivious Context Controllers

Definition 14. *A* state-oblivious context controller *over a background set S is an edge-labelled directed graph $\mathcal{E} = (Q, V)$ such that Q is a finite nonempty set and $V \subseteq Q \times 2^S \times Q$. We assume that, for every node $q \in Q$, there are $C \subseteq S$ and $q' \in Q$ such that $(q, C, q') \in V$.*

Note that a state-oblivious context controller over S is a labelled transition system over 2^S. The additional condition that each node of $\mathcal{E}$ has an outgoing edge means that $\mathcal{E}$ is always able to throw in a context.

Interactive processes generated under the control of state-oblivious context controllers are defined as follows.

Definition 15. *Let $\mathcal{A} = (S, A)$ be a reaction system, and $\mathcal{E} = (Q, V)$ a state-oblivious context controller over S.*

An interactive process in $\mathcal{A}$ controlled *by $\mathcal{E}$ and* initiated *at $(q, X) \in Q \times 2^S$ is an ordered pair $\pi = (\gamma, \delta)$ of finite sequences such that, for some $n \geq 1$, $\gamma = C_0, \ldots, C_n$ and $\delta = D_0, \ldots, D_n$, where $C_0, \ldots, C_n, D_0, \ldots, D_n \subseteq S$. Moreover, there are $q_0, \ldots, q_n \in Q$ satisfying:*

- *$(q, C_0, q_0) \in V$ and $D_0 = res_{\mathcal{A}}(X)$,*
- *$(q_i, C_{i+1}, q_{i+1}) \in V$ for $i = 0, \ldots, n-1$, and*
- *$D_i = res_{\mathcal{A}}(C_{i-1} \cup D_{i-1})$ for $i = 1, \ldots, n$.*

The state sequence of π is $STS(\pi) = X, W_0, W_1, \ldots, W_n$, where $W_i = C_i \cup D_i$ for $i = 0, \ldots, n$.

Note that the state sequence of π begins with X rather than W_0 as was the case previously. We refer to (q, X) as the *origin of* π. The intuition behind it is that when $\mathcal{A}$ begins in state X, the state of $\mathcal{E}$ is q.

We use $PROC_{(q,X)}(\mathcal{A}, \mathcal{E})$ to denote the set of all interactive processes in $\mathcal{A}$ controlled by $\mathcal{E}$ and initiated at (q, X), and $PROC(\mathcal{A}, \mathcal{E})$ to denote the set of all interactive processes in $\mathcal{A}$ controlled by $\mathcal{E}$, i.e.,

$$PROC(\mathcal{A}, \mathcal{E}) = \bigcup_{(q,X) \in Q \times 2^S} PROC_{(q,X)}(\mathcal{A}, \mathcal{E}) \, .$$

One can visualise the last definition using the progression in terms of columns now also incorporating the state of the context controller, following the origin (q, X) which provides the initial parameters in the last definition:

$$\begin{array}{lcccccc} & q & q_0 & q_1 & \cdots & q_{n-1} & q_n \\ \gamma : & & C_0 & C_1 & \ldots & C_{n-1} & C_n \\ \delta : & & D_0 & D_1 & \ldots & D_{n-1} & D_n \\ \tau : & X & W_0 & W_1 & \ldots & W_{n-1} & W_n \end{array} \qquad (2)$$

Note that, for every $1 \leq k < n$:

$$((C_k, \ldots, C_n), (D_k, \ldots, D_n)) \in PROC_{(q_{k-1}, W_{k-1})}(\mathcal{A}, \mathcal{E}) \, .$$

In other words, a suffix of an interactive process is also an interactive process.

The motivation behind the above *extended* notion of interactive process (and one which is particularly needed in the case of state-aware context controllers introduced below) is to be able to have a clear concept of *extended* states and state sequences in which not only sets of entities are present, but also states of the controller. By an extended state sequence we mean $(q, X), W'_0, W'_1, \ldots, W'_n$, where $W'_i = (q_{i+1}, C_i \cup D_i)$ for $i = 0, \ldots, n$. Such a treatment also allows one to easily define a transition system representation of state sequences of interactive processes.

We introduce two kinds of transition systems for reaction systems controlled by state-oblivious context controllers. The first does not distinguish between entities thrown in by the environment and those produced by reaction system, and the second (to be used in Sect. 10) takes account of the contexts produced by controllers.

Definition 16. *Let $\mathcal{A} = (S, A)$ be a reaction system and $\mathcal{E} = (Q, V)$ be a state-oblivious context controller over S.*

1. *The* transition system *of $\mathcal{A}$ controlled by $\mathcal{E}$ is an ordered pair $TS(\mathcal{A}, \mathcal{E}) = (Q', U)$ such that*
   - *$Q' = Q \times 2^S$, and*
   - *U is the set of all ordered pairs $((q, X), (q', X')) \in Q' \times Q'$ such that there exists $(q, C, q') \in V$ satisfying $X' = C \cup res_{\mathcal{A}}(X)$.*
2. *The* labelled transition system *of $\mathcal{A}$ controlled by $\mathcal{E}$ is an ordered pair $LTS(\mathcal{A}, \mathcal{E}) = (Q', U)$ such that*
   - *$Q' = Q \times 2^S$, and*
   - *U is the set of all triples $((q, X), C, (q', X')) \in Q' \times Q'$ such that there exists $(q, C, q') \in V$ satisfying $X' = C \cup res_{\mathcal{A}}(X)$.*

For $TS(\mathcal{A}, \mathcal{E})$, a state sequence starting at a node (q, X) is obtained by taking the node sequence defined by a path originating at (q, X) and then deleting the first components (i.e., the states of $\mathcal{E}$). The set of all such sequences is denoted by $sts_{(q,X)}(TS(\mathcal{A}, \mathcal{E}))$, and then

$$sts(TS(\mathcal{A}, \mathcal{E})) = \bigcup_{(q,X) \in Q \times 2^S} sts_{(q,X)}(TS(\mathcal{A}, \mathcal{E}))$$

is the set of all state sequences of reaction system $\mathcal{A}$ controlled by $\mathcal{E}$.

Proposition 8. *Let $\mathcal{A} = (S, A)$ be a reaction system, and $\mathcal{E} = (Q, V)$ be a state-oblivious context controller over S.*

1. *$sts_{(q,X)}(TS(\mathcal{A}, \mathcal{E})) = STS(PROC_{(q,X)}(\mathcal{A}, \mathcal{E}))$ for every $(q, X) \in Q \times 2^S$.*
2. *$sts(TS(\mathcal{A}, \mathcal{E})) = STS(PROC(\mathcal{A}, \mathcal{E}))$.*

Proof. Both statements follow directly from the definitions. □

Both the full transition system of $\mathcal{A} = (S, A)$ and the context-independent transition system of $\mathcal{A}$ may, in essence, be obtained using suitable state-oblivious context controllers, as follows.

- Let $\mathcal{E}^{full} = (\{q\}, \{(q, C, q) \mid C \in 2^S\})$ be a state-oblivious context controller with exactly one state. Then $TS(\mathcal{A})$ can be obtained from $TS(\mathcal{A}, \mathcal{E}^{full})$ by replacing each node (q, X) by X.
- Let $\mathcal{E}^{cind} = (\{q\}, \{(q, \varnothing, q)\})$ be a state-oblivious context controller with exactly one state. Then $CITS(\mathcal{A})$ can be obtained from $TS(\mathcal{A}, \mathcal{E}^{cind})$ by replacing each node (q, X) by X.

It can be shown that the state sequences of transition systems defined as above coincide with those defined in the standard way.

8.2 State-Aware Context Controllers

The totally arbitrary nature of the context sequences in interactive processes and, on the other hand, the extreme restriction of such sequences in context-independent interactive processes means that it is rather natural to allow and then analyse ways of varying the degree of such constraints. We have already made a first step in the previous sub-section. Now we will go one step further, by allowing a controller to access the current system state in order to decide what context to throw in. Such a feature is useful in modelling mutual interactions between (biological) systems and their surrounding environment.

Whereas a state-oblivious context controller had its 'own' set of states which were used to provide the desired contexts depending on its current state, a state-aware context controller will have states structured as ordered pairs (h, X), where $h \in H$ is as previously a state of the controller, and X represents the current state of the reaction system being controlled. This simple device allows one to make the generation of contexts dependent on the current state of the reaction system.

Definition 17. *A* state-aware context controller *over a background set S is an edge-labelled directed graph $\mathcal{E} = (Q, V)$ such that, for some finite nonempty set H, $Q = H \times 2^S$ and $V \subseteq Q \times 2^S \times Q$, the following hold:*

- *For every $q \in Q$, there exist $C \subseteq S$ and $q' \in Q$ such that $(q, C, q') \in V$.*
- *If $((h, X), C, (h', X')) \in V$ then*
  - *$C \subseteq X'$, and*
  - *$((h, X), C, (h', C \cup X'')) \in V$, for every $X'' \subseteq S$.*

Intuitively, the first condition means that the controller always provides some context and the last condition means that the controller provides only contexts and cannot block reactions of any reaction system operating under it from producing their products.

The notion of an interactive process is now defined as follows.

Definition 18. *Let $\mathcal{A} = (S, A)$ be a reaction system, and $\mathcal{E} = (Q, V)$ be a state-aware context controller over S.*

An interactive process in $\mathcal{A}$ controlled *by $\mathcal{E}$ and* initiated *at $q = (h, X) \in Q$ is an ordered pair $\pi = (\gamma, \delta)$ of finite sequences such that, for some $n \geq 1$, $\gamma = C_0, \ldots, C_n$ and $\delta = D_0, \ldots, D_n$, where $C_0, \ldots, C_n, D_0, \ldots, D_n \subseteq S$. Moreover, there are $q_i = (h_i, C_i \cup D_i) \in Q$ for $i = 0, \ldots, n$ satisfying:*

- *$(q, C_0, q_0) \in V$ and $D_0 = res_{\mathcal{A}}(X)$.*
- *$(q_i, C_{i+1}, q_{i+1}) \in V$ for $i = 0, \ldots, n-1$.*
- *$D_i = res_{\mathcal{A}}(C_{i-1} \cup D_{i-1})$ for $i = 1, \ldots, n$.*

The state sequence of π is $STS(\pi) = X, W_0, W_1, \ldots, W_n$, where $W_i = C_i \cup D_i$ for $i = 0, \ldots, n$.

Note that (as was the case in Definition 15) the state sequence of π begins with X.

We use $PROC_q(\mathcal{A},\mathcal{E})$ to denote the set of all interactive processes in $\mathcal{A}$ controlled by $\mathcal{E}$ and initiated at q, and $PROC(\mathcal{A},\mathcal{E})$ to denote the set of all interactive processes in $\mathcal{A}$ controlled by $\mathcal{E}$, i.e.,

$$PROC(\mathcal{A},\mathcal{E}) = \bigcup_{q\in Q} PROC_q(\mathcal{A},\mathcal{E}) \ .$$

Transition systems of $\mathcal{A}$ controlled by $\mathcal{E}$ are defined analogously to those introduced for state-oblivious context controllers.

Definition 19. *Let $\mathcal{A} = (S,A)$ be a reaction system, and $\mathcal{E} = (Q,V)$ be a state-aware context controller over S.*

1. *The* transition system *of $\mathcal{A}$ controlled by $\mathcal{E}$ is an ordered pair $TS(\mathcal{A},\mathcal{E}) = (Q,U)$, where U is the set of all ordered pairs $((h,X),(h',X')) \in V$ such that there exists $((h,X),C,(h',X')) \in V$ satisfying $X' = C \cup res_{\mathcal{A}}(X)$.*
2. *The* labelled transition system *of $\mathcal{A}$ controlled by $\mathcal{E}$ is an ordered pair $LTS(\mathcal{A},\mathcal{E}) = (Q,U)$, where U is the set of all triples $((h,X),C,(h',X')) \in V$ such that $X' = C \cup res_{\mathcal{A}}(X)$.*

For $TS(\mathcal{A},\mathcal{E})$ and $q = (h,X) \in Q$, the sets $sts_q(TS(\mathcal{A},\mathcal{E}))$ and $sts(TS(\mathcal{A},\mathcal{E}))$ of state sequences are defined similarly as in the case of a transition system of $\mathcal{A}$ controlled by a state-oblivious context controller.

Proposition 9. *Let $\mathcal{A} = (S,A)$ be a reaction system, and $\mathcal{E} = (Q,V)$ be a state-aware context controller over S.*

1. *$sts_q(TS(\mathcal{A},\mathcal{E})) = STS(PROC_q(\mathcal{A},\mathcal{E}))$ for every $q \in Q$.*
2. *$sts(TS(\mathcal{A},\mathcal{E})) = STS(PROC(\mathcal{A},\mathcal{E}))$.*

Proof. Both statements follow directly from the definitions. □

State-aware context controllers are more general than the state-oblivious ones, in the sense that the former can be used to simulate the latter. Consider, for example, a reaction system $\mathcal{A} = (S,A)$ controlled by a state-oblivious context controller $\mathcal{E} = (Q,V)$. Then $sts(TS(\mathcal{A},\mathcal{E})) = sts(TS(\mathcal{A},\mathcal{E}'))$, where $\mathcal{E}' = (Q \times 2^S, V')$ is a state-aware context controller with V' comprising all triples $((q,X),C,(q',X' \cup C))$ such that $X, X' \subseteq S$ and $(q,C,q') \in V$.

9 Equivalences Based on Transition Systems

The full transition systems associated with reaction systems lead in a natural way to an equivalence notion:

Definition 20. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set. Then $\mathcal{A}$ and $\mathcal{B}$ are* transition system equivalent*, denoted by $\mathcal{A} \sim_{TS} \mathcal{B}$, if $TS(\mathcal{A}) = TS(\mathcal{B})$.*

Both $\sim_{PROC}$ and $\sim_{TS}$ are equivalences defined in terms of states. They are based on manifestations of behaviour expressed in terms of interactive processes and transition systems respectively. In general for a given reaction system, there will be loss of behavioural information when one focusses solely on the state sequences of its interactive processes, or of its transition system: in general it will not be clear for a given state, what has been produced from a previous state and what has been thrown in as context. We will later remove this restriction and consider labelled transition systems associated with reaction systems.

Clearly, $\sim_{TS}$ is an equivalence relation. Moreover, it turns out that two reaction systems that have the same associated transition system, are also process equivalent and, moreover, are equivalent in the classical sense.

Proposition 10. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set. Then $\mathcal{A} \sim \mathcal{B}$ if and only if $\mathcal{A} \sim_{PROC} \mathcal{B}$ if and only if $\mathcal{A} \sim_{TS} \mathcal{B}$.*

Proof. Follows immediately from Corollary 1, Proposition 7, and the definition of the set of state sequences $sts(\mathcal{T})$ of a transition system $\mathcal{T}$. ◻

10 Bisimulation-Based Behavioural Equivalence

In the previous sections, transition systems associated with reaction systems as well as the environments in which they operate were basically concerned with state sequences. This means, implicitly, that we made no distinction between entities produced by reaction systems and those in contexts thrown-in by the environment. We will now reassess the way of observing behaviours and introduce new equivalences between reaction systems.

We base our discussion in this section on the well-known notion of bisimulation between transition systems. Such a notion has been successfully applied to capture a range of equivalence notions for communicating computing systems [26,27,35]. A standard way of introducing bisimulation-based equivalence is to consider two labelled transition systems, $LTS = (Q, V)$ and $LTS' = (Q', V')$, and then find a *bisimulation* relation $B \subseteq Q \times Q'$ aiming at identifying equivalent (or *bisimilar*) states. What is required is that if $(q, q') \in B$, then all transitions from q can be simulated by transitions from q' which lead to bisimilar states, and vice versa (i.e., if $(q, a, r) \in V$, then there is a $(q', a, r') \in V'$ such that $(r, r') \in B$, and vice versa). The notion of equivalence obtained in this way does not depend on a specific relation B as there is always the largest bisimulation[1] relation for a given pair of transition systems. Moreover, two initialised transition systems are bisimulation equivalent if their initial states are bisimilar.

We now make two general observations. First, bisimulation-based equivalences are usually label-based, and the states of the transition systems have no structure. This, of course, is no longer the case for reaction system behaviours,

[1] Since the union of two bisimulation relations is also a bisimulation relation, there is always the largest (in the set inclusion sense) bisimulation relation for a given pair of transition systems.

where states are sets of entities. Second, since we decided to accept that contexts thrown-in by the environment can be observed, the arcs of transition systems representing reaction systems and their environments should now be labelled with sets of entities. As a consequence of these two observations, in this section we will use labelled transition systems rather than transition systems of reaction system controlled by context controllers.

By using bisimulation-based equivalence, we expect to capture precisely what it means that two systems 'react' in an 'equivalent' manner to stimuli provided by a given context controller.

The next definition considers two reaction systems and a context controller over a common background set. Moreover, a set Y identifies entities considered as 'observable'.

Definition 21. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S, $\mathcal{E}$ be a state-oblivious or state-aware context controller over S, and $Y \subseteq S$.*

A Y-bisimulation *for $LTS(\mathcal{A},\mathcal{E}) = (Q,V)$ and $LTS(\mathcal{B},\mathcal{E}) = (Q,U)$ is a relation $B \subseteq Q \times Q$ such that if $((h,X),(h',X')) \in B$, then the following hold:*

- $X \cap Y = X' \cap Y$.
- *If $((h,X),C,(\widehat{h},\widehat{X})) \in V$, then there is a $((h',X'),C',(\widehat{h}',\widehat{X}')) \in U$ such that $C \cap Y = C' \cap Y$ and $((\widehat{h},\widehat{X}),(\widehat{h}',\widehat{X}')) \in B$.*
- *If $((h',X'),C',(\widehat{h}',\widehat{X}')) \in U$, then there is a $((h,X),C,(\widehat{h},\widehat{X})) \in V$ such that $C \cap Y = C' \cap Y$ and $((\widehat{h},\widehat{X}),(\widehat{h}',\widehat{X}')) \in B$.*

The above formalisation of bisimulation-based equivalence leads to a range of meaningful equivalences based on the idea of bisimulation, which are capable of capturing subtle aspects of the 'reactiveness' of $\mathcal{A}$ and $\mathcal{B}$.

Proposition 11. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S, $\mathcal{E}$ be a state-oblivious or state-aware context controller over S, and $Y \subseteq S$. Then there exists a Y-bisimulation $B^Y_{\mathcal{A},\mathcal{B},\mathcal{E}}$ for $LTS(\mathcal{A},\mathcal{E})$ and $LTS(\mathcal{B},\mathcal{E})$ such that $B \subseteq B^Y_{\mathcal{A},\mathcal{B},\mathcal{E}}$, for every Y-bisimulation B for $LTS(\mathcal{A},\mathcal{E})$ and $LTS(\mathcal{B},\mathcal{E})$.*

Proof. Follows directly from the definitions. □

Note that $B^Y_{\mathcal{A},\mathcal{B},\mathcal{E}}$ is well-defined since $B = \varnothing$ is always a Y-bisimulation.

Thus $B^Y_{\mathcal{A},\mathcal{B},\mathcal{E}}$ is the largest Y-bisimulation for $LTS(\mathcal{A},\mathcal{E})$ and $LTS(\mathcal{B},\mathcal{E})$. It allows one to establish, for example, which initial states for $\mathcal{A}$ and $\mathcal{B}$ controlled by $\mathcal{E}$ are equivalent (bisimilar) when one is interested in obtaining equivalent reactive behaviour assuming that only entities in Y are observable.

Proposition 12. *Let $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ be reaction systems with the same background set S, $\mathcal{E}$ be a state-oblivious or state-aware context controller over S, and $Y \subseteq S$. Then $(q,q'') \in B^Y_{\mathcal{A},\mathcal{C},\mathcal{E}}$, for all $(q,q') \in B^Y_{\mathcal{A},\mathcal{B},\mathcal{E}}$ and $(q',q'') \in B^Y_{\mathcal{B},\mathcal{C},\mathcal{E}}$.*

Proof. Follows directly from the definitions. □

The above result states that bisimilarity is transitive while the next one states that it is preserved by making the set of observable entities smaller.

Proposition 13. *Let $\mathcal{A}$ and $\mathcal{B}$ be reaction systems with the same background set S, $\mathcal{E}$ be a state-oblivious or state-aware context controller over S, and $Z \subseteq Y \subseteq S$. Then $B^Y_{\mathcal{A},\mathcal{B},\mathcal{E}} \subseteq B^Z_{\mathcal{A},\mathcal{B},\mathcal{E}}$.*

Proof. Follows directly from the definitions. □

Note that the notion of bisimulation-based equivalence introduced in this section is immediately extendable to equivalence based on a set of context controllers rather than a single one.

11 Discussion

In this paper we proposed a framework for considering equivalences of reaction systems. The following aspects of this paper are ‘immediate’ candidates for a follow-up line of research.

(1) We have established a new bridge between the theory of reaction systems and the theory of transition systems by proposing various representations of reaction systems by transition systems. This should lead to a transfer of many methods of investigating the behaviour of dynamic systems through transition systems into the domain of reaction systems. Among the relevant specific aspects of theory of transition systems that would be good candidates for a transfer into the realm of reaction systems are: various variants of weak bisimulation equivalence and testing equivalence, and efficient verification techniques and tools.

(2) We introduced a formalisation of finite-state environments through state-oblivious and state-aware context controllers. This naturally leads to a novel line of research concerning properties of state sequences. The investigation of properties of state sequences of context-independent processes of reaction systems resulted in a rich and elegant theory (see, e.g., [9,15,20,30,31,33,34]). Reaction systems with context determined by context controllers form a natural next level of a systematic and thorough investigation of state sequences (clearly, not much can be said when context is arbitrary as in the general model of reaction systems).

(3) In considering equivalences defined directly on reaction systems, functional equivalence played a central role in this paper. In [10] we considered the more subtle *enabling equivalence*. Incorporating also this equivalence in our framework is certainly a very natural research topic.

(4) The study of model checking techniques for reaction systems controlled by state-oblivious context controllers has been initiated in [25]. In this paper, we introduced the much more general framework of reaction systems controlled by state-aware context controllers, the ensuing model checking problem is both important and challenging.

Acknowledgements. The authors are indebted to Robert Brijder and the anonymous referee for useful comments on this paper.

References

1. Arnold, A.: Finite Transition Systems: Semantics of Communicating Systems. Prentice Hall, Upper Saddle River (1994)
2. Azimi, S., Iancu, B., Petre, I.: Reaction system models for the heat shock response. Fundam. Inform. **131**, 1–14 (2014)
3. Azimi, S., Panchal, C., Czeizler, E., Petre, I.: Reaction systems models for the self-assembly of intermediate filaments. Ann. Univ. Bucharest LXI **I**(2), 9–24 (2015)
4. Baier, C., Katoen, J.-P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
5. Brijder, R., Ehrenfeucht, A., Main, M., Rozenberg, G.: A tour of reaction systems. Int. J. Found. Comput. Sci. **22**(07), 1499–1517 (2011)
6. Brijder, R., Ehrenfeucht, A., Rozenberg, G.: Reaction systems with duration. In: Kelemen, J., Kelemenová, A. (eds.) Computation, Cooperation, and Life. LNCS, vol. 6610, pp. 191–202. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20000-7_16
7. Corolli, L., Maj, C., Marini, F., Besozzi, D., Mauri, G.: An excursion in reaction systems: from computer science to biology. Theor. Comput. Sci. **454**, 95–108 (2012)
8. Dennunzio, A., Formenti, E., Manzoni, L.: Extremal combinatorics of reaction systems. In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) LATA 2014. LNCS, vol. 8370, pp. 297–307. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04921-2_24
9. Dennunzio, A., Formenti, E., Manzoni, L., Porreca, A.E.: Ancestors, descendants, and gardens of Eden in reaction systems. Theor. Comput. Sci. **608**, 16–26 (2015)
10. Ehrenfeucht, A., Kleijn, J., Koutny, M., Rozenberg, G.: Evolving reaction systems. Theor. Comput. Sci. **682**, 79–99 (2017)
11. Ehrenfeucht, A., Kleijn, J., Koutny, M., Rozenberg, G.: Modelling reaction systems with Petri nets. In: International Workshop on Biological Processes & Petri Nets, CEUR-WS Workshop Proceedings, BioPPN-2011, pp. 36–52 (2011)
12. Ehrenfeucht, A., Kleijn, J., Koutny, M., Rozenberg, G.: Reaction systems: a natural computing approach to the functioning of living cells. In: Hector, Z. (ed.) A Computable Universe: Understanding and Exploring Nature as Computation, Chapter 10, pp. 189–208. World Scientific, Singapore (2012)
13. Ehrenfeucht, A., Main, M.G., Rozenberg, G.: Functions defined by reaction systems. Int. J. Found. Comput. Sci. **22**, 167–178 (2011)
14. Ehrenfeucht, A., Main, M.G., Rozenberg, G.: Combinatorics of life and death for reaction systems. Int. J. Found. Comput. Sci. **21**, 345–356 (2010)
15. Ehrenfeucht, A., Main, M., Rozenberg, G., Thompson Brown, A.: Stability and chaos in reaction systems. Int. J. Found. Comput. Sci. **23**, 1173–1184 (2012)
16. Ehrenfeucht, A., Petre, I., Rozenberg, G.: Reaction systems: a model of computation inspired by the functioning of the living cell. In: Konstantinidis, S., Moreira, N., Reis, R., Shallit, J. (eds.) The Role of Theory in Computer Science World Scientific, pp. 1–32 (2017)
17. Ehrenfeucht, A., Rozenberg, G.: Reaction systems. Fundam. Inform. **75**, 263–280 (2007)
18. Ehrenfeucht, A., Rozenberg, G.: Events and modules in reaction systems. Theor. Comput. Sci. **376**, 3–16 (2007)
19. Ehrenfeucht, A., Rozenberg, G.: Introducing time in reaction systems. Theor. Comput. Sci. **410**, 310–322 (2009)

20. Genova, D., Hoogeboom, H.J., Jonoska, N.: A graph isomorphism condition and equivalence of reaction systems. Theor. Comput. Sci. **701**, 109–119 (2017)
21. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Prentice Hall, Upper Saddle River (2006)
22. Kari, L., Rozenberg, G.: The many facets of natural computing. Commun. ACM **51**, 72–83 (2008)
23. Keller, R.M.: Formal verification of parallel programs. Commun. ACM **19**, 371–384 (1976)
24. Lehninger, A.: Bioenergetics: The Molecular Basis of Biological Energy Transformations. W.A. Benjamin Inc., Amsterdam (1965)
25. Meski, A., Penczek, W., Rozenberg, G.: Model checking temporal properties of reaction systems. Inf. Sci. **313**, 22–42 (2015)
26. Milner, R.: Communication and Concurrency. PHI Series in Computer Science. Prentice Hall, Upper Saddle River (1989)
27. Park, D.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) GI-TCS 1981. LNCS, vol. 104, pp. 167–183. Springer, Heidelberg (1981). https://doi.org/10.1007/BFb0017309
28. Reisig, W.: Petri Nets (An Introduction). EATCS Monographs on Theoretical Computer Science. Springer, Heidelberg (1985). https://doi.org/10.1007/978-3-642-69968-9
29. Rozenberg, G., Bäck, T., Kok, J.: Handbook of Natural Computing. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-540-92910-9
30. Salomaa, A.: Functions and sequences generated by reaction systems. Theor. Comput. Sci. **466**, 87–96 (2012)
31. Salomaa, A.: On state sequences defined by reaction systems. In: Constable, R.L., Silva, A. (eds.) Logic and Program Semantics. LNCS, vol. 7230, pp. 271–282. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29485-3_17
32. Salomaa, A.: Functional constructions between reaction systems and propositional logic. Int. J. Found. Comput. Sci. **24**, 147–160 (2013)
33. Salomaa, A.: Minimal and almost minimal reaction systems. Nat. Comput. **12**, 369–376 (2013)
34. Salomaa, A.: Two-step simulations of reaction systems by minimal ones. Acta Cybern. **22**, 247–257 (2015)
35. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press, Cambridge (2011)

On Usefulness of Information: Framework and NFA Case

Branislav Rovan(✉) and Šimon Sádovský

Department of Computer Science, Comenius University, Mlynská Dolina, 842 48 Bratislava, Slovakia
{rovan,sadovsky}@dcs.fmph.uniba.sk

Abstract. In this paper we present a brief overview of the results of our research aimed at identifying and formalising various aspects of the notion of information, especially its usefulness, i.e., its ability to simplify a solution of a problem. We formalize the problem via decompositions of automata and present new results in the nondeterministic finite automata setting. We characterise some subfamilies of regular languages that are nondeterministically decomposable/undecomposable and exhibit an infinite sequence of regular languages that are nondeterministically undecomposable but decomposable in the deterministic finite automata setting.

Keywords: Usefulness of information
Nondeterministic finite automata · Decomposition of finite automata
Supplementary information · Advice · Descriptional complexity
State complexity

1 Introduction

In this paper we present a brief overview of the research aimed at identifying and formalising various aspects of the notion of *information* initiated at the Department of Computer Science of the Comenius University about fifteen years ago and pursued by the first author and his master and PhD students. We also present some new recent results.

In the early days of Shannon's theory of information the main concern was in transferring information reliably and fast over possibly noisy channels. The *amount* of information was the important attribute of information considered. Over half a century later we can identify additional aspects, like usefulness, timeliness, etc. The first and so far the basic aspect of information we started to explore was the information usefulness. We shall now describe our approach to handling the problem.

The essence of our approach can be stated as follows: *Information is useful if it can help us to solve a problem easier.* We thus take a computational approach to studying information. There are several basic questions we need to clarify:

(i) What is a problem?
(ii) What do we mean by solving the problem?
(iii) How do we measure the complexity of the solution?
(iv) How do we provide information (advice) that possibly simplifies the solution of the problem?

It is clear that taking some powerful advice (e.g., providing the solution to the problem) could render the solution to a problem trivial. In our approach we took the position that information should be 'reasonable', i.e., we should also be able to 'simply' check whether the advice information is correct. Thus the complexity of checking the information (advice) should not exceed that of the problem itself.

The idea of providing 'additional information' appeared in probability theory, automata theory (e.g., promise problems [3]) and more recently in on-line algorithms [2,8]. A brief overview of approaches and merging the views on additional information in formal languages, probability theory and Kolmogorov complexity can be found in [12] together with main aspects of our approach.

Following some notation and basic facts in Sect. 2 we present a brief overview of our framework and results on usefulness and other aspects of information in Sect. 3. In Sect. 4 we present some new results concerning usefulness of information in the nondeterministic finite automata setting and its comparison to the deterministic case.

2 Preliminaries

We shall develop our framework for dealing with the notion of usefulness of information using the formalism of formal languages and automata theory. The notions and basic results can be found, e.g., in [7].

We shall denote the length of a word by $|w|$ and a number of elements of a finite set S by $|S|$. The empty word is denoted by ε. To denote the prefix of a word u of length k we shall write $\mathit{pref}(u,k)$ and to denote the suffix of a word u of length k we shall write $\mathit{suf}(u,k)$. Notation $A \subseteq B$ means, that the set A is a (not necessarily proper) subset of the set B. A proper subset is denoted by $\subset$.

A deterministic finite state automaton (DFA) is a 5-tuple $A = (K, \Sigma, \delta, q_0, F)$ with a standard meaning of its components. We assume the transition function $\delta\colon K \times \Sigma \to K$ is total. In the nondeterministic finite state automata (NFA) case we do not allow ε transitions, i.e., $\delta\colon K \times \Sigma \to 2^K$. The language accepted by A, denoted by $L(A)$, is defined by $L(A) = \{w \in \Sigma^* \mid \exists q \in F\colon (q_0, w) \vdash^* (q, \varepsilon)\}$, where the relation 'step of computation' $\vdash$ on configurations in $K \times \Sigma^*$ is defined as usual.

We shall measure complexity of automata by the number of their states, i.e., we shall use the complexity measure $\#_S$ defined by $\#_S(A) = |K|$. We extend this complexity measure to regular languages in a natural way. The (deterministic) state complexity of L, denoted by $sc(L)$, is the number of states of the minimal DFA for L. The nondeterministic state complexity of L is defined by $nsc(L) =$

$\min\{\#_S(A) \mid A \ is \ NFA \ and \ L(A) = L\}$.[1] In this paper we shall often need to argue that we found a minimal NFA A for some regular language L. To obtain a lower bound for $nsc(L)$ we shall often employ the fooling set technique from [6].

Theorem 1 (Fooling Set Technique). *Let L be a regular language, $n \in \mathbb{N}$. Suppose there exists a set of pairs $P = \{(x_i, y_i) \mid 1 \leq i \leq n\}$ such that:*

(a) $x_i y_i \in L$ *for* $1 \leq i \leq n$
(b) $x_i y_j \notin L$ *for* $1 \leq i, j \leq n$ *and* $i \neq j$

Then any NFA accepting L has at least n states.

3 Overview of Past Results

As mentioned in the introduction there were many attempts to incorporate supplementary information in various settings. It is generally perceived (see, e.g., the discussion in [8]) that more effort should be spent on trying to better understand the very notion of information.

In this section we illustrate the main ingredients of our framework for studying various aspects of information mentioned in the introduction. We also present a brief overview of the results obtained by our research group so far. The first and so far the basic aspect of information we started to explore was the information usefulness. There are many possible ways to specify the four basic ingredients of our approach: (i) what is a problem; (ii) what do we mean by solving the problem; (iii) how do we measure the complexity of the solution; and (iv) how do we provide information (advice) that possibly simplifies the solution of the problem. An important aspect of our framework is that the complexity of the advice should be smaller than that of the problem to be solved, i.e., the advice should be 'reasonable'.

We first addressed these questions in the deterministic finite automata setting [4]. Here the problem was given by some regular language L (more precisely, the problem is to answer a question, whether any given word w belongs to L) and solving the problem meant to find a DFA A such that $L = L(A)$. We decided to measure the complexity of the solution by taking the state complexity of A. Since we are interested in the simplest possible solutions it makes sense to take A to be the minimal automaton for L. Suppose we have some additional information about the input word (*advice*). It might be possible, that we could than find a simpler solution (a smaller automaton A_{new}) for deciding whether the input word belongs to L. We shall provide the advice information by another regular language L_{adv}, i.e., the advice is that the input word belongs to L_{adv} (given by the minimal DFA A_{adv}). Note that the minimal automaton A_{new} may by itself accept some superset of L but together with the advice it properly defines $L = L(A_{new}) \cap L_{adv}$.

[1] Note that our assumption to only consider NFA without ε moves does not influence the measure, since for each A there is an equivalent A' without ε moves having the same number of states. We made this assumption to simplify our presentation later.

To illustrate the above ideas let us take for L the language consisting of all words in a^* of length divisible by 6. Clearly the simplest solution to this problem is the automaton A having 6 states. Now suppose we have additional information, advice, that the input word is of even length (i.e., we take L_{adv} to be the language in a^* consisting of all words of even length). It thus suffices to check whether the length of the input word is divisible by 3 which can be done by A_{new} having just 3 states. Thus the information L_{adv} enabled us to find a simpler solution to the problem L.

Let us now return to the general considerations and illustrate what we mean by reasonable information (advice). Taking some powerful advice could render the solution to a problem trivial (e.g., taking $L_{adv} = L$, which would in the above example lead to A_{new} having just one state and recognising a^*). In this case we cannot 'simply' check, whether the advice information is correct. The complexity of this checking is the same as the complexity of the original problem. Thus we expect *both* A_{new} and A_{adv} to be simpler than A. In such case we consider the information useful.

The main question we considered in [4] was, whether there is a useful information, advice, that could simplify the solution to a given problem L. We reformulated this question in purely automata theoretic terms as follows: Does there exist a (nontrivial) parallel decomposition of a minimal DFA for L? It should be no surprise, that there are problems whose solution cannot be simplified by any useful information – advice. We also identify problems, for which a 'small' advice can substantially reduce the complexity of the solution. We also considered cases of a 'more precise' advice, e.g., instead of providing information whether the input word is accepted by A_{adv} we can provide information about the state A_{adv} reaches after reading the input word. This led to various types of decompositions of automata which we compared.

In order to further explore the notion of usefulness of information we decided to proceed by varying the four basic ingredients mentioned above. In [10] we take a problem L to be a deterministic context-free language and its solution the corresponding deterministic push-down automaton ($DPDA$) A. We measure the complexity of the solution via the number of states and auxiliary push-down symbols of A. The advice is given by a regular language L_{adv} (resp. its minimal DFA A_{adv}). Here we consider A_{adv} with any number of states to be a 'reasonable' advice since regular languages are 'simpler' than deterministic context-free languages formalising the problem here. We first show that no suitable complexity measure for DPDA combining the number of states and stack symbols exists. Next we prove tight bounds on the state complexity of an infinite sequence of deterministic context-free languages and show that no supplementary regular information can decrease the state complexity of the DPDAs recognizing them. We also exhibited an infinite sequence of deterministic context-free languages for which a suitable supplementary regular information enables a construction of a significantly simpler DPDA.

Our next exploration in this direction [13] was into the way the advice information is given. So far it was in a form to be directly used. This is not always the case in real life situations. One may need to apply some transformation (e.g., into a foreign language or a different encoding) in order to use the advice. We proposed a framework for studying the possibility to transform the instance of a problem to match the format of the advisory information. We stay in the deterministic finite automata setting, i.e., the problem is a regular language L, the solution is a DFA A for L, and the complexity measure is the number of states of A. However, the advice information is given by a pair (L_{adv}, M) where M is a mapping and L_{adv} is an advisory language. The 'advice' is that the input word mapped by M belongs to L_{adv}. The mapping M is given by a finite state device (a-transducer or sequential transducer - see [5] for definitions). We made our reasonability requirement stricter, i.e., we consider the advice useful if all of the three finite state devices – A_{new}, A_{adv}, and M – have together fewer states than A. We have shown that nondeterminism of a-transducers can absorb too much of the (deterministic) complexity of the problem and concentrated on deterministic sequential transducers. We were again able to show the existence of an infinite sequence of regular languages for which no useful information simplifying their solution exists and another infinite sequence of regular languages, for which useful information simplifying their solution exists.

There is another aspect of information commonly used informally. We often say that information is 'actual' or 'received in the right time'. In [11] we introduced a framework enabling us to define and study *time criticality* of information (advice). It is based on the notion of usefulness discussed above. Once again we use the deterministic finite automata setting. We formalise time criticality using the following scenario. Given a problem L and its solution A and a particular instance w we obtain information about w at some point during the computation of A on w. This information (advice) A_{adv} might enable us to use a different (possibly simpler) automaton A_{new} to finish the computation. This automaton may be more or less simple depending on the time the advice is received. Note that we may have different A_{new} automaton depending on the time t the advice is received but we prefer not to overload the notation in this general description. We define time criticality (of L_{adv} given A and w) as the function of time, measuring the usefulness of this information via a possible decrease in the number of states needed to finish the computation. We define time criticality as the function giving the ratio of the number of states we can spare by obtaining the information at time t to the number of states required without the advice information. One might expect that the usefulness of advice decreases with time. However, we were able to show, that time criticality may have complex behaviour (in fact having any number of local maxima). We also considered a dynamic version of time criticality, where the advice concerned the remaining part of the input, obtaining similar results.

In [9] we departed slightly from our basic scenario. Inspired by the reoptimization approach (see, e.g., [1]) we embarked to develop a framework where the information, advice, received does not concern the input word itself but some 'close enough' or 'similar' word. To be able to process the additional information, or advice, about a similar word, the standard model of deterministic finite automaton is augmented by an advice tape. There is a wide variety of possibilities how to specify similarity of words and especially, what is 'admissible' information. We explored some of them, many remain unexplored. We analysed the 'strength' of some of these scenarios by comparing and characterising the corresponding language families. However our main interest was not in the computational power of the new device. In line with our previous research we concentrated on regular languages and the possibility to reduce the complexity (measured again by the number of states) of their acceptance using the advice information on a similar word. We exhibited families of regular languages where the information provided by the given advice schema was useful and where no useful information could be found.

4 Usefulness and Nondeterministic Finite Automata

In this section we present new results of our exploration of the notion of usefulness of information in the nondeterministic finite state automata setting. Similar to the deterministic case studied in [4] the problem L and the advice L_{adv} are regular languages. However a solution to L is given by a nondeterministic finite state automaton A. Similarly A_{adv} for L_{adv} and a new solution utilising the advice A_{new} are nondeterministic finite state automata. We keep the number of states as a complexity measure used. We also keep our requirement of 'reasonability' of advice, i.e., for information to be useful we require both A_{adv} and A_{new} to be smaller than A.

We shall address two questions in this section. In Subsect. 4.1 we shall look for problems (regular languages) for which there exists useful information (advice) that simplifies their solution. On the other hand we shall also look for problems, where no useful information simplifying their solution exists. In Subsect. 4.2 we shall explore differences arising between the deterministic and nondeterministic finite state automata settings.

We shall again (just like in [4]) reformulate our problems in terms of parallel decompositions of NFAs. Given an NFA A we say that two NFAs A_1, A_2 such that $L(A) = L(A_1) \cap L(A_2)$ form a *parallel decomposition of the automaton* A. If it holds that $\#_S(A_1) < \#_S(A)$ and $\#_S(A_2) < \#_S(A)$, we call this decomposition *nontrivial*. If there exists nontrivial parallel decomposition of an NFA A we call A *decomposable*. We say that a regular language L is *nondeterministically decomposable* if a minimal NFA A for L is decomposable. It can be easily shown, that the decomposability of a language is well-defined and does not depend upon the choice of a minimal NFA for the language. In what follows we shall say just *decomposable* instead of nondeterministically decomposable when it is clear from the context.

Thus we can reformulate our question about existence of some useful advice for a given problem L to the question about decomposability of L. We shall use this formulation in the rest of this section.

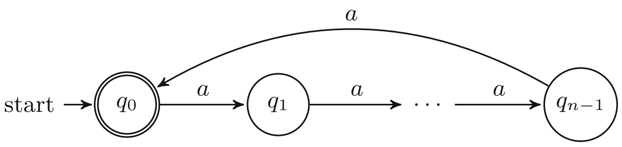


Fig. 1. The automaton A_n

4.1 Decomposability of Regular Languages

In this section we consider two infinite subfamilies of regular languages. We give a necessary and sufficient conditions for a language to be decomposable, i.e., we exhibit infinite number of problems – regular languages – for which a useful advice reducing the complexity of their solution exists and an infinite number of problems for which no such advice exists.

Note 1. There are some regular languages for which it is easy to see that they are undecomposable, e.g., a regular language L with $nsc(L) \leq 2$ is clearly undecomposable, since one-state NFAs can accept only one of the languages $\emptyset, \{\varepsilon\}$, or Σ^*.

First, let us consider a family of regular languages of the form $\{a^{kn} \mid k \in \mathbb{N}\}$ for a given $n \in \mathbb{N}$.

Theorem 2. *Let $n \in \mathbb{N}$, $L_n = \{a^{kn} \mid k \in \mathbb{N}\}$. The language L_n is decomposable if and only if n is not a power of a prime.*

Proof. First we shall show that $nsc(L_n) = n$, for all n. We construct an NFA A_n such that $L(A_n) = L_n$. We define A_n by the transition diagram in Fig. 1.

It is easy to see that $L(A_n) = L_n$. Thus $nsc(L_n) \leq \#_S(A_n) = n$. Using the fooling set technique of Theorem 1 and the set of pairs of words $M_n = \{(a^i, a^{n-i}) \mid 0 \leq i \leq n-1\}$ of size $|M_n| = n$ we can see that $nsc(L_n) \geq n$. Thus $nsc(L_n) = n$ and A_n is a minimal NFA for L_n.

(i) We shall now prove that if n is a power of a prime then the language L_n is undecomposable. Let $n = p^m$ for some prime p and $m \in \mathbb{N}$. From above we have $nsc(L_n) = p^m$.
Suppose that the language L_n is decomposable. Thus, there exists a nontrivial decomposition of the automaton A_n. Thus, there exist NFAs $A_1^{p^m}$ and $A_2^{p^m}$ such that it holds $\#_S(A_1^{p^m}) < p^m$, $\#_S(A_2^{p^m}) < p^m$, and $L(A_1^{p^m}) \cap L(A_2^{p^m}) = L_n$.
Consider the word a^{p^m} in $L(A_1^{p^m}) \cap L(A_2^{p^m})$. Let $(q_0, a^{p^m}) \vdash (q_1, a^{p^m-1}) \vdash \cdots \vdash (q_{p^m-1}, a) \vdash (q_{p^m}, \varepsilon)$, where q_0 is the initial state of $A_1^{p^m}$ and q_{p^m} is one of the accepting states of $A_1^{p^m}$, be an accepting computation of $A_1^{p^m}$ on a^{p^m}. Since $\#_S(A_1^{p^m}) < p^m$, there exist i and j in $\mathbb{N}$ such that $0 \leq i < j < p^m$ and $q_i = q_j$. It follows that we can pump part of the accepted word which is shorter than p^m, i.e., there exists $r_1 \in \mathbb{N}, 1 \leq r_1 < p^m$, such that for all k in

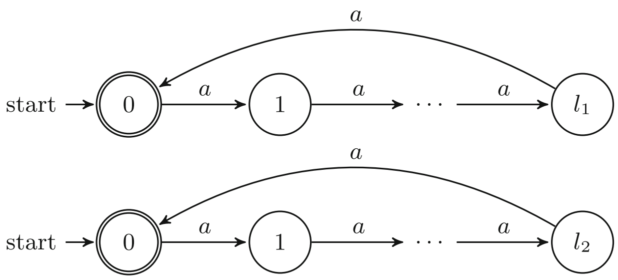


Fig. 2. Automata A_1^n and A_2^n forming a decomposition of A_n

$\mathbb{N}$ it holds $a^{p^m+kr_1}$ is in $L(A_1^{p^m})$. Similarly there exists $r_2 \in \mathbb{N}, 1 \leq r_2 < p^m$, such that for all k in $\mathbb{N}$ it holds $a^{p^m+kr_2}$ is in $L(A_2^{p^m})$.
The numbers r_1 and r_2 can be written as follows: $r_1 = p^{l_1}s_1$, where $0 \leq l_1 < m$ and s_1 is not divisible by p. Similarly $r_2 = p^{l_2}s_2$, where $0 \leq l_2 < m$ and s_2 is not divisible by p. Let $t = s_1 s_2 p^{max(l_1,l_2)}$. From above it follows that $a^{p^m+t} \in L(A_1^{p^m}) \cap L(A_2^{p^m})$. However, t is not divisible by p^m, thus, $a^{p^m+t} \notin L_n$ which is a contradiction to the assumption that the automata $A_1^{p^m}$ and $A_2^{p^m}$ form a nontrivial decomposition of the automaton A_n.

(ii) We shall now prove, that if n is not a power of prime, then the language L_n is decomposable. Let $p_1^{m_1}p_2^{m_2}\dots p_r^{m_r}$ be the prime factorisation of n. Since n is not a power of prime $r \geq 2$ holds. Let us denote $l_1 = p_1^{m_1}$ and $l_2 = p_2^{m_2}\dots p_r^{m_r}$. Let A_1^n and A_2^n be the NFA given by the transition diagrams in Fig. 2.
It is easy to see that these automata form a decomposition of A_n. [For, a word a^s belongs to $L(A_1^n) \cap L(A_2^n)$ iff s is divisible by both l_1 and l_2, i.e., iff s is divisible by $l_1 l_2$ which is iff s is divisible by $n = l_1 l_2$ and thus iff a^s belongs to $L_n = L(A_n)$.]
Since $\#_S(A_1^n) < \#_S(A_n)$ and $\#_S(A_2^n) < \#_S(A_n)$, this decomposition is nontrivial and the proof is complete. $\square$

The second subfamily of regular languages considered is the family of singleton languages, i.e., languages consisting of one word only.

Theorem 3. *Let w be a word, let $L_w = \{w\}$. Then L_w is decomposable if and only if w contains at least two distinct symbols.*

Proof. First we shall show that $nsc(L_w) = |w| + 1$. Let $w = c_1 \dots c_n$ where $c_1, \dots, c_n$ are symbols. Let A_w be the automaton for L_w given by its transition diagram in Fig. 3. Thus $nsc(L_w) \leq \#_S(A_w) = |w|+1$. Using the fooling set technique of Theorem 1 and the set of pairs of words $F = \{(pref(w,i), suf(w,|w|-i)) \mid 0 \leq i \leq |w|\}$ of size $|w| + 1$ we can see that $nsc(L_n) \geq |w| + 1$. Thus $nsc(L_n) = |w| + 1$ and A_w is a minimal NFA for L_w.

(i) Suppose w does not contain two distinct symbols, i.e., $w = a^n$ for some $n \in \mathbb{N}$.

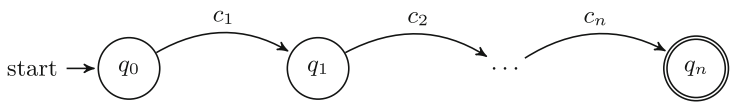


Fig. 3. The automaton A_w.

As mentioned in Note 1, L_w is undecomposable for $n = 0$ or $n = 1$. Consider $n \geq 2$. Suppose the language L_w is decomposable, i.e., there exists a nontrivial decomposition of the automaton A_w given by automata A_1^w and A_2^w.
Let $(p_0, a^n) \vdash (p_1, a^{n-1}) \vdash \ldots \vdash (p_{n-1}, a) \vdash (p_n, \varepsilon)$, where p_0 is the initial state of A_1^w and p_n is one of the accepting states of A_1^w, be an accepting computation of A_1^w on w. Since $\#_S(A_1^w) < |w| + 1$, there exist i and j in $\mathbb{N}$ such that $0 \leq i < j \leq n$ and $q_i = q_j$. Thus there exists r_1 in $\mathbb{N}, 1 \leq r_1 \leq n$, such that for all k in $\mathbb{N}$ we have a^{n+kr_1} in $L(A_1^w)$. Similarly there exists r_2 in $\mathbb{N}, 1 \leq r_2 \leq n$, such that for all k in $\mathbb{N}$ we have a^{n+kr_2} in $L(A_2^w)$. We thus have $a^{n+r_1r_2} \in L(A_1^w) \cap L(A_2^w) = L_w$ which is a contradiction.

(ii) Let $w = c_1 \ldots c_n$ contain at least two distinct symbols which we shall denote a and b. Hence we can write $w = c_1 \ldots c_{i-1}abc_{i+2} \ldots c_n$. Thus the automaton A_w defined above can be redrawn as shown in Fig. 4.

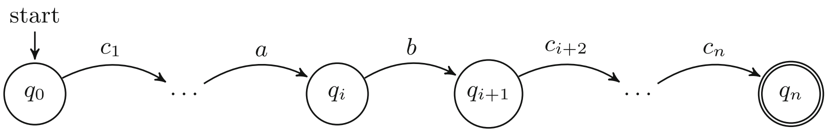


Fig. 4. The automaton A_w with two distinct symbols in w.

We shall construct a nontrivial decomposition of A_w. Automata A_w^a and A_w^b forming this decomposition are given by their transition diagrams in Fig. 5. Let $w_1 = c_1 \ldots c_{i-1}$ and $w_2 = c_{i+2} \ldots c_n$. Clearly $L(A_w^a) = \{w_1a^kbw_2 \mid k \in \mathbb{N}\}$, $L(A_w^b) = \{w_1ab^kw_2 \mid k \in \mathbb{N}\}$ and $L(A_w^a) \cap L(A_w^b) = \{w\}$. Since $\#_S(A_w^a) < \#_S(A_w)$ and $\#_S(A_w^b) < \#_S(A_w)$ the automata A_w^a and A_w^b form a nontrivial decomposition of A_w. □

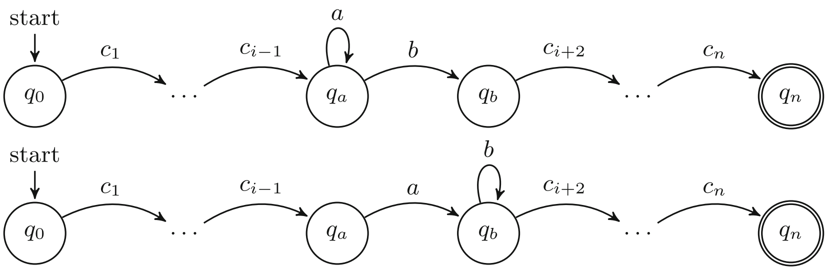


Fig. 5. A decomposition of A_w into automata A_w^a and A_w^b.

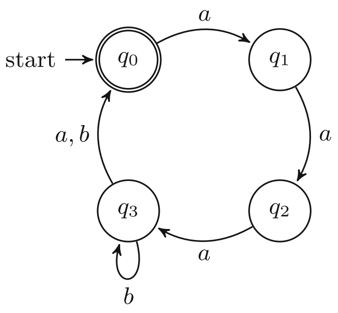


Fig. 6. The automaton A_4^N

4.2 Deterministic vs. Nondeterministic Decomposability

In this section we shall explore differences arising between the deterministic and nondeterministic finite state automata settings. We shall exhibit an infinite sequence of regular languages such that every language in that sequence is deterministically decomposable but nondeterministically undecomposable. Thus, for the problems in this sequence there is no useful information simplifying their solution in the nondeterministic setting but one can find useful advice to simplify the solution in the deterministic setting. Moreover, the advice in the deterministic setting helps substantially in the sense that for almost all decompositions in this sequence it holds that the size of both automata in the decomposition is about one half of the size of the original automaton.

Theorem 4. *There exists a sequence of regular languages* $(L_i)_{i=2}^{\infty}$ *such that the following holds:*

(a) The language L_i *is nondeterministically undecomposable but deterministically decomposable for every* $i \in \mathbb{N}, i \geq 2$*.*
(b) For each $i \in \mathbb{N}, i \geq 3$ *let* A_i *be the minimal DFA accepting* L_i*. Then there exists a decomposition of* A_i *into DFAs* A_1^i *and* A_2^i *such that* $\#_S(A_1^i) = \#_S(A_2^i) = \frac{\#_S(A_i)-1}{2} + 2$*.*

Proof. Let $(L_i')_{i=2}^{\infty}$ be a sequence of languages defined by $L_i' = (\{a^{i-1}\}\{b\}^*\{a,b\})^*$ for every $i \geq 2$. We shall construct a sequence $(L_i)_{i=2}^{\infty}$ satisfying (a) and (b) of the theorem by selecting some (infinitely many) members of $(L_i')_{i=2}^{\infty}$.

First, we shall show that for (infinitely many) indices i such that i is a power of prime the language L_i' is nondeterministically undecomposable. Let us construct NFAs A_i^N such that $L(A_i^N) = L_i'$. Let $A_i^N = (K_i^N, \{a,b\}, \delta_i^N, q_0, \{q_0\})$ where $K_i^N = \{q_j \mid 0 \leq j < i\}$ and the transition function δ_i^N is defined as follows - $\delta_i^N(q_{i-1}, b) = \{q_0, q_{i-1}\}$, $\forall j \in \mathbb{N}, 0 \leq j \leq i-1\colon \delta_i^N(q_j, a) = \{q_{(j+1) \bmod i}\}$. We illustrate this construction by showing the automaton A_4^N by its transition diagram in Fig. 6.

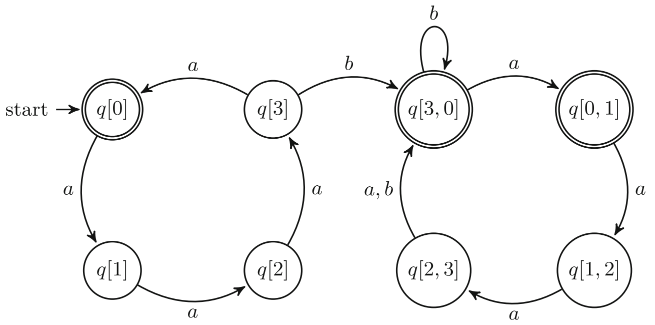


Fig. 7. The automaton A_4^D

Clearly $L(A_i^N) = L'_i$ for all i. Note that $\{a^{ki} \mid k \in \mathbb{N}\} \subset L'_i$. Using the fooling set technique of Theorem 1 with the set of pairs of words $M_i = \{(a^j, a^{i-j}) \mid 0 \leq j < i\}$ of size i we obtain $i \leq nsc(L'_i) \leq \#_S(A_i^N) = i$. Hence $nsc(L'_i) = i$ and the automaton A_i^N is a minimal NFA for the language L'_i.

We shall now show that for those i which are a power of prime the languages L'_i are nondeterministically undecomposable. Let us suppose that, to the contrary, there exists a nontrivial decomposition of A_i^N given by NFAs $A_1^{N,i}$ and $A_2^{N,i}$ with $\#_S(A_1^{N,i}) < i$, $\#_S(A_2^{N,i}) < i$, and $L(A_1^{N,i}) \cap L(A_2^{N,i}) = L'_i$ for each such i.

Since i is a power of a prime, $i = p^n$ for some prime p and n in $\mathbb{N}$. Since $a^{p^n} \in L(A_1^{N,i})$ and $a^{p^n} \in L(A_2^{N,i})$ there exist accepting computations of both $A_1^{N,i}$ and $A_2^{N,i}$ on the word a^{p^n}. Using an analogous argumentation as in part (i) of the proof of Theorem 2 one can find a word that belongs to $L(A_1^{N,i}) \cap L(A_2^{N,i})$ but does not belong to L'_i which is a contradiction.

We shall now show that L'_i is deterministically decomposable for arbitrary $i \geq 2$. Let the DFA A_i^D accepting L'_i be obtained from NFA A_i^N using the standard subset construction. Thus we define A_i^D as follows:
$A_i^D = (K_i^D \cup \{q_T\}, \{a, b\}, \delta_i^D, q[0], F_i^D)$, where $F_i^D = \{q[0], q[i-1, 0], q[0, 1]\}$, $K_i^D = \{q[j], q[j, (j+1) \bmod i] \mid 0 \leq j < i\}$ and the transition function δ_i^D is defined in an obvious way. For illustration we exhibit A_4^D by its transition diagram in Fig. 7. For simplicity we omit the trash state q_T in the diagram.

One can show minimality of A_i^D using standard techniques (e.g., Myhill-Nerode theorem).

We shall now construct a nontrivial decomposition of the DFA A_i^D. The main idea of the decomposition is, that the automata of the decomposition will not contain both cycles, which are present in A_i^D, but one of them will be 'collapsed' to one state. When considering words accepted by both automata, they will have each of their 'two parts' checked by one of the two automata thus guaranteeing they have the proper form for L'_i. The decomposition will work properly thanks to the fact that both cycles in A_i^D are separated by just one transition on b from the first cycle to the second one while we do not use transitions on b in the first

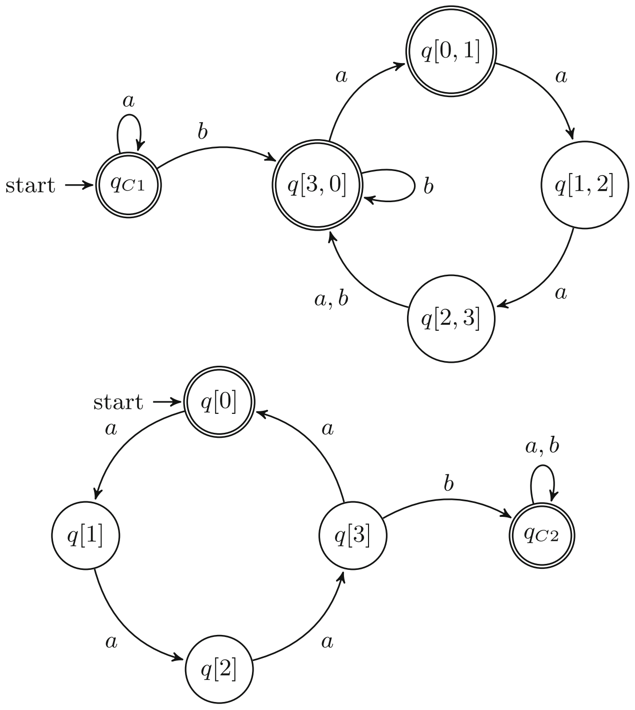


Fig. 8. A decomposition of A_4^D to $A_1^{D,4}$ and $A_2^{D,4}$

cycle. We denote the automata in the decomposition by $A_1^{D,i}$ and $A_2^{D,i}$. We shall illustrate the construction by presenting the decomposition of the automaton A_4^D to automata $A_1^{D,4}$ and $A_2^{D,4}$ by transition diagrams in Fig. 8. For simplicity we omit trash state q_T in the diagrams.

Formally, let us define the automata $A_1^{D,i}$ and $A_2^{D,i}$ as follows:

1. $A_1^{D,i} = (K_1^{D,i} \cup \{q_{C1}, q_T\}, \{a,b\}, \delta_1^{D,i}, q_{C1}, F_1^{D,i})$, where $K_1^{D,i} = \{q[j, (j+1) \bmod i] \mid 0 \le j < i\}$, $F_1^{D,i} = \{q_{C1}, q[i-1,0], q[0,1]\}$ where the transition function $\delta_1^{D,i}$ is defined as follows: $\delta_1^{D,i}(q_{C1}, a) = q_{C1}$, $\delta_1^{D,i}(q_{C1}, b) = q[i-1,0]$, $\delta_1^{D,i}(q[i-1,0], b) = q[i-1,0]$, $\delta_1^{D,i}(q[i-2,i-1], b) = q[i-1,0]$, $\forall q[j,k] \in K_1^{D,i}$: $\delta_1^{D,i}(q[j,k], a) = q[(j+1) \bmod i, (k+1) \bmod i]$. All transitions not explicitly mentioned are defined to lead to the trash state q_T.
2. $A_2^{D,i} = (K_2^{D,i} \cup \{q_{C2}, q_T\}, \{a,b\}, \delta_2^{D,i}, q[0], F_2^{D,i})$, where $K_2^{D,i} = \{q[j] \mid 0 \le j < i\}$, $F_2^{D,i} = \{q[0], q_{C2}\}$ where the transition function $\delta_2^{D,i}$ is defined as follows: $\delta_2^{D,i}(q[i-1], b) = q_{C2}$, $\delta_2^{D,i}(q_{C2}, a) = q_{C2}$, $\delta_2^{D,i}(q_{C2}, b) = q_{C2}, \forall q[j] \in K_2^{D,i}$: $\delta_2^{D,i}(q[j], a) = q[(j+1) \bmod i]$. All transitions not explicitly mentioned are defined to lead to the trash state q_T.

We show that $L(A_i^D) = L(A_1^{D,i}) \cap L(A_2^{D,i})$. Since the inclusion $L(A_i^D) \subseteq L(A_1^{D,i}) \cap L(A_2^{D,i})$ follows directly from the construction we shall concentrate on the reverse inclusion.

Let w be in $L(A_1^{D,i}) \cap L(A_2^{D,i})$. There has to be an accepting computation on w in both automata. Thus there exist states $q_{F1} \in F_1^{D,i}, q_{F2} \in F_2^{D,i}$ so that $(q_{C1}, w) \vdash^*_{A_1^{D,i}} (q_{F1}, \varepsilon), (q[0], w) \vdash^*_{A_2^{D,i}} (q_{F2}, \varepsilon)$. We shall consider two cases based on the possibilities for q_{F1}.

(i) $q_{F1} = q_{C1}$.
It follows from the construction of $A_1^{D,i}$ that the computation $(q_{C1}, w) \vdash^*_{A_1^{D,i}} (q_{F1}, \varepsilon)$ uses only state q_{C1}. Therefore there exists $n \in \mathbb{N}$ such that $w = a^n$. It follows from the construction of $A_2^{D,i}$ that the accepting computation $(q[0], w) \vdash^*_{A_2^{D,i}} (q_{F2}, \varepsilon)$ of $A_2^{D,i}$ on w uses only the states in $\{q[j] \mid 0 \leq j < i\}$. Thus $q_{F2} = q[0]$ and the computation $(q[0], w) \vdash^*_{A_i^D} (q[0], \varepsilon)$ is an accepting computation of A_i^D on the word w. Informally, the automaton A_i^D uses only its first cycle, which is completely contained also in $A_2^{D,i}$.

(ii) $q_{F1} \in \{q[i-1, 0], q[0, 1]\}$.
It follows from the construction of $A_1^{D,i}$ that there exist some $n \in \mathbb{N}$ and $u \in \{a, b\}^*$ so that $w = a^n bu$. The computation of $A_1^{D,i}$ on the word w then looks as follows: $(q_{C1}, a^n bu) \vdash^*_{A_1^{D,i}} (q_{C1}, bu) \vdash_{A_1^{D,i}} (q[i-1, 0], u) \vdash^*_{A_1^{D,i}} (q_{F1,\varepsilon})$. Since w contains the symbol b it follows from the construction of $A_2^{D,i}$ that $q_{F2} = q_{C2}$ and the computation of the automaton $A_2^{D,i}$ on the word w looks as follows: $(q[0], a^n bu) \vdash^*_{A_2^{D,i}} (q[i-1], bu) \vdash_{A_2^{D,i}} (q_{C2}, u) \vdash^*_{A_2^{D,i}} (q_{C2}, \varepsilon)$. It follows from the construction of $A_1^{D,i}$ that the computation $(q[i-1, 0], u) \vdash^*_{A_1^{D,i}} (q_{F1}, \varepsilon)$ in $A_1^{D,i}$ is also a computation $(q[i-1, 0], u) \vdash^*_{A_i^D} (q_{F1}, \varepsilon)$ in A_i^D. It follows from the construction of $A_2^{D,i}$ that the computation $(q[0], a^n bu) \vdash^*_{A_2^{D,i}} (q[i-1], bu)$ in $A_2^{D,i}$ is also a computation $(q[0], a^n bu) \vdash^*_{A_i^D} (q[i-1], bu)$ in A_i^D. Moreover, it holds $\delta_i^D(q[i-1], b) = q[i-1, 0]$. Therefore we have $(q[0], a^n bu) \vdash^*_{A_i^D} (q[i-1], bu) \vdash_{A_i^D} (q[i-1, 0], u) \vdash^*_{A_i^D} (q_{F1,\varepsilon})$. Since $q_{F1} \in F_i^D$, this computation is an accepting computation in the automaton A_i^D on the word w. Informally, both automata in the decomposition check part of the input in one of the cycles of the original automaton and just wait in the other. So we have the input checked by both cycles in the intersection.

Thus in all possible cases for q_{F1} we have w in $L(A_i^D)$ and the proof is complete.

Clearly $\#_S(A_1^{D,i}) < \#_S(A_i^D), \#_S(A_2^{D,i}) < \#_S(A_i^D)$, so that automata $A_1^{D,i}$ and $A_2^{D,i}$ form a nontrivial decomposition of the automaton A_i^D. Therefore L_i' is deterministically decomposable for arbitrary $i \geq 2$.

Let us summarize what we have proven. For the sequence of languages $(L_i')_{i=2}^{\infty}$ it holds that it contains infinitely many languages, which are nondeterministically undecomposable but deterministically decomposable. These languages are those L'_i for which i is a power a prime. Thus, we obtain $(L_i)_{i=2}^{\infty}$ as a subsequence of $(L'_i)_{i=2}^{\infty}$ by selecting those L'_i for which i is a power of a prime. Clearly $(L_i)_{i=2}^{\infty}$ satisfies (a). Moreover, from the construction of automata $A_1^{D,i}$ and $A_2^{D,i}$ in the deterministic decomposition of A_i^D it follows that (b) of the statement of the theorem holds as well. □

5 Conclusion

We have shown that a formal framework can be defined to formalise and study some informally used attributes of information. Our framework opens many possible avenues to explore. Varying possible instances of (i) to (iv) in our setting should provide more insight and may bring us closer to understanding the notion of information. Formalizing additional aspects of information may lead to extensions of our framework.

References

1. Böckenhauer, H.-J., Hromkovič, J., Mömke, T., Widmayer, P.: On the hardness of reoptimization. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 50–65. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77566-9_5
2. Dobrev, S., Královič, R., Pardubská, D.: How much information about the future is needed? In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 247–258. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77566-9_21
3. Even, S., Selman, A.L., Yacobi, Y.: The complexity of promise problems with applications to public-key cryptography. Inf. Control **61**(2), 159–173 (1984)
4. Gaži, P., Rovan, B.: Assisted problem solving and decompositions of finite automata. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 292–303. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77566-9_25
5. Ginsburg, S.: Algebraic and Automata-Theoretic Properties of Formal Languages. Elsevier Science Inc., New York (1975)
6. Glaister, I., Shallit, J.: A lower bound technique for the size of nondeterministic finite automata. Inf. Process. Lett. **59**(2), 75–77 (1996)
7. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2006)
8. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15155-2_3
9. Kováč, I.: Supplementary information and complexity of finite automata. Ph.D. thesis, Comenius University (2015)
10. Labath, P., Rovan, B.: Simplifying DPDA using supplementary information. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 342–353. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21254-3_27

11. Rovan, B., Zeman, M.: Modeling time criticality of information. Inf. Process. Lett. **114**(3), 147–151 (2014)
12. Steskal, L.: On usefulness of information: a computational approach. Ph.D. thesis, Comenius University (2010)
13. Vida, B.: Using transformation in solving problems with supplementary information. Master thesis, Comenius University (2015)

Parikh Matrices: Subword Indicators and Degrees of Ambiguity

Arto Salomaa(✉)

Turku Centre for Computer Science, University of Turku, Quantum Building 392, 20014 Turun yliopisto, Finland
asalomaa@utu.fi

Abstract. The quantity $|w|_u$, the number of occurrences of a word u as a (scattered) subword of a word w gives important numerical information about the word w. Properly chosen values $|w|_u$, for different u's, characterize the word w completely. Certain upper triangular matrices, customarily referred to as *Parikh matrices* have turned out to be very useful for computing numbers $|w|_u$. This partially expository paper discusses some highlights and open problems of the theory of Parikh matrices and subword occurrences. Special emphasis is on subword indicators and degrees of ambiguity.

1 Introduction

The most direct numerical fact about a word w is its *length*, in symbols $|w|$. The *Parikh vector*, [6], $\Psi(w) = (i_1, \ldots, i_n)$ indicates the number of occurrences of the letter a_j, $1 \leq j \leq n$, in w, provided w is over the alphabet $\{a_1, \ldots, a_n\}$. To get more information about a word, one has to focus the attention to the number of occurrences of a specific subword in the given word. The word u being a *subword* of w means that w, as a sequence of letters, contains u as a subsequence. More specifically, there exist letters $x_1, \ldots, x_k$ and words $y_0, \ldots, y_k$, some of them possibly empty, such that

$$u = x_1 \ldots x_k \quad \text{and} \quad w = y_0 x_1 y_1 \ldots x_k y_k \ .$$

We also consider *factors* u of a word w: u is a factor of w if there are words x and y such that $w = xuy$. Throughout this paper, we understand subwords and factors in the way mentioned. (In classical language theory our subwords are usually called "scattered subwords", whereas our factors are called "subwords".)

We use the notation $|w|_u$ for the *number* of occurrences of the word u as a subword of the word w. This number can be defined also as follows. Occurrences can be viewed as vectors. If $|u| = t$, each occurrence of u in w can be identified as the t-tuple $(i_1, \ldots, i_t)$ of increasing positive integers, where for $1 \leq j \leq t$, the jth letter of u is the i_jth letter of w. Then $|w|_u$ equals the number of such vectors. For instance, the 3 occurrences of $u = ab$ in $w = ababa$ are $(1, 2)$, $(1, 4)$ and $(3, 4)$. We also make the *convention* that, for any w and the empty word λ, $|w|_\lambda = 1$.

© Springer Nature Switzerland AG 2018
H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 100–112, 2018.
https://doi.org/10.1007/978-3-319-98355-4_7

In [2] the number $|w|_u$ is denoted as a "binomial coefficient" $|w|_u = \binom{w}{u}$. If w and u are words over a one-letter alphabet, $w = a^i$, $u = a^j$, then $|w|_u$ equals the ordinary binomial coefficient: $|w|_u = \binom{i}{j}$. Our convention concerning the empty word reduces to the fact that $\binom{i}{0} = 1$. However, the notation using binomial coefficients is not convenient in case of general alphabets.

Assume that Σ is an alphabet containing the letters a and b. Then, for any word w,

$$(|w|_a) \cdot (|w|_b) = |w|_{ab} + |w|_{ba} \; .$$

This simple equation is one of the few general facts about occurrences of subwords. A slight variation immediately leads to difficulties. No explicit characterization is known, [4], for the relation between $(|w|_u, |w|_v)$ and $(|w|_{uv}, |w|_{vu})$, where u, v, w are arbitrary words.

2 Parikh Matrices and Generalizations

We now consider a sharpening of the Parikh mapping, introduced in [3], which gives information also about the location of different letters in a word. The mapping uses upper triangular square matrices, with non-negative integer entries, 1's on the main diagonal and 0's below it. The set of all such triangular matrices is denoted by $\mathcal{M}$, and the subset of all matrices of dimension $k \geq 1$ is denoted by $\mathcal{M}_k$. Two words with the same Parikh matrix always have the same Parikh vector, but two words with the same Parikh vector have in general different Parikh matrices.

The notion of a Parikh matrix mapping is defined as follows. In what follows, small letters a, b, c, d, possibly with indices, from the beginning of the English alphabet denote letters of the formal alphabet.

Definition 1. *Let $\Sigma = \{a_1, \ldots, a_k\}$ be an ordered alphabet. The* Parikh matrix mapping, *denoted Ψ_k, is the morphism $\Psi_k \colon \Sigma^* \to \mathcal{M}_{k+1}$, defined as follows. Consider an integer q, $1 \leq q \leq k$. If we denote $\Psi_k(a_q) = (m_{i,j})_{1 \leq i,j \leq (k+1)}$, then, for each $1 \leq i \leq (k+1)$, $m_{i,i} = 1$, $m_{q,q+1} = 1$, all other elements of the matrix $\Psi_k(a_q)$ being 0.*

The following theorem, describes the basic properties of the Parikh matrix mapping. The proof is an easy induction on the length of the word w. Details can be found in [3]. For $\Sigma = \{a_1, \ldots, a_k\}$, we denote by $a_{i,j}$ the word $a_i a_{i+1} \ldots a_j$, where $1 \leq i \leq j \leq k$.

Theorem 1. *Consider $\Sigma = \{a_1, \ldots, a_k\}$ and $w \in \Sigma^*$. The matrix $\Psi_k(w) = (m_{i,j})_{1 \leq i,j \leq (k+1)}$, has the following properties:*

- *$m_{i,j} = 0$, for all $1 \leq j < i \leq (k+1)$,*
- *$m_{i,i} = 1$, for all $1 \leq i \leq (k+1)$,*
- *$m_{i,j+1} = |w|_{a_{i,j}}$, for all $1 \leq i \leq j \leq k$.*

Parikh matrices give much more information about a word w than Parikh vectors. Theorem 1 tells that the second diagonal of the Parikh matrix of w gives the Parikh vector of w. The next diagonals give information about the order of letters in w.

There is an extensive literature concerning Parikh matrices. The reader is referred to [1,5,7–10,13–19] for various aspects of the theory, many of which will be discussed below. Our emphasis lies in the core of the theory. We do not discuss related issues such as *subword histories*, [4], or *position indices*, [11]. Some new notions are introduced in the two last sections of the paper.

The Parikh matrix mapping yields the numbers $|w|_u$, where u is a factor of the word $a_1 \dots a_k$. Knowledge about numbers such as $|w|_{baa}$ will not be obtained. This drawback can be corrected by considering *generalized Parikh matrix mappings*, originally introduced in [13,14].

The *generalized Parikh matrix mapping* Ψ_v tells the number of occurrences (as a subword) of factors of an arbitrary word $v = c_1 \cdots c_t$, where each c is a letter. Repetitions of letters are possible.

Definition 2. *Consider a word $v = c_1 \cdots c_t$ of length t over the alphabet Σ. The* generalized Parikh matrix mapping*, denoted Ψ_v, is the morphism: $\Psi_v \colon \Sigma^* \to \mathcal{M}_{t+1}$, defined as follows. Assume that the letter a occurs in the word v in positions $j_1, \dots, j_r$. Denote $\Psi_v(a) = (n_{i,j})_{1 \le i,j \le (t+1)}$. Then for each $1 \le i \le (t+1)$, $n_{i,i} = 1$. Moreover, $n_{i,i+1} = 1$ if i is one of the numbers $j_1, \dots, j_r$. All other elements of the matrix $\Psi_v(a)$ are 0.*

A result analogous to Theorem 1 is that the entry $n_{i,j+1}$, $1 \le i \le j \le t$, in the matrix corresponding to a word w equals the number $|w|_{c_i \cdots c_j}$. The proof of this result, based on using matrix products, is again straightforward.

For instance, let $v = ababa$ and let w be arbitrary. Then

$$\Psi_{ababa}(w) = \begin{pmatrix} 1 & |w|_a & |w|_{ab} & |w|_{aba} & |w|_{abab} & |w|_{ababa} \\ 0 & 1 & |w|_b & |w|_{ba} & |w|_{bab} & |w|_{baba} \\ 0 & 0 & 1 & |w|_a & |w|_{ab} & |w|_{aba} \\ 0 & 0 & 0 & 1 & |w|_b & |w|_{ba} \\ 0 & 0 & 0 & 0 & 1 & |w|_a \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Choosing $w = baabaabaab$, we obtain

$$\Psi_{ababa}(baabaabaab) = \begin{pmatrix} 1 & 6 & 12 & 12 & 16 & 8 \\ 0 & 1 & 4 & 12 & 20 & 16 \\ 0 & 0 & 1 & 6 & 12 & 12 \\ 0 & 0 & 0 & 1 & 4 & 12 \\ 0 & 0 & 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The following fundamental result, [13], shows how every generalized Parikh matrix is related to a Parikh matrix. The result can be used for transferring facts concerning subword occurrences based on Parikh matrices to corresponding facts based on generalized Parikh matrices. A typical example of this is the inequality considered in [12].

Theorem 2. *For every generalized Parikh matrix mapping $\Psi_v(w)$, $|v| = t$, an ordered alphabet $\Sigma_t = \{a_1, \ldots, a_t\}$ and a word $w' \in \Sigma_t^*$ such that*

$$\Psi_v(w) = \Psi_t(w')$$

can be effectively constructed. Here, Ψ_t is a Parikh matrix mapping.

Proof. Number the occurrences of the letters in v by $1, \ldots, t$. The numbers $1, \ldots, t$ constitute the alphabet Σ_t. Consider the alphabet Σ of v. A morphism h of Σ^* into Σ_t^* is defined as follows. Let $a \in \Sigma$ appear in v in positions

$$p_1, p_2, \ldots, p_q,\ 1 \le q \le t,\ p_i < p_{i+1}, 1 \le i \le q-1 .$$

Then we define

$$h(a) = p_q p_{q-1} \cdots p_1 .$$

Finally, for $w \in \Sigma^*$, we choose $w' = h(w)$.

In our example $v = ababa$ above, $h(a) = 531$ and $h(b) = 42$. Hence, for $w = baabaabaab$, we obtain

$$w' = 4253153142531531425315314 2 .$$

For the (ordered) alphabet $\Sigma_5 = \{1, 2, 3, 4, 5\}$, we now have

$$\Psi_5(42531531425315314253153142) = \begin{pmatrix} 1 & 6 & 12 & 12 & 16 & 8 \\ 0 & 1 & 4 & 12 & 20 & 16 \\ 0 & 0 & 1 & 6 & 12 & 12 \\ 0 & 0 & 0 & 1 & 4 & 12 \\ 0 & 0 & 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} .$$

The result equals the generalized Parikh matrix obtained above.

The reverse order of the numbers p_i in the definition of the morphism is necessary because, otherwise, additional false occurrences are created. For instance, choose $v = aa$ and define $h(a) = 12$. Then $\Psi_v(aaa) \in \mathcal{M}_3$ has the number 3 in the upper right-hand corner. However, $\Psi_2(121212)$ has the number 6 in the upper right-hand corner.

The proof of Theorem 2 is now completed by induction on the length of the word w. Consider the morphism h and the word w' defined above. Clearly the theorem holds if w is a letter. Assuming that Theorem 2 holds for words w of length n and consider $h(a) \in \Sigma_t^+,\ a \in \Sigma$. Now we obtain $\Psi_v(wa) = \Psi_v(w)\Psi_v(a)$. By the induction hypothesis, $\Psi_v(w) = \Psi_t(h(w))$. Hence,

$$\Psi_v(wa) = \Psi_t(h(w))\Psi_v(a) = \Psi_t(h(w))\Psi_t(h(a)) .$$

The equation

$$\Psi_v(wa) = \Psi_t(h(wa))$$

follows by matrix multiplication. The factor $\Psi_t(h(a))$ increases appropriate entries in the matrix $\Psi_t(h(w))$. □

Coming back to our example, assume that we have established that

$$\Psi_{ababa}(baaba) = \Psi_5(4253153142531) = \begin{pmatrix} 1&3&2&2&0&0\\ 0&1&2&4&2&2\\ 0&0&1&3&2&2\\ 0&0&0&1&2&4\\ 0&0&0&0&1&3\\ 0&0&0&0&0&1 \end{pmatrix},$$

and want to establish

$$\Psi_{ababa}(baabaa) = \Psi_5(4253153142531531)\ .$$

Clearly,

$$\Psi_5(531) = \begin{pmatrix} 1&1&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&1&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&1\\ 0&0&0&0&0&1 \end{pmatrix}.$$

Hence,

$$\Psi_{ababa}(baabaa) = \begin{pmatrix} 1&3&2&2&0&0\\ 0&1&2&4&2&2\\ 0&0&1&3&2&2\\ 0&0&0&1&2&4\\ 0&0&0&0&1&3\\ 0&0&0&0&0&1 \end{pmatrix} \cdot \begin{pmatrix} 1&1&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&1&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&1\\ 0&0&0&0&0&1 \end{pmatrix}.$$

Finally, we obtain

$$\Psi_{ababa}(baabaa) = \begin{pmatrix} 1&4&2&4&0&0\\ 0&1&2&6&2&4\\ 0&0&1&4&2&4\\ 0&0&0&1&2&6\\ 0&0&0&0&1&4\\ 0&0&0&0&0&1 \end{pmatrix} = \Psi_5(4253153142531531)\ .$$

3 Ambiguity and Decision Problems

Only in some special cases the matrix $\Psi(w)$ or the generalized matrix $\Psi_v(w)$ defines uniquely the word w. We begin with the following definition. (We skip the corresponding definition for generalized Parikh matrices.)

Definition 3. *Let k and Ψ_k be as in Definition 1. Two words $w_1, w_2 \in \Sigma_k^*$ are termed M-*equivalent*, in symbols $w_1 \equiv_M w_2$, if $\Psi_k(w_1) = \Psi_k(w_2)$. A word $w \in \Sigma_k^*$ is termed M-*unambiguous* if there is no word $w' \neq w$ such that $w \equiv_M w'$. Otherwise, w is termed M-*ambiguous*. If $w \in \Sigma_k^*$ is M-unambiguous (resp. M-ambiguous), then also the Parikh matrix $\Psi_k(w)$ is called unambiguous (resp. ambiguous).*

A word being M-unambiguous means that it is uniquely determined by its Parikh matrix. If the analogous notion is defined for Parikh vectors, then only the words over a one-letter alphabet are Parikh vector unambiguous.

In the sequel we will often drop the letter M and speak simply of *unambiguous and ambiguous words.* It should be emphasized that in all notions involving Parikh matrices we deal with a specific ordering of the basic alphabet Σ_k.

Obviously, unambiguous words cannot possess ambiguous factors. More specifically, if a word $y \in \Sigma_k^*$ is ambiguous, so is every word xyz, where $x, z \in \Sigma_k^*$. However, unambiguous words may possess ambiguous subwords. For instance, the unambiguous word *abcba* has the ambiguous subword *abba.*

Some short ambiguous words are listed in the following lemma. The proof is easy and is omitted.

Lemma 1. *Consider the alphabet $\Sigma_k = \{a_1, \ldots, a_k\}$. The words*

$$a_i a_{i+j} \text{ and } a_{i+j} a_i,\ 1 \le i \le k-2,\ 2 \le j \le k-i\ ,$$

are ambiguous. So are each of the words

$$\begin{array}{ll} a_i a_{i+1} a_{i+1} a_i, & a_{i+1} a_i a_i a_{i+1}, \\ a_i a_{i+1} a_i^j a_{i+1} a_i, & a_{i+1} a_i^{j+2} a_{i+1}, \\ a_{i+1} a_i a_{i+1}^j a_i a_{i+1}, & a_i a_{i+1}^{j+2} a_i, \end{array}$$

where $1 \le i \le k-1$ and $j \ge 0$.

Ambiguous words can be characterized explicitly in case of a two-letter alphabet. For a proof of the following result the reader is referred to, for instance, [7]. An explicit characterization of unambiguous words over a three-letter alphabet was given in [14].

Theorem 3. *A word in $\{a, b\}^*$ is ambiguous if and only if it contains factors ab and ba which do not overlap. A word is unambiguous if and only if it belongs to the language denoted by the regular expression*

$$a^*b^* + b^*a^* + a^*ba^* + b^*ab^* + a^*bab^* + b^*aba^*\ .$$

The regular language mentioned in Theorem 3 is accepted by a five-state non-deterministic finite automaton.

A notion important in connection with M-equivalence is the notion of a *print* introduced and investigated in [14, 15].

Definition 4. *A word w' is the* print *of w if w' results from w by replacing every factor a^i, $i > 1$, where a is a letter, with a. A word is* printed *if it equals its print.*

For instance, the print of $ac^4a^2c^2ab^8abc^3b^5$ is *acacababcb.*

It was shown in [15] that, for each alphabet, the set of unambiguous printed words is finite. An upper bound for the size of this set, depending on the size of the alphabet, can be effectively computed.

The following related theorem is established in also [15].

Theorem 4. *Consider the alphabet $\Sigma_k = \{a_1, \ldots, a_k\}$, $k \geq 3$. There is an integer $N(k)$ such that every factor of length $\geq N(k)$ of the infinite word $(a_1a_2a_3 \cdots a_k)^\omega$ is ambiguous.*

The following *Şerbănuţă's conjecture* was open for a long time.

Conjecture 1. The unambiguity of a word implies the unambiguity of its print.

The conjecture holds for alphabets with 2 or 3 letters. It has been shown false in [19] for bigger alphabets. Indeed, the printed word *cbcbabcdcbabcbc* is ambiguous because it is M-equivalent with the word *bccabcbdbcbaccb*. However, the word $cbcbabc^2dcbabcbc$ is unambiguous.

In view of the result concerning unambiguous prints, Conjecture 1 would imply that all unambiguous words are found among those having "short" prints. Thus, all words with prints whose length exceeds a certain bound are ambiguous. This might still be true, since the counterexample to Conjecture 1 concerns a word with a short print. We would like to present the following conjecture.

Conjecture 2. There is a constant k, depending on the cardinality of the alphabet, such that, if the length of the print of a word w exceeds k, then word w is ambiguous.

Typical decision problems concerning Parikh matrices are the following ones.

1. Is a given matrix a Parikh matrix?
2. Given a word, decide whether or not it is M-ambiguous.
3. Are two given words M-equivalent?

The problem (3) amounts to computing two Parikh matrices and is clearly of polynomial time complexity. Straightforward algorithms for the problems (1) and (2), where one tests through all of the finitely many possibilities, result in an exponential time complexity. However, a really explicit characterization of Parikh matrices and unambiguous words is still missing.

It is easy to give examples of different Parikh matrices whose second and third diagonals coincide. However, it is not so easy to give examples of different Parikh matrices whose second, third and fourth diagonals coincide. Indeed, it was sometimes conjectured that such examples do not exist. It turns out that, no matter how many diagonals we fix, the Parikh matrix is still not necessarily determined.

Let Σ_k, $k \geq 2$, and Ψ_k be as in Definition 1. Consider integers i and j, $1 \leq i < j \leq k+1$. The entry (i, j) is said to be *independent* in the mapping Ψ_k if there are words $w, w' \in \Sigma_k^*$ such that the matrices $\Psi_k(w)$ and $\Psi_k(w')$ coincide elsewhere, but the (i, j)th entries in the two matrices are different. The following result was established in [9].

Theorem 5. *Every entry is independent in the mapping Ψ_k.*

4 Subword Indicators

In the original definition of a Parikh matrix mapping, Definition 1, we considered the ordered alphabet $\Sigma_k = \{a_1, \ldots, a_k\}$. The resulting matrices provided information about the subword occurrences (in the argument word of the mapping) of all factors of the word $a_1 \cdots a_k$. Similar information about the subword occurrences of all factors of an arbitrary word v was obtained in the matrices of a generalized Parikh matrix mapping. Going back to Definition 2, we refer to the word v as the *subword indicator* of the generalized matrix mapping.

Thus, the word $a_1 \cdots a_k$ is the subword indicator of the mapping in Definition 1. An extensive problem area, not at all investigated so far, consists of studying the change in the matrices affected by a change of the subword indicator. A special case is to study cases, where a change in the subword indicator does not affect the matrix, for some specific argument words. The following result gives a simple example.

Lemma 2. *The equation*

$$\Psi_{ab}(w) = \Psi_{ba}(w), \ \ w \in \{a, b\}^*$$

holds only for ambiguous words w.

Proof. Considering the entries in the matrices, we infer that a necessary and sufficient condition for the equation $\Psi_{ab}(w) = \Psi_{ba}(w)$ is that both of the equations $|w|_a = |w|_b$ and $|w|_{ab} = |w|_{ba}$ hold. According to Theorem 3, no unambiguous word satisfies these equations. $\square$

Another simple observation is the following

Lemma 3. *The equation*

$$\Psi_{ab}(w) = \Psi_{ba}(w), \ \ w \in \{a, b\}^*$$

implies that $|w|_a = |w|_b$ is even.

Proof. As seen above, we must have $|w|_a = |w|_b$. This number being odd would contradict the equation $(|w|_a) \cdot (|w|_b) = |w|_{ab} + |w|_{ba}$. $\square$

Consider now permutations p of the (ordered) set $\{a_1, \ldots, a_k\}$. We apply such permutations to words letterwise. For instance, if p is the circular permutation (abc), then $p(acabcb) = babcac$.

Definition 5. *A permutation p of the set $\{a_1, \ldots, a_k\}$, $k \geq 2$, is* Parikh-friendly *if there is a word w, containing all of the letters $a_1, \ldots, a_k$ and referred to as a* witness*, such that*

$$\Psi_{a_1 \cdots a_k}(w) = \Psi_{p(a_1 \cdots a_k)}(w) \ .$$

It is necessary to assume that the witness word w contains all the letters because, otherwise, every permutation is trivially Parikh-friendly.

For instance, consider the alphabet $\{a, b, c, d, e\}$. The circular permutation $p = (abcde)$ is Parikh-friendly. Choosing $w = abcdeedcba$ we get

$$\Psi_{abcde}(w) = \Psi_{bcdea}(w) = \begin{pmatrix} 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The example is a special case of the following result.

Theorem 6. *For any $k \geq 2$, the circular permutation $p = (a_1 a_2 \cdots a_k)$ is Parikh-friendly.*

Proof. The argument is as in the example. We choose $w = a_1 a_2 \cdots a_k a_k \cdots a_2 a_1$ and observe that both $\Psi_{a_1 a_2 \cdots a_k}(w)$ and $\Psi_{a_2 a_3 \cdots a_k a_1}(w)$ assume as their value the Parikh matrix, where every entry above the main diagonal equals 2. □

The proof of Theorem 6 does not work for arbitrary circular permutations.

It is well-known that every permutation can be represented as a product of transpositions. The product of transpositions (in any order), where no two transpositions have a common element, is Parikh-friendly. Consider the alphabet $\{a, b, c, d, e, f, g\}$ and the permutation $p = (ae)(bc)(df)$. (Thus, $p(g) = g$.) Choose $w = aeeabccbdffdgg$. Then

$$\Psi_{abcdefg}(w) = \Psi_{p(abcdefg)}(w) = \begin{pmatrix} 1 & 2 & 4 & 4 & 8 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 4 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

In the witness word w the order of the transpositions, as well as the fixed letter g, is irrelevant. It is also irrelevant whether we have one or two copies of g.

The example is a special case of the following result. However, it is an open problem to characterize Parikh-friendly permutations explicitly.

Theorem 7. *Every product of transpositions, where no two transpositions have a common element, is Parikh-friendly.*

Proof. The construction of the witness follows the lines of the example. Consider the permutation p of the alphabet $\{a_1, \ldots, a_k\}$, defined by

$$p = \prod_{i=1}^{t}(b_i, c_i) \prod_{j=1}^{u}(d_j),\ 2t + u = k \ .$$

Here, the b's, c's and d's are all different and exhaust the alphabet $\{a_1, \ldots, a_k\}$ in some order. Define now

$$w = \prod_{i=1}^{t} b_i c_i^2 b_i \prod_{j=1}^{u} d_j \ .$$

It is easy to verify that

$$\Psi_{a_1 \cdots a_k}(w) = \Psi_{p(a_1 \cdots a_k)}(w) \ . \qquad \square$$

5 Degree of M-Ambiguity

We conclude this paper with some remarks concerning a very natural notion not at all studied by now.

A context-free language can be of any degree of ambiguity, including infinity. For unambiguous languages the degree of ambiguity is one. Analogous notions can be defined for M-ambiguous words and Parikh matrices.

Definition 6. *The* degree of ambiguity $d(M)$ *of a Parikh matrix M is the number of words w such that $\Psi(w) = M$. If u is one of such words, then the* degree of M-ambiguity *of u is defined to be $d(M)$.*

M-unambiguous words are understood to have the degree of M-ambiguity 1. Contrary to the situation in language theory, no word can be of infinite degree of M-ambiguity. This follows because all words w satisfying the condition $\Psi(w) = M$ have to be of same length.

However, the degree of M-ambiguity of a word can be exponential in terms of the length of the word. Consider the alphabet $\{a, b\}$ and the word

$$w(n) = a^n b^n b^n a^n, \quad n \geq 1 \ .$$

Clearly,

$$\Psi(w(n)) = \begin{pmatrix} 1 & 2n & 2n^2 \\ 0 & 1 & 2n \\ 0 & 0 & 1 \end{pmatrix} .$$

The same matrix results whenever one replaces in $w(n)$ a factor ab with the factor ba and *simultaneously* replaces a factor ba with the factor ab. The words obtainable in this fashion can be characterized as follows. Consider the $2n$ occurrences of a in a word u resulting from $w(n)$ after a sequence of such simultaneous replacements. Denote by $x_i, 1 \leq i \leq 2n$, the number of b's occurring to the right if the ith occurrence of a. Thus, for $w(n)$ we have

$$x_1 = \ldots x_n = 2n, \quad x_{n+1} = \ldots x_{2n} = 0 .$$

The words M-equivalent to $w(n)$ are determined by the following system of equations and inequalities:

$$\sum_{i=1}^{2n} x_i = 2n^2, \ 2n \geq x_i \geq 0, \ x_i \geq x_{i+1} .$$

Here in the first (resp. second) system of inequalities i ranges from 1 to $2n$ (resp. $2n-1$). We will see that the number of solutions is exponential in n.

For instance, take $n = 4$. The solution

$$x_1 = 7, \ x_2 = x_3 = 5, \ x_4 = 4, \ x_5 = x_6 = x_7 = 3, \ x_8 = 2$$

defines the word *babbaababaaababb*.

We now prove that the number of words with the Parikh matrix $\Psi(w(n))$ is exponential in n.

Lemma 4. *There is no upper bound polynomial in n for the number of words M-equivalent with $w(n)$.*

Proof. Clearly,

$$0 \leq |w|_{ab} \leq 4n^2 ,$$

for words w such that $|w|_a = |w|_b = 2n$, $n \geq 1$. We denote by $N(x)$, $0 \leq x \leq 4n^2$, the number of words w such that

$$|w|_a = |w|_b = 2n, \ |w|_{ab} = x .$$

By definition, $N(2n^2)$ equals the number of words M-equivalent with $w(n)$. On the other hand, $N(0) = N(4n^2) = 1$, and $N(x)$ increases for $0 \leq x \leq 2n^2 - 1$, and decreases for other values of x. The number of all words w with $|w|_a = |w|_b = 2$ equals the binomial coefficient $\binom{4n}{2n}$, because these words are obtained by plotting $2n$ occurrences of b into $4n$ positions. Consequently,

$$\begin{aligned} N(2n^2) &\geq \binom{4n}{2n} / (4n^2 + 1) = (4n)!/(2n)! \cdot (2n)! \cdot (4n^2 + 1) \\ &= (4n) \cdots (2n+1)/(2n) \cdots 1 \cdot (4n^2 + 1) > 2^{2n}/(4n^2 + 1) , \end{aligned}$$

which proves the lemma. □

A straightforward way of obtaining words whose degree of M-ambiguity is exponential in terms of word length is to consider the alphabet $\{a, b, c\}$. Then all words w satisfying, for a given n,

$$|w|_a = |w|_c = n, \ |w|_b = 0 ,$$

have the Parikh matrix

$$\begin{pmatrix} 1 & n & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & n \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The number of these words equals the binomial coefficient $\binom{2n}{n} \geq 2^n$. Thus, we obtain at least $2^{\sqrt{m}}$ M-equivalent words of length m.

We have established the following result.

Theorem 8. *Let Σ be an alphabet with cardinality at least* 2*. There is no polynomial $P(n)$ such that, for all $n \geq 1$, the degree of M-ambiguity of an arbitrary word of length n over Σ is less than $P(n)$.*

6 Conclusion

Many problems remain open concerning M-ambiguity and subword indicators. Tex has introduced the notion of *strong M-equivalence* and the resulting notion of *strong M-ambiguity*, [17]. By definition, two words are strongly M-equivalent if they are M-equivalent for all orderings of the alphabet. For instance, the word *babcbabcbabcbab* is strongly M-ambiguous. Tex has also initiated, [16], detailed studies about *Parikh rewriting systems.* We hope to return to these matters in another paper.

Acknowledgements. The author would like to thank the referee for careful reading of the paper and the editors for technical help concerning the final version of the paper.

Dedication

This article is dedicated to *Juraj Hromkovič* on the occasion of his 60th birthday. The article is not connected to any of Juraj's work. However, I still consider it suitable for this volume, in view of Juraj's most diverse and variegated work. I got to know Juraj already in the 80's in Bratislava, and he visited Turku and participated in the Tampere ICALP in 1988. After that he paid numerous visits to Turku and formed close contacts to some members of my research group and to researchers close to my group. I am very grateful for Juraj's participation in some festival conferences organized in Turku. In my view, Juraj's scientific work is deep, sometimes seminal. I admire his ability to combine science with innovative ideas about teaching. I think that some of his new ideas, such as the ones concerning the P versus NP problem, are not yet fully recognized. Dear Juraj, I wish you many more successful years in science and teaching, as well as health and happiness in life.

Turku, February 2018, *Arto Salomaa*

References

1. Ding, C., Salomaa, A.: On some problems of Mateescu concerning subword occurrences. Fundam. Inform. **73**, 65–79 (2006)
2. Eilenberg, S.: Automata, Languages and Machines, vol. B. Academic Press, New York (1976)
3. Mateescu, A., Salomaa, A., Salomaa, K., Yu, S.: A sharpening of the Parikh mapping. Theor. Inform. Appl. **35**, 551–564 (2001)
4. Mateescu, A., Salomaa, A., Yu, S.: Subword histories and Parikh matrices. J. Comput. Syst. Sci. **68**, 1–21 (2004)
5. Mateescu, A., Salomaa, A.: Matrix indicators for subword occurrences and ambiguity. Int. J. Found. Comput. Sci. **15**, 277–292 (2004)
6. Parikh, R.J.: On context-free languages. J. Assoc. Comput. Mach. **13**, 570–581 (1966)
7. Salomaa, A.: On the injectivity of Parikh matrix mappings. Fundam. Inform. **64**, 391–404 (2005)
8. Salomaa, A.: Connections between subwords and certain matrix mappings. Theor. Comput. Sci. **340**, 188–203 (2005)
9. Salomaa, A.: Independence of certain quantities indicating subword occurrences. Theor. Comput. Sci. **362**, 222–231 (2006)
10. Salomaa, A.: Criteria for the matrix equivalence of words. Theor. Comput. Sci. **411**, 1818–1827 (2010)
11. Salomaa, A.: Subword balance, position indices and power sums. J. Comput. Syst. Sci. **76**, 861–871 (2010)
12. Salomaa, A.: On a Cauchy-type inequality for subword occurrences. Analele Universitate Bucuresti **62**(2), 101–112 (2015)
13. Şerbănuţă, T.-F.: Extending Parikh matrices. Theor. Comput. Sci. **310**, 233–246 (2004)
14. Şerbănuţă, V.G., Şerbănuţă, T.F.: Injectivity of the Parikh matrix mappings revisited. Fundam. Inform. **73**, 265–283 (2006)
15. Şerbănuţă, V.G.: On Parikh matrices, ambiguity and prints. Int. J. Found. Comput. Sci. **20**, 151–165 (2009)
16. Tex, W.C.: Parikh matrices and Parikh rewriting systems. Fundam. Inform. **146**, 305–320 (2016)
17. Tex, W.C.: On M-ambiguity for Parikh matrices. In: Asian Mathematical Conference, AMC 2016 (2016)
18. Teh, W.C., Atanasiu, A.: On a conjecture about Parikh matrices. Theor. Comput. Sci. **628**, 30–39 (2016)
19. Tex, W.C., Atanasiu, A., Poovanandran, G.: On strongly M-ambiguous prints and Şerbănuţă's conjecture for Parikh matrices (2017, submitted)

Probabilism versus Alternation for Automata

Georg Schnitger(✉)

Institut für Informatik, Goethe-Universität Frankfurt, Frankfurt, Germany
georg.schnitger@googlemail.com

Abstract. We compare the number of states required for one-way probabilistic finite automata with a positive gap ($1P_2FAs$) and the number of states required for one-way alternating automata (1AFAs). We show that a $1P_2FA$ P can be simulated by 1AFA with an at most polynomial increase in the number $s(P)$ of states of P, provided only inputs of length at most $\text{poly}(s(P))$ are considered. On the other hand we gather evidence that the number of states grows super-polynomially if the number of alternations is bounded by a fixed constant. Thus the behavior of one-way automata seems to be in marked contrast with the behavior of polynomial-time computations.

Many thanks to Juraj for many ideas, many questions, many answers and lots of hiking.

1 Introduction

Sipser [16] showed in 1983 that the complexity class BPP of languages recognizable in polynomial time by probabilistic Turing machines with bounded error probability is contained in the polynomial hierarchy. Later Peter Gacz and independently Lautemann [15] strengthened his results to imply that $\mathsf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$ holds.

What is the relation between probabilism and alternation for one-way automata if we consider the number of states as a resource? We follow the notation in [14] and compare the number of states required by

(a) $1P_2FAs$, i.e., one-way ***p***robabilistic ***f***inite ***a***utomata with two-sided bounded error,
(b) $1\exists_k FAs$ and $1\forall_k FAs$, i.e., one-way alternating ***f***inite ***a***utomata which start in an existential resp. universal state and alternate at most $k-1$ times between existental and universal states,
(c) 1AFAs, i.e., one-way ***a***lternating ***f***inite ***a***utomata.

(See Sect. 2 for formal definitions.) For an automata M let $s(M)$ be the number of states of M. To investigate the state complexity we again borrow from [14] and

© Springer Nature Switzerland AG 2018
H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 113–126, 2018.
https://doi.org/10.1007/978-3-319-98355-4_8

define complexity classes not consisting of languages but of families of languages. In particular, for a computation mode χ define the class

$$\{(L_n)_{n\geq 1} \mid \text{there are one-way } \chi\text{-automata } M_n \text{ with } L(M_n) = L_n \\ \text{such that } s(M_n) \text{ is polynomial in } n\}.$$

As a consequence, by choosing the computation mode accordingly, we may introduce $1\mathrm{P}_2$ as the automata-version of BPP and $1\Sigma_k, 1\Pi_k, 1H, 1A$ as the automata versions of Σ_k^p, Π_k^p, PH and AP respectively.

We gather evidence that a *fixed* number of alternations is insufficient to simulate $1\mathrm{P}_2$FAs efficiently, i.e., that $1\mathrm{P}_2$ is not a subset of $1\Sigma_k$ for any $k \in \mathbb{N}$. To do so we investigate "circuit automata", a class of 1AFAs whose state transitions depend – for all but one input position – on the position and not on the symbol found in this position. Circuit automata with few alternations are quite powerful since, if states and circuit size are compared, they turn out to be "equivalent" to alternating circuits of AND, OR and NOT-gates with small depth (see Proposition 1). But for the same reason circuit automata are quite weak since they are incapable of recognizing parity efficiently. We construct languages $L_{n,k}$ implementing the idea of "nested equalities" (see Definition 2) such that the family $(L_{n,k})_{n\geq 1}$ belongs to $1\mathrm{P}_2$ but show in Theorem 1 that circuit automata with $k-1$ alternations require a number of states which is super-polynomial in n. At least intuitively, alternations and not automata-specific abilities seem to matter when recognizing $L_{n,k}$ and hence we conjecture that $(L_{n,k})_{n\geq 1}$ does not belong to $1\Sigma_{o(k)}$.

Geffert [6] has separated all levels of the "polynomial" hierarchy $1H$ and was even able to do so for two-way automata. However, we could not apply his methods to languages which are easy probabilistically, but hard when using not too many alternations. Instead we apply methods of circuit complexity and in particular the Switching Lemma for bounded depth alternating circuits [7].

On the other hand we show in Theorem 2 that 1AFAs with an unbounded number of alternations simulate $1\mathrm{P}_2$FAs efficiently when input of "too large" length are excluded. This result is made possible by a normal form for probabilistic automata (see Proposition 2), namely as a weighted majority of DFAs.

Hromkovič has made, among others, many fundamental contributions to the understanding of the state complexity of nondeterministic and probabilistic automata. In [9] the surprising result is shown that DFAs require at most quadratically more states than Las Vegas automata and that this gap is largest possible. This result was made possible by investigations into communication complexity [4,5,10].

He showed how to apply communication complexity to the state complexity of various automata models, predominantly NFAs [8,12] and followed this approach systematically. For instance it was possible to show a rather fine-grained state hierarchy for NFAs with bounded ambiguity, where the ambiguity of an NFA is the maximal number of accepting computations for a given input size. Another example of the power of communication arguments is the proof in [11] that the standard construction of converting NFAs with ε-transitions into NFAs without ε-transitions is almost optimal.

We formally introduce $1P_2$FAs, $1\Sigma_k$FAs, $1\Pi_k$FAs, 1AFAs and circuit automata in Sect. 2. Theorems 1 and 2 are shown in Sects. 3 and 4 respectively. Conclusions are given in Sects. 5.

2 Basics

We introduce probabilistic and alternating automata. Finally circuit automata are defined and their relation to unbounded fan-in circuits is made explicit.

Probabilistic Finite Automata. A one-way probabilistic finite automaton (1PFA) $P = (Q, \Sigma, \delta, \lambda, q_0, F)$ with *cut-point* λ is defined by a sequence

$$\delta = (\, \delta_a \, : \, a \in \Sigma)$$

of stochastic $|Q| \times |Q|$-matrices δ_a where $\delta_a[p,q]$ is the probability that P enters state q, when reading letter a in state p. If $w = w_1 \cdots w_k \in \Sigma^*$ is a word of length k over Σ, then the matrix product

$$\delta_w[p,q] := \big(\delta_{w_1} \cdots \delta_{w_k}\big)[p,q]$$

is the probability that P reaches state q when reading w in state p. We define the acceptance probability of w for any state $p \in Q$ as

$$\text{prob}[P \text{ accepts input } w \text{ with } p \text{ as initial state}] := \sum_{q \in F} \delta_w[p,q]$$

and say that P accepts w with probability $\sum_{p \in F} \delta_w[q_0, p]$. The language accepted by P is

$$L(P) := \{w \in \Sigma^* \mid P \text{ accepts } w \text{ with probability at least } \lambda\}.$$

We say that P is a $1P_2$FA iff it accepts $L(P)$ with *gap* $\gamma > 0$, i.e.,

$$\left| \sum_{q \in F} \delta_w[q_0, q] - \lambda \right| \geq \gamma$$

holds for all words $w \in \Sigma^*$.

Alternating Automata. We say that $A = (Q_\forall, Q_\exists, \Sigma, \delta, q_0, F)$ is an alternating automaton (1AFA) iff A has disjoint sets $Q_\exists$, $Q_\forall$ of *existential* resp. *universal* states. For the set $Q := Q_\exists \cup Q_\forall$ of all states the transition function

$$\delta \colon Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$$

has the same structure as for NFAs with ε-moves. In particular we say that $q = (q_0, \ldots, q_k) \in Q^{k+1}$ is a *computation* of A on input $w = w_1 \cdots w_n \in \Sigma^n$ iff there is a sequence $v \in (\Sigma \cup \{\varepsilon\})^*$ which results from w by possibly introducing the empty word several times such that $q_i \in \delta(q_{i-1}, v_i)$ holds for all i $(1 \leq i \leq k)$.

The notion of acceptance is defined recursively:

(a) A accepts the empty word in initial state $q \in Q$ iff q belongs to F,
(b) Let $a \in \Sigma$ and $u \in \Sigma^*$. Then A accepts au with initial state $q \in Q$ iff
  - $q \in Q_\exists$ and A accepts u for *at least one* state in $\delta(q, a)$ as initial state
  - $q \in Q_\forall$ and A accepts u for *every* state in $\delta(q, a)$ as initial state.

We say that A accepts word $w \in \Sigma^*$ iff A accepts w with initial state q_0 and set $L(A) := \{u \in \Sigma^* \mid A \text{ accepts } u\}$. A is a Σ_k-automaton iff q_0 belongs to $Q_\exists$ and A, for all words $w \in \Sigma^*$ and all computations on w, alternates at most $k-1$ times between existential and universal states. A Π_k-automaton is defined analogously.

Circuit Automata. How difficult is it to derive lower bounds on the number of states of an alternating automaton? Geffert [6] gives non-trivial lower bounds for the state complexity of languages in the polynomial hierarchy 1H. However we were unable to apply these methods to languages which are "easy" for 1P$_2$FAs but hard for 1Σ_kFAs and had to work with *finite* languages. Therefore, to rephrase the above question: how difficult is it to derive non-trivial lower bounds on the number of states of alternating automata for finite languages? To at least partially answer this question we compare automata models with alternating unbounded fan-in circuits.

We fix notation first. For a directed graph $H = (U, D)$ and a node $u \in U$ we define *fan-in*(u) as the number of edges in D which are directed into u. A *circuit* C is specified by a pair $C = (G, \text{gate})$, where $G = (V, E)$ is a directed acyclic graph and gate is a function which assigns to each node $v \in V$ an input position iff fan-in$(v) = 0$, respectively a boolean operation $f\colon \{0,1\}^k \to \{0,1\}$ iff fan-in$(v) = k > 0$. The *size* of C is the number of nodes of G and its *fan-in* is the maximal fan-in of a node of V.

Remark 1. ACC is the complexity class of all boolean functions computable by unbounded fan-in circuits with AND, OR, NOT and modulo-gates in bounded depth and polynomial size. There are well known connections between bounded depth circuit classes such as ACC and automata. For instance ACC is the family of languages accepted by a nonuniform DFA (NUDFA) over a monoid that does not contain an unsolvable group as a subsemigroup. (A NUDFA accepts iff the product of the input bits belongs to a given list of monoid elements [1].) Williams [17] showed that not all languages computable in quasi-polynomial nondeterministic time belong to ACC. Super-polynomial lower bounds for languages in P however are missing.

A Σ_k-circuit (Π_k-circuit) C is a circuit composed of AND-, OR- as well as NOT-gates. It has an OR-gate (AND-gate) as its top gate and its fan-in is unbounded. We require that NOT-gates appear only at the bottom of C and that there are at most k AND- resp. OR-gates on any path in C (and hence that the depth of C is $k-1$).

Circuit automata, a restricted version of alternating automata, are tailor-made to simulate Σ_k- resp. Π_k-circuits.

Definition 1 (Circuit Automata). *Let A be a $1\Sigma_k$-automaton over the binary alphabet. We say that A is a $1\Sigma_k$-circuit automaton if for all inputs w, for all computations of A on input w and for all but one input position: all state transitions of A depend only on the position of the current input bit and not on its value.*

$1\Pi_k$-circuit automata are defined analogously.

Observe that circuit automata process only binary words.

Example 1. We later investigate "nested versions" of the language

$$L_{n,1} := \{uv \mid u, v \in \{0,1\}^n,\, u = v\}$$

of equality. $L_{n,1}$ can be recognized by a Π_2-circuit $C_{n,1}$: An AND-gate as the top gate of $C_{n,1}$ "checks" for all positions i ($1 \leq i \leq n$) whether the two disjunctions $u_i \vee \neg v_i$ and $\neg u_i \vee v_i$ hold. Observe that $C_{n,1}$ has $\mathcal{O}(n)$ gates.

We build a Π_2-circuit automaton $A_{n,1}$ from $C_{n,1}$. Starting from its *universal* initial state, $A_{n,1}$ universally selects an input position i ($1 \leq i \leq n$) *and* one of the two disjunctions. It then decides with a single alternation whether to check u_i or v_i. It accepts iff the selected disjunction is verified to be true.

Observe that $A_{n,1}$ is indeed a circuit automaton since in any computation only one input bit (namely u_i or v_i) is checked. Moreover $\mathcal{O}(n^2)$ states suffice, since any specific input location can be found with $\mathcal{O}(n)$ states.

It turns out that Σ_k-circuit automata and Σ_k-circuits are strongly related.

Proposition 1 (Circuit Automata and Circuits).

(a) Let C be a Σ_k-circuit of size s over the input space $\{0,1\}^n$. Then C can be simulated by a Σ_k-circuit automaton with $\mathcal{O}(n \cdot s)$ states.
(b) Let A be a Σ_k-circuit automaton with s states. Then A can be simulated – for all binary inputs of length n – by a Σ_k-circuit of size $\mathcal{O}((n \cdot s)^k)$.

Proof. (a) Simulate C by a Σ_k-circuit automaton A_C which has a state q_v for any node v of C. If the gate of v is an AND-gate (OR-gate), then q_v is a universal state (existential state). The initial state of A_C is the state of the sink of C and hence the initial state is existential. A_C may transition from q_v to q_u only in an ε-move and only if (u, v) is an edge of C.

Thus initially A_C selects a path of C in a series of $k-1$ alternating ε-moves beginning with an existential initial state. If an input gate $(\neg)x_i$ is reached, then A_C travels to the ith input bit. To make this possible attach a path of i new states to the "state of" $(\neg)x_i$. Hence A_C has at most $\mathcal{O}(n \cdot s)$ states, where s is the size of C.

So far A_C has made only input-independent moves. Finally A_C accepts if the ith input bit satisfies the corresponding input gate.

(b) Let A be a Σ_k-circuit automaton with s states. W.l.o.g. we may assume that all computations of A on inputs of length n become deterministic once the single input-dependent transition is performed.

Since A performs a single input-dependent move in any computation, all computations – before performing the input-dependent move – define a single computation tree T_n for all inputs in $\{0,1\}^n$. Whenever the input-dependent move is made, we terminate the computation and hence reach a leaf in T_n.

The leaf is labeled with x_i iff the computation has reached the ith input bit and is accepting. Otherwise the leaf is labeled with $\neg x_i$. Finally, a node v of T_n receives an AND-gate if the state of v is universal and an OR-gate otherwise.

Remember that A is a Σ_k-automaton and we may assume that the root of T_n has an OR-gate. The depth of T_n may be large, however there are at most $k-1$ "alternations" between AND- and OR-gates on any path of T_n. Connect the root directly with all nodes of T_n which are reachable by a path of OR-gates only. Once this is done, compress all AND-gates at the top of the new tree as far as possible and continue this process. At the end we obtain a Σ_k-circuit T with size $\mathcal{O}((n \cdot s)^k)$. □

Remark 2. How expressive are circuit automata? We show that the state complexity of DFAs and circuit automata are incomparable.

First observe that DFAs may require exponentially more states than circuit automata. Namely, as a consequence of Proposition 1, circuit automata and circuits – with alternations and depth coinciding – turn out to be "polynomially equivalent". As we have seen in Example 1 the language $L_{n,1}$ can be recognized by a Π_2-circuit automata and hence DFAs, when simulating Π_2- or Σ_2-circuit automata, require an exponential blowup in the number of states.

On the other hand, as a consequence of Proposition 1(b), Σ_k-circuit automata, for any $k \in \mathbb{N}$, are too weak to simulate DFAs efficiently, since n-bit parity requires alternating circuits of super-polynomial size in n if depth is bounded [7].

Remark 3. One may hope to obtain lower bounds for the number of states of alternating automata for finite languages using methods from circuit complexity. However Proposition 1(a) may be strengthened to allow for threshold gates[1] as bottom gates of a Σ_k-circuit. If C is an alternating circuit of size s and depth k containing threshold-gates with weight bound m as bottom gates, then C can be simulated by a Σ_k-automaton of size poly(n, s^k, m).

However no super-polynomial lower bounds for alternating circuits – with threshold gates as bottom gates – seem to exist for languages in P.

3 Separating Probabilism from Few Alternations

We define a family $L_{n,k}$ of languages which turn out to be easy for 1P$_2$FAs and 1AFAs with a sufficiently large number of alternations. However $L_{n,k}$ is shown to be hard for circuit automata with too few alternations.

[1] A threshold gate with weight bound m has integral weights $-m \leq w_0, w_1, \ldots, w_m \leq m$ and binary inputs $y_1, \ldots, y_m$. It accepts iff $\sum_{i=1}^{m} w_i y_i \geq w_0$ and rejects otherwise.

3.1 Nested Equality

The language $L_{n,k} \subseteq \{0,1\}^{(2n)^k}$ of "nested equality" is defined recursively. A bottom-up view of the definition is as follows. Assume that the input $x \in \{0,1\}^{(2n)^k}$ is partitioned into $2(2n)^{k-1}$ binary words $u_1, v_1, \ldots, u_{(2n)^{k-1}}, v_{(2n)^{k-1}}$ of respective length n with

$$x = u_1 v_1 \cdots u_{(2n)^{k-1}} v_{(2n)^{k-1}}.$$

Compress $x^{(1)} := x$ into the word $x^{(2)}$ of length $(2n)^{k-1}$ by replacing $u_i v_i$ by one if $u_i = v_i$ and by zero otherwise. Apply this compression repeatedly, each time partitioning the current word $x^{(i)}$ into consecutive words of length n, and accept x iff $x^{(k+1)} = 1$. Here is the top-down view.

Definition 2 (The Language of Nested Equality).

(a) The functions $f_{n,k}\colon \{0,1\}^{(2n)^k} \to \{0,1\}$ *are defined recursively.*

– *For* $u, v \in \{0,1\}^n$ *set* $f_{n,1}(uv) := \begin{cases} 1 & \text{if } u = v, \\ 0 & \text{otherwise.} \end{cases}$

– *For* $x \in \{0,1\}^{(2n)^k}$ *set*

$$f_{n,k}(x) := f_{n,1}\Big(f_{n,k-1}(u_1) \cdots f_{n,k-1}(u_n), f_{n,k-1}(v_1) \cdots f_{n,k-1}(v_n)\Big),$$

where $x = u_1 \cdots u_n v_1 \cdots v_n$ *with* $|u_i| = |v_i| = (2n)^{k-1}$ *for* $i = 1, \ldots, n$.

(b) Set $L_{n,k} := \{x \in \{0,1\}^{(2n)^k} \mid f_{n,k}(x) = 1\}$.

$L_{n,k}$ turns out to be easy for bounded-error automata as well as for alternating circuit automata with a sufficient number of alternations.

Lemma 1 (1P$_2$FAs and 1AFAs for $L_{n,k}$). *For any* $k \geq 1$,

(a) There are 1P$_2$FAs for $L_{n,k}$ *with* $\mathcal{O}\big(n^{2k^2+3k}\big)$ *states and gap* $1/4$.

(b) $L_{n,k}$ *can be simulated by a* Π_{2k}*-circuit automaton with* $\mathcal{O}((2n)^{2k})$ *states.*

Proof. We associate the $(2n)$-ary tree $T_{n,k}$ of depth k with $L_{n,k}$. $T_{n,k}$ helps to visualize the hierarchical decomposition of the input bits of x into the $1 + 2n + (2n)^2 + \cdots + (2n)^{k-1} = \frac{(2n)^k - 1}{2n-1}$ equality problems.

(a) We describe probabilistic automata $P_{n,k}$ for $L_{n,k}$ with cut-point $1/2$ and gap $1/4$ recursively. The automaton $P_{n,1}$ selects a prime $p \leq N_1$ at random, where N_1 is to be determined later and checks whether $\sum_{i=1}^{n} 2^{n-i} u_i \equiv \sum_{i=1}^{n} 2^{n-i} v_i \bmod p$ holds. $P_{n,1}$ accepts iff the answer is positive. Hence $P_{n,1}$ accepts $L_{n,k}$ with $\mathcal{O}(n \cdot N_1^2)$ states. It errs only if $uv \in \{0,1\}^{2n}$ does not belong to $L_{n,1}$ and in particular if it picked a prime divisor of the difference $D := \sum_{i=1}^{n} 2^{n-i} u_i - \sum_{i=1}^{n} 2^{n-i} v_i$. Hence the error probability of $P_{n,1}$ is bounded by the quotient of the number of prime divisors of D and the total number of

primes used by $P_{n,1}$. Hence, by the Prime Number Theorem, the error for the equality problem on n bits is bounded by

$$\mathcal{O}\left(\frac{n}{N_1/\log_2 N_1}\right). \tag{1}$$

We view $P_{n,k}$ as a vector $(P_{n,1}^{(1)}, \ldots, P_{n,1}^{(k)})$ of k variants $P_{n,1}^{(1)}, \ldots, P_{n,1}^{(k)}$ of $P_{n,1}$. If j is the current input position, then $P_{n,1}^{(i)}$ deals with its equality problem which is specified by the node of height $i-1$ in $T_{n,k}$ which is an ancestor of the leaf containing position j. In particular, $P_{n,1}^{(i)}$ waits until $P_{n,1}^{(i-1)}$ has completed its current equality problem, computes for one step after receiving the decision of $P_{n,1}^{(i-1)}$ and then waits for the next decision. If $P_{n,1}^{(i)}$ finishes its equality problem it sends the result to $P_{n,1}^{(i+1)}$ and begins with the next equality problem.

There is a total of $\frac{(2n)^k-1}{2n-1}$ equality problems to be solved. To enforce a gap of at least $\frac{1}{4}$, we choose the upper bound N_k for primes sufficiently large. In particular $N_k = \Theta(n^{k+1})$ will do since, by (1), $P_{n,k}$ errs with probability $1-o(1)$ on none of the $\Theta(n^{k-1})$ equality problems.

The states of $P_{n,k}$ are k-tuples with the ith component corresponding to a state of $P_{n,1}^{(i)}$. The number of states of $P_{n,k}$ is hence asymptotically bounded by $(nN_k^2)^k = n^k \cdot n^{2(k+1)k} = n^{2k^2+3k}$.

(b) We construct a Π_{2k}-circuit $C_{n,k}$ for $L_{n,k}$ recursively and then apply Proposition 1(a).

The Π_2-circuit $C_{n,1}$ (with $\mathcal{O}(n)$ gates) is described in Example 1. Inputs of $C_{n,k}$ have the form $x = u_1 v_1 \cdots u_{(2n)^{k-1}} v_{(2n)^{k-1}}$ with $|u_i| = |v_i| = n$. We obtain circuit $C_{n,k}$ after feeding the outputs of $(2n)^{k-1}$ copies of $C_{n,1}$ into a copy of $C_{n,k-1}$, where the ith copy of $C_{n,1}$ checks whether $u_i = v_i$ holds.

With an inductive argument one may verify that circuit $C_{n,k}$ has depth $2k$ and size $\mathcal{O}((2n)^k)$. Hence we obtain a Π_{2k}-circuit automaton with $\mathcal{O}((2n)^{2k})$ states for $L_{n,k}$, if we apply Proposition 1(a). □

3.2 Circuit Automata Require Many Alternations

We show that Σ_{k+1}-circuit automata for $L_{n,k}$ require a super-polynomial number $f_k(n)$ of states.

Theorem 1. *Let $k \in \mathbb{N}$ be even and assume that $n \in \mathbb{N}$ is sufficiently large. Then any Σ_k-circuit automaton for $L_{n,k-1}$ has to have size super-polynomial in n. However polynomial size is sufficient for $\Pi_{2(k-1)}$-circuit automata.*

Proof. $L_{n,k-1}$ is accepted by $\Pi_{2(k-1)}$-circuit automata of polynomial size in n as a consequence of Lemma 1(b). Hence the claim follows from Proposition 1 (b) if we show that any Σ_k-circuit $C_{n,k}$ for $L_{n,k-1}$ has to have size super-polynomial in n, provided $k \geq 2$.

We apply a variant of the Switching Lemma for bounded depth alternating circuits [7] to the two bottom layers of $C_{n,k}$. The original Switching Lemma, applied to circuits with N input bits, is proven by selecting a random restriction $\rho\colon [N] \to \{0, 1, *\}$, where a star has probability p. Positions are fixed with a zero or a one with respective probability $(1 - p)/2$.

Remember that an input x for $C_{n,k}$ has the form $x = u_1 v_1 \cdots u_{(2n)^{k-1}} v_{(2n)^{k-1}}$ for strings u_i, v_i of length n. To obtain lower size bounds for $C_{n,k}$ we place stars with probability p in $u_1 \cdots u_{(2n)^{k-1}}$ and fix the remaining positions in u_i at random. Moreover *all* positions in $v_1 \cdots v_{(2n)^{k-1}}$ are fixed such that equality $u_i = v_i$ is still possible for all i $(1 \leq i \leq (2n)^{k-1})$. A straightforward argument shows that the results of the original Switching Lemma also hold with adjusted parameters in our situation. Finally, for any i, fix all but one star in u_i such that equality remains possible. Here we utilize that fixing inputs bits does not increase size or depth of the circuit.

If s is the size of $C_{n,k}$, then $p = \Theta(\frac{1}{\log_2 s})$ will do. Hence to show that superpolynomial size is required, we may by way of contradiction assume $p = \frac{\alpha}{\log_2 n}$ for an arbitrarily large constant α. As a consequence with high probability there will be exactly one star in each u_i.

We apply the variant of the Switching Lemma, observing that all bits of v_i are fixed, and obtain with high probability that the new circuit $C_{n,k-1}$ has depth $k - 1$ and that its size is polynomial in the size of $C_{n,k}$.

Observe that $C_{n,k-1}$ accepts $L_{n,k-2}$ and we may repeat this procedure. Since k is even, the circuit $C_{n,2}$ is a Σ_2-circuit. Hence there are restrictions such that there is a Σ_2-circuit for $L_{n,1}$ with size polynomial in the size of $C_{n,k}$.

Remember that an implicant of a boolean function $f\colon \{0, 1\}^N \to \{0, 1\}$ is a conjunction of literals which implies f. But $L_{n,1}$, interpreted as a boolean function $f\colon \{0, 1\}^{2n} \to \{0, 1\}$, has only implicants of length $2n$, namely one conjunction for any pair uu. But $C_{n,2}$ is a disjunction of implicants and all implicants are required to appear. Hence $C_{n,2}$ has to have size 2^n and the claim follows. □

4 Simulating Probabilism with Alternations

Our goal is to efficiently simulate a 1P$_2$FA $P = (Q, \Sigma, \delta, \lambda, q_0, F)$ by an 1AFA A. The constructions of Sipser, Lautemann or Canetti [2,15,16] in some shape or form require the capability of performing and evaluating many simulations of the probabilistic machine. For instance, Lautemann assumes an error probability of $2^{-\Omega(n)}$ for input size n, a requirement which in general cannot be fulfilled for 1P$_2$FAs. We therefore proceed differently and in particular do not impose any bound on the number of alternations.

We begin by deriving a normal form for P when restricted to "short" inputs. To do so we randomly fix P-transitions to obtain DFAs $D_1, \ldots, D_r$ as well as a distribution $\mu = (\mu_1, \ldots, \mu_r)$ on the DFAs. Define the 1P$_2$FA

$$P_r^* := \text{majority}_{\lambda,\mu}(D_1, \ldots, D_r)$$

to pick the DFA D_i with probability μ_i and to simulate the input w by D_i. The input w is accepted iff the combined acceptance probabilities, summed over all accepting DFAs D_i, is at least λ.

We show that P_r^* is equivalent with P on all inputs of approximate length at most $r \cdot \gamma^2$ and that P_r^* even has a gap of at least $\gamma/2$.

Proposition 2. *Let $P = (Q, \Sigma, \delta, \lambda, q_0, F)$ be a 1P$_2$FA with gap $\gamma > 0$ and let r be a natural number. Then there are DFAs $D_1, \ldots, D_r$, each of size $s(P)$, as well as a distribution $\mu = (\mu_1, \ldots, \mu_r)$ such that $P_r^* :=$ majority$_{\lambda,\mu}(D_1, \ldots, D_r)$ and P agree on all inputs of length at most K where*

$$K = \Omega\Big(\frac{r \cdot \gamma^2}{\ln(|\Sigma|)}\Big),$$

provided $|\Sigma| \geq 2$. If $|\Sigma| = 1$, then $K = \exp^{\Omega(r\cdot\gamma^2)}$ holds. Moreover P_r^* has a gap of least $\gamma/2$ on all inputs of length at most K.

Proof. For an integer K set $W_K := \bigcup_{i=0}^{K} \Sigma^i$. We randomly select DFAs $D_i = (Q^*, \Sigma, \delta_i, q_0, F^*)$ for $i = 1, \ldots, 2r+1$, where $Q^* := Q \times [K] \cup \{q_0\}$ and $F^* = F \times [K]$ – if $q_0 \in F$, then q_0 has to be inserted into F^*. The transition functions δ_i are obtained by setting $\delta_i((p,t), a) = (q, t+1)$ with probability $\delta_a[p,q]$.

Let $\mathcal{D}$ be the set of DFAs which can be build this way and let p_D be the probability for a DFA $D \in \mathcal{D}$ to be picked. We define the distribution μ by setting $\mu_i = p_{D_i} / \sum_{j=1}^{r} p_{D_j}$ and work with the 1P$_2$FA $P_r^* :=$ majority$_{\lambda,\mu}(D_1, \ldots, D_r)$.

Fix some input $w \in W_K$. Assume that w is accepted with probability p_w by P and with probability p_w^* by P_r^*. By how much does p_w^* deviate from p_w and with which probability does that happen?

Assume first that P accepts w and as a consequence $p_w \geq \lambda + \gamma$ follows. For any path $\mathcal{P}$ in P, which starts in q_0 and is consistent with w, the probability of $\mathcal{P}$ equals the probability of a DFA $D \in \mathcal{D}$ to possess path $\mathcal{P}$. Hence p_w coincides with the probability $p_w^{\mathcal{D}}$ that a DFA $D \in \mathcal{D}$, selected with probability p_D, accepts w and $p_w^{\mathcal{D}} = p_w \geq \lambda + \gamma$ follows.

How likely is it that p_w^* is significantly smaller than p_w, i.e., that $p_w^* \leq \lambda + \gamma/2$ holds? If r DFAs are selected in $\mathcal{D}$, then $r \cdot p_w$ is the expected number of DFAs in $\mathcal{D}$ accepting w. If P_r^* errs on w or if its gap is less than $\gamma/2$, then $r \cdot p_w^*$, as the result of r independent random trials of picking a DFA from $\mathcal{D}$, deviates from its expected value $r \cdot p_w$ by a factor μ with $\mu = \frac{\lambda+\gamma/2}{p_w} \leq \frac{\lambda+\gamma/2}{\lambda+\gamma} = 1 - \frac{\gamma/2}{\lambda+\gamma}$. We apply the Chernoff bound and obtain for $|\Sigma| \geq 2$,

$$\begin{aligned}
&\text{prob[there is } w \in W_K \text{ such that } p_w^* \leq \lambda + \gamma/2 \leq \lambda + \gamma \leq p_w] \\
&\quad\leq \sum_{w \in W_K, p_w \geq \lambda+\gamma} \text{prob}[p_w^* \leq \lambda + \gamma/2] \\
&\quad\leq |W_K| \cdot \exp^{-(\frac{\gamma/2}{\lambda+\gamma})^2 \cdot r \cdot p_w/2} \leq |W_K| \cdot \exp^{-\frac{\gamma^2}{\lambda+\gamma}\cdot r/8} \qquad \text{(Chernoff bound)} \\
&\quad\leq 2\exp^{K\ln(|\Sigma|)} \cdot \exp^{-\frac{\gamma^2}{\lambda+\gamma}\cdot r/8} = 2\exp^{K\ln(|\Sigma|) - \frac{\gamma^2}{\lambda+\gamma}\cdot r/8}.
\end{aligned}$$

To obtain a sufficiently failure probability it suffices to (asymptotically) demand $r = \frac{\lambda+\gamma}{\gamma^2} \cdot K\ln(|\Sigma|) = \mathcal{O}\big(\frac{K\ln(|\Sigma|)}{\gamma^2}\big)$, respectively $K = \Omega\big(\frac{r\cdot\gamma^2}{\ln(|\Sigma|)}\big)$. If $|\Sigma| = 1$

we obtain, again asymptotically, $r = \frac{\lambda+\gamma}{\gamma^2} \cdot \ln(K+1) = \mathcal{O}\big(\frac{\ln(K+1)}{\gamma^2}\big)$ and hence $K = \exp^{\Omega(r\gamma^2)}$ follows.

The case that P rejects w is treated analogously. We find a majority-of-DFAs with the required properties if the probability for P_r^* to err on an input w of length at most K or to have a gap smaller than $\gamma/2$ is less than 1. The claim follows. □

Let $P = (Q, \Sigma, \delta, \lambda, q_0, F)$ be a 1P$_2$FA with gap γ. We just found out that P can be simulated by the weighted majority

$$P_r^* = \text{majority}_{\lambda,\mu}(D_1, \ldots, D_r)$$

with gap $\gamma/2$, provided correctness is required only for inputs of approximate length $K = \Omega\big(\frac{r\cdot\gamma^2}{\ln(|\Sigma|)}\big)$. We now show how to simulate P_r^* by a 1AFA

$$A_r = (Q_\forall, Q_\exists, \Sigma, \delta_A, q_0, F_A)$$

on inputs of length at most K. Assume that q_0 is the starting state of P_r^* and $Q \times \{i\}$ is the set of states of D_i. We begin the definition of A_r by choosing q_0 as its (existential) starting state and set

$$\begin{aligned} Q_\forall &:= \{!\} \times Q \times [r] \times [r^2], \\ Q_\exists &:= \{?\} \times Q \times [r] \times [r^2] \cup \{?\} \times \{\text{continue}\} \times [r] \times [r^2] \cup \{q_0\}. \end{aligned}$$

Only universal states are accepting, since we set

$$F_A = \{!\} \times F \times [r] \times [r^2].$$

Also, insert q_0 into F_A iff P accepts the empty word. We describe the transitions of A_r next.

Let w be an arbitrary input. The alternating automaton A_r guesses which DFAs accept w and then verifies its guess each time. In particular, A_r may guess that D_i accepts w and therefore introduces for each i $(1 \leq i \leq r)$, the ε-transitions

$$q_0 \xrightarrow{\varepsilon} (?, q_0, i, j) \xrightarrow{\varepsilon} (!, q_0, i, j),$$

where $j := \lfloor r^2 \cdot \mu_i \rfloor$ keeps track of the probability of D_i. The ε-transition from the existential to the universal version of (q_0, i, j) is the only applicable transition and therefore A_r immediately challenges its guess. It verifies its guess by simulating D_i with $(!, q_0, i, j)$ as the starting state.

But A_r may have to search for further DFAs $D_{i'}$ accepting w and therefore introduces all ε-transitions of the form

$$(!, q_0, i, j) \xrightarrow{\varepsilon} (?, \text{continue}, i, j) \xrightarrow{\varepsilon} (?, q_0, i', j + j') \xrightarrow{\varepsilon} (!, q_0, i', j + j'),$$

where $j' := \lfloor r^2 \cdot \mu_{i'} \rfloor$ keeps track of the probability of $D_{i'}$. However, to avoid that DFAs are selected more than once, $i < i'$ has to hold. Hence, again the guess is immediately challenged and subsequently verified. This process continues until a universal state $(!, q_0, i^*, j^*)$ is reached with $\frac{1}{r^2} \cdot j^* \geq \lambda$.

In summary, A_r accepts an input w iff there are DFAs $D_{i_1}, \ldots, D_{i_s}$ accepting w such that $i_1 < i_2 < \cdots < i_s$ and

$$\frac{1}{r^2} \cdot \sum_{t=1}^{s} \lfloor r^2 \mu_i \rfloor \geq \lambda$$

hold.

Theorem 2 (1P$_2$FAs and 1AFAs). *Assume that $P = (Q, \Sigma, \delta, \lambda, q_0, F)$ is a 1P$_2$FA with gap γ and let r be a natural number. Then there is a 1AFA A_r such that A_r and P agree on all inputs of length at most K where*

$$K = \Omega\left(\frac{r \cdot \gamma^2}{\ln(|\Sigma|)}\right),$$

provided $|\Sigma| \geq 2$. If $|\Sigma| = 1$, then $K = \exp^{\Omega(r \cdot \gamma^2)}$ holds. The size of A_r is bounded by $\mathcal{O}(r^3 \cdot s(P))$.

Proof. We apply Proposition 2 and transform the 1P$_2$FA P into the weighted majority automaton P_r^* which agrees with P on all inputs of length at most K.

We may assume that $r \geq 2/\gamma^2$, since otherwise $K < 2$ and the claim is trivial. In particular $\gamma/2 \geq \gamma^2/2 \geq 1/r$ follows. We do a case analysis.

Case 1: P_r^* accepts w. But P_r^* has gap $\gamma/2$ and hence

$$\sum_{i:D_i \text{ accepts } w} \mu_i \geq \lambda + \gamma/2. \tag{2}$$

A_r accepts w as well, since

$$\begin{aligned} \frac{1}{r^2} \cdot \sum_{i:D_i \text{ accepts } w} \lfloor r^2 \mu_i \rfloor &\geq \frac{1}{r^2} \cdot \sum_{i:D_i \text{ accepts } w} \left(r^2 \mu_i - 1 \right) \\ &\geq \sum_{i:D_i \text{ accepts } w} \left(\mu_i - \frac{1}{r^2} \right) \overset{(2)}{\geq} \lambda + \gamma/2 - \frac{1}{r} \geq \lambda. \end{aligned}$$

Case 2: P_r^* rejects w. We again utilize that P_r^* has gap $\gamma/2$ and obtain

$$\sum_{i:D_i \text{ accepts } w} \mu_i \leq \lambda - \gamma/2. \tag{3}$$

This time A_r rejects w, since

$$\frac{1}{r^2} \cdot \sum_{i:D_i \text{ accepts } w} \lfloor r^2 \mu_i \rfloor \leq \frac{1}{r^2} \cdot \sum_{i:D_i \text{ accepts } w} r^2 \mu_i \overset{(3)}{\leq} \lambda - \gamma/2.$$

Finally observe that the size of A_r is asymptotically bounded by $r^3 \cdot s(P)$. □

5 Conclusions

We gather evidence that a *fixed* number of alternations is insufficient to simulate a 1P$_2$FA P efficiently and thus the behavior of one-way automata is quite different from unrestricted computations in, say, polynomial time or at least logarithmic space. However such an efficient simulation exists for inputs of length at most polynomial in the number $s(P)$ of states of P, if an unbounded number of alternations is allowed.

Quite a few fundamental problems remain unresolved. First, showing nontrivial lower bounds on the number of states of 1AFAs for finite languages seems to be an important extension of the lower size bounds for circuits of bounded depth. For instance we conjecture that 1AFAs with $o(k)$ alternations may recognize $L_{n,k}$ only if their size is super-polynomial in n. It certainly seems also viable to construct infinite languages which are presumably easy for 1P$_2$FAs and hard for $1\Sigma_k$FAs for some k, since then automata specific arguments may be applied.

Second, does $1P_2 \subseteq 1A$ hold? We conjecture that the answer is positive. Observe that even $1P_2 \subseteq 1H$ may hold.

Third, is it possible to simulate arbitrary 1P$_2$FAs with positive gap efficiently by a majority-of-DFAs *without* any restriction on the size of inputs? Or, in other words, can the restriction on input length in the claim Proposition 2 be dropped? Of course a positive answer implies that $1P_2 \subseteq 1A$ holds.

Acknowledgement. Many thanks to a referee for many valuable comments.

References

1. Barrington, D.A., Thérien, D.: Finite monoids and the fine structure of NC^1. J. ACM **35**(4), 941–952 (1988)
2. Canetti, R.: More on BPP and the polynomial-time hierarchy. Inf. Process. Lett. **57**, 237–241 (1996)
3. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM **28**(1), 114–133 (1981)
4. Ďuriš, P., Hromkovič, J., Jukna, S., Sauerhoff, M., Schnitger, G.: On multi-partition communication complexity. Inf. Comput. **194**(1), 49–75 (2004)
5. Dietzfelbinger, M., Hromkovič, J., Schnitger, G.: A comparison of two lower-bound methods for communication complexity. Theor. Comput. Sci. **168**(1), 39–51 (1996)
6. Geffert, V.: An alternating hierarchy for finite automata. Theor. Comput. Sci. **445**, 1–24 (2012)
7. Håstad, J.: Almost optimal lower bounds for small depth circuits. In: STOC, pp. 6–20 (1986)
8. Hromkovič, J., Seibert, S., Karhumäki, J., Klauck, H., Schnitger, G.: Communication complexity method for measuring nondeterminism in finite automata. Inf. Comput. **172**(2), 202–217 (2002)
9. Hromkovič, J., Schnitger, G.: On the power of Las Vegas for one-way communication complexity, OBDDs, and finite automata. Inf. Comput. **169**(2), 284–296 (2001)
10. Hromkovič, J., Schnitger, G.: Nondeterministic communication with a limited number of advice bits. SIAM J. Comput. **33**(1), 43–68 (2003)

11. Hromkovič, J., Schnitger, G.: Comparing the size of NFAs with and without epsilon-transitions. Theor. Comput. Sci. **380**(1–2), 100–114 (2007)
12. Hromkovič, J., Petersen, H., Schnitger, G.: On the limits of the communication complexity technique for proving lower bounds on the size of minimal NFA's. Theor. Comput. Sci. **410**(30–32), 2972–2981 (2009)
13. Hromkovič, J., Schnitger, G.: Ambiguity and communication. Theory Comput. Syst. **48**(3), 517–534 (2011)
14. Kapoutsis, C.A.: Size complexity of two-way finite automata. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 47–66. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02737-6_4
15. Lautemann, C.: BPP and the polynomial hierarchy. Inf. Process. Lett. **17**, 215–217 (1983)
16. Sipser, M.: A complexity theoretic approach to randomness. In: ACM Symposium on Theoretical Computer Science, pp. 330–335 (1983)
17. Williams, R.: Non-uniform ACC circuit lower bounds. In: IEEE Conference on Computational Complexity, pp. 115–125 (2011)

Algorithmics

Classical and Quantum Computations with Restricted Memory

Farid Ablayev[1(✉)], Marat Ablayev[1], Kamil Khadiev[1,2], and Alexander Vasiliev[1]

[1] Kazan Federal University, Kazan, Russia
fablayev@gmail.com, mablayev@gmail.com, kamilhadi@gmail.com, alexander.ksu@gmail.com
[2] University of Latvia, Riga, Latvia

Abstract. Automata and branching programs are known models of computation with restricted memory. These models of computation were in focus of a large number of researchers during the last decades. Streaming algorithms are a modern model of computation with restricted memory. In this paper, we present recent results on the comparative computational power of quantum and classical models of branching programs and streaming algorithms.

In addition to comparative complexity results, we present a quantum branching program for computing a practically important quantum function (quantum hash function) and prove optimality of this algorithm.

Keywords: Complexity · Quantum computing
Branching programs · OBDDs · Hierarchy · Hashing

1 Introduction

The class of problems solvable with restricted memory and efficient algorithms (classical and quantum) for such problems are of great interest nowadays. There are many different models that process data streams and use few memory resources: automata, branching programs (BPs), or streaming algorithms. Streaming algorithms are a new model in computer science, while branching programs and automata were in the focus of a great number of computer science researchers during the last decades.

Our research group is focused on the study of automata, branching programs with the OBDD type restrictions, and models of streaming algorithms. These research interests largely coincide with the research interests of Juraj Hromkovič and his research group.

In this paper, we present an overview of our results on classical and quantum restricted computational models, especially on the comparison of the computational power of classical and quantum models.

Branching programs are a well-known model of computation that has proven useful in a variety of domains such as hardware verification, model checking, and other applications [64]. It is known that the class of Boolean functions computed by polynomial-sized branching programs coincides with the class of functions computed by non-uniform log-space Turing machines. For theoretical computer science the model of branching programs is interesting, in particular, because of the possibility to define different natural kinds of restrictions on branching programs. It is interesting that the computational model, which is known in theory as branching program was independently defined and investigated in applied computer science as well.

Streaming algorithms are a new model of computation that came to play as a tool for investigations of big data processing. This direction or research uses in part methods developed for the model of branching programs. We show that such connections (between the streaming algorithm model and the branching program model) are very close.

The paper is organized as follows. We start with the definition of various models of branching programs and hierarchy results for these models. Note that hierarchy results need both lower bound and upper bound techniques. To prove some hierarchy one needs appropriate examples of functions that separate different classes. We present a bunch of such functions and use them for hierarchy proofs. To show the practical importance of the model of branching programs one needs examples of practically important functions that can be efficiently computed by BPs. We present a quantum hash function and show a corresponding quantum branching program that is optimal for such a function.

2 Branching Programs

One of the important restricted variants of branching programs are oblivious read-once branching programs or *ordered binary decision diagrams* (OBDD) [64]. The OBDD model can be considered as a nonuniform automaton (see, for example, [10]) and is a good model for data streaming algorithms.

2.1 Definitions

A branching program over the set X of n Boolean variables is a directed acyclic graph with two distinguished nodes s (a source node) and t (a sink node). We denote such a program by $P_{s,t}$ or simply P. Each inner node v of P is associated with a variable $x \in X$. A *deterministic* P has exactly two outgoing edges labeled $x = 0$ and $x = 1$ respectively for such a node v. The program P computes the Boolean function $f(X)$ ($f\colon \{0,1\}^n \to \{0,1\}$) as follows: for each $\sigma \in \{0,1\}^n$ we let $f(\sigma) = 1$ if and only if there exists at least one $s - t$ path (called *accepting path* for σ) such that all edges along this path are consistent with σ.

A branching program is *leveled* if the nodes can be partitioned into levels $V_1, \ldots, V_\ell$ and a level $V_{\ell+1}$ such that the nodes in $V_{\ell+1}$ are the sink nodes, and the nodes in each level V_j with $j \leq \ell$ have outgoing edges only to nodes in the next level V_{j+1}. For a leveled $P_{s,t}$ the source node s is a node from the first level V_1 of nodes and the sink node t is a node from the last level $V_{\ell+1}$.

The *width* $width(P)$ of a leveled branching program P is the maximum number of nodes over all levels of P, $width(P) = \max_{1\leq j\leq \ell} |V_j|$. The *size* of a branching program P is the number of nodes in P.

A leveled branching program is called *oblivious* if all inner nodes of each level are labeled by the same variable. A branching program is called *read once* if each variable is tested on each path only once. An oblivious leveled read once branching program is also called *ordered binary decision diagram* (OBDD). OBDD P reads variables in its individual order $\pi = (j_1, \ldots, j_n)$, $\pi(i) = j_i$; $\pi^{-1}(j)$ is the position of j in permutation π. We call $\pi(P)$ the order of P. Let us denote the natural order by $id = (1, \ldots, n)$. Sometimes we will use the notation id-OBDD P, which means that $\pi(P) = id$. Let $width(f) = \min_P width(P)$ for an OBDD P which computes f, and id-$width(f)$ is the same but for id-OBDD.

A branching program P is called a k-OBDD if it consists of k layers, where the i-th $(1 \leq i \leq k)$ layer P^i of P is an OBDD. Let π_i be an order of P^i, where $1 \leq i \leq k$ and $\pi_1 = \cdots = \pi_k = \pi$. We call $\pi(P) = \pi$ the order of P.

A *nondeterministic* OBDD (NOBDD) is the nondeterministic counterpart of OBDD. A *probabilistic* OBDD (POBDD) can have more than two outgoing edges for a node, and the choice between them is done probabilistically during computation. A POBDD P computes a Boolean function f with bounded error $0.5 - \varepsilon$ if the probability of a correct answer is at least $0.5 + \varepsilon$.

Next, we recall the definition of a quantum version of the branching program model [4,5].

Definition 1. *A* quantum branching program *Q over the Hilbert space $\mathcal{H}^d$ is defined as*

$$Q = \langle \mathbb{T}, |\psi_0\rangle, \mathrm{Accept}\rangle \ , \tag{1}$$

where $\mathbb{T}$ is a sequence of l instructions: $\mathbb{T}_j = \big(x_{i_j}, U_j(0), U_j(1)\big)$ is determined by the variable x_{i_j} tested on the step j, and $U_j(0)$, $U_j(1)$ are unitary transformations in $\mathcal{H}^d$.

Vectors $|\psi\rangle \in \mathcal{H}^d$ are called states (state vectors) of Q, $|\psi_0\rangle \in \mathcal{H}^d$ is the initial state of Q, and Accept $\subseteq \{1, 2, \ldots, d\}$ *is the set of indices of accepting basis states.*

We define a computation of Q on an input $\sigma = \sigma_1 \ldots \sigma_n \in \{0,1\}^n$ as follows:

1. *A computation of Q starts from the initial state $|\psi_0\rangle$;*
2. *The j-th instruction of Q reads the input symbol σ_{i_j} (the value of x_{i_j}) and applies the transition matrix $U_j = U_j(\sigma_{i_j})$ to the current state $|\psi\rangle$ to obtain the state $|\psi'\rangle = U_j(\sigma_{i_j})|\psi\rangle$;*
3. *The final state is*

$$|\psi_\sigma\rangle = \left(\prod_{j=l}^{1} U_j(\sigma_{i_j})\right) |\psi_0\rangle \ . \tag{2}$$

4. *After the l-th (last) step of quantum transformation the configuration $|\psi_\sigma\rangle = (\alpha_1, \ldots, \alpha_d)^T$ of Q is measured, and the input σ is accepted if and only if the result is in* Accept, *which happens with probability*

$$\Pr_{\text{accept}}(\sigma) = \sum_{i \in \text{Accept}} |\alpha_i|^2 \ . \tag{3}$$

We say that a Boolean function f is computed by Q with *bounded error* if there exists an $\varepsilon \in (0, \frac{1}{2}]$ such that Q accepts all inputs from $f^{-1}(1)$ with probability at least $\frac{1}{2} + \varepsilon$ and Q accepts all inputs from $f^{-1}(0)$ with probability at most $\frac{1}{2} - \varepsilon$. We can say that the probability of error is bounded by $\frac{1}{2} - \varepsilon$.

Two main complexity measures of a quantum branching program are its *width* (the dimension of the state space d) and its *length* (the number of instructions l). We can also introduce the complexity measures of the function f being computed by a quantum branching program (QBP):

$$\mathit{Width}(f) = \min \mathit{Width}(Q), \qquad \mathit{Length}(f) = \min \mathit{Length}(Q),$$

where the minimum is taken over all QBPs that compute f.

We can naturally define restricted variants of the QBP model. For instance, we call a quantum branching program a *quantum OBDD* (QOBDD) if each input variable appears in the sequence of instructions at most once. In this case we can define π to be a permutation of $\{1, \ldots, n\}$ that defines the order of testing input variables.

We can also define a quantum k-OBDD (k-QOBDD) to be a quantum branching program with k layers, where each layer is a QOBDD, and each layer has the same order π. We allow measurements for k-QOBDDs during the computation, but after that it should stop and accept an input or continue the computation. A k-*id*-QOBDD is a k-QOBDD with the natural order $id = (1, \ldots, n)$ of input bits.

Let k-$\mathit{QOBDD}_{\mathcal{W}}$ be a set of Boolean functions that can be computed by the bounded-error k-QOBDDs of width w, for $w \in \mathcal{W}$. k-*id*-$\mathit{QOBDD}_{\mathcal{W}}$ is defined in a similar way for bounded-error k-*id*-QOBDDs.

Let BQP$_{\varepsilon}$-kQOBDD be a set of Boolean functions that can be computed by polynomial-sized k-QOBDDs with the probability of error bounded by ε. We can consider similar classes for the deterministic model (P-kOBDD) and bounded-error probabilistic model (BP$_{\varepsilon}$-kOBDD):

– P-kOBDD and BP$_{\delta}$-kOBDD are defined for polynomial-sized k-OBDDs, the first one is for the deterministic case and the second one is for the bounded-error probabilistic k-OBDD with error probability at most δ;
– SUPERPOLY-kOBDD and BSUPERPOLY$_{\delta}$-kOBDD are similar classes for superpolynomial-sized models;
– SUBEXP$_{\alpha}$-kOBDD and BSUBEXP$_{\alpha,\delta}$-kOBDD are similar classes for a size of at most $2^{O(n^{\alpha})}$, for $0 < \alpha < 1$.

2.2 Hierarchies for Classical Read *k*-Times Branching Programs (*k*-OBDDs)

One of the first explicit hard functions for the *syntactic* BPs was introduced by Borodin, Razborov, and Smolensky [20]. For each $k \leq c \log n$ they presented an explicit function, which needs superpolynomial size syntactic k-BPs for some appropriate constant $c > 0$.

Let P-kBP be the set of Boolean functions that can be computed by syntactic k-BP of polynomial size. The classes NP-kBP and BPP-kBP are nondeterministic and bounded-error probabilistic counterparts of the P-kBP class.

Thathachar [61] presented a family of explicit Boolean functions depending on integer parameter k, which cannot be represented by a kn-length polynomial-sized *syntactic* nondeterministic k-BP. In addition, the technique from [61] allows to prove the following proper inclusion: NP-$(k-1)$BP $\subsetneq$ NP-kBP, for $k = o(\log\log n)$.

Probabilistic k-BPs were investigated by Hromkovič and Sauerhoff in 2003 [35]. They proved a lower bound for the explicit Boolean function *m-Masked-$PJ_{k,n}$* and showed that bounded-error probabilistic k-BPs should have a size of at least $2^{\Omega(N^\alpha/k^3)}$, for $\alpha = 1/(1+2\log 3)$. Using these results Hromkovič and Sauerhoff obtained a hierarchy for polynomial-sized bounded-error probabilistic k-BPs: BPP-$(k-1)$BP $\subsetneq$ BPP-kBP, for $k \le \log n/3$.

For the case of k-OBDD models, Bolling, Sauerhoff, Sieling, and Wegener suggested an explicit Boolean function which cannot be represented by non-linear-length $o(n^{3/2}/\log^{3/2} n)$-polynomial-sized k-OBDDs. In addition their technique allows to prove the following proper inclusions: P-$(k-1)$OBDD $\subsetneq$ P-kOBDD, for $k = o(n^{1/2}/\log^{3/2} n)$.

Ablayev and Karpinski [8,12] introduced an explicit Boolean function which is hard for polynomial-sized nondeterministic k-OBDDs, for $k = o(n/\log n)$, but can be computed by bounded-error probabilistic k-OBDDs.

Brosenne, Homeister, and Waack [25] showed that for any constant k it holds that NP-OBDD = NP-kOBDD.

The k-OBDD model of small width is also interesting, because, for example, the class of functions computed by constant-width *poly*(n)-OBDD equals the well-known complexity class NC^1 for logarithmic-depth circuits [17,62].

Khadiev [42] considered polynomial-width deterministic and nondeterministic k-OBDDs, and sublinear-width deterministic, nondeterministic and probabilistic k-OBDDs. For polynomial, superpolynomial and subexponential width (size) he proved a lower bound for an explicit function and obtained the following hierarchies.

Theorem 1 (Khadiev [42]).

- *For $k = o(n/\log n)$ and $k > \log^2 n$, the following statements are true:*

$$\textit{P-}\left(\frac{k}{\log^2 n}\right)\textit{OBDD} \subsetneq \textit{P-kOBDD},\quad \textit{NP-}\left(\frac{k}{\log^2 n}\right)\textit{OBDD} \subsetneq \textit{NP-kOBDD}.$$

- *For $k = o(n/\log^\alpha n)$ and $\log^{\alpha+1} n = o(k)$, the following statements are true:*

$$\textit{SUPERPOLY-}\left\lfloor k/\log^{\alpha+1} n\right\rfloor \textit{NOBDD} \subsetneq \textit{SUPERPOLY-kNOBDD},$$
$$\textit{SUPERPOLY-}\left\lfloor k/(\log^{\alpha+1} n)\right\rfloor \textit{OBDD} \subsetneq \textit{SUPERPOLY-kOBDD}.$$

– *For $k = o(n^{1-\alpha})$ and $n^{\alpha} \log n = o(k)$, the following statements are true:*

$$SUBEXP_{\alpha}\text{-}\lfloor k/(n^{\alpha}\log n)\rfloor\, NOBDD \subsetneq SUBEXP_{\alpha}\text{-}kNOBDD,$$
$$SUBEXP_{\alpha}\text{-}\lfloor k/(n^{\alpha}\log n)\rfloor\, OBDD \subsetneq SUBEXP_{\alpha}\text{-}kOBDD.$$

Proof (Sketch). The *shuffled equality function* was used, which was defined in [6,7]. The lower and upper bounds for separation of the classes are proven via this function.

The idea of the lower bound is presenting k-NOBDDs as 1-NOBDDs of bigger width using a product technique. Then we can use the NOBDD complexity of the shuffled equality function from [6,7].

Let us define the shuffled equality function. Let d be a multiple of 4 such that $4 \leq d \leq n/4$. The Boolean function EQS_d depends only on the first d bits. For any given input $\nu \in \{0,1\}^n$, we can define two binary strings $\alpha(\nu)$ and $\beta(\nu)$ in the following way. We call odd bits of the input *marker bits* and even bits *value bits*. For any i satisfying $1 \leq i \leq d/2$, the value bit ν_{2i} belongs to $\alpha(\nu)$ if the corresponding marker bit $\nu_{2i-1} = 0$ and ν_{2i} belongs to $\beta(\nu)$ otherwise. $EQS_d(\nu) = 1$ if $\alpha(\nu) = \beta(\nu)$, and $EQS_d(\nu) = 0$ otherwise. □

Later the results for polynomial width (size) deterministic and probabilistic k-OBDDs were improved by Khadiev and Khadieva [44]. They developed a reordering method for constructing hard functions that allow to show the following hierarchies.

Theorem 2 (Khadiev and Khadieva [44]).

– *For polynomial size (width) we have:*

$$P\text{-}kOBDD \subsetneq P\text{-}2kOBDD, \text{ for } k = o(n/\log^3 n).$$
$$BP_{1/3}\text{-}kOBDD \subsetneq BP_{1/3}\text{-}2kOBDD, \text{ for } k = o(n^{1/3}/\log n).$$

– *For superpolynomial size (width) we have:*

$$SUPERPOLY\text{-}kOBDD \subsetneq SUPERPOLY\text{-}2kOBDD,$$
$$\text{for } k = o(n^{1-\delta}), \delta > 0.$$
$$BSUPERPOLY_{1/3}\text{-}kOBDD \subsetneq BSUPERPOLY_{1/3}\text{-}2kOBDD,$$
$$\text{for } k = o(n^{1/3-\delta}), \delta > 0.$$

– *For subexponential size (width) we have:*

$$SUBEXP_{\alpha} - kOBDD \subsetneq SUBEXP_{\alpha}\text{-}2kOBDD,$$
$$\text{for } k = o(n^{1-\delta}), 1 > \delta > \alpha + \varepsilon, \varepsilon > 0.$$
$$BSUBEXP_{\alpha,1/3} - kOBDD \subsetneq BSUBEXP_{\alpha,1/3} - 2kOBDD,$$
$$\text{for } k = o(n^{1/3-\delta/3}), 1/3 > \delta > \alpha + \varepsilon, \varepsilon > 0.$$

Proof (Sketch). We can use the reordered version of the *pointer jumping function* from [19,57]. We use the lower bound for the pointer jumping function that is based on communication complexity results from [57], and modify it for the reordered version of the function. Additionally, we show the upper bound for the function.

Let us define the function. Let V_A, V_B be two disjoint sets (of vertices) with $|V_A| = |V_B| = m$ and $V = V_A \cup V_B$. Let $F_A = \{f_A : V_A \to V_B\}$, $F_B = \{f_B : V_B \to V_A\}$, and $f = (f_A, f_B): V \to V$ is defined by $f(v) = f_A(v)$ if $v \in V_A$, and $f = f_B(v)$ if $v \in V_B$. For each $k \geq 0$ we define $f^{(k)}(v)$ as $f^{(0)}(v) = v$, $f^{(k+1)}(v) = f(f^{(k)}(v))$. Let $v_0 \in V_A$. We will be interested in computing $g_{k,m}: F_A \times F_B \to V$ defined by $g_{k,m}(f_A, f_B) = f^{(k)}(v_0)$. The function $PJ_{t,n}: \{0,1\}^n \to \{0,1\}$ is the Boolean version of $g_{k,m}$, where we encode f_A by a binary string using $m \log m$ bits, and do this for f_B as well. The result of the function is the parity of the binary representation of the resulting vertex.

Let us apply the reordering method from [44] to the $PJ_{k,n}$ function. $RPJ_{k,n}$ is the total version of the reordered $PJ_{k,n}$. Formally, the Boolean function $RPJ_{k,n}: \{0,1\}^n \to \{0,1\}$ is defined as follows. Let us separate the whole input $X = (x_1, \ldots, x_n)$ into b blocks, such that $b\lceil \log_2 b + 1\rceil = n$; therefore $b = O(n/\log n)$. Let $Adr(X,i)$ be the address whose binary representation is given by the first $\lceil \log_2 b\rceil$ bits of the i-th block and let $Val(X,i)$ be the value of the bit at position $\lceil \log_2 b + 1\rceil$ of block i, for $i \in \{0,\ldots,b-1\}$. Let a be a number such that $b = 2a\lceil \log_2 a\rceil$ and $V_A = \{0,\ldots,a-1\}$, $V_B = \{a,\ldots,2a-1\}$.

Let the function $BV: \{0,1\}^n \times \{0,\ldots,2a-1\} \to \{0,\ldots,a-1\}$ be defined as follows:

$$BV(X,v) = \sum_{i:(v-1)\lceil \log_2 b\rceil < Adr(X,i) \leq v\lceil \log_2 b\rceil} 2^{Adr(X,i)-(v-1)\lceil \log_2 b\rceil} \cdot Val(X,i) \pmod{a}.$$

Then $f_A(v) = BV(X,v) + a$, $f_B(v) = BV(X,v)$.

Let $r = g_{k,a}(f_A, f_B)$; then

$$RPJ_{k,n}(X) = \bigoplus_{i:(r-1)\lceil \log_2 b\rceil < Adr(X,i) \leq r\lceil \log_2 b\rceil} Val(X,i).$$

□

Khadiev [42] also investigated the sublinear-width case. He showed the following results for deterministic, nondeterministic and probabilistic models of constant, polylogarithmic and sublinear width.

Theorem 3 (Khadiev [42]).

- *For $k = o(n/\log n)$, the following statements are true:*

$$\lfloor k/(\log_2 \log_2 n)\rfloor - OBDD_{\mathtt{const}} \subsetneq k - OBDD_{\mathtt{const}}, for \log n = o(k);$$
$$\lfloor k/(\log_2 \log_2 n)\rfloor - NOBDD_{\mathtt{const}} \subsetneq k - NOBDD_{\mathtt{const}}, for \log n = o(k);$$
$$\lfloor k/(\log_2 n \cdot \log_2 \log_2 n)\rfloor - POBDD_{\mathtt{const}} \subsetneq k - POBDD_{\mathtt{const}},$$
$$for \log_2 n \cdot \log_2 \log_2 n = o(k).$$

– *For* $\varepsilon, \varepsilon_1 > 0, k = o(n^{1-\varepsilon}), n^{\varepsilon_1} < k$, *the following statements are true:*

$$\lfloor k/n^{\varepsilon_1} \rfloor - OBDD_{\texttt{polylog}} \subsetneq k - OBDD_{\texttt{polylog}};$$
$$\lfloor k/n^{\varepsilon_1} \rfloor - NOBDD_{\texttt{polylog}} \subsetneq k - NOBDD_{\texttt{polylog}};$$
$$\lfloor k/n^{\varepsilon_1} \rfloor - POBDD_{\texttt{polylog}} \subsetneq k - POBDD_{\texttt{polylog}}.$$

– *the following statements are true:*

$$\lfloor k/(n^{\alpha}(\log_2 n)^2) \rfloor - OBDD_{\texttt{sublinear}_\alpha} \subsetneq k - OBDD_{\texttt{sublinear}_\alpha},$$
$$\textit{for } 0 < \alpha < 0.5 - \varepsilon, \varepsilon > 0, k > n^{\alpha}(\log_2 n)^2, k = o(n^{1-\alpha}/\log_2 n);$$
$$\lfloor k/(n^{\alpha}(\log_2 n)^2) \rfloor - OBDD_{\texttt{sublinear}_\alpha} \subsetneq k - OBDD_{\texttt{sublinear}_\alpha},$$
$$\textit{for } 0 < \alpha < \frac{1}{3} - \varepsilon, \varepsilon > 0, k > n^{2\alpha}(\log_2 n)^2, k = o(n^{1-\alpha}/\log_2 n);$$
$$\lfloor k/(n^{2\alpha}(\log_2 n)^2) \rfloor - POBDD_{\texttt{sublinear}_\alpha} \subsetneq k - POBDD_{\texttt{sublinear}_\alpha},$$
$$\textit{for } 0 < \alpha < 0.25 - \varepsilon, \varepsilon > 0, k > n^{2\alpha}(\log_2 n)^2, k = o(n^{1-2\alpha}/(\log_2 n)^2).$$

Proof (Sketch). The *shuffled address function* from [41] is used. The results also can be shown using the modification of the pointer jumping function from [19,57] that is called *matrix XOR pointer jumping function* [3]. We use lower bounds for memoryless communication protocols from [11,42] and prove upper bound for the shuffled address function.

Let us define the matrix XOR pointer jumping that is easier and also useful for the proof. The matrix XOR pointer jumping function ($MXPJ_{2k,d}$) is a modification of PJ. First, we give the definition of the $MatrixPJ_{2k,d}$ function. Let us consider functions $f_{A,1}, \dots, f_{A,k} \in F_A$ and $f_{B,1}, \dots, f_{B,k} \in F_B$. On iteration $j+1$ function $f^{(j+1)}(v) = f_{j+1}(f^{(j)}(v))$, where $f_i(v) = f_{A,\lceil \frac{i}{2} \rceil}(v)$ if i is odd, and $f_i(v) = f_{B,\lceil \frac{i}{2} \rceil}(v)$ otherwise. The value of $MatrixPJ_{2k,d}(f_{A,1}, \dots, f_{A,k}, f_{B,1}, \dots, f_{B,k})$ is given by $f^{(k)}(v_0)$. $MXPJ_{2k,d}$ is a modification of $MatrixPJ_{2k,d}$. Here, we take

$$f^{(j+1)}(v) = f_{j+1}(f^{(j)}(v)) \oplus f^{(j-1)}(v), \text{ for } j \geq 0.$$

Finally, we consider a Boolean version of these functions. The Boolean function $PJ_{t,n}\colon \{0,1\}^n \to \{0,1\}$ is $g_{k,d}$, where we encode f_A in a binary string using $d \log d$ bits, and do this for f_B as well. The result of the function is the parity of the bits from the binary representation of the resulting vertex's number. For encoding input functions of $MXPJ_{2k,d}$ we use the following order: $f_{A,1}, \dots, f_{A,k}, f_{B,1}, \dots, f_{B,k}$. The function $f_{A,i}$ is encoded by $a_{i,1}, \dots, a_{i,d}$, and $f_{B,i}$ is encoded by $b_{i,1}, \dots, b_{i,d}$, for $i \in \{1, \dots, k\}$. We assume that $v_0 = 0$. □

The results for deterministic k-OBDDs extend the Bolling-Sauerhoff-Sieling-Wegener hierarchy for larger k, but they are not tight. If we compare the results of [44] and [42], then we can see that the latter results are true for larger k. At the same time, the hierarchy of [44] has a smaller "jump" between the classes. Additionally, these two papers show hierarchies for sublinear, superpolynomial and subexponential width. These cases were not considered before.

In the nondeterministic case, the best possible hierarchy for polynomial width is shown in [42]. A smaller jump is not possible, because increasing k by a constant factor does not give more power for the model [25]. So Khadiev's result shows a hierarchy which extends Thathachar's and Okolnishnikova's hierarchies but for the model with a more regular structure (k-NOBDDs). As for the deterministic case, the improvement was shown for sublinear, superpolynomial and subexponential width.

For the probabilistic case, the authors of [44] prove a hierarchy which extends the Hromkovič-Sauerhoff hierarchy but for the model with a more regular structure (probabilistic k-OBDDs). The papers [42,44] show an extension of the Hromkovič-Sauerhoff hierarchy in case of sublinear, superpolynomial and subexponential width.

2.3 Hierarchies for Quantum Read *k*-Times Branching Programs (*k*-OBDDs)

In the last decades, a quantum model of OBDDs was considered [4,54,58,59]. Researchers are also interested in the read-k-times quantum model of OBDDs (k-QOBDDs, see, for example, [34]). k-QOBDDs can be explored from an automata point of view. In that situation, we can find good algorithms for two-way quantum classical automata and related models [15,65]. It is known that if we consider unbounded-error k-OBDDs, then we don't have known hierarchies of complexity classes. Let us consider two classes of Boolean functions: functions computed by polynomial-sized unbounded-error k-QOBDDs and 1-QOBDDs. Homeister and Waack [34] have shown equality of these two classes.

Ablayev, Ambainis, Khadiev and Khadieva [3] modified the technique from [44] and proved hierarchies for quantum k-OBDDs with bounded error.

Theorem 4 (Ablayev et al. [3]). *$BQP_{1/8}$-$kOBDD \subsetneq BQP_{1/8}$-$(2k)QOBDD$, for $k > 0, k = const$.*

Proof (Sketch). We use the XOR-reordered version of the pointer jumping function from Theorem 2 and the modified version of the lower bound for the pointer jumping function based on communication complexity results from [39,48]. Additionally, we show an upper bound for the function. $\square$

In [3] we also showed hierarchies for sublinear width and the natural order of the input variables:

Theorem 5 (Ablayev et al. [3]).

– *For $k = o(\sqrt{n})$, $\sqrt{k} > r$, $1 = o(r)$, we have*

$$\lfloor\sqrt{k}/r\rfloor\text{-}id\text{-QOBDD}_{\mathtt{const}} \subsetneq k\text{-}id\text{-QOBDD}_{\mathtt{const}}.$$

– *For $k = o(n^{0.5-\delta})$, $\sqrt{k} > n^r$, $r > 0, \delta > 0$, we have*

$$\lfloor\sqrt{k}/n^r\rfloor\text{-}id\text{-QOBDD}_{\mathtt{polylog}} \subsetneq k\text{-}id\text{-QOBDD}_{\mathtt{polylog}}.$$

– *For $k = o(n^{0.5-\delta})$, $\sqrt{k} > n^r$, $r > 0, \delta > 0$, we have*

$$\lfloor \sqrt{k}/n^{\alpha+r} \rfloor\text{-}id\text{-QOBDD}_{\texttt{sublinear}_\alpha} \subsetneq k\text{-}id\text{-QOBDD}_{\texttt{sublinear}_\alpha}.$$

Proof (Sketch). We use the matrix XOR pointer jumping function and lower bounds for memoryless communication protocols that are modifications of results from [11,42]. Also we present an upper bound for the matrix XOR pointer jumping function. □

The hierarchies in [3] are the first such hierarchies for bounded-error quantum k-OBDDs, and these results show that unbounded-error and bounded-error models have different properties in this point of view.

3 Las-Vegas Models

The Las-Vegas model is an alternative probabilistic computational model. In this case, an algorithm gives the right answer 0 or 1 with probability $1-\varepsilon$ and with probability ε it fails. The relations between Las-Vegas and deterministic versions of automata and OBDDs were explored by different groups and using different techniques. The first group were Ďuriš, Hromkovič, Rolim, and Schnitger [29,37]; another result was produced by Klauck with the focus on quantum models [49]; and the third research group were Hirvensalo and Seibert [32].

The relations between Las-Vegas and deterministic automata complexities are the following.

Fact 1 (Ďuriš et al. [29,37], Klauck [49], Hirvensalo and Seibert [32]). *For any regular language L and error bound $\varepsilon < 1$, we have the following lower bounds for probabilistic finite automata and quantum finite automata:*

$$(\mathsf{DFA(L)})^{1-\varepsilon} \leq \mathsf{LV}_\varepsilon(\mathsf{L}) \text{ and } (\mathsf{DFA(L)})^{1-\varepsilon} \leq \mathsf{ULV}_\varepsilon(\mathsf{L}).$$

Here $\mathsf{DFA(L)}$ is the size of the best deterministic finite automaton for the language L, $\mathsf{LV}_\varepsilon(\mathsf{L})$ is the size of the best Las-Vegas automaton and $\mathsf{ULV}_\varepsilon(\mathsf{L})$ is the size of the best measure-once quantum automaton. So, if $\varepsilon \leq 1/2$, then the best gap can be quadratic.

Up to a constant, this quadratic gap is achieved [29,37] by using the language $\mathtt{END_k} = \{u1v \mid u,v \in \{0,1\}^* \text{ and } |v| = k-1\}$:

$$\mathsf{DFA}(\mathtt{END_k}) = 2^k \text{ and } \mathsf{LV}_{0.5}(\mathtt{END_k}) \leq 4 \cdot 2^{k/2}.$$

Later Ibrahimov, Khadiev, Prūsis and Yakaryılmaz [38] suggested a modification $\mathtt{MODXOR_k}$ of the language. The language $\mathtt{MODXOR_k}$ for $k > 0$ is formed by the strings

$$\{0,1\}^{<2k} x_1 \{0,1\}^{2k-1} x_2 \{0,1\}^{2k-1} \cdots x_m \{0,1\}^{2k-1},$$

where $m > 0$, each $x_i \in \{0,1\}$ for $1 \leq i \leq m$, and $\bigoplus_{i=0}^m x_i = 1$, taking $x_0 = 0$.

Using this language, better gaps for the probabilistic and quantum cases were shown.

Theorem 6 (Ibrahimov et al. [38]). *For each $k > 0$, we have:*

$$\mathsf{DFA}(\mathtt{MODXOR_k}) \geq 2^{2k}, \quad \mathsf{LV}_{0.5}(\mathtt{MODXOR_k}) \leq 2 \cdot 2^k, \quad \mathsf{ULV}_{0.5}(\mathtt{MODXOR_k}) \leq 2 \cdot 2^k.$$

The same relation exists for the OBDD model. The relation between the quantum Las-Vegas model and the deterministic one was shown by Sauerhoff and Sieling [60]. For the probabilistic model, the relation was investigated by Ibrahimov, Khadiev, Prūsis and Yakaryılmaz [38].

Theorem 7 (Sauerhoff and Sieling [60], Ibrahimov et al. [38]). *For any Boolean function f over $X = (X_1, \ldots, X_n)$ and error bound $\varepsilon < 1$:*

$$(\mathsf{OBDD(f)})^{1-\varepsilon} \leq \mathsf{ULV{-}OBDD}_\varepsilon(\mathsf{f}) \text{ and } (\mathsf{OBDD(f)})^{1-\varepsilon} \leq \mathsf{LV{-}OBDD}_\varepsilon(\mathsf{f}).$$

For OBDDs with $\varepsilon = \frac{1}{2}$, the lower bound can be at most quadratic. Up to a logarithmic factor, this quadratic gap was achieved by using the $\mathtt{SA_d}$ function in [60] for id-OBDDs. The authors of [38] presented result for OBDDs (for any order) using the $\mathtt{SSA_n}$ function.

Let us define this function. We start with the well-known *storage access* Boolean function $\mathtt{SA_d}(x, y) = x_y$, where the input is split into the *storage* $x = (x_1, \ldots, x_{2^d})$ and the *address* $y = (y_1, \ldots, y_d)$. By using the idea of "shuffling" from [1,6,7,12] we define the *shuffled storage access* Boolean function $\mathtt{SSA_n}\colon \{0,1\}^n \to \{0,1\}$, for even n. Let $x \in \{0,1\}^n$ be an input. We form two disjoint sorted lists of even indexes of bits $I_0 = (2i_1, \ldots, 2i_m)$ and $I_1 = (2j_1, \ldots, 2j_k)$ with the following properties: (i) if $x_{2i-1} = 0$ then $2i \in I_0(x)$, (ii) if $x_{2i-1} = 1$ then $2i \in I_1(x)$, (iii) $i_r < i_{r+1}$, for $r \in \{1, \ldots, m-1\}$ and $j_r < j_{r+1}$, for $r \in \{1, \ldots, k-1\}$.

Let d be a number such that $2^d + d = n/2$. We can construct two binary strings by the following procedure: Initially we set $\alpha(x) := 0^{2^d}$ (the all-zero sequence of length 2^d); then for r from 1 to m we do $\alpha(x) := \mathtt{ShiftX}(\alpha(x), x_{2i_r})$, where $\mathtt{ShiftX}((a_1, \ldots, a_m), b) = (a_2, \ldots, a_m, a_1 \oplus b)$. We initially set $\beta(x) := 0^d$ (the all-zero sequence of length d); then for r from 1 to k: $\beta(x) := \mathtt{ShiftX}(\beta(x), x_{2j_r})$. Then, $\mathtt{SSA_n}(x) = \mathtt{SA_d}(\alpha(x), \beta(x))$.

We can provide a lower bound for OBDDs and upper bounds for Las-Vegas-OBDDs (LV-OBDDs), computing the $\mathtt{SSA_n}$ function.

Theorem 8 (Ibrahimov et al. [38]). *For $2^d + d = n/2$, we have:*

$$\mathsf{OBDD}(\mathtt{SSA_n}) \geq 2^{2^d}, \ \mathsf{LV{-}OBDD}_{0.5}(\mathtt{SSA_n}) \leq 2^{2^d/2+d+3},$$
$$\mathsf{ULV{-}OBDD}_{0.5}(\mathtt{SSA_n}) \leq 2^{2^d/2+d+3}.$$

4 Quantum Hashing

Hashing has a lot of fruitful applications in cryptography. Note that in cryptography functions satisfying (i) the *one-way property* and (ii) the *collision resistance property* (in different specific meanings) are called hash functions, and we use this notion when we are considering cryptographic aspects of quantum functions with the above properties. So we suggest to call a quantum function (which maps classical words into quantum states) that satisfies properties (i) and (ii) (in the quantum setting) a *cryptographic quantum hash function* or just quantum hash function. One of the first considerations of a quantum function as a cryptographic primitive, having the one-way property and the collision resistance property, is due to [31], where the quantum fingerprinting function from [26] was used. Another approach towards constructing quantum hash functions from quantum walks was considered in [51,52,66], and it resulted in privacy amplification in quantum key distribution and other useful applications.

In this section, we recall the notion of a quantum (δ, ε)-resistant hash function based on [9].

We let $\mathbb{Z}_q$ be the finite additive group of $\mathbb{Z}/q\mathbb{Z}$, i.e., the integers modulo q. Let Σ^k be the set of words of length k over a finite alphabet Σ. Let $\mathbb{X}$ be a finite set (in [63] we have considered $\mathbb{X}$ as an arbitrary finite Abelian group, and in Subsect. 4.3, we will restrict ourselves to $\mathbb{X} = \mathbb{Z}_q$). For $K = |\mathbb{X}|$ and integer an $s \geq 1$ we define a $(K; s)$ classical-quantum function (or just quantum function) to be the following mapping:

$$\psi \colon \mathbb{X} \to (\mathcal{H}^2)^{\otimes s} \quad \text{or} \quad \psi \colon w \mapsto |\psi(w)\rangle.$$

In order to outline a computational aspect and present a procedure for the quantum function ψ, we define a unitary transformation (determined by an element $w \in \mathbb{X}$) of the initial state $|\psi_0\rangle \in (\mathcal{H}^2)^{\otimes s}$ to a quantum state $|\psi(w)\rangle \in (\mathcal{H}^2)^{\otimes s}$:

$$|\psi(w)\rangle = U(w)|\psi_0\rangle,$$

where $U(w)$ is a unitary matrix.

Extracting information about w from $|\psi(w)\rangle$ is a result of the measurement of the quantum state $|\psi(w)\rangle$. In this paper, we consider quantum transformations and measurements of quantum states with respect to computational basis.

4.1 One-Way δ-Resistance

We present the following definition of a quantum δ-resistant one-way function. Let M be a function $M \colon (\mathcal{H}^2)^{\otimes s} \to \mathbb{X}$. Informally speaking, an "information extracting" mechanism M makes some measurement of the state $|\psi\rangle \in (\mathcal{H}^2)^{\otimes s}$ and decodes the result to $\mathbb{X}$.

Definition 2 (Ablayev and Ablayev [9]). *Let X be a random variable with* $\Pr[X = w]$ *for all $w \in \mathbb{X}$. Let $\psi \colon \mathbb{X} \to (\mathcal{H}^2)^{\otimes s}$ be a quantum function. Let Y be a random variable over $\mathbb{X}$ obtained by some M making a measurement to the encoding ψ of X and decoding the result to $\mathbb{X}$. Let $\delta > 0$. We call a quantum function ψ a one-way δ-resistant function if*

1. *it is easy to compute, i.e., a quantum state $|\psi(w)\rangle$ for a particular $w \in \mathbb{X}$ can be computed using a polynomial-time algorithm;*

2. for any mechanism M, the probability $\Pr[Y = X]$ *that M successfully decodes Y is bounded by δ, i.e.,*

$$\Pr[Y = X] \leq \delta.$$

For cryptographic purposes it is natural to expect (and we do this in the rest of the paper) that the random variable X is uniformly distributed.

A quantum state of $s \geq 1$ qubits can theoretically contain an infinite amount of information. On the other hand, Holevo's Theorem [33] states that $O(s)$ bits of information can be extracted from this state. Here we use the result of [56] motivated by Holevo's Theorem.

Property 1 (Nayak [56]*).* Let X be a random variable uniformly distributed over $\{0,1\}^k$. Let $\psi\colon \{0,1\}^k \to (\mathcal{H}^2)^{\otimes s}$ be a quantum function. Let Y be a random variable over $\{0,1\}^k$ obtained by some M making a measurement of the encoding ψ of X and decoding the result to $\{0,1\}^k$. Then the probability of a correct decoding is given by

$$\Pr[Y = X] \leq \frac{2^s}{2^k}.$$

So, extracting information about the input σ from the state $|\psi(\sigma)\rangle$ is rather limited. The efficiency of computing $|\psi(\sigma)\rangle$ depends on the construction of the quantum hash function ψ. In Subsect. 4.3, we consider the quantum hash function that is based on small biased sets and prove the efficiency of this construction.

4.2 Collision ε-Resistance

The following definition was presented in [2].

Definition 3. *Let $\varepsilon > 0$. We call a quantum function $\psi\colon \mathbb{X} \to (\mathcal{H}^2)^{\otimes s}$ a collision ε-resistant function if for any pair w, w' of different inputs,*

$$|\langle\psi(w)|\psi(w')\rangle| \leq \varepsilon.$$

Informally speaking, we need two states $|\psi(w)\rangle$ and $|\psi(w')\rangle$ to be almost orthogonal in order to get a small probability of collision, i.e., if one is testing different states $|\psi(w)\rangle$ and $|\psi(w')\rangle$ for equality, then a testing procedure should give a positive result with a small probability .

The following result [2] proves that a quantum collision ε-resistant function needs at least $\log\log K - c(\varepsilon)$ qubits.

Property 2 (Ablayev and Ablayev [2]*).* Let $s \geq 1$ and $K = |\mathbb{X}| \geq 4$. Let $\psi\colon \mathbb{X} \to (\mathcal{H}^2)^{\otimes s}$ be a collision ε-resistant quantum hash function. Then

$$s \geq \log\log K - \log\log\left(1 + \sqrt{2/(1-\varepsilon)}\right) - 1.$$

Properties 1 and 2 provide a basis for building a "balanced" one-way δ-resistant and collision ε-resistant quantum hash function. Roughly speaking, if we need to hash elements w from the domain $\mathbb{X}$ with $|\mathbb{X}| = K$ and if we can build for an $\varepsilon > 0$ a collision ε-resistant $(K; s)$ hash function ψ with $s \approx \log\log K - c(\varepsilon)$ qubits, then the function f is one-way δ-resistant with $\delta \approx (\log K/K)$. Such a function is balanced with respect to Property 2.

To summarize the above considerations we can state the following. A quantum (δ, ε)-hash function satisfies all of the properties that a "classical" hash function should satisfy. Pre-image resistance follows from Property 1. Second pre-image resistance and collision resistance follow, because all inputs are mapped to states that are nearly orthogonal. Therefore, we see that quantum hash functions can satisfy the properties of classical cryptographic hash functions.

4.3 Constructing Quantum Hash Functions via Small-Biased Sets

This section is based on the paper [63]. We first present a brief background on ε-biased sets. For more information see [27]. Note that ε-biased sets are generally defined for arbitrary finite groups, but here we restrict ourselves to $\mathbb{Z}_q$.

For an $a \in \mathbb{Z}_q$ a character χ_a of $\mathbb{Z}_q$ is a homomorphism $\chi_a \colon \mathbb{Z}_q \to \mu_q$, where μ_q is the (multiplicative) group of complex q-th roots of unity. That is, $\chi_a(x) = \omega^{ax}$, where $\omega = e^{\frac{2\pi i}{q}}$, is a primitive q-th root of unity. The character $\chi_0 \equiv 1$ is called a trivial character.

Definition 4. *A set $S \subseteq \mathbb{Z}_q$ is called ε-biased if for any nontrivial character $\chi \in \{\chi_a \mid a \in \mathbb{Z}_q\}$*

$$\frac{1}{|S|}\left|\sum_{x \in S} \chi(x)\right| \leq \varepsilon.$$

These sets are interesting when $|S| \ll |\mathbb{Z}_q|$ (as $S = \mathbb{Z}_q$ is 0-biased). In their seminal paper [55] Naor and Naor defined these small-biased sets, gave the first explicit constructions of such sets, and demonstrated the power of small-biased sets for several applications.

Remark 1. Note that a set S of $O(\log q/\varepsilon^2)$ elements selected uniformly at random from $\mathbb{Z}_q$ is ε-biased with positive probability [14].

Many other constructions of small-biased sets followed during the last decades.

Vasiliev [63] showed that ε-biased sets generate (δ,ε)-resistant hash functions. We present the result of [63] in the following form.

Theorem 9. *Let $S \subseteq \mathbb{Z}_q$ be an ε-biased set. Let*

$$H_S = \{h_a(x) = ax \pmod q), \quad a \in S, h_a \colon \mathbb{Z}_q \to \mathbb{Z}_q\}$$

be a set of functions determined by S. Then the quantum function $\psi_{H_S} : \mathbb{Z}_q \to (\mathcal{H}^2)^{\otimes \log |S|}$

$$|\psi_{H_S}(x)\rangle = \frac{1}{\sqrt{|S|}} \sum_{a \in S} \omega^{h_a(x)} |a\rangle$$

is a (δ, ε)-resistant quantum hash function, where $\delta \leq |S|/q$.

As a corollary from Theorem 9 and the above considerations we can state the following.

Property 3. For a small-size ε-biased set $S = \{a_1, \ldots, a_T\} \subset \mathbb{F}_q$ with $T = O(\log q/\varepsilon^2)$, for $s = \log T$, for $\delta = O(1/(q\varepsilon^2))$, H_S generates the quantum (δ, ε)-hash function

$$\psi_{H_S} : \mathbb{F}_q \to (\mathcal{H}^2)^{\otimes s} \tag{4}$$

$$|\psi_{H_S}(x)\rangle = \frac{1}{\sqrt{T}} \sum_{j=0}^{T-1} \omega^{a_j x} |j\rangle. \tag{5}$$

5 Computing Quantum Hash Function by QOBDD

The complexity of computing the quantum hash function ψ_{H_S} in the QOBDD model is given by the following theorem.

Theorem 10. *The quantum (δ, ε)-hash function (4)*

$$\psi_{H_S} : \mathbb{F}_q \to (\mathcal{H}^2)^{\otimes s}$$

can be computed by a quantum OBDD Q composed of $s = O(\log \log q)$ qubits.

Proof. The quantum function ψ_{H_S} (4) maps an input $x \in \mathbb{F}_q$ to a quantum state (5), i.e.,

$$|\psi_{H_S}(x)\rangle = \frac{1}{\sqrt{T}} \sum_{j=0}^{T-1} \omega^{a_j x} |j\rangle,$$

which is the result of a transformation (similar to the well-known *quantum Fourier transformation*, QFT) of the initial state

$$|\psi_0\rangle = \frac{1}{\sqrt{T}} \sum_{j=0}^{T-1} |j\rangle.$$

Such a QFT-like operator is controlled by the input x. We represent an integer $x \in \{0, \ldots, q-1\}$ as the bit string $x = x_0 \ldots x_{\log q - 1}$, i.e., $x = x_0 + 2^1 x_1 + \cdots + 2^{\log q - 1} x_{\log q - 1}$.

For a binary string $x = x_0 \dots x_{\log q - 1}$ a quantum OBDD Q over the space $(\mathcal{H}^2)^{\otimes s}$ for computing $\psi_{H_S}(x)$ (composed of $s = \log T$ qubits) is defined as

$$Q = \langle \mathbb{T}, |\psi_0\rangle \rangle,$$

where $|\psi_0\rangle$ is the initial state and $\mathbb{T}$ is a sequence of $\log q$ instructions:

$$\mathbb{T}_j = (x_j, U_j(0), U_j(1))$$

is determined by the variable x_j tested on step j, and $U_j(0)$, $U_j(1)$ are unitary transformations in $(\mathcal{H}^2)^{\otimes s}$. More precisely, $U_j(0)$ is the $T \times T$ identity matrix. $U_j(1)$ is the $T \times T$ diagonal matrix whose diagonal entries are $\omega^{a_0 2^j}, \omega^{a_1 2^j}, \dots, \omega^{a_{T-1} 2^j}$ and the off-diagonal elements are all zero, i.e.,

$$U_j(1) = \begin{bmatrix} \omega^{a_0 2^j} & & & \\ & \omega^{a_1 2^j} & & \\ & & \ddots & \\ & & & \omega^{a_{T-1} 2^j} \end{bmatrix}.$$

Note that here we compute a quantum function instead of a Boolean function, and thus we abandoned the measurement phase of the computation and the corresponding accepting set in the construction of a quantum branching program.

We define a computation of Q on an input $x = x_0 \dots x_{\log q - 1} \in \{0,1\}^{\log q}$ as follows:

1. A computation of Q starts from the initial state $|\psi_0\rangle$;
2. The j-th instruction of Q reads the input symbol x_j (the value of x) and applies the transition matrix $U_j(x_j)$ to the current state $|\psi\rangle$ to obtain the state $|\psi'\rangle = U_j(x_j)|\psi\rangle$;
3. The final state is

$$|\psi_{H_S}(x)\rangle = \left(\prod_{j=0}^{\log q - 1} U_j(x_j) \right) |\psi_0\rangle.$$

□

Upper bounds. From Theorem 10 we have the following corollary.

Corollary 1. *It holds that*

$$\begin{aligned} \mathit{Width}(\psi_{H_S}) &= O(\log q), \\ \mathit{Length}(\psi_{H_S}) &= O(\log q). \end{aligned}$$

Proof. $O(\log q)$ width corresponds to $O(\log \log q)$ qubits used by the QOBDD, and $O(\log q)$ length is due to the binary encoding of the input $x \in \mathbb{F}_q$.

Lower bounds. Here we show that the quantum OBDD from Theorem 10 is optimal for the function ψ_{H_S}.

Theorem 11. *It holds that*

$$Width(\psi_{H_S}) = \Omega(\log q), \tag{6}$$

$$Length(\psi_{H_S}) = \Omega(\log q). \tag{7}$$

Proof. Let Q be a QOBDD computing the function ψ_{H_S}. Since QOBDDs are defined for binary inputs, we use the following encoding for ψ_{H_S}:

$$\psi_{H_S} \colon \{0,1\}^{\log q} \to (\mathcal{H}^2)^{\otimes s}.$$

The lower bound (6) for $Width(\psi_{H_S})$ follows immediately from the minimal number of qubits required by Property 2.

$$s \geq \log\log q - \log\log(1 + \sqrt{2/(1-\varepsilon)}).$$

The lower bound (7) for $Length(\psi_{H_S})$ follows from the fact that ψ_{H_S} is a collision ε-resistant function. Indeed, the assumption that the QOBDD Q for ψ_{H_S} can test less than $\log q$ (that is, not all $\log q$) variables encoding the input $x \in \mathbb{F}_q$ means the existence of (at least) two different inputs $w, w' \in \mathbb{F}_q$, such that Q produces the same quantum hashes $|\psi(w)\rangle$ and $|\psi(w')\rangle$ for w and w', that is, $|\psi(w)\rangle = |\psi(w')\rangle = |\psi\rangle$. The latter contradicts the fact that the states $|\psi(w)\rangle$ and $|\psi(w')\rangle$ are ε-orthogonal:

$$|\langle\psi(w)|\psi(w')\rangle| \leq \varepsilon.$$

□

6 Online Streaming Algorithms

Online algorithms are well-known as a computational model for solving optimization problems. The defining property of this model is that the algorithm reads an input piece by piece and should return output variables after some of the input variables immediately, even if the answer depends on whole input. An online algorithm should return an output for minimizing an objective function. There are different methods to define the efficiency of online algorithms [22,23,28], but the most standard is the competitive ratio [40].

Typically, online algorithms have unlimited computational power and the main restriction is the lack of knowledge on future input variables. There are many problems that can be formulated in these terms. At the same time it is quite interesting to solve online minimization problems in the case of large input streams. We consider a large stream such that it cannot be stored in memory. In this situation, we can discuss online algorithms with restricted memory. In this paper, we consider streaming algorithms as online algorithms. This classical model was considered in [18,24,30,47]. Automata for online minimization problems were considered in [43]. We are interested in the investigation of a new model of online algorithms, the *quantum online algorithms*, that use the power of quantum computing for solving online minimization problems. This model was introduced in [47]. Here, we focus on quantum online streaming algorithms.

6.1 Definitions

Let us define online optimization problems. We give all following definitions with respect to [42,45,47,67]. An *online minimization problem* consists of a set $\mathcal{I}$ of inputs and a cost function. Every input $I \in \mathcal{I}$ is a sequence of requests $I = (x_1, \dots, x_n)$. Furthermore, a set of feasible outputs (or solutions) is associated with every I; every output is a sequence of answers $O = (y_1, \dots, y_n)$. The cost function assigns a positive real value $cost(I, O)$ to every input I and any feasible output O. For every input I, we call any feasible output O for I that has the smallest possible cost (i.e., that minimizes the cost function) an optimal solution for I.

Let us define an online algorithm for a given online problem as an algorithm which gets requests x_i from $I = (x_1, \dots, x_n)$ one by one and should return answers y_i from $O = (y_1, \dots, y_n)$ immediately, even if an optimal solution can depend on future requests. A *deterministic online algorithm* A computes the output sequence $A(I) = (y_1, \dots, y_n)$ such that y_i is computed from $x_1, \dots, x_i$. This setting can also be regarded as a request-answer game: an adversary generates requests, and an online algorithm has to serve them one at a time [13].

We use the competitive ratio as the main measure of quality for online algorithms. It is the ratio of the cost of the algorithm's solution and the cost of a solution of an optimal offline algorithm in the worst case. We say that some deterministic online algorithm A is c-*competitive* if there exists a non-negative constant α such that, for every n and for any input I, we have: $cost(A(I)) \leq c \cdot cost(Opt(I)) + \alpha$, where Opt is an optimal offline algorithm for the problem, $|I| \leq n$ and $|I|$ is length of I. We also call c the *competitive ratio* of A. If $\alpha = 0$, then A is called strictly c-competitive; A is optimal if it is strictly 1-competitive.

Let us define an online algorithm with advice. We can say that advice is some information about the future input. An *online algorithm A with advice* computes the output sequence $A^{\phi}(I) = (y_1, \dots, y_n)$ such that y_i is computed from $\phi, x_1, \dots, x_i$, where ϕ is the message from an *adviser*, who knows the whole input. A is c-competitive with advice complexity $b = b(n)$ if there exists a non-negative constant α such that, for every n and for any input I, there exists some ϕ such that $cost(A^{\phi}(I)) \leq c \cdot cost(Opt(I)) + \alpha$ and $|\phi| \leq b$, $|I| \leq n$.

Next, let us define a randomized online algorithm. A *randomized online algorithm R* computes the output sequence $R^{\psi} := R^{\psi}(I) = (y_1, \dots, y_n)$ such that y_i is computed from $\psi, x_1, \dots, x_i$, where ψ is the content of a random tape, i.e., an infinite binary sequence, where every bit is chosen uniformly at random and independently of all the others. By $cost(R^{\psi}(I))$ we denote the random variable expressing the cost of the solution computed by R on I. R is c-competitive in expectation if there exists a non-negative constant α such that, for every I, $\mathbb{E}[cost(R^{\psi}(I))] \leq c \cdot cost(Opt(I)) + \alpha$.

We use streaming algorithms for online minimization problems as online algorithms with restricted memory. You can read more about streaming algorithms in [53]. Shortly, these are algorithms that use a small amount of memory and read input variables one by one. Suppose A is a *deterministic online streaming* *algorithm* with $s = s(n)$ bits of memory that processes the input $I = (x_1, \ldots, x_n)$. Then we can describe a state of the memory of A before reading input variable x_{i+1} by a vector $d^i = (d^i_1, \ldots, d^i_s) \in \{0,1\}^s$. The algorithm computes the output $A(I) = (y_1, \ldots, y_n)$ such that y_i depends on d^{i-1} and x_i; d^i depends on d^{i-1} and x_i. *Randomized online streaming algorithms* and *deterministic online streaming algorithms with advice* have similar definitions, but with respect to the definitions of the corresponding models of online algorithms.

Now we are ready to define a *quantum online algorithm.* A *quantum online algorithm* Q computes the output sequence $Q(I) = (y_1, \ldots, y_n)$ such that y_i depends on $x_1, \ldots, x_i$. The algorithm can measure qubits several times during a computation. Note that quantum computation is a probabilistic process. Q is c-competitive in expectation if there exists a non-negative constant α such that, for every I, $\mathbb{E}[cost(Q(I))] \leq c \cdot cost(Opt(I)) + \alpha$.

Let us consider a *quantum online streaming algorithm.* For a given $n > 0$, a quantum online algorithm Q with q qubits is defined on the input $I = (x_1, \ldots, x_n) \in \{0, \ldots, \alpha - 1\}^n$ and outputs $(y_1, \ldots, y_m) \in \{0, \ldots, \beta - 1\}^m$. The algorithm is a triple $Q = (\mathbb{T}, |\psi_0\rangle, Result)$, where $\mathbb{T} = \{\mathbb{T}_j \mid 1 \leq j \leq n$ and $\mathbb{T}_j = (U^0_j, \ldots, U^{\alpha-1}_j)\}$ are (left) unitary matrices representing the transitions. Here, $U^{x_j}_j$ is applied on the j-th step. $|\psi_0\rangle$ is an initial vector from the 2^q-dimensional Hilbert space. $Result = \{Result_1, \ldots, Result_n\}$, where $Result_i \colon \{0, \ldots, 2^q - 1\} \to \{0, \ldots, \beta - 1\}$ is a function that converts the result of the measurement to an output variable. For any given input I, the computation of A on I can be traced by a 2^q-dimensional vector from the Hilbert space. The initial one is $|\psi_0\rangle$. In each step $j \in \{1, \ldots, n\}$ the input variable x_j is tested and then the $U^{x_j}_j$ unitary operator is applied: $|\psi_j\rangle = U^{x_j}_j(|\psi_{j-1}\rangle)$, where $|\psi_j\rangle$ represents the state of the system after the j-th step. The algorithm can measure one of the qubits or more on any step after a unitary transformation; or it can skip the measurement. Suppose that Q is in the state $|\psi\rangle = (v_1, \ldots, v_{2^q})^T$ before the measurement, and the i-th qubit is measured. Let the states with numbers $a^0_1, \ldots, a^0_{2^{q-1}}$ correspond to the 0 value of the i-th qubit, and the states with numbers $a^1_1, \ldots, a^1_{2^{q-1}}$ correspond to the 1 value of the qubit. The result of the measurement of the qubit is 1 with probability $pr_1 = \sum_{j=1}^{2^{q-1}} |v_{a^1_j}|^2$ and 0 with probability $pr_0 = 1 - pr_1$. If r qubits were measured on the j-th step, then we get the number $\gamma \in \{0, \ldots, 2^r - 1\}$ as a result and A returns $Result_j(\gamma)$.

The following lemma describes relations between automata, *id*-OBDDs and streaming algorithms that are folklore.

Lemma 1. *If a quantum (probabilistic) id-OBDD P of width 2^w computes a Boolean function f, then there is a quantum (randomized) streaming algorithm computing f that uses $w + \lceil \log_2 n \rceil$ qubits (bits). If a quantum (probabilistic) automaton A of size 2^w recognizes a language L, then there is a quantum (randomized) streaming algorithm recognizing L that uses w qubits (bits). If any deterministic (probabilistic) id-OBDD P computing a Boolean function f has width at least 2^w, then any deterministic (randomized) streaming algorithm computing f uses at least w bits. If any deterministic (probabilistic) automaton A*

recognizing a language L has size at least 2^w, then any deterministic (randomized) streaming algorithm recognizing L uses at least w bits.

Let us describe the "black hats method" from [45] that allows to construct hard online minimization problems. In this paper, we discuss a Boolean function f, but in fact we consider a family of Boolean functions $f = \{f_1, f_2, \dots\}$, where $f_m\colon \{0,1\}^m \to \{0,1\}$. We use the notation $f(X)$ for $f_m(X)$ if the length of X is m and it is clear from context.

Definition 5 (Black Hats Method). *Let f be a Boolean function. Then $BH^t_{k,r,w}(f)$, for positive integers k, r, w, t, where $k \bmod t = 0$, is the following online minimization problem: Suppose we have an input $I = (x_1, \dots, x_n)$ and k positive integers $m_1, \dots, m_k$, where $n = \sum_{i=1}^{k}(m_i + 1)$. Lets assume that $I = 2, X_1, 2, X_2, 2, X_3, 2, \dots, 2, X_k$, where $X_i \in \{0,1\}^{m_i}$, for $i \in \{1, \dots, k\}$. Let O be an output and $O' = (y_1, \dots, y_k)$ be the output bits corresponding to input variables with value 2 (in other words, output variables for guardians). Output y_j corresponds to an input variable x_{i_j}, where $i_j = j + \sum_{r=1}^{j-1} m_r$. Let $g_j(I) = \bigoplus_{i=j}^{k} f_{m_i}(X_i)$. We split all output variables y_i into t blocks each of length $u = k/t$. The cost of the i-th block is c_i, where $c_i = r$ if $y_j = g_j(I)$ for $j \in \{(i-1)u + 1, \dots, i \cdot u\}$, and $c_i = w$ otherwise. The cost of the whole output is $cost^t(I, O) = c_1 + \dots + c_t$.*

6.2 Quantum vs. Classical Online Streaming Algorithms

Khadiev, Ziatdinov, Mannapov, Khadieva and Yamilov [46,47] showed that quantum online streaming algorithms can be better than classical ones in the case of sublogarithmic memory (see the following theorem).

Theorem 12 (Khadiev et al. [46], Khadiev et al. [47]). *There is an online minimization problem BHP with the following properties:*

- *There is a quantum online streaming algorithm Q for BHP such that it uses 1 qubit of memory and has expected competitive ratio c.*
- *Any deterministic online algorithm D with unlimited computational power solving BHP is c'-competitive, for $c' > c$.*
- *Any randomized online streaming algorithm R using $o(\log n)$ bits and solving BHP is c''-competitive in expectation, for $c'' > c$.*

Proof (Sketch). To define the *BHP* problem, we use the Boolean function $PartialMOD^s_n$ from [6,7,16] and the black hats method for constructing hard minimization problems from [45]. We propose a single qubit quantum online streaming algorithm for the problem and show lower bounds for classical models.

Let us present the definition of the Boolean function $PartialMOD^s_n$. The feasible inputs for the problem are $X = (x_1, \dots, x_n) \in \{0,1\}^n$ such that $\#_1(X) = v \cdot 2^s$, where $\#_1(X)$ is the number of 1s in X and $v \geq 2$. $PartialMOD^s_n(X) = v \bmod 2$. □

The case of polylogarithmic memory was considered by Khadiev, Khadieva, Kravchenko, Rivosh, Yamilov and Mannapov [45]. Similar supremacy is shown in the following result.

Theorem 13 (Khadiev et al. [46]). *There is an online minimization problem BHR with the following properties:*

- *There is a quantum online streaming algorithm Q for BHR such that it uses $\log^{O(1)} n$ qubits of memory and has expected competitive ratio c.*
- *Any deterministic online algorithm D with unlimited computational power solving BHR is c'-competitive, for $c' > c$.*
- *Any randomized online streaming algorithm R using $n^{o(1)}$ bits and solving BHR is c''-competitive in expectation, for $c'' > c$.*

Proof (Sketch). For the definition of the *BHR* problem, we use $R_{\nu,l,m,n}$ from [60] and the black hats method. We propose a single qubit quantum online streaming algorithm for the problem and show lower bounds for classical models.

Let us define $R_{\nu,l,m,n}$. Let $|1\rangle, \ldots, |n\rangle$ be the standard computational basis of the n-dimensional Hilbert space. Let V_0 and V_1 denote the subspaces spanned by the first and last $n/2$ of these basis vectors. Let $0 < \nu < 1/\sqrt{2}$. The input for the function $R_{\nu,l,m,n}$ consists of $3l(m+1)$ Boolean variables $a_{i,j}, b_{i,j}, c_{i,j}, 1 \leq i \leq l, 1 \leq j \leq m+1$, which are interpreted as universal (ε, l, m)-codes for three unitary $n \times n$ matrices A, B, C, where $\varepsilon = 1/(3n)$. The function takes the value $z \in \{0,1\}$ if the Euclidean distance between $CBA|1\rangle$ and V_z is at most ν; otherwise the function is undefined. □

6.3 Advice Complexity of Quantum and Classical Online Streaming Algorithms

We are also interested in the *advice complexity* of online algorithms [21,50]. In this model an online algorithm gets some bits of advice about the input. The trusted *adviser* sending these bits knows the whole input and has unlimited computational power. The question is "How many advice bits are sufficient to reduce the competitive ratio or to make the online algorithm as efficient as the offline algorithm?" This question has different interpretations. One of them is "How much information should an algorithm have about the future for solving a problem efficiently?" Another one is "If we have an expensive channel which can be used for pre-processed information about the future, then how many bits should we send via this channel to solve a problem efficiently?" Deterministic and probabilistic or randomized online algorithms with advice were investigated in [36,50]. It is interesting to compare the power of quantum online streaming algorithms and classical ones.

Let us consider the case of sublogarithmic memory. Khadiev, Ziatdinov, Mannapov, Khadieva and Yamilov [46,47] showed that quantum online streaming algorithms can be better than classical ones even if the classical algorithms get advice bits. Moreover, if a quantum online streaming algorithm gets a single advice bit, then it becomes optimal.

Theorem 14 (Khadiev et al. [46], Khadiev et al. [47]). *There is an online minimization problem BHP with the following properties:*

- *There is a quantum online streaming algorithm Q for BHP such that it uses a single qubit of memory and has expected competitive ratio c, for* $c \geq 1$*.*
- *There is an* optimal *quantum online streaming algorithm* Q' *for BHP such that it uses a single qubit of memory and a single advice bit.*
- *Any deterministic online streaming algorithm D using* $o(\log n)$ *bits of memory,* $o(\log n)$ *advice bits and solving BHP is* c'*-competitive, for* $c' > c$*.*
- *Any randomized online streaming algorithm R using* $o(\log n)$ *bits of memory,* $o(\log n)$ *advice bits and solving BHP is* c''*-competitive in expectation, for* $c'' > c$*.*

Proof (Sketch). We use the same problem *BHP* as in Theorem 12 that is based on the Boolean function $PartialMOD_n^s$ from [6,7,16]. We can construct an optimal quantum algorithm, because we have an exact quantum streaming algorithm for $PartialMOD_n^s$. □

Quantum online streaming algorithms can also be better than deterministic algorithms with unlimited computational power and a constant number of advice bits.

Theorem 15 (Khadiev et al. [46]). *There is an online minimization problem* $IBHP_\lambda$ *with the following properties:*

- *There is a quantum online streaming algorithm Q for* $IBHP_\lambda$ *such that it uses constant memory, less than* λ *advice bits and has expected competitive ratio c, for* $c \geq 1$*.*
- *Any deterministic online algorithm D with unlimited computational power using less than* λ *advice bits and solving* $IBHP_\lambda$ *is* c'*-competitive, for* $c' > c$*.*

Proof (Sketch). We use a modification of the *BHP* problem from Theorems 12 and 14. This modification uses λ copies of the *BHP* problem. □

At the same time, randomized online streaming algorithms cannot achieve a similar supremacy for the *BHP* problem as quantum online streaming algorithms. The reason is that there is no randomized streaming algorithm that can compute the Boolean function $PartialMOD_n^s$ using constant memory.

Let us again consider the case of polylogarithmic memory. Khadiev, Ziatdinov, Mannapov, Khadieva and Yamilov [46] and same authors with Kravchenko and Rivosh [45] considered this case. They showed that we have a situation similar to the sublogarithmic case. A quantum online streaming algorithm can be better than classical ones even if classical algorithms get advice.

Theorem 16 (Khadiev et al. [46], Khadiev et al. [45]). *There is an online minimization problem BHR with the following properties:*

- *There is a quantum online streaming algorithm Q for BHR such that it uses* $\log^{O(1)} n$ *qubits of memory and has expected competitive ratio c, for* $c \geq 1$*.*

– *Any deterministic online streaming algorithm D using $n^{o(1)}$ bits of memory, $o(\log n)$ advice bits and solving BHR is c'-competitive, for $c' > c$.*
– *Any randomized online streaming algorithm R using $n^{o(1)}$ bits of memory, $o(\log n)$ advice bits and solving BHR is c''-competitive in expectation, for $c'' > c$.*

Proof (Sketch). We use the same problem *BHP* as in Theorem 13 that is based on the Boolean function $R_{\nu,l,m,n}$ from [60]. □

Acknowledgements. Partially supported by ERC Advanced Grant MQC. The work is performed according to the Russian Government Program of Competitive Growth of Kazan Federal University. Work was in part supported by the Russian Foundation for Basic Research (under the grant 17-07-01606).

References

1. Ablayev, F.: Randomization and nondeterminism are incomparable for ordered read-once branching programs. In: Electronic Colloquium on Computational Complexity (ECCC) (021) (1997)
2. Ablayev, F., Ablayev, M.: Quantum hashing via ε-universal hashing constructions and classical fingerprinting. Lobachevskii J. Math. **36**(2), 89–96 (2015). https://doi.org/10.1134/S199508021502002X
3. Ablayev, F., Ambainis, A., Khadiev, K., Khadieva, A.: Lower bounds and hierarchies for quantum memoryless communication protocols and quantum ordered binary decision diagrams with repeated test. In: Tjoa, A.M., Bellatreche, L., Biffl, S., van Leeuwen, J., Wiedermann, J. (eds.) SOFSEM 2018. LNCS, vol. 10706, pp. 197–211. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73117-9_14
4. Ablayev, F., Gainutdinova, A., Karpinski, M.: On computational power of quantum branching programs. In: Freivalds, R. (ed.) FCT 2001. LNCS, vol. 2138, pp. 59–70. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44669-9_8
5. Ablayev, F., Gainutdinova, A., Karpinski, M., Moore, C., Pollett, C.: On the computational power of probabilistic and quantum branching program. Inf. Comput. **203**(2), 145–162 (2005)
6. Ablayev, F., Gainutdinova, A., Khadiev, K., Yakaryılmaz, A.: Very narrow quantum OBDDs and width hierarchies for classical obdds. Lobachevskii J. Math. **37**(6), 670–682 (2016). https://doi.org/10.1134/S199508021606007X
7. Ablayev, F., Gainutdinova, A., Khadiev, K., Yakaryılmaz, A.: Very narrow quantum OBDDs and width hierarchies for classical OBDDs. In: Jürgensen, H., Karhumäki, J., Okhotin, A. (eds.) DCFS 2014. LNCS, vol. 8614, pp. 53–64. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09704-6_6
8. Ablayev, F., Karpinski, M.: On the power of randomized ordered branching programs. In: Electronic Colloquium on Computational Complexity (ECCC) (004) (1998)
9. Ablayev, F., Ablayev, M.: On the concept of cryptographic quantum hashing. Laser Phys. Lett. **12**(12), 125204 (2015). http://stacks.iop.org/1612-202X/12/i=12/a=125204
10. Ablayev, F., Gainutdinova, A.: Complexity of quantum uniform and nonuniform automata. In: De Felice, C., Restivo, A. (eds.) DLT 2005. LNCS, vol. 3572, pp. 78–87. Springer, Heidelberg (2005). https://doi.org/10.1007/11505877_7

11. Ablayev, F., Khadiev, K.: Extension of the hierarchy for k-OBDDs of small width. Russ. Math. **53**(3), 46–50 (2013)
12. Ablayev, F., Karpinski, M.: On the power of randomized branching programs. In: Meyer, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 348–356. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61440-0_141
13. Albers, S.: BRICS, Mini-Course on Competitive Online Algorithms. Aarhus University (1996)
14. Alon, N., Roichman, Y.: Random Cayley graphs and expanders. Random Struct. Algorithms **5**(2), 271–284 (1994). https://doi.org/10.1002/rsa.3240050203
15. Ambainis, A., Watrous, J.: Two-way finite automata with quantum and classical states. Theor. Comput. Sci. **287**(1), 299–311 (2002)
16. Ambainis, A., Yakaryılmaz, A.: Superiority of exact quantum automata for promise problems. Inf. Process. Lett. **112**(7), 289–291 (2012)
17. Barrington, D.A.M.: Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. J. Comput. Syst. Sci. **38**(1), 150–164 (1989)
18. Becchetti, L., Koutsoupias, E.: Competitive analysis of aggregate max in windowed streaming. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 156–170. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02927-1_15
19. Bollig, B., Sauerhoff, M., Sieling, D., Wegener, I.: Hierarchy theorems for kOBDDs and kIBDDs. Theor. Comput. Sci. **205**(1), 45–60 (1998)
20. Borodin, A., Razborov, A., Smolensky, R.: On lower bounds for read-k-times branching programs. Comput. Complex. **3**(1), 1–18 (1993)
21. Boyar, J., Favrholdt, L., Kudahl, C., Larsen, K., Mikkelsen, J.: Online algorithms with advice: a survey. ACM Comput. Surv. (CSUR) **50**(2), 19 (2017)
22. Boyar, J., Irani, S., Larsen, K.S.: A comparison of performance measures for online algorithms. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 119–130. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03367-4_11
23. Boyar, J., Irani, S., Larsen, K.S.: A comparison of performance measures for online algorithms. Algorithmica **72**(4), 969–994 (2015)
24. Boyar, J., Larsen, K.S., Maiti, A.: The frequent items problem in online streaming under various performance measures. Int. J. Found. Comput. Sci. **26**(4), 413–439 (2015)
25. Brosenne, H., Homeister, M., Waack, S.: Nondeterministic ordered binary decision diagrams with repeated tests and various modes of acceptance. Inf. Process. Lett. **98**(1), 6–10 (2006)
26. Buhrman, H., Cleve, R., Watrous, J., de Wolf, R.: Quantum fingerprinting. Phys. Rev. Lett. **87**(16), 167902 (2001). https://doi.org/10.1103/PhysRevLett.87.167902. http://www.arXiv.org/quant-ph/0102001v1
27. Chen, S., Moore, C., Russell, A.: Small-bias sets for nonabelian groups. In: Raghavendra, P., Raskhodnikova, S., Jansen, K., Rolim, J.D.P. (eds.) APPROX/RANDOM -2013. LNCS, vol. 8096, pp. 436–451. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40328-6_31
28. Dorrigiv, R., López-Ortiz, A.: A survey of performance measures for on-line algorithms. SIGACT News **36**(3), 67–81 (2005)
29. Duriš, P., Hromkovič, J., Rolim, J.D.P., Schnitger, G.: Las Vegas versus determinism for one-way communication complexity, finite automata, and polynomial-time computations. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 117–128. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0023453

30. Giannakopoulos, Y., Koutsoupias, E.: Competitive analysis of maintaining frequent items of a stream. Theor. Comput. Sci. **562**, 23–32 (2015)
31. Gottesman, D., Chuang, I.: Quantum digital signatures. Technical report, Cornell University Library, November 2001. http://arxiv.org/abs/quant-ph/0105032
32. Hirvensalo, M., Seibert, S.: Lower bounds for Las Vegas automata by information theory. RAIRO-Theor. Inform. Appl. **37**(1), 39–49 (2003)
33. Holevo, A.S.: Some estimates of the information transmitted by quantum communication channel (russian). Probl. Pered. Inform. [Probl. Inf. Transm.] 9(3), 3–11 (1973)
34. Homeister, M., Waack, S.: Quantum ordered binary decision diagrams with repeated tests. arXiv preprint quant-ph/0507258 (2005)
35. Hromkovič, J., Sauerhoff, M.: The power of nondeterminism and randomness for oblivious branching programs. Theory Comput. Syst. **36**(2), 159–182 (2003)
36. Hromkovic, J.: Zámecniková. design and analysis of randomized algorithms: introduction to design paradigms (2005)
37. Hromkovič, J., Schnitger, G.: On the power of Las Vegas for one-way communication complexity, OBDDs, and finite automata. Inf. Comput. **169**(2), 284–296 (2001)
38. Ibrahimov, R., Khadiev, K., Prusis, K., Yakaryılmaz, A.: Zero-error affine, unitary, and probabilistic obdds. Technical report. arXiv 1703.07184 (2017)
39. Klauck, H., Nayak, A., Ta-Shma, A., Zuckerman, D.: Interaction in quantum communication and the complexity of set disjointness. In: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, pp. 124–133. ACM (2001)
40. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. In: 1986 27th Annual Symposium on Foundations of Computer Science, pp. 244–254. IEEE (1986)
41. Khadiev, K.: Width hierarchy for k-obdd of small width. Lobachevskii J. Math. **36**(2), 178–183 (2015)
42. Khadiev, K.: On the hierarchies for deterministic, nondeterministic and probabilistic ordered read-k-times branching programs. Lobachevskii J. Math. **37**(6), 682–703 (2016)
43. Khadiev, K., Khadieva, A.: Quantum automata for online minimization problems. In: Ninth Workshop on NCMA 2017 Short Papers, pp. 25–33. Institute fur Computersprachen TU Wien (2017)
44. Khadiev, K., Khadieva, A.: Reordering method and hierarchies for quantum and classical ordered binary decision diagrams. In: Weil, P. (ed.) CSR 2017. LNCS, vol. 10304, pp. 162–175. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58747-9_16
45. Khadiev, K., Khadieva, A., Kravchenko, D., Rivosh, A., Yamilov, R., Mannapov, I.: Quantum versus classical online algorithms with advice and logarithmic space. arXiv:1710.09595 (2017)
46. Khadiev, K., Ziatdinov, M., Mannapov, I., Khadieva, A., Yamilov, R.: Quantum online streaming algorithms with constant number of advice bits. arXiv:1802.05134 (2018)
47. Khadiev, K., Khadieva, A., Mannapov, I.: Quantum online algorithms with respect to space complexity. arXiv:1709.08409 (2017)
48. Klauck, H., Nayak, A., Ta-Shma, A., Zuckerman, D.: Interaction in quantum communication. IEEE Trans. Inf. Theory **53**(6), 1970–1982 (2007)

49. Klauck, H.: On quantum and probabilistic communication: Las vegas and one-way protocols. In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, STOC 2000, pp. 644–651 (2000)
50. Komm, D.: An Introduction to Online Computation: Determinism, Randomization, Advice. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-319-42749-2
51. Li, D., Zhang, J., Guo, F.Z., Huang, W., Wen, Q.Y., Chen, H.: Discrete-time interacting quantum walks and quantum hash schemes. Quantum Inf. Process. **12**(3), 1501–1513 (2013). https://doi.org/10.1007/s11128-012-0421-8
52. Li, D., Zhang, J., Ma, X.W., Zhang, W.W., Wen, Q.Y.: Analysis of the two-particle controlled interacting quantum walks. Quantum Inf. Process. **12**(6), 2167–2176 (2013). https://doi.org/10.1007/s11128-012-0516-2
53. Muthukrishnan, S., et al.: Data streams: algorithms and applications. Found. Trends® Theor. Comput. Sci. **1**(2), 117–236 (2005)
54. Nakanishi, M., Hamaguchi, K., Kashiwabara, T.: Ordered quantum branching programs are more powerful than ordered probabilistic branching programs under a bounded-width restriction. In: Du, D.-Z.-Z., Eades, P., Estivill-Castro, V., Lin, X., Sharma, A. (eds.) COCOON 2000. LNCS, vol. 1858, pp. 467–476. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44968-X_46
55. Naor, J., Naor, M.: Small-bias probability spaces: efficient constructions and applications. In: Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing, STOC 1990, pp. 213–223. ACM, New York (1990). https://doi.org/10.1145/100216.100244
56. Nayak, A.: Optimal lower bounds for quantum automata and random access codes. In: 1999 40th Annual Symposium on Foundations of Computer Science, pp. 369–376 (1999). https://doi.org/10.1109/SFFCS.1999.814608
57. Nisan, N., Widgerson, A.: Rounds in communication complexity revisited. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, pp. 419–429. ACM (1991)
58. Sauerhoff, M.: Quantum vs. classical read-once branching programs. In: Krause, M., Pudlák, P., Reischuk, R., van Melkebeek, D. (eds.) Complexity of Boolean Functions. No. 06111 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2006). http://drops.dagstuhl.de/opus/volltexte/2006/616
59. Sauerhoff, M., Sieling, D.: Quantum branching programs and space-bounded nonuniform quantum complexity. Theor. Comput. Sci. **334**(1–3), 177–225 (2005)
60. Sauerhoff, M., Sieling, D.: Quantum branching programs and space-bounded nonuniform quantum complexity. Theor. Comput. Sci. **334**(1), 177–225 (2005)
61. Thathachar, J.S.: On separating the read-k-times branching program hierarchy. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, pp. 653–662. ACM (1998)
62. Vasiliev, A.V.: Functions computable by Boolean circuits of logarithmic depth and branching programs of a special type. J. Appl. Ind. Math. **2**(4), 585–590 (2008). https://doi.org/10.1134/S1990478908040145
63. Vasiliev, A.: Quantum hashing for finite abelian groups. Lobachevskii J. Math. **37**(6), 751–754 (2016). http://arxiv.org/abs/1603.02209
64. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. SIAM, Philadelphia (2000)
65. Yakaryılmaz, A., Say, A.C.C.: Succinctness of two-way probabilistic and quantum finite automata. Discrete Math. Theor. Comput. Sci. **12**(2), 19–40 (2010)

66. Yang, Y.G., Xu, P., Yang, R., Zhou, Y.H., Shi, W.M.: Quantum hash function and its application to privacy amplification in quantum key distribution, pseudorandom number generation and image encryption. Sci. Rep. **6**, 19788 (2016). https://doi.org/10.1038/srep19788
67. Yuan, Q.: Quantum Online Algorithms. University of California, Santa Barbara (2009)

Stability of Reapproximation Algorithms for the β-Metric Traveling Salesman (Path) Problem

Annalisa D'Andrea[1], Luca Forlizzi[1], and Guido Proietti[1,2](✉)

[1] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università degli Studi dell'Aquila, Via Vetoio, 67100 L'Aquila, Italy
{annalisa.dandrea,luca.forlizzi,guido.proietti}@univaq.it

[2] Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", Consiglio Nazionale delle Ricerche, Via dei Taurini 19, 00185 Rome, Italy

Abstract. Inspired by the concept of *stability of approximation*, we consider the following (re)optimization problem: Given a *minimum-cost Hamiltonian cycle* of a complete non-negatively real weighted graph $G = (V, E, c)$ obeying the *strengthened* triangle inequality (i.e., for some *strength factor* $1/2 \leq \beta < 1$, we have that $\forall u, v, z \in V, c(u, z) \leq \beta(c(u, v) + c(v, z))$), and given a vertex v whose removal from G (resp., addition to G), along with all its incident edges, produces a new weighted graph still obeying the strengthened triangle inequality, find a minimum-cost Hamiltonian cycle of the modified graph. This problem is known to be NP-hard, but we show that it admits a PTAS, which just consists of either returning the old optimal cycle (after having by-passed the removed node), or instead computing (for finitely many inputs) a new optimal solution from scratch − depending on the required accuracy in the approximation. Then, we turn our attention to the case in which a *minimum-cost Hamiltonian path* is given instead, and the underlying graph obeys the *relaxed* triangle inequality. Here, if one edge weight is increased, and β_{O}, $\beta_{\mathsf{M}} \geq 1$ denotes the relaxation factor of the original and the modified graph, respectively, then we show how to obtain an approximation of $1 + 2\min\{\beta_{\mathsf{O}}, \beta_{\mathsf{M}}\}$, which improves over existing solutions as soon as $\min\{\beta_{\mathsf{O}}, \beta_{\mathsf{M}}\} \geq \frac{3+2\sqrt{6}}{5} \approx 1.58$.

1 Introduction

The hardness of optimization problems is commonly defined according to the computational complexity of worst-case input instances. However, in many cases one can find significant sets of input instances that behave substantially better in terms of tractability. In order to promote a broader notion of tractability, less tied to the tyranny of worst-case complexity, Hromkovič introduced the concept of *stability of approximation* [26,27]. This provides a formal tool to partition the set of input instances of optimization problems into possibly infinitely many subclasses according to the hardness of their approximability, and allows us to have an efficient algorithm for deciding the membership of any problem instance to one of the subclasses considered. The idea behind this concept is to find a distance measure between input instances that captures their difference from a computational complexity perspective. An algorithm that offers a good approximation for some set $\mathcal{I}$ of input instances is then called *stable* with respect to this measure if its approximation ratio grows with the distance of an input instance from $\mathcal{I}$ but not with the size of the input instances.

As a paradigmatic case study, the literature offers many examples of optimization problems which are computationally hard when defined on general weighted graphs, but become easier to solve/approximate when the input graph happens to be *metric* (i.e., its cost function obeys the *triangle inequality*). Following the concept of stability, it is then natural to assume, as distance measures from "good" instances, a parametrization of the metricity of the input graph. More precisely, we say that a non-negatively real weighted graph $G = (V, E, c)$ satisfies the β-triangle inequality, where $\beta \geq 1/2$, if $c(u, v) \leq \beta\,(c(u, z) + c(z, v)), \forall u, v, z \in V$. If $\beta < 1$, we speak about the *strengthened* triangle inequality, while if $\beta > 1$, we speak about the *relaxed* triangle inequality. Then the parameter β can be used to partition the set of all input instances of a hard optimization problem (from $\beta = 1/2$, i.e., unweighted graphs, up to $\beta = \infty$, i.e., general weighted graphs), and to classify them according to their respective computational properties. This idea proved to be very fruitful, giving rise to numerous results for different hard optimization problems, such as k-connectivity [11,12], Steiner tree [8,9,14,19], Traveling Salesman Problem (TSP for short) [1,2,5,15–17,22], and its variants [7,10,13,18,20,21,24,25].

In this paper we further explore the usefulness of the notion of stability of approximation in order to attack the *reoptimization version* of the TSP (and a Hamiltonian-path variant of it) when the input graph is β-metric.

1.1 The Reoptimization Version of the TSP

The TSP is that of finding a minimum-cost cycle spanning all the nodes of a complete real weighted graph, say $G = (V, E, c)$. It is well-known to be not only NP-hard, but also not approximable in polynomial time within a $2^{p(|V|)}$-factor, where p is any polynomial, unless $\mathsf{P} = \mathsf{NP}$. On the other hand, in the metric case, although it remains APX-hard, it can be approximated within a factor of 3/2 by means of the classic Christofides algorithm [23]. For an excellent survey on the TSP, we refer the reader to [32].

To get closer to the subject of this paper, imagine now a scenario in which, given an instance of the general TSP, along with an optimal solution, i.e., a minimum-cost Hamiltonian cycle of G, this instance is subject to a small *modification*, consisting in the insertion or the deletion of a new node into or from the graph. Then, two natural arising questions are the following: Is it still NP-hard to find an optimal solution for this problem? And if so, what about its approximability? This was exactly the pioneering formalization of a *reoptimization* version of the TSP given in [3]. More precisely, the authors focused on the metric TSP, and showed that both problems (i.e., insertion or deletion of a new node) remain

NP-hard. Moreover, they showed that for the former, the adaptation of the classic *cheapest insertion* heuristic guarantees a *linear-time* $3/2$-approximation ratio, as opposed to the $O(|V|^3)$ time required by the Christofides algorithm, while for the latter, they showed that simply deleting the edges of the solution incident to the node that we have to eliminate, and then linking the other endpoints of such edges, we obtain a $3/2$-approximation ratio.

Inspired by this work, in [13] the authors addressed another natural reoptimization variant of the metric TSP, namely that in which the instance undergoes the alteration of the weight of a single edge. Once again, the authors proved the NP-hardness of the problem, and therefore provided an extensive comparative analysis with the canonical metric TSP as far as the approximability of the problem was concerned. To this end, the authors developed their study by classifying the approximability of TSP depending on the metricity of both the input and the modified instance. More formally, they proved that: (i) if the input and the modified instance obey the *strengthened* triangle inequality, the problem admits a PTAS (this compares favorably with the APX-hardness of the counterpart); (ii) if the input and the modified instance are both metric, the problem can be approximated within a factor of $7/5$, which compares favorably with the approximation ratio guaranteed by the Christofides algorithm; this result was further improved in [6], where the authors proved the problem can be approximated within a factor of $4/3$; and finally, (iii) if the input and the modified instance obey the *relaxed* triangle inequality with $1 < \beta < 3.34899$, the problem admits a $\beta^2 \frac{15\beta^2+5\beta-6}{13\beta^2+3\beta-6}$-approximation algorithm, which is better than its counterparts given in [5,17]. In other words, for the considered range of values of β, all these reapproximation algorithms are stable.

Further reoptimization versions of the metric TSP were developed for both the strengthened and the relaxed case [4,13,21].

1.2 Our Results

In this paper we focus on two reoptimization variants of the TSP, and we show that they admit stable reapproximation algorithms.

First, we concentrate on a reoptimization version of the *strengthened metric* TSP ($\Delta_{\beta<1}$-TSP for short). We apply the same kind of perturbations introduced in [3], namely insertion and deletion of a node, trying to exploit both the optimal solution and the strengthened metric properties of the input instances, and we show that also in this case the problem on the perturbed instances admits a PTAS. Notice that, for the case of node deletion, our approach has a natural application counterpart in the so-called *transient failure* setting, namely that in which the malfunctioning of a network component (in our case, a node) is supposed to be promptly repaired. In such a scenario, it makes sense to exploit the knowledge that can be gathered from the optimal solution of the old instance (which was presumably found with much effort and time), in order to compute quickly an approximation to the new instance, and we will show that this is actually possible.

Then, we turn our attention to a more general version of the TSP, called the *Traveling Salesman Path Problem* (TSPP for short), in which we are also given two vertices $s, t \in V$ and the goal is to find a path from s to t visiting each vertex in V exactly once. More precisely, we address the *relaxed metric* TSPP ($\Delta_{\beta>1}$-TSPP for short), whose input instances obey the relaxed triangle inequality, and whose goal is the same as for the TSPP, i.e., finding a minimum-cost Hamiltonian path between the given vertices s and t. The best current approximation algorithm for this problem is the so-called *Path Matching Christofides Algorithm-TSPP* [25], with an approximation ratio of $\frac{5}{3}\beta^2$. Also in this case, we focus on the reoptimization version of the relaxed metric TSPP. More precisely, if one edge weight is (transiently) increased, and we assume that the original and the modified graph obey the triangle inequality within a relaxation factor of β_{O}, $\beta_{\mathsf{M}} \geq 1$, respectively, then we show how to obtain an approximation of $1 + 2\beta_{\mathsf{L}}$, where $\beta_{\mathsf{L}} = \min\{\beta_{\mathsf{O}}, \beta_{\mathsf{M}}\}$, which becomes the best approximation ratio for the $\Delta_{\beta>1}$-TSPP when $\beta \geq \frac{3+2\sqrt{6}}{5} \approx 1.58$.

2 Reoptimizing the Strengthened Metric TSP on Single Node Modifications

In this section we study the reoptimization of TSP subject to a (transient) single node modification. For the remaining part of this section, for an input graph G, let $\textsc{Opt}_G$ denote the cost of a minimum-cost Hamiltonian cycle of G. Formally, we define two problems, to cover both the insertion and the deletion of a single node of G.

Definition 1. *The* node-inserted TSP *problem (*TSP$^+$*) is defined as follows: Given*

- *two complete weighted graphs $G_{\mathsf{O}} = (V_{\mathsf{O}}, E_{\mathsf{O}}, c_{\mathsf{O}})$ and $G_{\mathsf{M}} = (V_{\mathsf{M}}, E_{\mathsf{M}}, c_{\mathsf{M}})$ such that $V_{\mathsf{O}} = \{v_1, \ldots, v_n\}$, $V_{\mathsf{M}} = V_{\mathsf{O}} \cup \{v_{n+1}\}$, and G_{O} coincides with the subgraph of G_{M} induced by V_{O};*
- *a Hamiltonian cycle $\overline{C}$ of G_{O} such that $\sum_{e \in \overline{C}} c(e) = \textsc{Opt}_{G_{\mathsf{O}}}$;*

find a Hamiltonian cycle C of G_{M} such that $\sum_{e \in C} c(e) = \textsc{Opt}_{G_{\mathsf{M}}}$.

Definition 2. *The* node-deleted TSP *problem (*TSP$^-$*) is defined as follows: Given*

- *two complete weighted graphs $G_{\mathsf{O}} = (V_{\mathsf{O}}, E_{\mathsf{O}}, c_{\mathsf{O}})$ and $G_{\mathsf{M}} = (V_{\mathsf{M}}, E_{\mathsf{M}}, c_{\mathsf{M}})$ such that $V_{\mathsf{O}} = \{v_1, \ldots, v_{n+1}\}$, $V_{\mathsf{M}} = V_{\mathsf{O}} \setminus \{v_i\}$, for any $i = 1, \ldots, n+1$, and G_{M} coincides with the subgraph of G_{O} induced by V_{M};*
- *a Hamiltonian cycle $\overline{C}$ of G_{O} such that $\sum_{e \in \overline{C}} c(e) = \textsc{Opt}_{G_{\mathsf{O}}}$;*

find a Hamiltonian cycle C of G_{M} such that $\sum_{e \in C} c(e) = \textsc{Opt}_{G_{\mathsf{M}}}$.

These problems are studied in [3,4], where it is shown that:

1. the problems are NP-hard;
2. in the case of an arbitrary distance between the vertices, the problems are not $2^{p(n)}$-approximable, if $\mathsf{P} \neq \mathsf{NP}$, for any polynomial p;
3. in the case where the distances between the vertices obey the triangle inequality, TSP$^+$ is approximable within a factor of 4/3 in $O(n^3)$ time [4], and within a factor of 3/2 in $O(n)$ time [3]; while TSP$^-$ is approximable, in constant time, within a factor of 3/2 [3].

In this paper we further improve these results for input instances where both G_{O} and G_{M} obey the strengthened triangle inequality up to a strength factor of β_{O} and β_{M}, respectively, with $\beta_{\mathsf{O}}, \beta_{\mathsf{M}} < 1$. We denote the restriction of TSP$^+$ (resp., TSP$^-$) to instances of such kind as $\Delta_{\beta<1}$-TSP$^+$ (resp., $\Delta_{\beta<1}$-TSP$^-$).

In contrast to the edge-perturbed TSP [24], where we know that changing the cost of even a single edge can either increase or decrease the minimum β-value such that the graph satisfies the β-triangle inequality, it is easy to see that by adding (resp., removing) a vertex to (resp., from) an instance, the corresponding β-value does not decrease (resp., increase), as shown by the following lemma:

Lemma 1. *Let G_{n+1} be a complete weighted graph of $n+1$ nodes that satisfies the $\overline{\beta}$-triangle inequality, for some $1/2 \leq \overline{\beta} < 1$, and let G_n be the subgraph of G_{n+1} induced by $V(G_n) = V(G_{n+1}) \setminus \{v\}$, for some $v \in V(G_{n+1})$. Then G_n satisfies the $\overline{\beta}$-triangle inequality.*

Proof. For any $e_1, e_2, e_3 \in E(G_n)$ that form a triangle, the edge costs in G_n are the same as those in G_{n+1}, hence they obey the $\overline{\beta}$-triangle inequality. $\square$

A natural way to reoptimize a TSP$^+$ instance is to add the vertex v_{n+1} to the optimal Hamiltonian cycle $\overline{C}$ of G_{O}, by means of a *vertex-insertion* operation that removes from $\overline{C}$ an edge (x, y), selected according to certain criteria, and replaces it with the edges (x, v_{n+1}) and (v_{n+1}, y). Concerning TSP$^-$, w.l.o.g. let us assume in the following that the removed vertex is v_{n+1}. Then, one can reoptimize a TSP$^-$ instance by using a *vertex-deletion* operation that removes from the optimal Hamiltonian cycle $\overline{C}$ of G_{O} the two edges incident to v_{n+1}, say (u, v_{n+1}) and (v_{n+1}, w), and then adds (u, w) to $\overline{C}$.

Such kind of operations are the foundations of the reoptimization algorithms for the metric case presented in [3,4], as well as of well-known approximation algorithms for the classic TSP [28]. Notice that for the TSP$^+$ one can specify different rules to select the edge to be removed from $\overline{C}$. The algorithms in [3,4] use, in particular, the *cheapest insertion* rule, that selects, among the edges in $\overline{C}$, any one that minimizes the cost increase of the Hamiltonian cycle C built by the operation. For our results, it is sufficient to employ the slightly simpler *cheapest-removal insertion* rule that just removes from $\overline{C}$ an edge of minimum cost.

Formally, our algorithms are based on the following operations:

Definition 3 (Insertion Operation). *Given a cycle C and a node v not in C, select an edge (x^*, y^*) of minimum cost in C, and build a new cycle C' from C, by removing (x^*, y^*) and then adding edges (x^*, v) and (v, y^*).*

Definition 4 (Deletion Operation). *Given a cycle C and a node v such that (x, v) and (v, y) are the edges incident to v in C, build a new cycle C' from C by removing edges (x, v) and (v, y), and then adding edge (x, y).*

The strengthened triangle inequality induces a strong relationship between edge costs, that can be used to bound the increase (resp., decrease) of the cost of the cycle modified by an insertion (resp., deletion) operation. We show this in Lemma 3, that in turn uses the following technical lemma proven in [16]:

Lemma 2 (Böckenhauer et al. [16]). *Given a complete weighted graph $G = (V, E, c)$ that satisfies the β-triangle inequality, for some $1/2 \leq \beta < 1$:*

– *For any two edges $e_1, e_2 \in E$ with a common endpoint:*

$$c(e_1) \leq \frac{\beta}{1-\beta} c(e_2). \tag{1}$$

– *Let $c_{\max}$ and $c_{\min}$ denote the maximum and the minimum, respectively, among all edge costs. Then,*

$$\frac{c_{\max}}{c_{\min}} \leq \frac{2\beta^2}{1-\beta}. \tag{2}$$

Lemma 3. *Let G_{n+1} be a complete weighted graph that satisfies the β-triangle inequality, for some $1/2 \leq \beta < 1$, such that $|V(G_{n+1})| = n + 1$, and let G_n be a subgraph of G_{n+1} such that $V(G_{n+1}) = V(G_n) \cup \{v_{n+1}\}$. Let C_{n+1} and C_n be a minimum-cost Hamiltonian cycle for G_{n+1} and for G_n, respectively, of cost $\textsc{Opt}_{G_{n+1}}$ and $\textsc{Opt}_{G_n}$, respectively. Then we have*

$$\textsc{Opt}_{G_n} < \textsc{Opt}_{G_{n+1}} \leq \left(1 + \left(\frac{2\beta}{1-\beta} - 1\right)\frac{1}{n}\right)\textsc{Opt}_{G_n}. \tag{3}$$

Proof. Let s and t be the vertices of G_{n+1} that are adjacent to v_{n+1} in C_{n+1}. From C_{n+1} we can construct a Hamiltonian cycle C_a of G_n by removing v_{n+1} with a deletion operation. Since C_n is a minimum-cost Hamiltonian cycle for G_n, we have $\textsc{Opt}_{G_n} \leq c(C_a)$. On the other hand, the strengthened triangle inequality rules out the existence of edges of weight 0 (but for the degenerate case in which all the edges have weight 0), which implies $c(C_a) < \textsc{Opt}_{G_{n+1}}$, and so we have that

$$\textsc{Opt}_{G_n} < \textsc{Opt}_{G_{n+1}}.$$

From C_n we can build a Hamiltonian cycle C_b of G_{n+1} by inserting v_{n+1} with an insertion operation; let (x^*, y^*) be the edge removed by the operation. Since C_{n+1} is a minimum-cost Hamiltonian cycle for G_{n+1}, we have $\textsc{Opt}_{G_{n+1}} \leq c(C_b)$.

On the other hand, $c(C_b) = \textsc{Opt}_{G_n} + c(x^*, v_{n+1}) + c(v_{n+1}, y^*) - c(x^*, y^*)$. Since (x^*, v_{n+1}) and (v_{n+1}, y^*) have an endpoint in common with (x^*, y^*), from (1) we have that the cost of each of them is bounded from above by $\frac{\beta}{1-\beta}c(x^*, y^*)$. Since (x^*, y^*) has the minimum cost among the edges in C_n, we have $c(x^*, y^*) \leq \frac{1}{n}\textsc{Opt}_{G_n}$. Therefore it follows that

$$\textsc{Opt}_{G_{n+1}} \leq c(C_b) \leq \left(1 + \left(\frac{2\beta}{1-\beta} - 1\right)\frac{1}{n}\right)\textsc{Opt}_{G_n}. \qquad \square$$

The previous lemma immediately suggests that it is possible to achieve a very good approximation for the $\Delta_{\beta<1}$-TSP$^+$ and for the $\Delta_{\beta<1}$-TSP$^-$, and allows us to provide our main result. More precisely, we show that returning the cycle obtained by using an insertion or deletion operation, respectively, guarantees an approximation which can be made arbitrarily close to 1 for sufficiently large input graphs.

Theorem 1. *Let $(G_{\mathsf{O}}, G_{\mathsf{M}}, \overline{C})$ be an input instance of $\Delta_{\beta<1}$-TSP$^+$ (resp., $\Delta_{\beta<1}$-TSP$^-$) such that G_{O} and G_{M} obey the triangle inequality up to a strength factor of β_{O} and β_{M}, respectively, with $\beta_{\mathsf{O}}, \beta_{\mathsf{M}} < 1$. Let G_{O} and G_{M} agree except for a single node v_i along with all its incident edges. Let $\beta_{\max} := \max\{\beta_{\mathsf{O}}, \beta_{\mathsf{M}}\} < 1$ and $n = \min\{|V(G_{\mathsf{O}})|, |V(G_{\mathsf{M}})|\}$. Then, it is a $\left(1 + \left(\frac{2\beta_{\max}}{1-\beta_{\max}} - 1\right)\frac{1}{n}\right)$-approximation to simply output the cycle obtained by using the insertion (resp., deletion) operation.*

Proof. Let us start with the $\Delta_{\beta<1}$-TSP$^+$: let $n = |V(G_{\mathsf{O}})|$ and let v_{n+1} be the vertex of G_{M} that does not belong to G_{O}. From Lemma 1, we know that G_{O} satisfies the β_{M}-triangle inequality, hence $\beta_{\max} = \beta_{\mathsf{M}}$. Using the insertion operation, we obtain, from $\overline{C}$, a Hamiltonian cycle C of G_{M} that, as shown in Lemma 3, has a cost bounded from above by $\left(1 + \left(\frac{2\beta_{\max}}{1-\beta_{\max}} - 1\right)\frac{1}{n}\right)\textsc{Opt}_{G_{\mathsf{O}}}$. From Lemma 3, we also have that $\textsc{Opt}_{G_{\mathsf{O}}} \leq \textsc{Opt}_{G_{\mathsf{M}}}$, therefore

$$c(C) \leq \left(1 + \left(\frac{2\beta_{\max}}{1-\beta_{\max}} - 1\right)\frac{1}{n}\right)\textsc{Opt}_{G_{\mathsf{M}}}.$$

For the $\Delta_{\beta<1}$-TSP$^-$, let $n = |V(G_{\mathsf{M}})|$ and let v be the vertex of G_{O} that does not belong to G_{M}. From Lemma 1, again, we know that $\beta_{\max} = \beta_{\mathsf{O}}$. Now, from $\overline{C}$, using the deletion operation, we obtain a Hamiltonian cycle C of G_{M} that, by the β_{O}-triangle inequality, has a cost bounded from above by $\textsc{Opt}_{G_{\mathsf{O}}}$. Since, by Lemma 3, $\textsc{Opt}_{G_{\mathsf{O}}}$ is bounded from above by $\left(1 + \left(\frac{2\beta_{\max}}{1-\beta_{\max}} - 1\right)\frac{1}{n}\right)\textsc{Opt}_{G_{\mathsf{M}}}$, we have

$$c(C) \leq \left(1 + \left(\frac{2\beta_{\max}}{1-\beta_{\max}} - 1\right)\frac{1}{n}\right)\textsc{Opt}_{G_{\mathsf{M}}}. \qquad \square$$

Notice that for the $\Delta_{\beta<1}$-TSP$^+$, the above result holds true also if one uses, instead of the insertion operation, a similar one that selects the edge to be removed from the original cycle according to the *cheapest insertion* rule, since

Algorithm 1. $\Delta_{\beta<1}$-TSP$^{+/-}$

```
1 For the given input instance compute $\beta_{\max}$ and $n$;
2 By Theorem 1, compute the approximation $\alpha$ which can be guaranteed by
  simply returning the cycle $C$ obtained by using the insertion (resp., deletion)
  operation;
3 if $\alpha \leq \varepsilon$ then
4     output $C$.
5 else
6     Perform exhaustive search for an optimal solution.
7 end
```

the cost of the cycle constructed by this operation is not greater than that of the cycle constructed by our insertion operation.

The algorithms for the $\Delta_{\beta<1}$-TSP$^+$ and for the $\Delta_{\beta<1}$-TSP$^-$ obtained from Theorem 1 can be easily adapted to provide a $(1+\varepsilon)$-approximation guarantee, for a parameter $\varepsilon > 0$ (see Algorithm 1).

Theorem 2. *Algorithm 1 is a* PTAS.

Proof. Given an instance of $\Delta_{\beta<1}$-TSP$^+$ (resp., $\Delta_{\beta<1}$-TSP$^-$) with $n = |V(G_{\mathsf{O}})|$ (resp., $n = |V(G_{\mathsf{M}})|$), one can compute

$$c = 1 + \left(\frac{2\beta_{\max}}{1-\beta_{\max}} - 1\right)$$

in $O(n^3)$ time. Then, for any $\varepsilon > 0$, we proceed as follows:

1. If $n \geq \frac{c}{\varepsilon}$, we know from Theorem 1 that returning the cycle C obtained by using the insertion (resp., deletion) operation guarantees an approximation ratio of $1 + \frac{c}{n} \leq 1 + \varepsilon$, and thus this case costs $O(1)$ time;
2. otherwise, we perform an exhaustive search for a new optimal solution, and this costs $O(n!) = O(n^{\frac{c}{\varepsilon}+1})$ time.

Since Case 2 happens for finitely many inputs only, the claim follows. □

3 Reoptimizing the Relaxed Metric TSPPon Single Edge Weight Increases

In this section we study the reoptimization version of $\Delta_{\beta>1}$-TSPP subject to a (transient) single edge weight increase. The best current approximation algorithm for the $\Delta_{\beta>1}$-TSPP is the PMCA-TSPP [25] algorithm, which has an approximation ratio of $\frac{5}{3}\beta^2$. We analyze the impact of an edge weight increase in an instance for which we have an optimal solution, trying to exploit the relaxed metric properties of the instance. In the following, for a given graph G, let $\overline{P}$ denote a minimum-cost Hamiltonian path in G with predetermined endpoints s and t, and let $\textsc{Opt}_G$ be the cost of $\overline{P}$.

Now we can give a formal definition of our problem:

Definition 5. *The* edge-increased $\Delta_{\beta>1}$-TSPP *(*$\mathtt{ei}-\Delta_{\beta>1}$-TSPP *for short) is defined as follows: Given*

- *two complete non-negatively real weighted undirected graphs* $G_{\mathsf{O}} = (V, E, c_{\mathsf{O}})$ *and* $G_{\mathsf{M}} = (V, E, c_{\mathsf{M}})$ *such that* G_{O} *and* G_{M} *obey the triangle inequality up to a relaxation factor of* β_{O} *and* β_{M}*, respectively, with* $\beta_{\mathsf{O}}, \beta_{\mathsf{M}} \geq 1$*, and such that* c_{O} *and* c_{M} *coincide, except that, for an edge* e_i*, we have* $c_{\mathsf{O}}(e_i) < c_{\mathsf{M}}(e_i)$*;*
- *two predetermined vertices* s *and* t*;*
- *a Hamiltonian path* $\overline{P}$ *of* G_{O} *between* s *and* t *such that* $\sum_{e \in \overline{P}} c_{\mathsf{O}}(e) = \textsc{Opt}_{G_{\mathsf{O}}}$*;*

find a Hamiltonian path P *of* G_{M} *between* s *and* t *such that* $\sum_{e \in P} c_{\mathsf{M}}(e) = \textsc{Opt}_{G_{\mathsf{M}}}$*.*

In the following, we describe our approximation algorithm for the $\mathtt{ei}$-$\Delta_{\beta>1}$-TSPP. Let $e = (i, j)$ be the edge of G whose weight is increased (i.e., $c_{\mathsf{O}}(e) < c_{\mathsf{M}}(e)$). If $e \notin \overline{P}$ then, trivially, $\overline{P}$ is an optimal path in G_{M}. Thus we assume in the following $e \in \overline{P}$. Also it is worth noting that, if the number of nodes of the graph is at most 3, then the edge $e = (i, j)$ is always an edge of the optimal solution of the modified instance, and therefore $\textsc{Opt}_{G_{\mathsf{M}}} = \textsc{Opt}_{G_{\mathsf{O}}} - c_{\mathsf{O}}(e) + c_{\mathsf{M}}(e)$. Consequently, w.l.o.g., we assume that the instances have at least 4 nodes.

The idea is simple: if an edge $e = (i, j)$ of the optimal path $\overline{P}$ has increased its weight by a quantity c, either we keep $\overline{P}$ as a solution or we find an alternative s-t path that does not include e. The algorithm, in the general case, finds two alternative subpaths P_i and P_j of $\overline{P}$ (see Fig. 1), and then selects the cheaper of the two, say P'. Let P^* be the unique subpath of $\overline{P}$ that has the same endpoints as P'. Then the algorithm returns the cheaper (with respect to the increase in the cost of e) between $\overline{P}$ and the alternative path $\widetilde{P} = \overline{P} - P^* + P'$.

Notice that there is only one case in which both $i < 3$ and $j > t - 2$ holds, namely when $i = 2$ and $j = 3$ and the input instance has 4 nodes, but in this case, since $s = 1$ and $t = 4$ are fixed, the edge $e = (2, 3)$ is always an edge of the optimal solution of the modified instance. For all other instances with at least 5 nodes, it is not possible that both $i < 3$ and $j > t - 2$ holds. Indeed, if we consider the extreme case in which the instance has exactly 5 nodes we have the following:

- The vertices of $\overline{P}$ are numbered in non-decreasing order from 1 to 5, where $s = 1$ and $t = 5$ are the endpoints of $\overline{P}$;
- $e = (i, j)$, with $i < j$, is the edge of $\overline{P}$ whose weight is increased;
- thus, since $e \in \overline{P}$ and the vertices of $\overline{P}$ are numbered in nondecreasing order, then $i = j - 1$ and therefore it is not possible that $i < 3$ and $j > t - 2 = 3$.

This ensures that if L is empty after line 12, the algorithm returns $\overline{P}$ which is optimal for the modified instance, otherwise L is not empty.

We are now ready to give our main result, that is we show that Algorithm 2 guarantees an approximation of $1 + 2\beta_{\mathsf{L}}$ for the $\mathtt{ei}$-$\Delta_{\beta>1}$-TSPP, which outperforms the $\frac{5}{3}\beta^2$ approximation ratio of the PMCA-TSPP algorithm for $\beta \geq \frac{3+2\sqrt{6}}{5} \approx 1.58$.

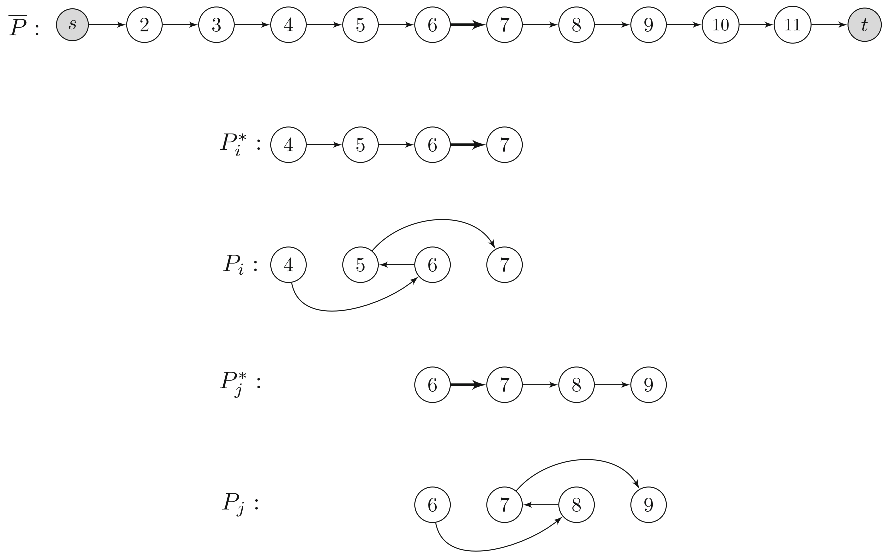


Fig. 1. From the top to the bottom, we have: $\overline{P}$ with the vertices numbered in non decreasing order and with $s = 1, t = 12$, the bold edge $e = (6, 7)$ is the edge whose weight is increased; P_i^* is the unique path in $\overline{P}$ having as endpoints $i - 2 = 4$ and $j = 7$; P_i is a path having the same endpoints as P_i^* and not including $e = (6, 7)$; P_j^* is the unique path in $\overline{P}$ having as endpoints $i = 6$ and $j + 2 = 9$; P_j is a path having the same endpoints as P_j^* and not including $e = (6, 7)$

Theorem 3. *Let* $(G_{\mathsf{O}}, G_{\mathsf{M}}, \overline{P})$ *be an input instance of* $\texttt{ei} - \Delta_{\beta>1}$-TSPP *such that* G_{O} *and* G_{M} *obey the triangle inequality up to, respectively,* β_{O} *and* β_{M}*, with* $\beta_{\mathsf{O}}, \beta_{\mathsf{M}} \geq 1$*. Let* G_{O} *and* G_{M} *agree except for an edge* e *such that* $c_{\mathsf{M}}(e) > c_{\mathsf{O}}(e)$*. Let* $\beta_{\mathsf{L}} = \min\{\beta_{\mathsf{O}}, \beta_{\mathsf{M}}\}$ *and let* $s, t \in V(G_{\mathsf{O}})$ *be the endpoints of* $\overline{P}$*. Then, Algorithm 2 provides a* $(1 + 2\beta_{\mathsf{L}})$*-approximation with respect to the cost of* P*.*

Proof. It is easy to verify that, if the number of nodes of the graph is 4, our algorithm always returns an optimal solution, so, w.l.o.g., we assume that the number of nodes is at least 5 and L is not empty. Let P' and P^* be the paths found by Algorithm 2 at lines 17 and 18, respectively. Since $e = (i, j) \notin P'$, we have $c_{\mathsf{O}}(P') = c_{\mathsf{M}}(P')$, while, on the other hand, since $e \in P^*$ and $c_{\mathsf{M}}(e) > c_{\mathsf{O}}(e)$, we have $c_{\mathsf{O}}(P^*) < c_{\mathsf{M}}(P^*)$. Now we show that

$$c_{\mathsf{M}}(P') - c_{\mathsf{O}}(P^*) \leq 2\beta_{\mathsf{O}}\textsc{Opt}_{G_{\mathsf{O}}} \tag{4}$$

and

$$c_{\mathsf{M}}(P') - c_{\mathsf{O}}(P^*) \leq 2\beta_{\mathsf{M}}\textsc{Opt}_{G_{\mathsf{M}}}. \tag{5}$$

By definition, we have

$$c_{\mathsf{O}}(P_i) = c_{\mathsf{O}}(i-2, i) + c_{\mathsf{O}}(i, i-1) + c_{\mathsf{O}}(i-1, j)$$

Algorithm 2. ei-$\Delta_{\beta>1}$-TSPP

```
1  Let $s$ and $t$ be the endpoints of $\overline{P}$;
2  Number the vertices of $\overline{P}$ in non decreasing order from 1 to $n$, with $s = 1$ and
   $t = n$ and let $e = (i, j)$, with $i < j$, be the edge whose weight is increased;
3  Let $L$ be a list of paths, initially empty;
4  if $i \geq 3$ then
5      Let $P_i^*$ be the unique path in $\overline{P}$ having as endpoints $i - 2$ and $j$;
6      Let $P_i = (i-2, i), (i, i-1), (i-1, j)$ be a path having the same endpoints
       as $P_i^*$;
7      Add $P_i$ to $L$ and set key($P_i$) to the value of $c(P_i)$;
8  end
9  if $j \leq t - 2$ then
10     Let $P_j^*$ be the unique path in $\overline{P}$ having as endpoints $i$ and $j + 2$;
11     Let $P_j = (i, j+1), (j+1, j), (j, j+2)$ be a path having the same endpoints
       as $P_j^*$;
12     Add $P_j$ to $L$ and set key($P_j$) to the value of $c(P_j)$;
13 end
14 if $L$ is empty then
15     output $\overline{P}$.
16 else
17     Let $P'$ be the path in $L$ having minimum cost;
18     Let $P^*$ be the unique subpath of $\overline{P}$ having the same endpoints as $P'$;
19     if $c_{\mathsf{M}}(P') \geq c_{\mathsf{M}}(P^*)$ then
20         output $\overline{P}$.
21     else
22         output $\widetilde{P} = \overline{P} - P^* + P'$.
23     end
24 end
```

and

$$c_{\mathsf{O}}(P_i^*) = c_{\mathsf{O}}(i-2, i-1) + c_{\mathsf{O}}(i-1, i) + c_{\mathsf{O}}(i, j).$$

Hence

$$c_{\mathsf{O}}(P_i) - c_{\mathsf{O}}(P_i^*) = c_{\mathsf{O}}(i-2, i) + c_{\mathsf{O}}(i-1, j) - c_{\mathsf{O}}(i-2, i-1) - c_{\mathsf{O}}(i, j). \tag{6}$$

Moreover, since the β-triangle inequality holds, we have that

$$c_{\mathsf{O}}(i-2, i) \leq \beta_{\mathsf{O}} \left(c_{\mathsf{O}}(i-2, i-1) + c_{\mathsf{O}}(i-1, i) \right)$$

and

$$c_{\mathsf{O}}(i-1, j) \leq \beta_{\mathsf{O}} \left(c_{\mathsf{O}}(i-1, i) + c_{\mathsf{O}}(i, j) \right).$$

Hence

$$c_{\mathsf{O}}(i-2, i) + c_{\mathsf{O}}(i-1, j) \leq 2\beta_{\mathsf{O}} c_{\mathsf{O}}(P_i^*), \tag{7}$$

and from (6) and (7) we have

$$c_{\mathsf{M}}(P_i) - c_{\mathsf{O}}(P_i^*) = c_{\mathsf{O}}(P_i) - c_{\mathsf{O}}(P_i^*) \leq 2\beta_{\mathsf{O}} c_{\mathsf{O}}(P_i^*) \leq 2\beta_{\mathsf{O}} \textsc{Opt}_{G_{\mathsf{O}}}. \tag{8}$$

Moreover we have that

$$c_{\mathsf{O}}(i-2,i) = c_{\mathsf{M}}(i-2,i) \leq \beta_{\mathsf{M}} \left(c_{\mathsf{M}}(i-2,i-1) + c_{\mathsf{M}}(i-1,i)\right)$$

and

$$c_{\mathsf{O}}(i-1,j) = c_{\mathsf{M}}(i-1,j) \leq \beta_{\mathsf{M}} \left(c_{\mathsf{M}}(i-1,i) + c_{\mathsf{M}}(i,j)\right).$$

Therefore

$$c_{\mathsf{M}}(i-2,i) + c_{\mathsf{M}}(i-1,j) \leq 2\beta_{\mathsf{M}} c_{\mathsf{M}}(P_i^*), \tag{9}$$

and from (6) and (9) we have

$$c_{\mathsf{M}}(P_i) - c_{\mathsf{O}}(P_i^*) \leq 2\beta_{\mathsf{M}} c_{\mathsf{M}}(P_i^*) \leq 2\beta_{\mathsf{M}} \textsc{Opt}_{G_{\mathsf{M}}}. \tag{10}$$

With a similar argument, one can prove that

$$c_{\mathsf{M}}(P_j) - c_{\mathsf{O}}(P_j^*) \leq 2\beta_{\mathsf{O}} c_{\mathsf{O}}(P_j^*) \leq 2\beta_{\mathsf{O}} \textsc{Opt}_{G_{\mathsf{O}}}$$

and

$$c_{\mathsf{M}}(P_j) - c_{\mathsf{O}}(P_j^*) \leq 2\beta_{\mathsf{M}} c_{\mathsf{M}}(P_j^*) \leq 2\beta_{\mathsf{M}} \textsc{Opt}_{G_{\mathsf{M}}}.$$

Hence (4) and (5) hold true. The rest is proven by a case distinction.

Case 1: If $c_{\mathsf{M}}(P^*) \leq c_{\mathsf{M}}(P')$, we return $\overline{P}$. Indeed:

$$\begin{aligned}
\frac{c_{\mathsf{M}}(\overline{P})}{\textsc{Opt}_{G_{\mathsf{M}}}} &= \frac{\textsc{Opt}_{G_{\mathsf{O}}} - c_{\mathsf{O}}(e) + c_{\mathsf{M}}(e)}{\textsc{Opt}_{G_{\mathsf{M}}}} \\
&= \frac{\textsc{Opt}_{G_{\mathsf{O}}} - c_{\mathsf{O}}(P^*) + c_{\mathsf{M}}(P^*)}{\textsc{Opt}_{G_{\mathsf{M}}}} \\
&\leq \frac{\textsc{Opt}_{G_{\mathsf{O}}} - c_{\mathsf{O}}(P^*) + c_{\mathsf{M}}(P')}{\textsc{Opt}_{G_{\mathsf{M}}}} \\
&\overset{(4)}{\leq} \frac{\textsc{Opt}_{G_{\mathsf{O}}} + 2\beta_{\mathsf{O}} \textsc{Opt}_{G_{\mathsf{O}}}}{\textsc{Opt}_{G_{\mathsf{O}}}} \\
&= 1 + 2\beta_{\mathsf{O}}.
\end{aligned} \tag{11}$$

But we also have

$$\begin{aligned}
\frac{c_{\mathsf{M}}(\overline{P})}{\textsc{Opt}_{G_{\mathsf{M}}}} &= \frac{\textsc{Opt}_{G_{\mathsf{O}}} - c_{\mathsf{O}}(e) + c_{\mathsf{M}}(e)}{\textsc{Opt}_{G_{\mathsf{M}}}} \\
&= \frac{\textsc{Opt}_{G_{\mathsf{O}}} - c_{\mathsf{O}}(P^*) + c_{\mathsf{M}}(P^*)}{\textsc{Opt}_{G_{\mathsf{M}}}} \\
&\leq \frac{\textsc{Opt}_{G_{\mathsf{O}}} - c_{\mathsf{O}}(P^*) + c_{\mathsf{M}}(P')}{\textsc{Opt}_{G_{\mathsf{M}}}} \\
&\overset{(5)}{\leq} \frac{\textsc{Opt}_{G_{\mathsf{M}}} + 2\beta_{\mathsf{M}} \textsc{Opt}_{G_{\mathsf{M}}}}{\textsc{Opt}_{G_{\mathsf{M}}}} \\
&= 1 + 2\beta_{\mathsf{M}}.
\end{aligned} \tag{12}$$

Thus, the combination of (11) and (12) yields

$$\frac{c_{\mathsf{M}}(\overline{P})}{\textsc{Opt}_{G_{\mathsf{M}}}} \leq 1 + 2\beta_{\mathsf{L}}. \tag{13}$$

Case 2: otherwise, if $c_{\mathsf{M}}(P^*) > c_{\mathsf{M}}(P')$, we return $\widetilde{P} = \overline{P} - P^* + P'$. Indeed:

$$\begin{aligned}\frac{c_{\mathsf{M}}(\widetilde{P})}{\textsc{Opt}_{G_{\mathsf{M}}}} &\leq \frac{\textsc{Opt}_{G_{\mathsf{O}}} - c_{\mathsf{O}}(P^*) + c_{\mathsf{M}}(P')}{\textsc{Opt}_{G_{\mathsf{M}}}} \\ &\overset{(4)}{\leq} \frac{\textsc{Opt}_{G_{\mathsf{O}}} + 2\beta_{\mathsf{O}}\textsc{Opt}_{G_{\mathsf{O}}}}{\textsc{Opt}_{G_{\mathsf{O}}}} \\ &= 1 + 2\beta_{\mathsf{O}}. \end{aligned} \tag{14}$$

But we also have

$$\begin{aligned}\frac{c_{\mathsf{M}}(\widetilde{P})}{\textsc{Opt}_{G_{\mathsf{M}}}} &\leq \frac{\textsc{Opt}_{G_{\mathsf{O}}} - c_{\mathsf{O}}(P^*) + c_{\mathsf{M}}(P')}{\textsc{Opt}_{G_{\mathsf{M}}}} \\ &\overset{(5)}{\leq} \frac{\textsc{Opt}_{G_{\mathsf{M}}} + 2\beta_{\mathsf{M}}\textsc{Opt}_{G_{\mathsf{M}}}}{\textsc{Opt}_{G_{\mathsf{M}}}} \\ &= 1 + 2\beta_{\mathsf{M}}. \end{aligned} \tag{15}$$

Thus, the combination of (14) and (15) yields

$$\frac{c_{\mathsf{M}}(\widetilde{P})}{\textsc{Opt}_{G_{\mathsf{M}}}} \leq 1 + 2\beta_{\mathsf{L}}, \tag{16}$$

and so the claim follows by (13) and (16). □

4 Conclusion

In this paper, we reviewed the influential concept of stability of approximation, introduced by Hromkovič [26,27], that has inspired many advances in the investigation of hard optimization problems. In turn, we used this idea to study two reoptimization variants of β-metric TSP and TSPPfor which no results were known in the literature. We obtained a PTAS for the reoptimization of the strengthened metric TSP subject to a single node modification, and a constant approximation algorithm for the reoptimization of the relaxed metric TSPPsubject to a single edge weight increase.

Although there are already many applications of stability of approximation, we think that this concept can still bring many interesting results. For example, we envision possible new results on TSPP, by combining such a powerful idea with recent developments in the study of this problem on metric input instances [29,30]. Furthermore, another interesting direction is that of analyzing under the stability perspective the *asymmetric* metric TSP, for which a recent breakthrough showed the existence of a constant-factor approximation algorithm [31].

Dedication

The authors would like to thank Juraj Hromkovič for having inspired their work on the topic discussed in this paper, as well as on several other research areas of interest.

References

1. Andreae, T.: On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality. Netw. Int. J. **38**(2), 59–67 (2001)
2. Andreae, T., Bandelt, H.-J.: Performance guarantees for approximation algorithms depending on parametrized triangle inequalities. SIAM J. Discrete Math. **8**(1), 1–16 (1995)
3. Archetti, C., Bertazzi, L., Speranza, M.G.: Reoptimizing the traveling salesman problem. Networks **42**(3), 154–159 (2003)
4. Ausiello, G., Escoffier, B., Monnot, J., Paschos, V.T.: Reoptimization of minimum and maximum traveling salesman's tours. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 196–207. Springer, Heidelberg (2006). https://doi.org/10.1007/11785293_20
5. Bender, M.A., Chekuri, C.: Performance guarantees for the TSP with a parameterized triangle inequality. In: Dehne, F., Sack, J.-R., Gupta, A., Tamassia, R. (eds.) WADS 1999. LNCS, vol. 1663, pp. 80–85. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48447-7_10
6. Berg, T., Hempel, H.: Reoptimization of traveling salesperson problems: changing single edge-weights. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 141–151. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00982-2_12
7. Bilò, D., Forlizzi, L., Proietti, G.: Approximating the metric TSP in linear time. Theory Comput. Syst. **49**(3), 615–631 (2011)
8. Bilò, D., Zych, A.: New advances in reoptimizing the minimum Steiner tree problem. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 184–197. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32589-2_19
9. Bilò, D., et al.: Reoptimization of Steiner trees. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 258–269. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69903-3_24
10. Böckenhauer, H.-J., Hromkovič, J., Kneis, J., Kupke, J.: On the approximation hardness of some generalizations of TSP. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 184–195. Springer, Heidelberg (2006). https://doi.org/10.1007/11785293_19
11. Böckenhauer, H.-J., et al.: On the hardness of constructing minimal 2-connected spanning subgraphs in complete graphs with sharpened triangle inequality. Theor. Comput. Sci. **326**(1–3), 137–153 (2004)
12. Böckenhauer, H.-J., et al.: On k-connectivity problems with sharpened triangle inequality. J. Discrete Algorithms **6**(4), 605–617 (2008)
13. Böckenhauer, H.-J., et al.: On the approximability of TSP on local modifications of optimally solved instances. Algorith. Oper. Res. **2**, 83–93 (2007)
14. Böckenhauer, H.-J., Freiermuth, K., Hromkovič, J., Mömke, T., Sprock, A., Steffen, B.: Steiner tree reoptimization in graphs with sharpened triangle inequality. J. Discrete Algorithms **11**, 73–86 (2012)

15. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: Approximation algorithms for the TSP with sharpened triangle inequality. Inf. Process. Lett. **75**, 133–138 (2000)
16. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: An improved lower bound on the approximability of metric TSP and approximation algorithms for the TSP with sharpened triangle inequality. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 382–394. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46541-3_32
17. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem. Theor. Comput. Sci. **285**(1), 3–24 (2002)
18. Böckenhauer, H.-J., Hromkovič, J., Kneis, J., Kupke, J.: The parameterized approximability of TSP with deadlines. Theory Comput. Syst. **41**(3), 431–444 (2007)
19. Böckenhauer, H.-J., Hromkovič, J., Královic, R., Mömke, T., Rossmanith, P.: Reoptimization of Steiner trees: changing the terminal set. Theor. Comput. Sci. **410**(36), 3428–3435 (2009)
20. Böckenhauer, H.-J., Kneis, J., Kupke, J.: Approximation hardness of deadline-TSP reoptimization. Theor. Comput. Sci. **410**(21–23), 2241–2249 (2009)
21. Böckenhauer, H.-J., Komm, D.: Reoptimization of the metric deadline TSP. J. Discrete Algorithms **8**(1), 87–100 (2010)
22. Böckenhauer, H.-J., Seibert, S.: Improved lower bounds on the approximability of the traveling salesman problem. ITA **34**(3), 213–255 (2000)
23. Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegy-Mellon University (1976)
24. D'Andrea, A., Proietti, G.: Reoptimizing the strengthened metric TSP on multiple edge weight modifications. In: Klasing, R. (ed.) SEA 2012. LNCS, vol. 7276, pp. 111–122. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30850-5_11
25. Forlizzi, L., Hromkovič, J., Proietti, G., Seibert, S.: On the stability of approximation for Hamiltonian path problems. Algorithm. Oper. Res. **1**(1), 31–45 (2006)
26. Hromkovič, J.: Stability of approximation algorithms and the knapsack problem. In: Karhumäki, J., Maurer, H., Păun, G., Rozenberg, G. (eds.) Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, pp. 238–249. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-642-60207-8_21
27. Hromkovič, J.: Stability of approximation algorithms for hard optimization problems. In: Pavelka, J., Tel, G., Bartošek, M. (eds.) SOFSEM 1999. LNCS, vol. 1725, pp. 29–47. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-47849-3_2
28. Rosenkrantz, D.J., Stearns, R.E., Lewis II, P.M.: An analysis of several heuristics for the traveling salesman problem. SIAM J. Comput. **6**(3), 563–581 (1977)
29. Sebö, A.: Eight-fifth approximation for the path TSP. In: Goemans, M., Correa, J. (eds.) IPCO 2013. LNCS, vol. 7801, pp. 362–374. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36694-9_31
30. Sebö, A., van Zuylen, A.: The salesman's improved paths: A 3/2+1/34 approximation. In: IEEE 57th Annual Symposium on Foundations of Computer Science. FOCS 2016, 9–11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA, pp. 118–127 (2016)

31. Svensson, O., Tarnawski, J., Végh, L.A.: A constant-factor approximation algorithm for the asymmetric traveling salesman problem. CoRR, abs/1708.04215 (2017)
32. Vygen, J.: New approximation algorithms for the TSP. OPTIMA **90**, 1–12 (2012)

Fully Online Matching with Advice on General Bipartite Graphs and Paths

Hans-Joachim Böckenhauer[1(✉)], Lucia Di Caro[2], and Walter Unger[3]

[1] Department of Computer Science, ETH Zurich, Zürich, Switzerland
hjb@inf.ethz.ch

[2] University of Applied Sciences and Arts Northwestern Switzerland FHNW, Brugg-Windisch, Switzerland
lucia.dicaro@fhnw.ch

[3] Department of Computer Science, RWTH Aachen University, Aachen, Germany
walter.unger@algo.rwth-aachen.de

Abstract. We consider the problem of finding a maximum-size matching in a fully online setting, where the vertices of a bipartite graph are given one after the other in discrete time steps, together with all edges to previously revealed vertices. In each time step, an online algorithm has to irrevocably decide which of the newly presented edges is going to be part of the matching.

It is known that this problem admits a 2-competitive online algorithm in the case of general graphs. We prove a tight lower bound that even holds for bipartite graphs, and tight bounds of $\frac{3}{2}$ for paths.

In the main part of the paper, we consider the model of advice complexity which contributes to a more fine-grained complexity analysis. Within this model, one asks how much information about the yet missing parts of the input is needed to compute an optimal or near-optimal solution. We prove almost tight linear upper and lower bounds on the amount of advice that is necessary for computing an optimal solution for paths. We complement these results by bounding the sufficient amount of advice for computing sub-optimal matchings. We furthermore prove that a single bit of advice does not help to improve over deterministic algorithms.

1 Introduction

Finding a maximum-size matching in a graph is one of the fundamental algorithmic tasks in graph theory with numerous practical applications. The classical offline version of the problem, where the whole graph is known in advance, can be solved in polynomial time (see, e.g., [19,29,30]). This problem has been widely investigated, an overview of matching theory can be found in [31].

In contrast to this, the online version of the problem is hard even for bipartite graphs. Most attention has been paid to the so-called one-sided matching in bipartite graphs, where the vertices from one shore of the bipartite graph are given beforehand, and the vertices from the other shore appear one after the other in an online fashion [26]. An online algorithm then has to irrevocably decide which of the edges incident to the newly presented vertex, if any, will be included in the matching. The performance of an online algorithm is classically analyzed using the *competitive ratio*, i.e., by comparing the cost of the computed solution to the optimal cost of a solution computed by an exact offline algorithm knowing the complete instance beforehand. This *competitive analysis* was introduced by Sleator and Tarjan [33], for an introduction, see the textbooks by Komm [27] or Borodin and El-Yaniv [12]. For the one-sided matching problem, Karp et al. [26] have shown that a simple greedy strategy reaches a competitive ratio of 2, i.e., that this strategy computes a matching of at least half the size of a maximum matching, and they have further proven that this is the best possible for any deterministic online algorithm.

In this paper, we consider the *fully online* version of the matching problem, where all vertices appear online. More precisely, in each time step, one vertex of the graph appears, together with all edges incident to already presented vertices. Thus, the problem is not limited to bipartite graphs. This version of the problem is less studied, but also very well motivated. Consider, for example, the work of a job center that receives job offers daily from companies and, independent from these offers, job applications of candidates with a certain profile. Clearly, the job center cannot wait infinitely long to find the best candidate for an open position.

Not much is known about the fully online matching problem besides a greedy algorithm by Favrholdt and Vatshelle [21] achieving a competitive ratio of 2 on general graphs and of 3/2 on paths. We show in this paper that this is the best possible ratio, even if we restrict the set of inputs to bipartite graphs. This result points to how coarse-grained classical competitive analysis can be. In terms of competitive ratio, the fully online matching problem is as hard on bipartite graphs as it is on general graphs.

As a means for a more fine-grained analysis of online algorithms, we use the concept of advice complexity of online problems. The idea behind this approach is to measure the information content of an online problem, i.e., the amount of information about the yet unknown parts of the input that is needed to compute an optimal solution or a solution with a certain competitive ratio. The advice complexity was introduced by Dobrev et al. [17] and refined by Emek et al. [20] and by Böckenhauer et al. [10,11]; see also Hromkovič et al. [25]. It has been successfully used for many online problems since then, including the k-server problem [8,9,16,20,24], the knapsack problem [6,7], bin packing [14], disjoint path allocation [1,10,11,23], set cover [18,28], coloring problems [2,3,15,22,32], and many others. For a comprehensive overview, see the textbook by Komm [27] and a recent survey by Boyar et al. [13].

The paper is organized as follows. In Sect. 2, we fix our notation and give a formal definition of online algorithms with advice and the fully online bipartite matching problem. We further review the known results on online computation without advice. In Sect. 3, we consider lower and upper bounds on the advice complexity for computing optimal solutions in general and bipartite graphs. In Sects. 4 and 5, we deal with restricting the problem to online paths. We consider online algorithms with advice computing optimal solutions in Sect. 4

and approximate solutions in Sect. 5, where we in particular prove the same lower bound for algorithms using a single advice bit as for those working without advice. In Sect. 6, we conclude the paper.

2 Preliminaries and Algorithms Without Advice

In this section, we formally define the notions used in the following. All logarithms in this paper are taken to be binary, unless stated otherwise.

Definition 1 (Online Maximization Problem). *An online maximization problem consists of a set $\mathcal{I}$ of inputs and a gain function. Every input $I \in \mathcal{I}$ is a sequence of* requests *$I = (x_1, \ldots, x_n)$. Furthermore, a set of feasible outputs (or solutions) is associated with every I; every output is a sequence of* answers *$O = (y_1, \ldots, y_n)$. The gain function assigns a positive real value* $\text{gain}(I, O)$ *to every input I and any feasible output O. If the input is clear from the context, we omit I and denote the gain of O as* $\text{gain}(O)$. *For every input I, we call any output O that is feasible for I and has largest possible gain an* optimal solution *of I, denoted by $Opt(I)$.*

We now formally define online algorithms with advice for online maximization problems, and their competitive ratios.

Definition 2 (Online Algorithm with Advice). *Consider an input I of an online maximization problem. An* online algorithm *A with advice computes the output sequence $A^{\phi}(I) = (y_1, \ldots, y_n)$ such that y_i is computed from $\phi, x_1, \ldots, x_i$, where ϕ is the content of the advice tape, i.e., an infinite binary sequence. We denote the gains of the computed output by* $\text{gain}(A^{\phi}(I))$. *The algorithm A is c-*competitive with advice complexity *$b(n)$ if there exists a constant α such that, for every n and for each I of length at most n, there exists some ϕ such that* $\text{gain}(A^{\phi}(I)) \geq \frac{1}{c} \cdot \text{gain}(\text{Opt}(I)) - \alpha$ *and at most the first $b(n)$ bits of ϕ have been accessed during the computation of $A^{\phi}(I)$. If A is c-competitive for $\alpha = 0$, we call it* strictly c-competitive.

Definition 2 follows the advice complexity model from Böckenhauer et al. [10,11]. A motivation and discussion of this model is found in [25].

We use the standard notions for dealing with graphs in this paper; an introduction to graph theory can, e.g., be found in [34]. Let $G = (V, E)$ be a graph. A subset $M \subseteq E$ is called a *matching* if the edges in M are pairwise not adjacent, i.e., if no two edges have a common end vertex. The *size of a matching* is measured by the number of edges in the matching. We call a vertex v *matched* in a matching M if there is an edge $e \in M$ such that v is incident to e. A *maximum matching* is a matching of maximum cardinality.

An *online graph instance* $G^{\prec} = (G, \prec)$ consists of a graph $G = (V, E)$ and a linear ordering $\prec$ on the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ with $v_i \prec v_j$ for $i < j$. In the online presentation $G^{\prec}$ of the graph G, the vertices of V appear in the order determined by $\prec$, together with the edges adjacent to already present vertices. Let $V_i = \{v_1, \ldots, v_i\}$. Then we denote by $G^{\prec}[V_i]$ the online subgraph of $G^{\prec}$ induced by V_i.

Definition 3 (Fully Online Matching Problem). *The* fully online matching problem *is the following online maximization problem: Given an online graph $G^{\prec} = (G, \prec)$ with $G = (V, E)$ and $V = \{v_1, v_2, \ldots, v_n\}$, the goal is to compute a sequence $(M_1, M_2, \ldots, M_n)$ such that $M_i \subseteq E$ is a matching on $G^{\prec}[V_i]$ and $M_i \subseteq M_j$ for all $i < j$ with $i \in \{1, 2, \ldots, n-1\}$ and $j \in \{2, 3, \ldots, n\}$ such as to maximize the size $|M_n|$ of the computed matching for $G^{\prec}$.*

We will use the following notation in the figures throughout this paper: a matching edge will be drawn as a wavy line, whereas all other edges are drawn as straight lines.

The following results are known for the classical online setting without advice. The lower bound of 2 on the competitive ratio for the one-sided online matching problem in general graphs from Karp et al. [26] carries over to the fully online matching problem. In [21], Favrholdt and Vatshelle complemented this result with a tight upper bound for the fully online matching problem on general graphs.

In [21], the authors moreover showed a matching lower and upper bound of $\frac{3}{2}$ on the competitive ratio of the fully online matching problem on paths. The upper bound can be achieved by a simple greedy strategy that takes every presented edge into the matching as long as both endpoints are not yet matched.

3 Optimality on General and Bipartite Graphs

In this section, we investigate how much advice is necessary and sufficient to compute an optimal matching in a general or bipartite graph. An upper bound for general graphs is easy to see: A vertex in a general online graph on n vertices has at most $n-1$ neighbors, i.e., it has a vertex degree of at most $n-1$. For every vertex, at most one incident edge can be a matching edge. Therefore, if an algorithm asks, for every of the n vertices, which of the neighboring edges should be a matching edge, it finds a maximum matching.

Theorem 1. *There exists an online algorithm with advice that reads at most $n \cdot (\log(n) - \log(\mathrm{e}) + 1)$ advice bits in order to find a maximum matching in a general online graph $G_n^{\prec}$ on n vertices.*

Proof. For each presented vertex v_k, except for the first one, the algorithm reads advice telling it which of the edges to the already presented vertices $v_1, \ldots, v_{k-1}$ should be included in the matching, if any. This advice can be encoded by some number from $\{0, \ldots, k-1\}$, where 0 stands for not choosing any edge in this time step. No self-delimiting is needed for encoding these numbers in binary since the algorithm knows the number of the current time step. Thus, the number of advice bits can be bounded by

$$\sum_{k=1}^{n} \lceil \log(k) \rceil \leq n + \sum_{k=1}^{n} \log(k) = n + \log(n!) \leq n + n(\log n - \log(\mathrm{e})). \qquad \square$$

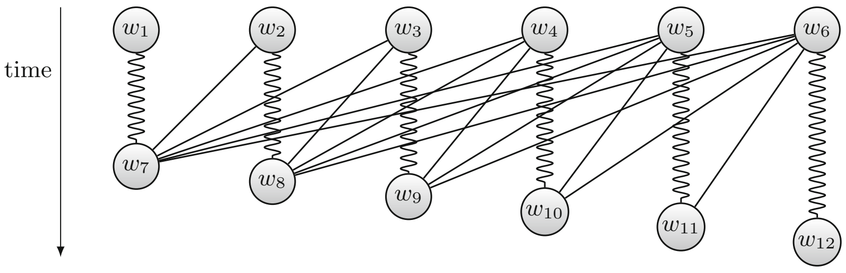


Fig. 1. Pattern of the online bipartite graphs used in the proof of Theorem 2. Here, we have $n = 12$. The adversary shows first a permutation of the vertices $w_1, w_2, \ldots, w_6$. Then, the vertices $w_7, w_8, \ldots, w_{12}$ are given in this order.

Although this algorithm is very simple, we will see in the following that no essentially better online algorithm with advice is possible, not even for bipartite graphs. The following lower bound is of the same order of magnitude $\Theta(n \log(n))$ as the upper bound from Theorem 1.

Theorem 2. *Any deterministic online algorithm for the problem of finding a maximum matching in an online bipartite graph instance with n vertices needs to read at least $\log\left(\left(\frac{n}{2}\right)!\right) \in \Theta(n\log(n))$ advice bits in order to find a maximum matching.*

Proof. In order to prove the claim, we describe a class of $(n/2)!$ online instances in which each two of the instances need a different advice string.

Let $n = 2k$ with $k \in \mathbb{N}$ be the number of vertices in the instances from this class, containing two shores of size k, $S_1 = \{w_1, w_2, \ldots, w_k\}$ and $S_2 = \{w_k, w_{k+1}, \ldots, w_{2k}\}$; see Fig. 1 for an example.

The edge set E is defined as

$$E = \{\{w_i, w_j\} \mid 1 \le i \le k \text{ and } k+1 \le j \le k+i\},$$

leading to a unique perfect matching in these types of bipartite graphs since the only possibility for all vertices w_{2i} being matched is that they are matched by the edges $\{w_i, w_{2i}\}$, for $1 \le i \le k$.

The online presentation of one of these bipartite graphs starts with any permutation of isolated vertices $w_1, w_2, \ldots, w_k$. Then, the adversary presents the vertices $w_{k+1}, w_{k+2}, \ldots, w_{2k}$ in this order. In every time step i, for any $k+1 \le i \le 2k$, the algorithm encounters the same prefix but it has to choose one of the $2k - i + 1$ edges incident to w_i to be a matching edge, and only one of these possibilities is correct. Therefore, the algorithm has to distinguish

$$\prod_{i=k+1}^{2k} (2k - i + 1) = \prod_{i=1}^{k} i = k! = \left(\frac{n}{2}\right)!$$

different possible online representations of this bipartite graph, leading to

$$\log\left(\left(\frac{n}{2}\right)!\right) \in \Theta(n\log(n))$$

advice bits that every online algorithm with advice needs to read at least in order to be optimal. □

We see that the class of hard instances as used in the proof of Theorem 2 has some nice additional properties. For example, none of these graphs contains an induced path on 5 vertices. Thus, the proof of Theorem 2 also shows that at least $\Omega(n \log(n))$ advice bits are necessary to compute an optimal matching in online P_5-free graphs. Moreover, the graphs from this instance class have a diameter of 3. Thus the same lower bound also holds for online graphs with a diameter bounded by 3.

Although the problem thus appears to be quite hard on very restricted classes of graphs, we will analyze a graph class in the subsequent sections, where less advice is sufficient, namely paths.

4 Optimality on Paths

In this section, we will show how much advice is necessary and sufficient in order to solve the online matching problem on paths optimally. For this, we will mostly focus on matchings which match all inner vertices.

In the online setting of a path P_n, the isolated edges play a special role. Since, at a certain time step i, they are not appended to any other subpath, the algorithm does not get any information from the neighborhood about whether they are matching or non-matching edges. Therefore, intuitively, an online algorithm for solving the online matching problem on a path should get some advice on these edges in order to be optimal. We show that there cannot appear too many isolated edges in an online path instance. By $\mathcal{P}_n$ we denote the set of all online paths on n vertices.

Theorem 3. *There exists an online algorithm with advice for finding a maximum matching in a path $P^{\prec} \in \mathcal{P}_n$ on n vertices which uses at most $\lceil \frac{n}{3} \rceil$ advice bits.*

Proof. If n is even, the maximum matching is unique and matches all inner vertices. If n is odd, there also exists a matching that matches all inner vertices. Algorithm 1 always constructs such a matching. The idea behind this algorithm is as follows. It asks advice for every new vertex if necessary and applies the answer of this bit to the adjacent edges. The intuition is that the algorithm has no information from the neighborhood for *isolated edges* or for an *isolated path of length 2* that arises in time step i by connecting two isolated vertices by a new vertex v_i. Therefore, the advice bits can also be directly associated with the edges. In all other cases, the algorithm uses a greedy strategy.

For the correctness of Algorithm 1, note that, due to the greedy strategy and since the advice is given in such a way as to force the algorithm to choose a matching that does not leave any inner vertex unmatched, a vertex v_i cannot be connected to two non-isolated unmatched vertices v_{j_1} and v_{j_2}. In this case, one of the two inner vertices v_{j_1} and v_{j_2} would remain unmatched. Moreover, v_i

Algorithm 1. Maximum Matching on Paths

```
1  input: $P^{\prec} \in \mathcal{P}_n$, for some $n \in \mathbb{N}$
2  $M = \emptyset$;
3  for $i = 1$ to $n$ do
4      if $v_i$ is connected by exactly one edge $e$ to a previously isolated vertex then
5          read an advice bit $\sigma$ to decide whether $e$ is a matching edge and update
           $M$ accordingly;
6      else if $v_i$ has two edges $e_1$ and $e_2$ to previously isolated vertices then
7          use an advice bit $\sigma$ to decide whether $e_1$ or $e_2$ is a matching edge and
           update $M$ accordingly;
8      else
9          update $M$ greedily, i.e., add each newly revealed edge to the matching if
           possible (with ties broken arbitrarily);
10     end
11 end
12 output: The constructed matching $M$
```

also cannot be connected to two matched non-isolated vertices, since it would remain unmatched in this case.

The number of advice bits used by the algorithm depends on the number of isolated edges or isolated subpaths of length 2. These isolated subpaths need to be separated by at least one vertex in order to ensure that the algorithm has to ask for an advice bit for each of these subpaths. In the worst case, the online path $P^{\prec}$ consists of subpaths of length 1, i.e., isolated edges, and each two consecutive subpaths will be separated by a single vertex. Such a presentation of the online path on n vertices contains $\lceil \frac{n}{3} \rceil$ isolated edges. Thus, $\lceil \frac{n}{3} \rceil$ advice bits suffice for any online path of length n. □

In the following, we prove asymptotically matching lower bounds for the online matching problem on paths.

Theorem 4. *Any deterministic online algorithm for the problem of finding a maximum matching in an online path instance $P^{\prec} \in \mathcal{P}_n$ needs to read at least*

$$\left\lfloor \frac{1}{3}n - \frac{1}{2}\log(n) + \log\left(\sqrt{\frac{3}{2\pi}}\right) \right\rfloor$$

advice bits in order to be optimal.

Proof. We give a proof by contradiction, based on the following idea. We assume that there exists an algorithm $A_{<}$ that uses less than

$$b(n) = \frac{1}{3}n - \frac{1}{2}\log(n) + \log\left(\sqrt{\frac{3}{2\pi}}\right)$$

advice bits to find a maximum matching in any online path instance with n vertices. Then, we describe a special set of pairwise different instances with

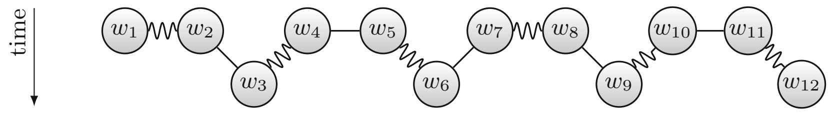


Fig. 2. Structure of the online instances for the lower bound on the advice bits used for solving the online matching problem on an online path. In this example, we have $n = 12$ and therefore $k = 2$.

pairwise different optimal solutions. The number of these instances will be at least $2^{b(n)}$. Therefore, by the pigeonhole principle, there exists a pair of instances such that $A_<$ has to use the same advice string to solve them both optimally. The set of instances will be constructed in such a way that they all have a long common prefix and so any online algorithm can perform differently on this long prefix if and only if the advice is different. Moreover, any optimal algorithm is forced to behave differently already in a time step when it cannot distinguish between them, because they have the same prefix. Therefore, the algorithm cannot use the same advice string to solve them both optimally.

For the proof, we will assume that $n = 6k$ is an even number such that all the instances contain exactly one maximum matching. The special instances all start with a common prefix, namely with $\frac{n}{3} = 2k$ isolated edges

$$\{\{w_{3i+1}, w_{3i+2}\} \mid i \in \{0, 1, 2, \ldots, 2k-1\}\}$$

given in all possible orders. Note that each online algorithm has to decide immediately which of these edges are part of the matching and which are not. In the second step, each two of those edges are connected by one of the vertices in $\{w_{3i} \mid i \in \{1, 2, \ldots, 2k\}\}$ in order to get a path (see Fig. 2).

The described instances only differ in the presentation order of the edge set $\{\{w_{3i+1}, w_{3i+2}\} \mid i \in \{0, 1, 2, \ldots, 2k-1\}\}$ containing $2k$ vertices. Thereby, it does not matter whether the vertex w_{3i+1} or the vertex w_{3i+2} appears first.

Note that the k edges $\{w_1, w_2\}, \ldots, \{w_{6k-5}, w_{6k-4}\}$ have to be matching edges and the remaining k edges $\{w_4, w_5\}, \ldots, \{w_{6k-2}, w_{6k-1}\}$ are non-matching edges. The solution only depends on the order in which the matching and the non-matching edges appear. This divides the set of all instances that can be reached with the described construction in equivalence classes, and for each of these classes, there is exactly one unique optimal solution. The unique solution is determined by a string $x_1 x_2 x_3 \ldots x_{2k}$ with $x_i \in \{0, 1\}$ for all $i \in \{1, 2, \ldots, 2k\}$ with

$$x_i = \begin{cases} 1 & \text{if the } i\text{th isolated edge is a matching edge,} \\ 0 & \text{else.} \end{cases}$$

These strings contain exactly k zeros and k ones. This leads to $\binom{2k}{k}$ equivalence classes. Using a well-known bound on the central binomial coefficient, we get

$$\binom{2k}{k} > \frac{1}{2} \cdot \frac{4^k}{\sqrt{\pi k}} = \frac{1}{2} \cdot \frac{4^{\frac{n}{6}}}{\sqrt{\pi \cdot \frac{n}{6}}} = \frac{1}{2} \cdot \frac{2^{\frac{n}{3}}}{\sqrt{\frac{\pi}{6}} \cdot \sqrt{n}}$$
$$= \sqrt{\frac{3}{2\pi}} \cdot \frac{2^{\frac{n}{3}}}{\sqrt{n}},$$

for any $k = \frac{n}{6}$. We claimed that $A_<$ uses less than

$$\log\left(\sqrt{\frac{3}{2\pi}} \cdot \frac{2^{\frac{n}{3}}}{\sqrt{n}}\right) = \frac{1}{3}n - \frac{1}{2}\log(n) + \log\left(\sqrt{\frac{3}{2\pi}}\right) < \log\binom{2k}{k}$$

advice bits. Therefore, there are two equivalence classes which get the same advice string. This means that the algorithm makes the same decisions on the first $\frac{n}{3}$ isolated edges on both instance classes. But since these equivalence classes are different, there is a first isolated edge e such that, w.l.o.g., e is a matching edge in the first instance but a non-matching edge in the second instance. Thus, the matching chosen by $A_<$ will not be optimal for one of these instances, which is a contradiction to the assumption. □

5 Tradeoffs on Paths

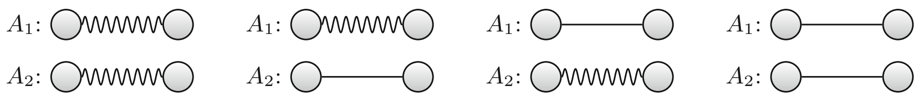
Until now, we have seen bounds on how much advice is needed to be optimal on general graphs or paths. In this section, we want to analyze what happens in the case when the algorithm has less advice bits at disposal.

We want to discuss some tradeoffs between the available amount of advice bits and the achievable competitive ratio. We know from Theorems 3 and 4 that approximately n/3 advice bits are necessary and sufficient for computing a maximal matching. On the other hand, as proven in [21], without advice, no competitive ratio better than $\frac{3}{2} - \varepsilon$ can be reached, for any $\varepsilon > 0$.

First, we want to show that, for one bit of advice, we can prove the same lower bound as for deterministic algorithms without advice on paths.

Theorem 5. *No online algorithm using one bit of advice for finding a maximum matching in a path on n vertices with $n \geq \frac{3(\alpha+1)}{\varepsilon} - 2$ can be better than $\left(\frac{3}{2} - \varepsilon\right)$-competitive (with additive constant α), for arbitrary non-negative constants α and ε.*

Proof. One bit of advice can be used to decide which of two given deterministic algorithms should be used in order to find a maximum matching in paths on n vertices. This makes the work of the adversary more involved since he has to provide a class of lower-bound instances for all pairs of algorithms. To cover all these pairs, we distinguish the algorithms with respect to the behavior on the isolated edges for a given online path instance $P^{\prec} \in \mathcal{P}_n$. Let A_1 and A_2 be two arbitrary online algorithms. There are four possibilities how these two algorithms can treat an isolated edge e (see Fig. 3):



(a) Group 1. **(b)** Group 2. **(c)** Group 3. **(d)** Group 4.

Fig. 3. The four possibilities how a pair of algorithms A_1 and A_2 can decide about an isolated edge.

Group 1: Both A_1 and A_2 assign e to the matching.
Group 2: Only A_1 takes e into the matching.
Group 3: Only A_2 takes e into the matching.
Group 4: Neither A_1 nor A_2 allocates e to the matching.

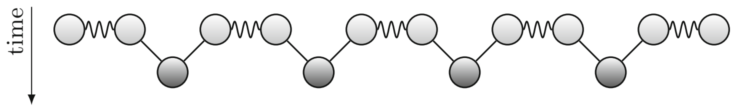
This gives us a classification of the isolated edges. Note that the adversary fights against two arbitrary deterministic algorithms. Therefore, some of these groups of edges can be empty. The algorithm with advice has to handle all subsets of these four groups.

It is easy to see that a matching with k unmatched vertices has $\lfloor \frac{k}{2} \rfloor$ less matching-edges than a maximum matching on this instance. Therefore, it suffices if we count the minimum number of unmatched vertices on the instances of this special instance class in order to calculate a lower bound on the competitive ratio for algorithms reading one bit of advice.

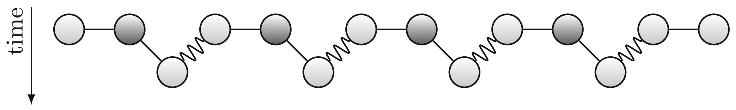
The construction is done as follows. Let $n = 6k + 4$ be the number of vertices in the online path $P^{\prec}$ for a $k \in \mathbb{N}$. The adversary starts with $\frac{n-1}{3}$ isolated edges which belong to one of the above-described edge groups. In order to reach a high number of unmatched vertices, the adversary arranges the isolated edges group by group to build four subpaths, connecting the isolated edges inside a group by single vertices. Note that, in each group, the neighboring isolated edges are either both matching or both non-matching edges. Therefore, if the algorithms act best possible, any of the four subpaths built by any of the two algorithms is of one of the two forms as shown in Fig. 4.

In both types of subpaths containing i isolated edges, the adversary forces the algorithms to leave at least $i - 1$ vertices unmatched. Therefore, we can guarantee $i - 1$ unmatched vertices in each group. In type-2 subpaths, we have possibly further unmatched vertices at the left and the right end. But we will count these possibly unmatched vertices in a second step, when the adversary connects the subpaths such that the two algorithms are forced to leave many vertices unmatched in between the at most four groups. Note that connecting two edges that are both matching or both non-matching forces the algorithms to leave one vertex unmatched. On the other hand, connecting two different edges enables the algorithms to construct at least locally a good matching (see Fig. 5).

The end edges of possibly four subpaths are depicted in Fig. 3. Therefore, these pairs of edges will symbolize the whole subpaths in the following figures. We will make a case distinction with respect to the subset of the groups that arise in the solution of a given pair of algorithms:

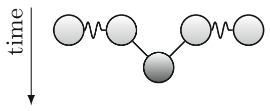


(a) Type 1: A group with isolated matching edges.

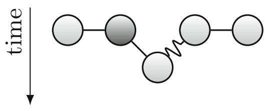


(b) Type 2: A group with isolated non-matching edges.

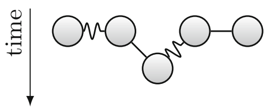
Fig. 4. The algorithm chooses, inside each group, one of the two patterns if it acts best possible connecting the isolated edges to a path. The marked vertices are the unmatched vertices within each group (ignoring the end vertices for type 2 subpaths). Note that we can assume w.l.o.g. that the algorithm always assigns the right edge to the matching when connecting the isolated edges in a subpath of type 2.



(a) Connecting two matching edges.



(b) Two non-matching edges.



(c) Two different edges.

Fig. 5. Connecting two isolated edges. The adversary can only force the algorithms to leave a vertex unmatched when connecting two edges of the same type.

Groups 1, 2, 3, 4. If all the edge groups are present in a solution, we arrange the subpaths in the order 2, 1, 3, 4, as shown in Fig. 6.
Both algorithms, A_1 and A_2, have to leave 3 vertices unmatched. Since in a subpath on i vertices, both algorithms leave at least $i-1$ vertices unmatched, we have $\frac{n-1}{3} - 4$ unmatched vertices within the four subpaths. Therefore, overall, the adversary can force both algorithms to leave at least

$$\frac{n-1}{3} - 4 + 3 = \frac{n-1}{3} - 1 = 2k$$

vertices unmatched, leading to a loss of k matching edges.

If we can show the same for all other subgroups, we are done, since in this case we get the same lower bound as in the deterministic case.

Groups 1, 2, 3. If no edges from group 4 appear in the solution, the adversary arranges the remaining groups as shown in Fig. 7.
Within the three subpaths, there are at least $\frac{n-1}{3} - 3$ unmatched vertices and together with the two additional unmatched vertices between the paths, we have again $\frac{n-1}{3} - 1$ unmatched vertices.

Groups 2, 3, 4. In this case, the adversary provides the order of Fig. 8, leading to at least $\frac{n-1}{3} - 1$ unmatched vertices.

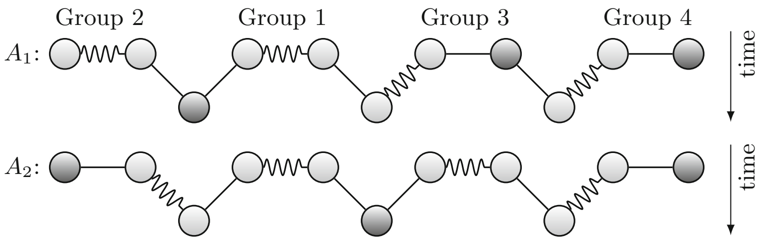


Fig. 6. All of the edge groups are present in the solution.

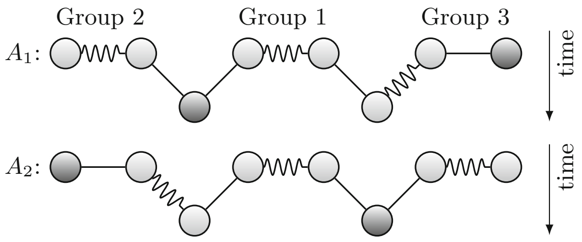


Fig. 7. The edge groups 1, 2 and 3 are present in the solution.

Groups 1, 2, 4 or groups 1, 3, 4. In these two equivalent cases, the order of Fig. 9 leads to $\frac{n-1}{3} - 1$ unmatched vertices.

Groups 1, 4 or groups 2, 4 or groups 3, 4. If only two edge groups appear in the solution, there are $\frac{n-1}{3} - 2$ unmatched vertices within the two subpaths. Therefore, one unmatched vertex between the two subpaths or at one end would be enough to reach the bound of $\frac{n-1}{3} - 1$ unmatched vertices in total. We see in Fig. 10 that this is given for all pairs of groups containing group 4, since the adversary can force the algorithms to leave at least the right end vertex unmatched.

Groups 1, 2 or groups 1, 3. Also in these two cases, the adversary can force the algorithms to leave $\frac{n-1}{3} - 1$ vertices unmatched (see Fig. 11).

Groups 2, 3. Also in this case, one of the end vertices will stay unmatched (see Fig. 11) and therefore the adversary can guarantee $\frac{n-1}{3}$ unmatched vertices.

Group 1 or 2 or 3 or 4. If only one group of edges is present in the solution, any algorithm leaves at least $\frac{n-1}{3}$ vertices unmatched within the path.

Summarizing, in all possible subsets of isolated edges, the solution has to contain at least $\frac{n-1}{3}$ unmatched vertices. A simple calculation shows that this leads to a competitive ratio of $\frac{3}{2} - \varepsilon$ on paths on n vertices with $n \geq \frac{3(\alpha+1)}{\varepsilon} - 2$, for arbitrary non-negative constants α and ε. □

We already mentioned above that, given the decision on the isolated edges whether to take them into the matching or not as advice, an algorithm solves the online matching problem on paths optimally. In the following, we investigate which competitive ratio an algorithm can reach for the online matching problem

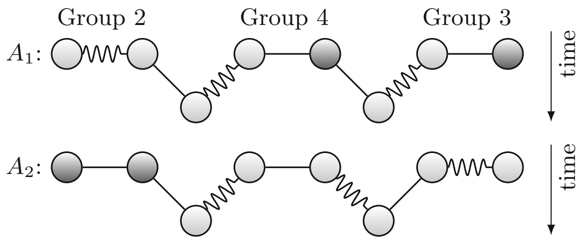


Fig. 8. The edge groups 2, 3 and 4 are present in the solution.

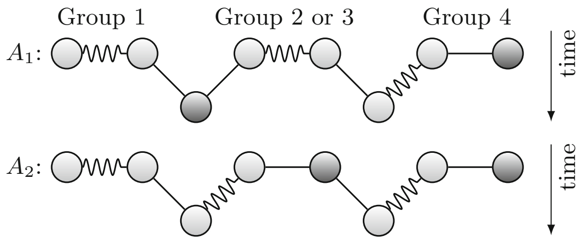


Fig. 9. The edge groups 1 and 4 and either group 2 or group 3 are present in the solution.

on paths if the given information about the decisions on isolated edges is only partially correct. To this end, we use the so-called *string guessing problem with known history* as introduced in [4,5].

In the string guessing problem with known history, the algorithm has to guess a bit string $b = b_1b_2b_3 \dots b_n$ of a given length n, one bit in each time step, and gets an immediate reply in the next time step, whether the guess was correct or not. With the first request, it additionally learns the length n of the sequence to be guessed. With the request $n+1$, the last bit b_n is revealed, and the algorithm does not have to respond in this last round. The goal of the online algorithm is to minimize the number of wrongly guessed bits. This very generic online problem has been proven very useful for providing both upper and lower bounds on the advice complexity of many different online problems. In particular, the following upper bound for the string guessing problem with known history is known.

Lemma 1 (Böckenhauer et al. [4]). *Let b be a bit string of length m. There is an online algorithm reading at most*

$$\left\lceil (1 + (1-\alpha)\log(1-\alpha) + \alpha\log(\alpha))\, m + \frac{3}{2}\log(m) + \frac{1}{2} + \log(\ln(2)) \right\rceil$$

advice bits in order to guess αm bits of b correctly, for some $\frac{1}{2} \leq \alpha < 1$.

This result can be used to design an algorithm for the online matching problem on paths.

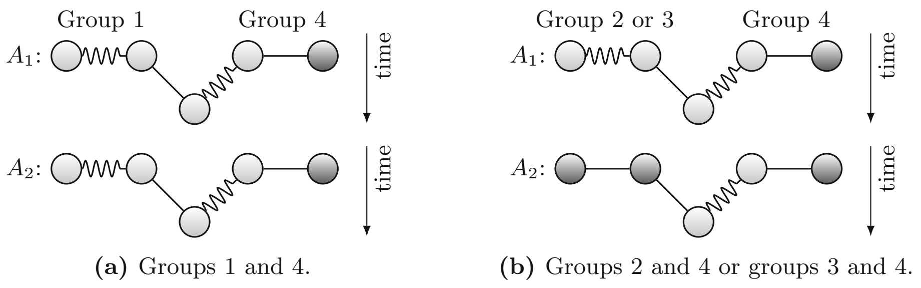


Fig. 10. All pairs of edge groups containing group 4 in the solution.

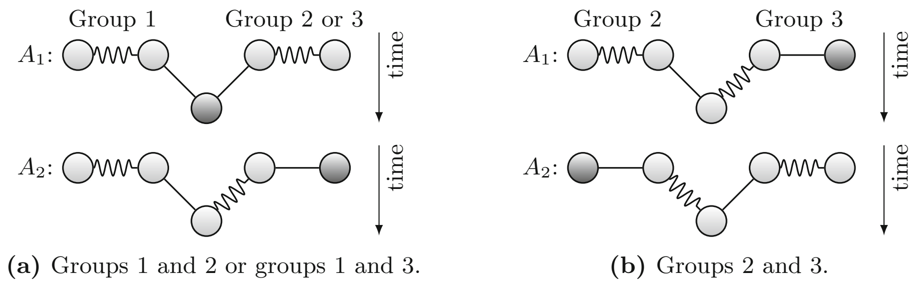


Fig. 11. All remaining pairs of two edge groups.

Theorem 6. *There exists an online algorithm that finds a matching on*

$$\left\lfloor \frac{n}{2} \right\rfloor - (1-\alpha)\left\lceil \frac{n}{3} \right\rceil$$

edges on a path on n vertices using at most

$$\left\lceil (1 + (1-\alpha)\log(1-\alpha) + \alpha\log(\alpha))\left\lceil \frac{n}{3} \right\rceil + \frac{3}{2}\log\left(\left\lceil \frac{n}{3} \right\rceil\right) + \frac{1}{2} + \log(\ln(2)) \right\rceil$$

advice bits, for some $\frac{1}{2} \leq \alpha < 1$.

Proof. Let $P^{\prec} \in \mathcal{P}_n$ be an online path instance for the online matching problem and let M^* be a fixed maximum matching that matches all inner vertices. Algorithm 2 works greedily on the non-isolated edges and computes its decisions on the isolated edges using an algorithm solving the string guessing problem with known history.

Recall that an online path instance on n vertices contains at most $\lceil \frac{n}{3} \rceil$ isolated edges. Let

$$e_1, e_2, \ldots, e_k, \text{ for some } k \in \left\{1, 2, \ldots, \left\lceil \frac{n}{3} \right\rceil\right\},$$

be these isolated edges. The algorithm transforms the decision on these isolated edges to the problem of finding a bit string $b = b_1 b_2 \ldots b_{\lceil \frac{n}{3} \rceil}$ with

Algorithm 2. Matching on Paths Using a Bit String

```
1  input: $P^{\prec} \in \mathcal{P}_n$, for some $n \in \mathbb{N}$
2  $M = \emptyset$;
3  Calculate $b = b_1 b_2 \ldots b_{\lceil \frac{n}{3} \rceil}$ by an algorithm for the string guessing problem with
   known history using the amount of advice specified in Theorem 6 to achieve the
   desired number of correct guesses;
4  for $i = 1$ to $n$ do
5      if (a) $v_i$ has exactly one edge $e$ to a previously isolated vertex then
6          read the next bit in $b$ to decide whether $e$ is a matching edge;
7          if $\sigma = 1$ then
8              $M \leftarrow M \cup \{e\}$;
9          else
10             $M \leftarrow M$;
11         end
12     else if (b) $v_i$ has two edges $e_1$ and $e_2$ to prev. isolated vertices then
13         use the next bit in $b$ to decide whether $e_1$ or $e_2$ is a matching edge;
14         if $\sigma = 0$ then
15             $M \leftarrow M \cup \{e_1\}$;
16         else
17             $M \leftarrow M \cup \{e_2\}$;
18         end
19     else if (c) $v_i$ is connected to some non-isolated and unmatched vertex by an
       edge $e$ then
20         $M \leftarrow M \cup \{e\}$;
21     else if (d) $v_i$ has an edge $e_1$ to a matched vertex and an edge $e_2$ to an
       isolated vertex then
22         $M \leftarrow M \cup \{e_2\}$;
23     else
24         $M \leftarrow M$;
25     end
26     output $M_i = M$;
27 end
28 output: $M = (e_{i_1}, e_{i_2}, \ldots, e_{i_m}) = \bigcup_{i=1}^{n} M_i \subseteq E$, for some $m \in \mathbb{N}$
```

$$b_i = \begin{cases} 1 & \text{if } e_i \text{ is an edge in } M^*, \\ 0 & \text{if } e_i \text{ is not present in } M^*, \end{cases}$$

for all $i \in \{1, 2, \ldots, k\}$ and $b_i = 0$ for all non-relevant bits $b_{k+1}, \ldots, b_{\lceil \frac{n}{3} \rceil}$; see Fig. 12 for an example.

Guessing αm bits correctly means guessing $(1-\alpha)m$ bits wrongly, implying $(1-\alpha)m$ wrongly set isolated edges of the online path on n vertices with $m = \lceil \frac{n}{3} \rceil$ with respect to M^*.

It can easily be seen that every wrongly set isolated edge leads to exactly two unmatched vertices if the adversary can arrange the isolated edges such that every correctly set edge lies between two wrongly set edges. This directly implies that every wrongly set isolated edge leads to exactly one edge less in

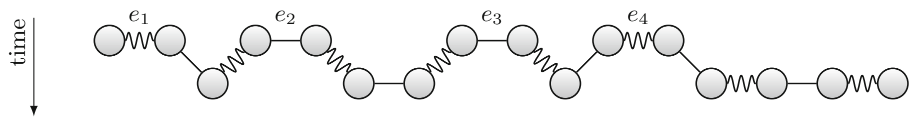


Fig. 12. This online path instance on 16 vertices contains the isolated edges e_1, e_2, e_3, and e_4. The maximum matching corresponds to the bit string $b = b_1b_2b_3b_4b_5b_6 = 100100$ in the bit string guessing problem with known history. b_1 to b_4 correspond to the decision on the edges e_1 to e_4, and b_5 and b_6 are non-relevant bits that occur since this online path does not contain the maximum possible number of isolated edges, which would be 6.

the matching computed by the algorithm, with respect to the fixed maximum matching M^*. Arranging the isolated edges in this way is possible as long as less than half of the isolated edges is set wrong, i.e., if $\alpha \geq \frac{1}{2}$ holds.

Therefore, we can use Lemma 1 and the fact that a path on n vertices has a matching containing $\left\lfloor \frac{n}{2} \right\rfloor$ matching edges to show that Algorithm 2 finds a matching of size

$$\left\lfloor \frac{n}{2} \right\rfloor - (1-\alpha)m = \left\lfloor \frac{n}{2} \right\rfloor - (1-\alpha)\left\lceil \frac{n}{3} \right\rceil,$$

for some $\frac{1}{2} \leq \alpha < 1$, using

$$\left\lceil (1 + (1-\alpha)\log(1-\alpha) + \alpha\log(\alpha))\left\lceil \frac{n}{3} \right\rceil + \frac{3}{2}\log\left(\left\lceil \frac{n}{3} \right\rceil\right) + \frac{1}{2} + \log(\ln(2)) \right\rceil$$

advice bits. This function is visualized in Fig. 13. □

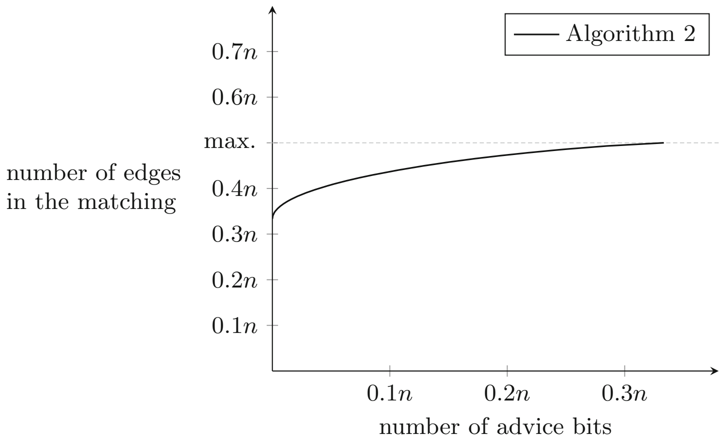


Fig. 13. Minimum number of edges found by Algorithm 2 on an arbitrary online path on n edges with respect to the number of advice bits that are at disposal.

6 Conclusion

We considered the fully online matching problem mainly on general bipartite graphs and paths. For general graphs, we constructed a deterministic online algorithm with advice that solves the problem optimally using $\mathcal{O}(n \log(n))$ advice bits. We complemented this result with a matching lower bound that holds already for P_5-free bipartite graphs with diameter 3.

For paths, we showed an almost tight upper and lower bound of approximately $\frac{n}{3}$ advice bits for optimality. These results can be quite easily generalized also to the case of cycles.

For paths, we also investigated the tradeoff between the number of advice bits and the competitive ratio using a reduction from string guessing. Moreover, we showed that one single advice bit does not help at all. It is open which ratio can be reached with two or a few more advice bits. Also the tradeoff for other graph classes would be of interest.

References

1. Barhum, K., et al.: On the power of advice and randomization for the disjoint path allocation problem. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 89–101. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04298-5_9
2. Bianchi, M.P., Böckenhauer, H.-J., Hromkovič, J., Keller, L.: Online coloring of bipartite graphs with and without advice. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 519–530. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32241-9_44
3. Bianchi, M.P., Böckenhauer, H.-J., Hromkovič, J., Krug, S., Steffen, B.: On the advice complexity of the online $L(2,1)$-coloring problem on paths and cycles. In: Du, D.-Z., Zhang, G. (eds.) COCOON 2013. LNCS, vol. 7936, pp. 53–64. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38768-5_7
4. Böckenhauer, H.-J., Hromkovič, J., Komm, D., Krug, S., Smula, J., Sprock, A.: The string guessing problem as a method to prove lower bounds on the advice complexity. Electron. Colloq. Comput. Complex. (ECCC) **19**, 162 (2012)
5. Böckenhauer, H.-J., Hromkovič, J., Komm, D., Krug, S., Smula, J., Sprock, A.: The string guessing problem as a method to prove lower bounds on the advice complexity. In: Du, D.-Z., Zhang, G. (eds.) COCOON 2013. LNCS, vol. 7936, pp. 493–505. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38768-5_44
6. Böckenhauer, H.-J., Komm, D., Královič, R., Rossmanith, P.: On the advice complexity of the knapsack problem. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 61–72. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29344-3_6
7. Böckenhauer, H.-J., Komm, D., Královič, R., Rossmanith, P.: The online knapsack problem: advice and randomization. Theor. Comput. Sci. **527**, 61–72 (2014)
8. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the k-server problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22006-7_18

9. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the k-server problem. J. Comput. Syst. Sci. **86**, 159–170 (2017)
10. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the advice complexity of online problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10631-6_35
11. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: Online algorithms with advice: the tape model. Inf. Comput. **254**, 59–83 (2017)
12. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
13. Boyar, J., Favrholdt, L.M., Kudahl, C., Larsen, K.S., Mikkelsen, J.W.: Online algorithms with advice: a survey. SIGACT News **47**(3), 93–129 (2016)
14. Boyar, J., Kamali, S., Larsen, K.S., López-Ortiz, A.: Online bin packing with advice. CoRR, abs/1212.4016 (2012)
15. Burjons, E., Hromkovič, J., Muñoz, X., Unger, W.: Online graph coloring with advice and randomized adversary. In: Freivalds, R.M., Engels, G., Catania, B. (eds.) SOFSEM 2016. LNCS, vol. 9587, pp. 229–240. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49192-8_19
16. Burjons, E., Komm, D., Schöngens, M.: The k-server problem with advice in d dimensions and on the sphere. In: Tjoa, A.M., Bellatreche, L., Biffl, S., van Leeuwen, J., Wiedermann, J. (eds.) SOFSEM 2018. LNCS, vol. 10706, pp. 396–409. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73117-9_28
17. Dobrev, S., Královič, R., Pardubská, D.: How much information about the future is needed? In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 247–258. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77566-9_21
18. Dobrev, S., Edmonds, J., Komm, D., Královič, R., Královič, R., Krug, S., Mömke, T.: Improved analysis of the online set cover problem with advice. Theor. Comput. Sci. **689**, 96–107 (2017)
19. Edmonds, J.: Paths, trees, and flowers. Can. J. Math. **17**, 449–467 (1965)
20. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 427–438. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02927-1_36
21. Favrholdt, L.M., Vatshelle, M.: Online greedy matching from a new perspective. http://www.ii.uib.no/~martinv/Papers/OnlineMatching.pdf
22. Forišek, M., Keller, L., Steinová, M.: Advice complexity of online coloring for paths. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 228–239. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28332-1_20
23. Gebauer, H., Komm, D., Královič, R., Královič, R., Smula, J.: Disjoint path allocation with sublinear advice. In: Xu, D., Du, D., Du, D. (eds.) COCOON 2015. LNCS, vol. 9198, pp. 417–429. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21398-9_33
24. Gupta, S., Kamali, S., López-Ortiz, A.: On advice complexity of the k-server problem under sparse metrics. Theory Comput. Sys. **59**, 476–499 (2016)
25. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15155-2_3
26. Karp, R.M., Vazirani, U.V., Vazirani, V.V.: An optimal algorithm for on-line bipartite matching. In: Proceedings of STOC 1990, pp. 352–358. ACM (1990)

27. Komm, D.: An Introduction to Online Computation: Determinism, Randomization, Advice. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-319-42749-2
28. Komm, D., Královič, R., Mömke, T.: On the advice complexity of the set cover problem. In: Hirsch, E.A., Karhumäki, J., Lepistö, A., Prilutskii, M. (eds.) CSR 2012. LNCS, vol. 7353, pp. 241–252. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30642-6_23
29. Mansour, Y., Vardi, S.: A local computation approximation scheme to maximum matching. CoRR, abs/1306.5003 (2013)
30. Micali, S., Vazirani, V.V.: An $O(\mathrm{sqrt}(|v|)|E|)$ algorithm for finding maximum matching in general graphs. In: Proceedings of FOCS 1980, pp. 17–27. IEEE (1980)
31. Plummer, M.D., Lovász, L.L.: Matching Theory. North-Holland Mathematics Studies. Elsevier Science, Amsterdam (1986)
32. Seibert, S., Sprock, A., Unger, W.: Advice complexity of the online coloring problem. In: Spirakis, P.G., Serna, M. (eds.) CIAC 2013. LNCS, vol. 7878, pp. 345–357. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38233-8_29
33. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)
34. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice Hall, Upper Saddle River (2001)

Sequence Hypergraphs: Paths, Flows, and Cuts

Kateřina Böhmová[1], Jérémie Chalopin[2], Matúš Mihalák[3], Guido Proietti[4,5], and Peter Widmayer[1](✉)

[1] Department of Computer Science, ETH Zurich, Zurich, Switzerland
asitak.kat@gmail.com, widmayer@inf.ethz.ch
[2] Aix-Marseille Université, CNRS, Université de Toulon, LIS, Marseille, France
jeremie.chalopin@lis-lab.fr
[3] Department of Data Science and Knowledge Engineering, Maastricht University, Maastricht, The Netherlands
matus.mihalak@maastrichtuniversity.nl
[4] DISIM, Università degli Studi dell'Aquila, L'Aquila, Italy
guido.proietti@univaq.it
[5] IASI, CNR, Rome, Italy

Abstract. We introduce *sequence hypergraphs* by extending the concept of a directed edge (from simple directed graphs) to hypergraphs. Specifically, every hyperedge of a sequence hypergraph is defined as a sequence of vertices (not unlike a directed path). Sequence hypergraphs are motivated by problems in public transportation networks, as they conveniently represent transportation lines. We study the complexity of several fundamental algorithmic problems, arising (not only) in transportation, in the setting of sequence hypergraphs. In particular, we consider the problem of finding a *shortest st-hyperpath*: a minimum set of hyperedges that "connects" (allows to travel to) t from s; finding a *minimum st-hypercut*: a minimum set of hyperedges whose removal "disconnects" t from s; or finding a *maximum st-hyperflow*: a maximum number of hyperedge-disjoint st-hyperpaths. We show that many of these problems are APX-hard, even in acyclic sequence hypergraphs or with hyperedges of constant length. However, if all the hyperedges are of length at most 2, we show that these problems become polynomially solvable. We also study the special setting in which for every hyperedge there also is a hyperedge with the same sequence, but in reverse order. Finally, we briefly discuss other algorithmic problems such as finding a minimum spanning tree, or connected components.

Keywords: Sequence hypergraphs · Colored graphs
Labeled problems · Transportation lines · Algorithms · Complexity

An extended abstract of this paper appeared at WG 2016, and it was at a workshop of this lovely series on graph-theoretic concepts in computer science where the last author had the joy of meeting the jubilarian for the first time.

© Springer Nature Switzerland AG 2018
H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 191–215, 2018.
https://doi.org/10.1007/978-3-319-98355-4_12

1 Introduction

Consider a public transportation network, e.g., a bus network, where every vertex corresponds to a bus stop, and where every bus line is specified as a fixed sequence of bus stops. One can travel in the network by taking a bus and then following the stops in the order that is fixed by the corresponding bus line. See Fig. 1 for an illustration. Note that we think of a line as a sequence of stops in one direction only, since there might be one-way streets or other obstacles that cause that the bus can travel the stops in a single direction only. Then, interesting questions arise: How can one travel from s to t using the minimum number of lines? How many lines must break down, so that t is not reachable from s? Are there two ways to travel from s to t that both use different lines?

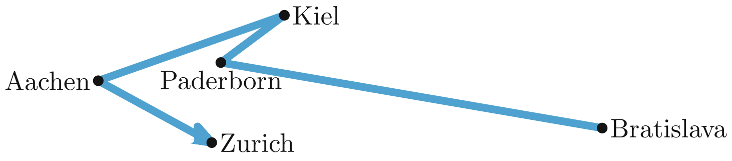


Fig. 1. A bus line (actually, the career path of the jubilarian) in a transportation network, and the corresponding hyperedge.

These kinds of questions are traditionally modeled and studied by algorithmic graph theory, but no model appears to capture all the necessary aspects of the problems above. We propose the following very natural way to extend the concept of directed graphs to hypergraphs.

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with an ordering of the vertices of every hyperedge is called a *sequence hypergraph*. Formally, the sequence hypergraph $\mathcal{H}$ consists of the set of vertices $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, and the set of *(sequence) hyperedges* $\mathcal{E} = \{E_1, E_2, \ldots, E_k\}$, where each hyperedge $E_i = (v_{i_1}, v_{i_2}, \ldots, v_{i_l})$ is defined as a sequence of vertices without repetition. We remark that this definition substantially differs from the commonly used definition of directed hypergraphs [1,2,14], where each directed hyperedge is a pair (From, To) of disjoint subsets of $\mathcal{V}$. We note that the order of vertices in a sequence hyperedge does not imply any order of the same vertices in other hyperedges. Furthermore, the sequence hypergraph does not impose any global order on $\mathcal{V}$.

There is another way to look at sequence hypergraphs coming from our motivation in transportation. For a sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, we construct a *directed colored multigraph* $G = (V, E, c)$ as follows. The set of vertices V is identical to $\mathcal{V}$, and for a hyperedge $E_i = (v_{i_1}, v_{i_2}, \ldots, v_{i_l})$ from $\mathcal{E}$, the multigraph G contains $l-1$ edges $(v_{i_j}, v_{i_{j+1}})$ for $j = 1, \ldots, l-1$, all colored with color $c(E_i)$, with $c(E_i) \neq c(E_{i'})$ for $i \neq i'$. Therefore, each edge of G is colored by one of the $k = |\mathcal{E}|$ different colors $\mathcal{C} = \{c(E_1), c(E_2), \ldots, c(E_k) \mid E_i \in \mathcal{E}\}$. Clearly, the edges of each color form a directed path in G. We refer to G as the *underlying colored graph* of $\mathcal{H}$. We denote by m the number of edges of G.

In this article, we study several standard graph-algorithmic problems in the setting of sequence hypergraphs. In particular, we consider the problem of finding a *shortest st-hyperpath*: an *st*-path that uses the minimum number of sequence hyperedges; the problem of finding a *minimum st-hypercut*: an *st*-cut that uses the minimum number of sequence hyperedges; and the problem of finding a *maximum st-hyperflow*: a maximum number of hyperedge-disjoint *st*-hyperpaths.

We show that the shortest *st*-hyperpath is NP-hard to approximate within a factor of $(1 - \varepsilon) \ln n$, for any $\varepsilon > 0$, in general sequence hypergraphs, but can be found in polynomial time if the given sequence hypergraph is acyclic (Sect. 3). On the other hand, we show that both maximum *st*-hyperflow and minimum *st*-hypercut are APX-hard to find even in acyclic sequence hypergraphs (Sects. 4 and 5). We then consider sequence hypergraphs with sequence hyperedges of constant length, where the length of a hyperedge is the number of its vertices minus one. We note that the shortest *st*-hyperpath problem remains hard to approximate even with hyperedges of length at most 5, and we show that the maximum *st*-hyperflow problem remains APX-hard even with hyperedges of length at most 3. On the other hand, we show that if all the hyperedges are of length at most 2, all 3 problems become polynomially solvable (Sect. 6). We also study the complexity in a special setting in which for each hyperedge there also is a hyperedge with the same sequence, but in the opposite direction. We show that the shortest *st*-hyperpath problem becomes polynomially solvable, but both maximum *st*-hyperflow and minimum *st*-hypercut are NP-hard to find also in this setting, and we give a 2-approximation algorithm for the minimum *st*-hypercut problem (Sect. 7). Finally, we briefly study the complexity of other algorithmic problems, namely, finding a minimum spanning tree, or connected components, in sequence hypergraphs (Sect. 8). For a summary of the results see Table 1. The table also shows known results for the related labeled graphs (discussed below). The result on the APX-hardness of the shortest *st*-hyperpath problem (Theorem 1) appeared also, in a different context, in [4].

Table 1. Summary of the complexities admitted by some classic problems in the setting of colored (labeled) graphs and sequence hypergraphs. The last row indicates whether the sizes of the maximum *st*-flow and the minimum *st*-cut equal in the considered setting. The cells in gray show our contribution.

	Colored/Labeled Graphs		Sequence Hypergraphs			
	General	Span 1	General	Acyclic	Backward	Length≤ 2
Shortest *st*-path	APX-hard	P	APX-hard	P	P	P
Minimum *st*-cut	APX-hard	P	APX-hard	APX-hard	NP-hard	P
Maximum *st*-flow	APX-hard	P	APX-hard	APX-hard	NP-hard	P
MaxFlow-MinCut Duality	$\times$	$\surd$	$\times$	$\times$	$\times$	$\surd$

2 Related Work

Recently, there has been a lot of research concerning optimization problems in (multi)graphs with colored edges, where the cost of a solution is measured by the number of colors used, e.g., one may ask for an st-path using the minimum number of colors. The motivation comes from applications in optical or other communication networks, where a group of links (i.e., edges) can fail simultaneously and the goal is to find resilient solutions. Similar situations may occur in economics, when certain commodities are sold (and priced) in bundles.

Formally, *colored graphs* or *labeled graphs* are (mostly undirected) graphs where each edge has one color, and in general there is no restriction on a set of edges of the same color. Note that some of the studies consider a slightly different definition of colored graphs, where to each edge a set of colors is associated instead of a single color. Since the computational complexity of some problems may differ in the two models, the transformations between the two models have been investigated [8].

The *minimum label path* problem, which asks for an st-path with a minimum number of colors, is NP-hard and hard to approximate [5–7,16,17,22]. The *2 label disjoint paths* problem, which asks for a pair of st-paths such that the sets of colors appearing on the two paths are disjoint, is NP-hard [18]. The *minimum label cut* problem, which asks for a set of edges with a minimum number of colors that forms an st-cut, is NP-hard and hard to approximate [7,23]. The *minimum label spanning tree* problem, which asks for a spanning tree using edges of minimum number of colors, is NP-hard and hard to approximate [17,20].

Hassin et al. [17] give a $\log(n)$-approximation algorithm for the minimum label spanning tree problem and a $\sqrt{n}$-approximation algorithm for the minimum label path problem, where n is the number of vertices of the input colored graph. Zhang et al. [23] give a $\sqrt{m}$-approximation algorithm for the minimum label cut problem, where m is the number of edges of the input colored graph. Fellows et al. [13] study the parameterized complexity of minimum label problems. Coudert et al. [7,8] consider special cases when the *span* is 1, i.e., each color forms a connected component; or when the graph has a *star property*, i.e., the edges of every color are adjacent to one vertex.

Note that since most of these results consider undirected labeled graphs, they provide almost no implications on the complexity of similar problems in the setting of sequence hypergraphs. In our setting, not only we work with *directed* labeled graphs, but we also require edges of each color to form a directed path, which implies a very specific structure that, to the best of our knowledge, has not been considered in the setting of labeled graphs.

On the other hand, we are not the first to define hypergraphs with hyperedges specified as sequences of vertices. However, we are not aware of any work that would consider and explore this type of hypergraphs from an algorithmic graph theory point of view. In fact, mostly, these hypergraphs are taken merely as a tool, convenient to capture certain relations, but they are not studied further. We shortly list a few articles where sequence hypergraphs appeared, but we do not give details, since there is very little relation to our area of study. Berry et al. [3] introduce and describe the basic architecture of a software tool for (hyper)graph drawing. Wachman et al. [21] present a kernel for learning from *ordered hypergraphs*, a formalization that captures relational data as used in

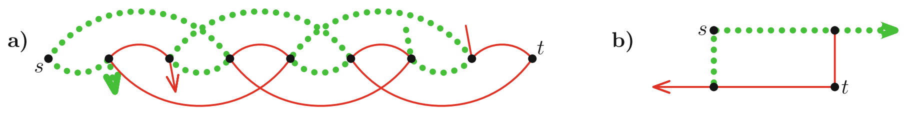


Fig. 2. In both figures, the green-dotted curve and the red solid curve depict two sequence hyperedges. (a) The length of the *st*-hyperpath is 2, but the number of switches is 7. (b) The *st*-hyperpath consists of two sequence hyperedges that also form a hypercycle.

Inductive Logic Programming. Erdős et al. [12] study Sperner-families and as an application of a derived result they study the maximum number of edges of a so-called *directed Sperner-hypergraph*.

3 On the Shortest *st*-Hyperpath

In this section, we consider the shortest *st*-hyperpath problem in general sequence hypergraphs and in acyclic sequence hypergraphs.

Definition 1 (*st*-Hyperpath). *Let s and t be two vertices of a sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. A set of hyperedges $P \subseteq \mathcal{E}$ forms a* hyperpath *from s to t if the underlying (multi)graph G' of the subhypergraph $\mathcal{H}' = (\mathcal{V}, P)$ contains an st-path, and P is minimal with respect to inclusion. We call such an st-path an* underlying path *of P. The* length *of an st-hyperpath P is defined as the number of hyperedges in P. The* number of switches *of an st-hyperpath P is the minimum number of changes between the hyperedges of P, when following any underlying st-path of P.*

We note that each hyperpath may have multiple underlying paths. Also note that, even though the number of switches of an *st*-hyperpath P gives an upper bound on the length of P, the actual length of P can be much smaller than the number of switches of P (see Fig. 2a).

Proposition 1. *Given a sequence hypergraph, and two vertices s and t, an st-hyperpath minimizing the number of switches can be found in polynomial time.*

Such an *st*-hyperpath can be found, e.g., by a modified Dijkstra algorithm (starting from s, following the outgoing sequence hyperedges and for each vertex storing the minimum number of switches necessary to reach it).

Conversely, we show that finding a shortest *st*-hyperpath (minimizing the number of hyperedges) is hard to approximate. On the other hand, if the given sequence hypergraph is acyclic, we show that the shortest *st*-hyperpath problem becomes polynomially solvable.

Definition 2 (Acyclic Sequence Hypergraph). *A set of hyperedges $O \subseteq \mathcal{E}$ forms a* hypercycle, *if there are two vertices $a \neq b$ such that O contains both a hyperpath from a to b, and a hyperpath from b to a. A sequence hypergraph without hypercycles is called* acyclic.

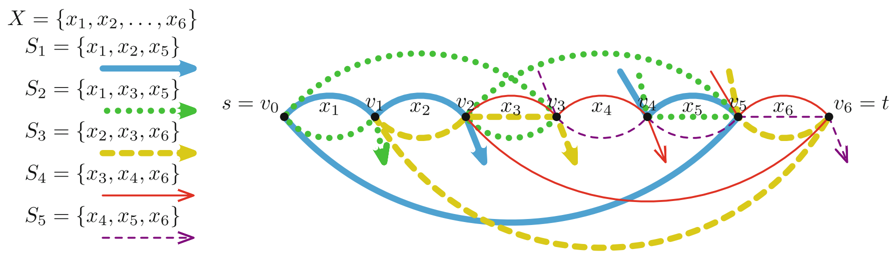

$X = \{x_1, x_2, \ldots, x_6\}$
$S_1 = \{x_1, x_2, x_5\}$
$S_2 = \{x_1, x_3, x_5\}$
$S_3 = \{x_2, x_3, x_6\}$
$S_4 = \{x_3, x_4, x_6\}$
$S_5 = \{x_4, x_5, x_6\}$
$s = v_0$
v_1
v_2
v_3
v_4
v_5
$v_6 = t$
x_1
x_2
x_3
x_4
x_5
x_6

Fig. 3. Finding a shortest st-hyperpath is at least as hard as the minimum set cover problem.

Observe that an st-hyperpath may also be a hypercycle (see Fig. 2b).

Definition 3 (Edges of a Hyperedge). *Let $E = (v_1, v_2, \ldots, v_k)$ be a hyperedge of a sequence hypergraph $\mathcal{H}$. We call the set of directed edges $\{e_i = (v_i, v_{i+1}) |\ i = 1, \ldots, k-1\}$ the* edges of E. *The edges of E are exactly the edges of color $c(E)$ in the underlying colored graph of $\mathcal{H}$. The* length *of a hyperedge is defined as the number of its edges (which is the number of its vertices minus one).*

For a fixed order $V^O = (v_1, v_2, \ldots, v_n)$ of vertices $\mathcal{V}$, an edge e of a hyperedge E is called a forward *edge with respect to V^O if its orientation agrees with the order V^O. Similarly, e is a* backward *edge if its orientation disagrees with V^O.*

Theorem 1. *Finding a shortest st-hyperpath in sequence hypergraphs is* NP*-hard to approximate within a factor of $(1-\varepsilon)\ln n$ for any $\varepsilon > 0$, unless* P = NP. *The problem remains* APX*-hard even if every hyperedge has length at most 5.*

Proof. We construct an approximation-preserving reduction from the set cover problem. The reduction is similar to that presented in [22] for the minimum label path problem in colored graphs. An instance $I = (X, \mathcal{S})$ of the set cover problem is given by a ground set $X = \{x_1, \ldots, x_n\}$, and a family of its subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$. The goal is to find a smallest subset $\mathcal{S}' \subseteq \mathcal{S}$ such that the union of the sets in $\mathcal{S}'$ contains all elements from X. The set cover problem is known to be NP-hard to approximate within a factor of $(1-\epsilon)\ln n$, unless P = NP [10]. Moreover, if each subset of $\mathcal{S}$ is of size at most 3, the problem remains APX-hard [9].

From I we construct a sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ as follows (cf. Figure 3 along with the construction). The set of vertices $\mathcal{V} = \{v_0, v_1, \ldots, v_n\}$ contains one vertex v_i for each element x_i of the ground set X, plus one additional vertex v_0. Let V^O be the order of vertices in $\mathcal{V}$ naturally defined by their indices. The set of sequence hyperedges $\mathcal{E} = \{E_1, \ldots, E_m\}$ contains one hyperedge for each set in $\mathcal{S}$. For a set $S_i \in \mathcal{S}$, consider the set of vertices that correspond to the elements in S_i and order them according to V^O, to obtain a sequence $Q = (v_{i_1}, v_{i_2}, \ldots, v_{i_r})$, where $i_1 < i_2 < \cdots < i_r$. First, let us consider the simplest case where none of the v_{i_j} and $v_{i_{j+1}}$ (for $j = 1, \ldots, r-1$) are consecutive in the order V^O, that is, $i_j + 1 \neq i_{j+1}$. Then the sequence of the hyperedge E_i corresponding to S_i is $(v_{i_r-1}, v_{i_r}, v_{i_{(r-1)}-1}, v_{i_{(r-1)}}, \dots, v_{i_1-1}, v_{i_1})$ (e.g., the hyperedge corresponding to S_2 in Fig. 3). In other words, E_i consists of forward edges: one forward edge (v_{i_j-1}, v_{i_j}) for each v_{i_j} in Q; and backward edges that connect the forward edges in the order opposite to Q. Now, for the more general case, if for some j, v_{i_j} and $v_{i_{j+1}}$ are consecutive vertices with respect to V^O, i.e., $i_j + 1 = i_{j+1}$, the sequence constructed as above would contain vertices repeatedly, which is not allowed (each sequence hyperedge has to be mapped to a path in the underlying graph). To avoid this, we construct E_i as follows. For simplicity of the explanation, instead of describing the sequence of the hyperedge, we specify E_i by listing the edges of the hyperedge and the path to which E_i is mapped. The hyperedge E_i consists of the same forward edges as before: one forward edge (v_{i_j-1}, v_{i_j}) for each v_{i_j} in Q. Whenever two or more vertices of Q are consecutive in V^O, their corresponding forward edges form a path. Clearly, the forward edges of E_i then determine a set of (non-overlapping) paths $p_1, p_2, \dots, p_{r'}$ (uniquely ordered according to V^O). The backward edges of E_i then connect these paths in the order opposite to V^O into a single path (which specifies E_i). In particular, the last vertex of $p_{r'}$ connects to the first vertex of $p_{r'-1}$, the last vertex of $p_{r'-1}$ connects to the first vertex of $p_{r'-2}$, ..., and the last vertex of p_2 connects to the first vertex of p_1.

Note that the length of each sequence hyperedge E_i is bounded by $2|S_i| - 1$, where $|S_i|$ is the size of the set $S_i \in \mathcal{S}$ corresponding to E_i. This follows from the fact that E_i consists of $|S_i|$ forward edges and at most $|S_i| - 1$ backward edges to connect the forward edges. In particular, if each subset of $\mathcal{S}$ is of size at most 3, all the hyperedges are of length at most 5.

We set the source vertex s to v_0, and the target vertex t to v_n, and we show that a shortest st-hyperpath in $\mathcal{H}$ of length k provides a minimum set cover for I of the same size, and vice versa. First, notice that all the forward edges (with respect to V^O) of the hyperedges in $\mathcal{E}$ are of the form (v_i, v_{i+1}) for some $i = 0, \dots, n-1$. Together with the fact that t is smaller than s in the order V^O, it follows that any path from s to t in the underlying graph of $\mathcal{H}$ goes via all the vertices, in the order V^O. Thus, there is an underlying path p of the shortest st-hyperpath P in $\mathcal{H}$, such that p does not use any backward edges of the hyperedges in $\mathcal{E}$. Clearly, by choosing a hyperedge E_i into the st-hyperpath P, one also chooses its forward edges and this way "covers" some sections of the underlying path of P. Since there is a one-to-one mapping between the hyperedges in $\mathcal{E}$ and the sets in $\mathcal{S}$, by finding an st-hyperpath P of length k, one finds a set cover of size k for the given instance. On the other hand, each set cover of size k can be mapped, using the same direct one-to-one mapping in the opposite direction, to an st-hyperpath of length k.

Thus, the described reduction is approximation-preserving. By reducing from the general set cover problem we obtain the first part of the claim, and by reducing from the set cover problem with all subsets of size at most 3 we obtain the second part of the claim. □

We complement the hardness result with a positive one.

Theorem 2. *The problem of finding the shortest st-hyperpath in acyclic sequence hypergraphs can be solved in polynomial time.*

Proof. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be an acyclic sequence hypergraph. Since $\mathcal{H}$ is acyclic, let V^O be an order of the vertices $\mathcal{V}$ such that all the edges of each hyperedge are forward edges with respect to this order. This implies that for every st-hyperpath, there is an underlying path where all the edges of each hyperedge appear consecutively (the last edge of a hyperedge E appearing in an underlying path is reachable by E from the first appearing edge of E). Therefore, finding the shortest st-hyperpath P in $\mathcal{H}$ is the same as finding a hyperpath minimizing the number of switches, which can be done in polynomial time by Proposition 1. □

4 On the Maximum *st*-Hyperflow

We consider the problem of finding a number of hyperedge-disjoint st-hyperpaths. Capturing a similar relation as in graphs (between a set of k edge-disjoint st-paths and an st-flow of size k, when all the capacities are 1), for simplicity and brevity, we refer to a set of hyperedge-disjoint st-hyperpaths as an st-hyperflow.

Definition 4 (*st*-Hyperflow). *Let s and t be two vertices of a sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. Let $\mathcal{F} \subseteq 2^{\mathcal{E}}$ be a set of pairwise hyperedge-disjoint st-hyperpaths $\mathcal{F} = \{P_1, \dots, P_k\}$. Then, $\mathcal{F}$ is an st-*hyperflow *of size $|\mathcal{F}| = k$.*

We show that deciding whether a given sequence hypergraph contains an st-hyperflow of size 2 is NP-hard, and thus finding a maximum st-hyperflow is inapproximable within a factor of $2 - \varepsilon$ unless P = NP. This remains true even for acyclic sequence hypergraphs with all the hyperedges of length at most 3.

Theorem 3. *Given an acyclic sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with all hyperedges of length at most 3, and two vertices s and t, it is* NP*-hard to decide whether there are two hyperedge-disjoint st-hyperpaths.*

Proof. We construct a reduction from the NP-complete 3-Sat problem [15]. Let I be an instance of the 3-Sat problem, given as a set of m clauses $C = \{c_1, \dots, c_m\}$ over a set $X = \{x_1, \dots, x_n\}$ of Boolean variables. Recall that the goal of the 3-Sat problem is to find an assignment to the variables of X that satisfies all clauses of C.

From I we construct a sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ as follows (cf. Figure 4 along with the construction). The set $\mathcal{V}$ consists of $2 + (m + 1) + (n + 1) + \sum_{c_i \in C} |c_i|$ vertices: a source vertex s and a target vertex t; a vertex c_i for each clause $c_i \in C$ and a dummy vertex c_{m+1}; a vertex x_j for each variable $x_j \in X$ and a dummy vertex x_{n+1}; and finally a vertex $x_j c_i$ for each pair (x_j, c_i) such that $x_j \in c_i$, and similarly, $\overline{x_j} c_i$ for each $\overline{x_j} \in c_i$. Let us fix an arbitrary order C^O of the clauses in C. The set $\mathcal{E}$ consists of $4 + 2n + |I|$

$$\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3})$$

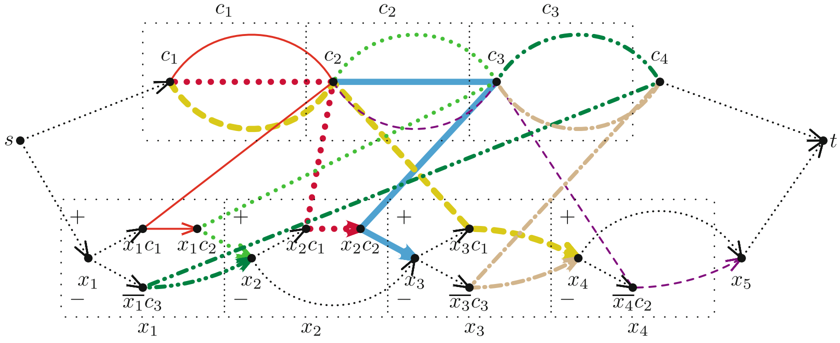


Fig. 4. Deciding st-hyperflow of size 2 is at least as hard as 3-SAT.

hyperedges: There are 2 *source hyperedges* (s, c_1) and (s, x_1), and 2 *target hyperedges* (c_{m+1}, t) and (x_{n+1}, t). There are $2n$ *auxiliary hyperedges* (x_i, x_ic_k) and $(x_i, \overline{x_i}c_{k'})$ for $i = 1, \ldots, n$, where c_k, or $c_{k'}$ is always the first clause (with respect to C^O) containing x_i or $\overline{x_i}$, respectively. If there is no clause containing x_i (or $\overline{x_i}$), the corresponding auxiliary hyperedge is (x_i, x_{i+1}). Finally, there are $|I|$ *lit-in-clause hyperedges* as follows. For each appearance of a variable x_j in a clause c_i as a positive literal there is one lit-in-clause hyperedge $(c_i, c_{i+1}, x_jc_i, x_jc_k)$, where c_k is the next clause (with respect to C^O) after c_i where x_j appears as a positive literal (in case, there is no such c_k, then the hyperedge ends in x_{j+1} instead). Similarly, if x_j is in c_i as a negative literal, there is one lit-in-clause hyperedge $(c_i, c_{i+1}, \overline{x_j}c_i, \overline{x_j}c_k)$, where c_k is the next clause containing the negative literal $\overline{x_j}$ (or it ends in x_{j+1}).

Clearly, each hyperedge is of length at most 3. We now observe that the constructed sequence hypergraph $\mathcal{H}$ is acyclic. All the hyperedges of $\mathcal{H}$ agree with the following order: the source vertex s; all the vertices $c_i \in C$ ordered according to C^O, and the dummy vertex c_{m+1}; the vertex x_1 followed by all the vertices x_1c_i ordered according to C^O, and then followed by the vertices $\overline{x_1}c_i$ again ordered according to C^O; the vertex x_2 followed by all x_2c_i and then all $\overline{x_2}c_i$; …; the vertex x_n followed by all x_nc_i and then all $\overline{x_n}c_i$; and finally the dummy vertex x_{n+1}; and the target vertex t.

We show that the formula I is satisfiable if and only if $\mathcal{H}$ contains two hyperedge-disjoint st-hyperpaths. There are 3 possible types of st-paths in the underlying graph of $\mathcal{H}$: the first one leads through all the vertices $c_1, c_2, \ldots, c_{m+1}$ in this order; the second one leads through all the vertices $x_1, x_2, \ldots, x_{m+1}$ in this order and between x_j and x_{j+1} it goes either through all the x_jc_* (here, and later, $*$ is used as a wildcard) vertices or through all the $\overline{x_j}c_*$ vertices (this may differ for different j); and the third possible st-path starts the same as the first option and ends as the second one. Based on this observation, notice that there can be at most 2 hyperedge-disjoint st-hyperpaths: necessarily, one of them has

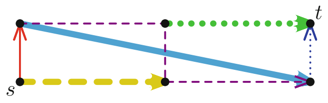


Fig. 5. Acyclic sequence hypergraph with minimum st-hypercut of size 2, and no two hyperedge-disjoint st-hyperpaths.

an underlying path of the first type, while the other one has an underlying path of the second type.

From a satisfying assignment A to the variables of I we can construct the two disjoint st-hyperpaths as follows. The underlying path of one hyperpath leads from s to t via the vertices $c_1, c_2, \ldots, c_{m+1}$, and to move from c_i to c_{i+1} it uses a lit-in-clause hyperedge that corresponds to a pair (l, c_i) such that l is one of the literals that satisfy the clause c_i in A. The second hyperpath has an underlying path of the second type, it leads via $x_1, x_2, \ldots, x_{n+1}$, and from x_j to x_{j+1} it uses the vertices containing only the literals that are not satisfied by the assignment A. Thus, the second hyperpath uses only those lit-in-clause hyperedges that correspond to pairs containing literals that are not satisfied by A. This implies that the two constructed st-hyperpaths are hyperedge-disjoint.

Let P and Q be two hyperedge-disjoint st-hyperpaths of $\mathcal{H}$. Let P have an underlying path p of the first type and Q have an underlying path q of the second type. We can construct a satisfying assignment for I by setting to `TRUE` the literals opposite to those that occur in the vertices on q. Then, the hyperpath P describes how the clauses of I are satisfied by this assignment. $\square$

5 On the Minimum *st*-Hypercut

Quite naturally, we define an st-hypercut of a sequence hypergraph $\mathcal{H}$ as a set C of hyperedges whose removal from $\mathcal{H}$ leaves s and t disconnected.

Definition 5 (*st*-Hypercut). *Let s and t be two vertices of a sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. A set of hyperedges $X \subseteq \mathcal{E}$ is an st-*hypercut *if the subhypergraph $\mathcal{H}' = (\mathcal{V}, \mathcal{E} \setminus X)$ does not contain any hyperpath from s to t. The* size *of an st-hypercut X is $|X|$, i.e., the number of hyperedges in X.*

For directed (multi)graphs, the famous MaxFlow-MinCut Duality Theorem [11] states that the size of a maximum st-flow is equal to the size of a minimum st-cut. In sequence hypergraphs, this duality does not hold, not even in acyclic sequence hypergraphs as Fig. 5 shows. But, of course, the size of any st-hyperflow is a lower bound on the size of any st-hypercut. We showed the maximum st-hyperflow problem to be APX-hard even in acyclic sequence hypergraphs (see Theorem 3). It turns out that also the minimum st-hypercut problem in acyclic sequence hypergraphs is APX-hard.

Theorem 4. *Minimum st-hypercut in acyclic sequence hypergraphs is* NP*-hard to approximate within a factor of $2 - \varepsilon$ under UGC, or within a factor $7/6 - \varepsilon$ unless* $\mathrm{P} = \mathrm{NP}$.

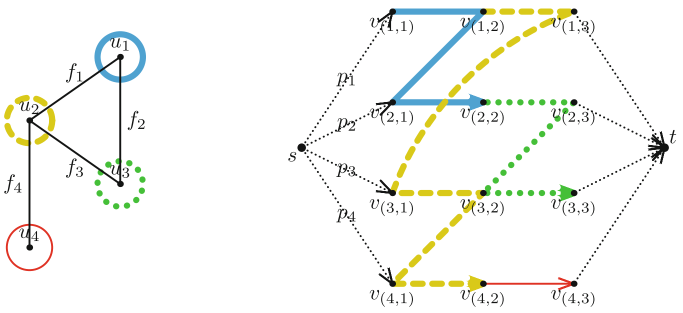


Fig. 6. Minimum st-hypercut is at least as hard as minimum vertex cover.

Proof. We construct an approximation-preserving reduction from the vertex cover problem, which has the claimed inapproximability [19]. An instance of the vertex cover problem is an undirected graph $I = (U, F)$, with the vertex set $U = \{u_1, \dots, u_n\}$ and the edge set $F = \{f_1, \dots, f_m\}$. The goal is to find a smallest subset $U' \subseteq U$ such that U contains at least one vertex from each edge $f \in F$.

To construct from the instance I an instance I' of the minimum st-hypercut problem in acyclic sequence hypergraphs, we fix an order of the vertices and edges of I as follows. Let $U^O = (u_1, \dots, u_n)$ be an arbitrary order on the vertices of U. From every edge in F we create an ordered edge, where the vertices of the edge are ordered naturally according to U^O. Let F' denote the set of the created ordered edges, and let F^O be the edges F' ordered lexicographically according to U^O.

We construct the sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ of I' as follows (cf. Figure 6 along with the construction). The set of vertices $\mathcal{V}$ consists of $3m + 2$ vertices: a source vertex s, a target vertex t, and for each edge $f_i \in F^O$, $i = 1, \dots, m$, there are three vertices $v_{(i,1)}, v_{(i,2)}$, and $v_{(i,3)}$. The set of hyperedges $\mathcal{E}$ consists of $2m+n$ hyperedges. There are m *source hyperedges* of the form $(s, v_{(i,1)})$, each of them connects s to one vertex $v_{(i,1)}$. There are m *target hyperedges* of the form $(v_{(i,3)}, t)$, each of them connects one vertex $v_{(i,3)}$ to t. And finally, there are n *vertex hyperedges*, each corresponding to one of the vertices in U, and is constructed as follows.

For a vertex $u \in U$ of I, we describe the sequence of vertices in the corresponding vertex hyperedge E_u iteratively. We start with an empty sequence. We consider the edges F^O in order, and for each edge $f_i \in F^O$ that contains u we prolong the sequence of E_u as follows. If f_i contains u as the first vertex, we append $v_{(i,1)}$ and $v_{(i,2)}$ to E_u. Otherwise, f_i contains u as the second vertex, and we append $v_{(i,2)}$ and $v_{(i,3)}$ to E_u. Now consider the edges of the obtained hyperedge E_u and let us distinguish two types. First, there are edges of the form $(v_{(i,1)}, v_{(i,2)})$ and $(v_{(i,2)}, v_{(i,3)})$ for some i, and second, there are edges of the form $(v_{(i,2)}, v_{(j,1)})$, $(v_{(i,3)}, v_{(j,2)})$, and $(v_{(i,3)}, v_{(j,1)})$ for some $i < j$. Note that, due to the fact that the edges in F^O are ordered lexicographically, all the edges of the vertex hyperedge E_u take one of the forms described above. Also note that, due to the direct correspondence between the vertices in U and vertex hyperedges, each tuple $(v_{(i,j)}, v_{(i,j+1)})$ is part of (i.e., an edge of) exactly one of the hyperedges.

Clearly, by the construction, the sequence hypergraph is acyclic, since the ordering $V^O = (s, v_{(1,1)}, v_{(1,2)}, v_{(1,3)}, \ldots, v_{(m,1)}, v_{(m,2)}, v_{(m,3)}, t)$ is a topological sorting of the underlying graph G.

Let us now observe that there always exists a minimum st-hypercut that does not contain any source or target hyperedges. Notice that in the underlying graph G of $\mathcal{H}$, the only outgoing edge from $v_{(i,1)}$ leads to $v_{(i,2)}$, for $i = 1, \ldots, m$, and similarly, the only incoming edge to $v_{(i,3)}$ comes from $v_{(i,2)}$. Since for each i, the tuple $(v_{(i,j)}, v_{(i,j+1)})$ is an edge of exactly one hyperedge, every source hyperedge $(s, v_{(i,1)})$ in an st-hypercut C can be substituted by the vertex hyperedge containing the edge $(v_{(i,1)}, v_{(i,2)})$; and every target hyperedge $(v_{(i,3)}, t)$ can be substituted by the vertex hyperedge containing the edge $(v_{(i,2)}, v_{(i,3)})$; and the resulting set is an st-hypercut of size equal or smaller than C. Thus, there exists an optimal solution that contains only vertex hyperedges.

Now observe that any minimum st-hypercut C consisting of vertex hyperedges only, must for each $i = 1, \ldots, m$ "hit" either the edge $(v_{(i,1)}, v_{(i,2)})$ or $(v_{(i,2)}, v_{(i,3)})$ (i.e., one of those two edges is an edge of some hyperedge in C). Otherwise, the underlying graph of $(\mathcal{V}, \mathcal{E} \setminus C)$ would contain an st-path $p_i = s, v_{(i,1)}, v_{(i,2)}, v_{(i,3)}, t$.

We show that the described construction gives us an approximation-preserving reduction: for an st-hypercut of size k, we can construct a solution for the vertex cover problem of the same size, and vice versa. Let $\mathcal{S}' \subseteq \mathcal{H}$ be an optimal solution to the instance I' that contains only vertex hyperedges. Recall that there is a direct one-to-one mapping between the vertex hyperedges and the vertices U of the instance I. There is also a direct mapping between each triple $(v_{(i,1)}, v_{(i,2)}, v_{(i,3)})$ and an edge from F. Since the solution $\mathcal{S}'$ hits one of the edges $(v_{(i,1)}, v_{(i,2)})$ or $(v_{(i,2)}, v_{(i,3)})$ for each i, we can use the mapping to construct a solution $\mathcal{S} \subseteq U$ to the instance I of the original minimum vertex cover problem, such that $|\mathcal{S}| = |\mathcal{S}'|$. On the other hand, every solution to the original vertex cover instance can be mapped, using the same direct one-to-one mapping in the opposite direction, to an st-hypercut of $\mathcal{H}$ of the same size. $\square$

6 Sequence Hypergraphs with Hyperedges of Length ≤ 2

We have seen that some of the classic, polynomially solvable problems in (directed) graphs become APX-hard in sequence hypergraphs. Note that this often remains true even if all the hyperedges are of constant length. In particular, Theorem 1 states that the shortest st-hyperpath is hard to approximate even if all the hyperedges are of length at most 5; Fig. 4 illustrates that the duality between minimum st-hypercut and maximum st-hyperflow breaks already with a single hyperedge of length 3; and Theorem 3 yields that the maximum st-hyperflow is hard to approximate even if all hyperedges are of length at most 3.

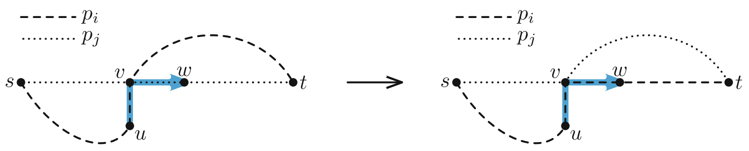


Fig. 7. Transforming st-paths into hyperedge-disjoint st-hyperpaths.

It is an interesting question to investigate the computational complexity of the problems for hyperedge lengths smaller than 5 or 3. We show that, if all the hyperedges of the given sequence hypergraph are of length at most 2, the shortest st-hyperpath, the minimum st-hypercut, and the maximum st-hyperflow can all be found in polynomial time.

Theorem 5. *The shortest st-hyperpath problem in sequence hypergraphs with hyperedges of length at most 2 can be solved in polynomial time.*

Proof. Consider a shortest st-hyperpath P in a given sequence hypergraph with hyperedges of length at most 2. Clearly, whenever both edges of a hyperedge are part of an underlying path of P, they must appear consecutively on it. Thus, all the edges of each hyperedge appear consecutively on any underlying path of P. Therefore, the length of the shortest st-hyperpath P is again the same as the minimum number of switches of P, and such a shortest st-hyperpath can be found in polynomial time (Proposition 1). $\square$

Theorem 6. *The maximum st-hyperflow problem in sequence hypergraphs with hyperedges of length at most 2 can be solved in polynomial time.*

Proof. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a sequence hypergraph with hyperedges of length at most 2, and let s and t be two of its vertices. Then, using standard graph algorithms we can find a maximum st-flow f in the underlying directed multigraph G of $\mathcal{H}$ with edge capacities 1. Thus, the flow f of size $|f|$ gives us a set of $|f|$ edge-disjoint st-paths $p_1, \ldots, p_{|f|}$ in G (note that any directed cycle in f can be easily removed).

We iteratively transform $p_1, \ldots, p_{|f|}$ into a set of st-paths such that all the edges of each hyperedge appear on only one of these paths. Let $E = (u, v, w)$ be a hyperedge that lies on two different paths (see Fig. 7), i.e., $(u, v) \in p_i$ and $(v, w) \in p_j$, for some $i, j \in [|f|]$. Then, p_i consists of an su-path, edge (u, v), and a vt-path. Similarly, p_j consists of an sv-path, edge (v, w), and a wt-path. Since all these paths and edges are pairwise edge-disjoint, by setting p_i to consist of the su-path, edge (u, v), edge (v, w), and the wt-path, and at the same time setting p_j to consist of the sv-path, and the vt-path, we again obtain two edge-disjoint st-paths p_i and p_j. However, now the hyperedge E is present only on p_i. At the same time, since each hyperedge is of length at most 2, all the edges of any hyperedge appear on any st-path consecutively, and any hyperedge that was present on only one of p_i or p_j is not affected by the above rerouting and still is present on one of the two paths only.

Thus, the rerouting decreased the number of hyperedges present on more paths, and after at most $|\mathcal{E}|$ iterations of this transformation we obtain $|f|$ hyperedge-disjoint st-paths, which gives us an st-hyperflow of size $|f|$. It is easy to observe that the size of the hyperflow is bounded from above by the size of the flow in the underlying multigraph. Thus, we obtain a maximum st-hyperflow in $\mathcal{H}$. $\square$

Theorem 7. *The minimum st-hypercut problem in sequence hypergraphs with hyperedges of length at most 2 can be solved in polynomial time.*

Proof. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a sequence hypergraph with hyperedges of length at most 2, and let s and t be two of its vertices. As in proof of Theorem 6, we find a maximum st-flow f (of size $|f|$) in the underlying directed multigraph G of $\mathcal{H}$ and obtain a maximum st-hyperflow F in $\mathcal{H}$ of the same size, i.e., $|F| = |f|$. Since in directed multigraphs the size of the minimum cut equals the size of the maximum flow [11], it follows that we can find $|F|$ edges $e_1, \ldots, e_{|F|}$ of G that form a minimum cut of G. Observe that each of these edges corresponds to exactly one hyperedge. Thus, we obtain a set C of at most $|F|$ hyperedges that forms an st-hypercut. Since the size of any st-hypercut is bounded from below by the size of the hyperflow, C is a minimum st-hypercut. $\square$

Note that we proved Theorem 7 by first constructing an st-hyperflow and then finding an st-hypercut of the same size. Since this is always possible, it follows that the equivalent of the MaxFlow-MinCut Duality Theorem holds in this setting with hyperedges of length at most 2.

7 Sequence Hypergraphs with Backward Hyperedges

We consider a special class of sequence hypergraphs where for every hyperedge, there is the exact same hyperedge, but oriented in the opposite direction.

Definition 6 (Backward Hyperedges). *Let $E = (v_1, v_2, \ldots, v_k)$ be a hyperedge of a sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. We say that E' is a* backward hyperedge[1] *of E if $E' = (v_k, \ldots, v_2, v_1)$. If for every E of $\mathcal{E}$, there is exactly one backward hyperedge in $\mathcal{E}$, we refer to $\mathcal{H}$ as* sequence hypergraph with backward hyperedges.

Such a situation arises naturally in urban public transportation networks, for instance most of the tram lines of the city of Zurich have also a "backward" line (which has the exact same stops as the "forward" line, but goes in the opposite direction). We study the complexities of shortest st-hyperpath, minimum st-hypercut, and maximum st-hyperflow under this setting.

We show that, in this setting, we can find a shortest st-hyperpath in polynomial time. On the other hand, we show that minimum st-hypercut and maximum st-hyperflow remain NP-hard, and we give a 2-approximation algorithm for the

[1] Note, if E' is a backward hyperedge of E, also E is a backward hyperedge of E'.

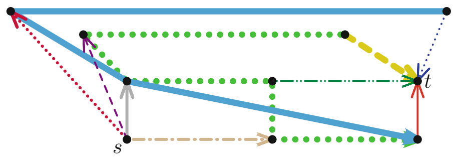


Fig. 8. Sequence hypergraph with backward hyperedges with minimum st-hypercut of size 4, and only three hyperedge-disjoint st-hyperpaths. For every displayed hyperedge, there is also a backward hyperedge, which is for simplicity omitted from the figure.

minimum st-hypercut. Also observe in Fig. 8 that the equivalent of the MaxFlow-MinCut Duality Theorem does not hold in sequence hypergraphs with backward hyperedges. The positive results in this section are based on existing algorithms for standard hypergraphs, the negative results are obtained by a modification of the hardness proofs given in Sects. 4 and 5.

Theorem 8. *The shortest st-hyperpath problem in sequence hypergraphs with backward hyperedges can be solved in polynomial time.*

Proof. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a sequence hypergraph with backward hyperedges, and let s and t be two vertices of $\mathcal{H}$. We construct a (standard) hypergraph $\mathcal{H}^* = (\mathcal{V}^* = \mathcal{V}, \mathcal{E}^*)$ from $\mathcal{H}$ in such a way that for each sequence hyperedge E of $\mathcal{E}$, $\mathcal{E}^*$ contains a (non-oriented) hyperedge E^* that corresponds to the set of vertices of E. Note that E and its backward hyperedge E' consist of the same set of vertices, thus the corresponding E^* and E'^* are the same. A shortest st-hyperpath[2] P^* in the (standard) hypergraph $\mathcal{H}^*$ can be found in polynomial time. Observe that the size of P^* gives us a lower bound $|P^*|$ on the length of the shortest path in the sequence hypergraph $\mathcal{H}$.

In fact, we can construct from P^* an st-hyperpath in $\mathcal{H}$ of size $|P^*|$ as follows. Let us fix p^* to be an underlying path of P^*. Let $(s = v_1, v_2, \ldots, v_{|P^*|+1} = t)$ be a sequence of vertices, subsequence of p^*, such that for each $i = 1, \ldots, |P^*|$, there is a hyperedge E^* in P^* that contains both v_i and v_{i+1}, v_i is the first vertex of E^* seen on p^*, and v_{i+1} is the last vertex of E^* seen on p^*. Since every hyperedge E^* of $\mathcal{E}^*$ corresponds to the set of vertices of some hyperedge E of $\mathcal{E}$, there is a sequence of sequence hyperedges $(E_1, E_2, \ldots, E_{|P^*|})$, $E_i \in \mathcal{E}$, such that v_i, v_{i+1} are vertices in E_i. Since $\mathcal{H}$ is a sequence hypergraph with backward hyperedges, for every hyperedge E of $\mathcal{E}$ and a pair of vertices v_i and v_{i+1} of E, there is a $v_i v_{i+1}$-hyperpath in $\mathcal{H}$ of size 1, which consists of E or its backward hyperedge E'. Therefore, there is an st-hyperpath of size $|P^*|$ in $\mathcal{H}$. □

Theorem 9. *The maximum st-hyperflow problem in sequence hypergraphs with backward hyperedges is* NP*-hard.*

[2] An st-hyperpath P^* and its underlying path are defined as in sequence hypergraphs.

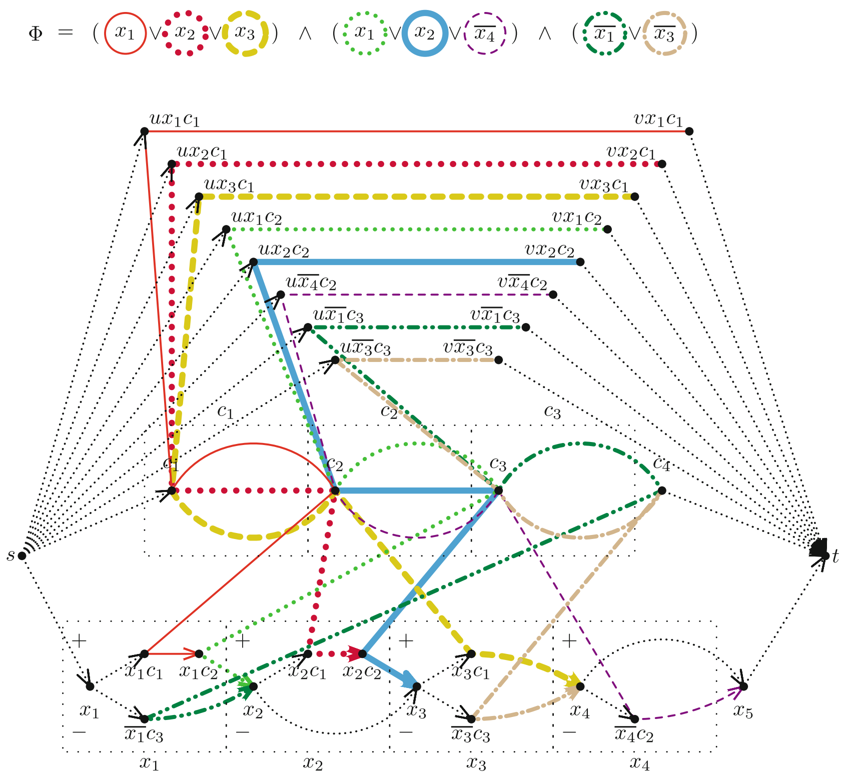


Fig. 9. Maximum *st*-hyperflow in sequence hypergraphs with backward hyperedges is NP-hard.

Proof. We construct a reduction from the NP-complete 3-SAT problem [15]. Let I be an instance of the 3-SAT problem, given as a set of m clauses $C = \{c_1, \ldots, c_m\}$ over a set $X = \{x_1, \ldots, x_n\}$ of Boolean variables.

From I we construct a sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ as follows (cf. Figure 9 along with the construction). The construction is very similar to that in the proof of Theorem 3, so we highlight the changes in bold. One major change is that now for every hyperedge there is a backward hyperedge. For simplicity, we divide all the sequence hyperedges into pairs of mutually backward sequence hyperedges, and we refer to one sequence hyperedge of each pair as *forward* hyperedge (and to the other as its *backward* hyperedge). For simplicity of the construction, we describe explicitly only the forward hyperedges, and each of them implicitly defines a backward hyperedge.

Let $|I|$ be the size of I (i.e., $|I| = \sum_{c_i \in C} |c_i|$). The set of vertices $\mathcal{V}$ consists of $2 + (m+1) + (n+1) + \mathbf{3}|I|$ vertices: a source vertex s and a target vertex t; a vertex c_i for each clause $c_i \in C$ and a dummy vertex c_{m+1}; a vertex x_j for each variable $x_j \in X$ and a dummy vertex x_{n+1}; and finally **three** vertices x_jc_i, $\boldsymbol{ux_jc_i}$, and $\boldsymbol{vx_jc_i}$ for each pair (x_j, c_i) such that $x_j \in c_i$, and similarly,

$\overline{x_j}c_i$, $\boldsymbol{u\overline{x_j}c_i}$, and $\boldsymbol{v\overline{x_j}c_i}$ for each $\overline{x_j} \in c_i$. Let us fix an arbitrary order C^O of the clauses in C. The set of hyperedges $\mathcal{E}$ contains $4 + 2n + \mathbf{3}|I|$ forward hyperedges (plus the same amount of the corresponding backward hyperedges that we do not specify explicitly). There are $2 + |\boldsymbol{I}|$ *source hyperedges:* (s, c_1) and (s, x_1); for each pair (x_j, c_i), $x_j \in c_i$, there is $\boldsymbol{(s, ux_jc_i)}$, and for each $\overline{x_j} \in c_i$, there is $\boldsymbol{(s, u\overline{x_j}c_i)}$. There are $2 + |\boldsymbol{I}|$ *target hyperedges:* (c_{m+1}, t) and (x_{n+1}, t); for each pair (x_j, c_i), $x_j \in c_i$, there is $\boldsymbol{(vx_jc_i, t)}$, and for each $\overline{x_j} \in c_i$, there is $\boldsymbol{(v\overline{x_j}c_i, t)}$. There are $2n$ *auxiliary hyperedges* (x_i, x_ic_k) and $(x_i, \overline{x_i}c_{k'})$ for $i = 1, \ldots, n$, where c_k or $c_{k'}$ is always the first clause (with respect to C^O) containing x_i or $\overline{x_i}$, respectively. In case there is no clause containing x_i (or $\overline{x_i}$), the corresponding auxiliary hyperedge is (x_i, x_{i+1}). Finally, there are $|I|$ *lit-in-clause hyperedges* as follows. For each appearance of a variable x_j in a clause c_i as a positive literal there is one lit-in-clause hyperedge $(\boldsymbol{vx_jc_i, ux_jc_i,} c_i,\ c_{i+1}, x_jc_i, x_jc_k)$, where c_k is the next clause (with respect to C^O) after c_i where x_j appears as a positive literal (in case there is no such c_k, the hyperedge ends in x_{j+1} instead). Similarly, if x_j is in c_i as a negative literal, there is one lit-in-clause hyperedge $(\boldsymbol{v\overline{x_j}c_i, u\overline{x_j}c_i,} c_i,\ c_{i+1}, \overline{x_j}c_i, \overline{x_j}c_k)$, where c_k is the next clause containing the negative literal $\overline{x_j}$ (or the hyperedge ends in x_{j+1}).

We show that the formula I is satisfiable if and only if the sequence hypergraph $\mathcal{H}$ contains $2+|\boldsymbol{I}|$ hyperedge-disjoint st-hyperpaths. Since there are exactly $2 + |\boldsymbol{I}|$ source hyperedges, and no other hyperedge (including backward hyperedges) originates from the source vertex s, all these source hyperedges have to be used to get $2 + |I|$ hyperedge-disjoint st-hyperpaths. Similarly, all the target hyperedges have to be used. But then, each of the $|I|$ vertices vx_jc_i or $v\overline{x_j}c_i$ has to be on one of the underlying st-paths. However, vx_jc_i can only be reached (unless passing via t) from ux_jc_i using a backward hyperedge of a lit-in-clause hyperedge. Similarly, $v\overline{x_j}c_i$ can only be reached from $u\overline{x_j}c_i$ using a backward hyperedge of a lit-in-clause hyperedge. This all implies that there can be $2 + |I|$ hyperedge-disjoint st-hyperpaths only if $|I|$ of them are composed in one of the two following ways: a source hyperedge (s, ux_jc_i), a backward hyperedge of some lit-in-clause hyperedge to get from ux_jc_i to vx_jc_i, and a target hyperedge (vx_jc_i, t); or a source hyperedge $(s, u\overline{x_j}c_i)$, a lit-in-clause backward hyperedge to get from $u\overline{x_j}c_i$ to $v\overline{x_j}c_i$, and a target hyperedge $(v\overline{x_j}c_i, t)$. Thus, the backward hyperedges of all $|I|$ lit-in-clause hyperedges are used and cannot appear in the remaining st-hyperpaths. Also note that all other backward hyperedges are useless to reach t from s, since they lead only backwards. This implies a situation equivalent to that in the proof of Theorem 3. That is, the formula I is satisfiable if and only if the sequence hypergraph $\mathcal{H}$ contains 2 hyperedge-disjoint st-hyperpaths, when considering forward hyperedges only, without the $|I|$ source and $|I|$ target hyperedges already used above.

Then, there are 3 possible types of st-paths in the underlying graph of $\mathcal{H}$: the first one leads through all the vertices $c_1, c_2, \ldots, c_{m+1}$ in this order; the second one leads through all the vertices $x_1, x_2, \ldots, x_{m+1}$ in this order and between x_j and x_{j+1} it goes either through all the x_jc_* vertices or through all the $\overline{x_j}c_*$ vertices (again, $*$ is used here as a wildcart); and the third possible st-path starts the same as the first option and ends as the second one. Based on this observation, notice that there can be at most 2 hyperedge-disjoint st-hyperpaths: necessarily, one of them has an underlying path of the first type, while the other one has an underlying path of the second type.

From a satisfying assignment A of I we can construct the two disjoint st-hyperpaths as follows. One hyperpath leads from s to t via the vertices $c_1, c_2, \ldots, c_{m+1}$, and to move from c_i to c_{i+1} it uses a lit-in-clause hyperedge that corresponds to a pair (l, c_i) such that l is one of the literals that satisfy the clause c_i in A. The second hyperpath has an underlying path of the second type, it leads via $x_1, x_2, \ldots, x_{n+1}$, and from x_j to x_{j+1} it uses the vertices containing only the literals that are not satisfied by the assignment A. Thus, the second hyperpath uses only those lit-in-clause hyperedges that correspond to pairs containing literals that are not satisfied by A. This implies that the two constructed st-hyperpaths are hyperedge-disjoint.

Let P and Q be two hyperedge-disjoint st-hyperpaths of $\mathcal{H}$, considering only the forward hyperedges, without the source hyperedges that lead to vertices ux_*c_* or $u\bar{x_*}c_*$. Let P have an underlying path p of the first type and Q has an underlying path q of the second type. We can construct a satisfying assignment for I by setting to `TRUE` the literals opposite to those that occur in the vertices on q. Then, the hyperpath P suggests how the clauses of I are satisfied by this assignment. □

Theorem 10. *The minimum st-hypercut problem in sequence hypergraphs with backward hyperedges is* NP*-hard.*

Proof. We construct a reduction from the NP-hard vertex cover problem. Let I be an instance of the vertex cover problem, given as an undirected graph $I = (U, F)$, with the vertex set $U = \{u_1, \ldots, u_n\}$, and the edge set $F = \{f_1, \ldots, f_m\}$.

To construct from I an instance I' of the minimum st-hypercut problem in acyclic sequence hypergraphs, we fix an order of the vertices and edges of I as follows. Let $U^O = (u_1, \ldots, u_n)$ be an arbitrary order on the vertices of U. From every edge in F we create an ordered edge, where the vertices of the edge are ordered naturally according to U^O. Let F' denote the set of the created ordered edges, and let F^O be the edges F' ordered lexicographically according to U^O.

We construct the sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ of I' as follows (cf. Figure 10 along with the construction). The construction is very similar to that in the proof of Theorem 4, so we highlight the changes in bold. One major change is that now for every hyperedge there is a backward hyperedge. For simplicity, we divide all the sequence hyperedges into pairs of mutually backward sequence hyperedges, and we refer to one sequence hyperedge of each pair as *forward* hyperedge (and to the other as its backward hyperedge). For simplicity of the construction, we describe explicitly only the forward hyperedges, and each of them implicitly defines a backward hyperedge.

The set of vertices $\mathcal{V}$ consists of $3m + 2 + \boldsymbol{2n}$ vertices: a source vertex s and a target vertex t; for each edge $f_i \in F^O$, $i = 1, \ldots, m$, there are three vertices $v_{(i,1)}, v_{(i,2)}$, and $v_{(i,3)}$; and finally, for each vertex $u_k \in U^O$, $k = 1, \ldots, n$, there

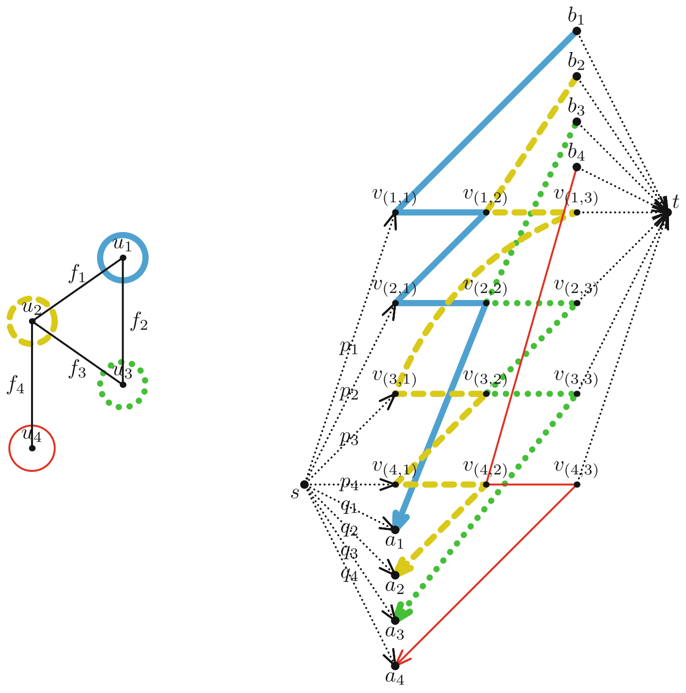


Fig. 10. Minimum *st*-hypercut in sequence hypergraphs with backward hyperedges is NP-hard.

are two vertices $\boldsymbol{a_k}$ and $\boldsymbol{b_k}$. The set of hyperedges $\mathcal{E}$ contains $2m + \boldsymbol{3}n$ forward hyperedges (plus the same amount of the corresponding backward hyperedges that we do not specify explicitly). There are m *type-1 source hyperedges* of the form $(s, v_{(i,1)})$, each of them connects s to one vertex $v_{(i,1)}$, and there are n *type-2 source hyperedges* of the form $\boldsymbol{(s, a_k)}$, connecting s to $\boldsymbol{a_k}$, for $k = 1, \dots, n$. There are m *type-1 target hyperedges* of the form $(v_{(i,3)}, t)$, each of them connects one vertex $v_{(i,3)}$ to t, and there are n *type-2 hyperedges* of the form $\boldsymbol{(b_k, t)}$, connecting $\boldsymbol{b_k}$ to t, for $k = 1, \dots, n$. And finally, there are n *vertex hyperedges*, each corresponds to one of the vertices in U, that are constructed as follows.

For a vertex $u_k \in U$ of the graph I, we describe the sequence of vertices in the corresponding vertex hyperedge E_{u_k} iteratively. We start with a sequence containing $E_{u_k} = \boldsymbol{b_k}$. We consider the edges F^O in order, and for each edge $f_i \in F^O$ that contains u_k we prolong the sequence of E_{u_k} as follows. If f_i contains u_k as the first vertex, we append $v_{(i,1)}$ and $v_{(i,2)}$ to E_{u_k}. Otherwise, f_i contains u_k as the second vertex, and we append $v_{(i,2)}$ and $v_{(i,3)}$ to E_{u_k}. Once all edges containing u_k have been considered, we append $\boldsymbol{a_k}$ to E_{u_k}. We denote by E'_{u_k} the corresponding backward vertex hyperedge.

Now consider the edges of the obtained hyperedge E_{u_k} and note the form they can have. The first edge is an edge $(\boldsymbol{b_k}, v_{(i,j)})$ for some $j \in \{1, 2\}$ and some $i \in [m]$. The last edge is an edge $(\boldsymbol{v_{(i,j)}}, a_k)$ for some $j \in \{2, 3\}$ and some

$i \in [m]$. The remaining edges are of two main types: There are edges of the form $(v_{(i,1)}, v_{(i,2)})$ and $(v_{(i,2)}, v_{(i,3)})$ for some $i \in [m]$, and there are edges of the form $(v_{(i,2)}, v_{(j,1)})$, $(v_{(i,3)}, v_{(j,2)})$, and $(v_{(i,3)}, v_{(j,1)})$ for some $i < j$, $i, j \in [m]$. Note that, due to the fact that the edges in F^O are ordered lexicographically, all the edges of the vertex hyperedge E_{u_k} take one of the forms described above. Also note that, due to the direct correspondence between the vertices in U and vertex hyperedges, each tuple $(v_{(i,j)}, v_{(i,j+1)})$ is part of (i.e., an edge of) exactly one of the hyperedges.

Let us now observe that there always exists a minimum st-hypercut that does not contain any type-2 source hyperedges or type-2 target hyperedges. Clearly, none of the backward type-2 source hyperedges or type-2 target hyperedges are in a minimum st-hypercut, since each consists of a single edge that either leads to s or from t. Notice that in the underlying graph G of $\mathcal{H}$, for each $k = 1, \ldots, n$, there is only one outgoing edge from $\boldsymbol{a_k}$ (other than to s), and this edge belongs only to the backward vertex hyperedge E'_{u_k}. Similarly, there is only one incoming edge from $\boldsymbol{b_k}$ (other than from t) and this edge belongs only to the backward vertex hyperedge E'_{u_k}. Consequently, every forward type-2 source hyperedge or type-2 target hyperedge in an st-hypercut C can be substituted for a backward vertex hyperedge, and the resulting set is an st-hypercut of size equal to or smaller than C. Thus, there exists a minimum st-hypercut C which does not contain type-2 source or type-2 target hyperedges. Now observe that such an st-hypercut C has to contain *all* the backward vertex hyperedges, as otherwise, for some k, the underlying graph of $(\mathcal{V}, \mathcal{E} \setminus C)$ would contain an st-path using the type-2 source hyperedge from s to $\boldsymbol{a_k}$, the backward vertex hyperedge E'_{u_k} from $\boldsymbol{a_k}$ to $\boldsymbol{b_k}$, and the type-2 target hyperedge from $\boldsymbol{b_k}$ to t. Now we show that each type-1 source or type-1 target hyperedge in C can be substituted by a forward vertex hyperedge. Let B be the set of all backward vertex hyperedges. Notice that in the underlying graph of $(\mathcal{V}, \mathcal{E} \setminus B)$, the only outgoing edges from $v_{(i,1)}$ lead to s or $v_{(i,2)}$, for $i = 1, \ldots, m$; and similarly, the only incoming edges to $v_{(i,3)}$ come from t or $v_{(i,2)}$. (And clearly, the hyperedges $(v_{(i,1)}, s)$ and $(t, v_{(i,3)})$ are not part of any st-hyperpath.) Since for each i, the tuple $(v_{(i,j)}, v_{(i,j+1)})$ is an edge of exactly one hyperedge, every type-1 source hyperedge $(s, v_{(i,1)})$ in an st-hypercut C can be substituted by the forward vertex hyperedge containing the edge $(v_{(i,1)}, v_{(i,2)})$; and every type-1 target hyperedge $(v_{(i,3)}, t)$ can be substituted for the forward vertex hyperedge containing the edge $(v_{(i,2)}, v_{(i,3)})$, and the resulting set is an st-hypercut of size equal to or smaller than C. Thus, there exists an optimum solution that contains only vertex hyperedges, and in particular, it contains all the backward vertex hyperedges.

Now observe that any minimum st-hypercut C consisting of vertex hyperedges only must for each $i = 1, \ldots, m$ "hit" either the edge $(v_{(i,1)}, v_{(i,2)})$ or $(v_{(i,2)}, v_{(i,3)})$ (i.e., one of those two edges is an edge of some hyperedge in C). Otherwise, the underlying graph of $(\mathcal{V}, \mathcal{E} \setminus C)$ would contain an st-path $p_i = s, v_{(i,1)}, v_{(i,2)}, v_{(i,3)}, t$.

Let $\mathcal{S}' \subseteq \mathcal{H}$ be an optimum solution to the instance I' that contains only vertex hyperedges. Since $\mathcal{S}'$ contains all n backward vertex hyperedges, it contains

$|\mathcal{S}'| - n$ forward vertex hyperedges. Recall that there is a direct one-to-one mapping between the forward vertex hyperedges and the vertices U of the instance I. There is also a direct mapping between each triple $(v_{(i,1)}, v_{(i,2)}, v_{(i,3)})$ and an edge from F. Since the solution $\mathcal{S}'$ hits one of the edges $(v_{(i,1)}, v_{(i,2)})$ or $(v_{(i,2)}, v_{(i,3)})$ for each i, we can use the mapping to construct a solution $\mathcal{S} \subseteq U$ to the instance I of the original minimum vertex cover problem, such that $|\mathcal{S}| = |\mathcal{S}'| - n$. On the other hand, every solution of size k to the original vertex cover instance can be mapped, using the same direct one-to-one mapping in the opposite direction, to an st-hypercut of $\mathcal{H}$ of size $k + n$. Therefore, an optimum solution for I' gives us an optimum solution for I. □

Theorem 11. *The minimum st-hypercut problem in sequence hypergraphs with backward hyperedges can be 2-approximated.*

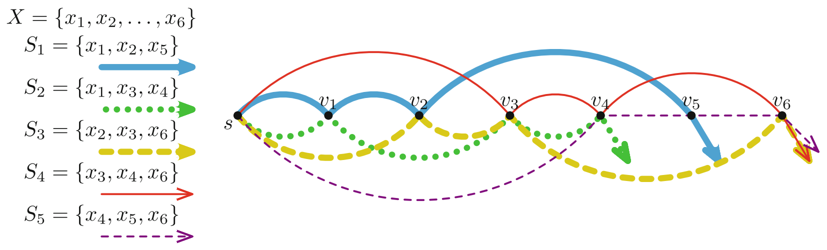

$X = \{x_1, x_2, \ldots, x_6\}$
$S_1 = \{x_1, x_2, x_5\}$
$S_2 = \{x_1, x_3, x_4\}$
$S_3 = \{x_2, x_3, x_6\}$
$S_4 = \{x_3, x_4, x_6\}$
$S_5 = \{x_4, x_5, x_6\}$
s
v_1
v_2
v_3
v_4
v_5
v_6

Fig. 11. Minimum s-rooted spanning hypergraph is at least as hard as the minimum set cover problem.

Proof. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a sequence hypergraph with backward hyperedges, and let s and t be two vertices of $\mathcal{H}$. Note that we can partition $\mathcal{E}$ into $|\mathcal{E}|/2$ pairs of hyperedges (each pair contains a hyperedge and its backward hyperedge). We construct a (standard) hypergraph $\mathcal{H}^* = (\mathcal{V}^* = \mathcal{V}, \mathcal{E}^*)$ from $\mathcal{H}$ in such a way that for each pair of mutually backward sequence hyperedges E and E' of $\mathcal{E}$, $\mathcal{E}^*$ contains exactly one (non-oriented) hyperedge E^* that corresponds to the set of vertices of E (and thus also E'). Note that $\mathcal{E}^*$ may contain multiple hyperedges corresponding to the same set of vertices, and we get that $\mathcal{E}^*$ contains exactly $|\mathcal{E}|/2$ hyperedges.

Next, we find a minimum st-hypercut X^* in $\mathcal{H}^*$ (this can be done in polynomial time by a transformation of $\mathcal{H}^*$ into a directed graph and solving a maximum flow problem in it). Clearly, $|X^*|$ is a lower bound on the size of a minimum st-hypercut in $\mathcal{H}$. Recall that every hyperedge in X^* corresponds to 2 sequence hyperedges in $\mathcal{H}$, thus, by removing all sequence hyperedges corresponding to hyperedges in X^*, we obtain an st-hypercut in $\mathcal{H}$ of size $2\,|X^*|$. □

8 On Other Algorithmic Problems

We briefly consider some other standard graph algorithmic problems in sequence hypergraphs.

Definition 7 (Rooted Spanning Hypergraph). *Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a sequence hypergraph. An* s-rooted spanning hypergraph *T is a subset of $\mathcal{E}$ such that for every $v \in \mathcal{V}$, T is an sv-hyperpath. The* size *of T is $|T|$.*

Theorem 12. *Finding a minimum s-rooted spanning hypergraph in acyclic sequence hypergraphs is* NP*-hard to approximate within a factor of $(1-\varepsilon)\ln n$, unless* P = NP*.*

Proof. We construct an approximation-preserving reduction from the set cover problem, which has the claimed inapproximability [10]. Let $I = (X, \mathcal{S})$ be an instance of the set cover problem, given by a ground set $X = \{x_1, \ldots, x_n\}$, and a family of its subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$.

We construct from I an instance I' of the minimum s-rooted spanning hypergraph problem in sequence hypergraphs (see Fig. 11 for an example). Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be the following sequence hypergraph. The set of vertices $\mathcal{V} = \{s, v_1, \ldots, v_n\}$ contains one vertex v_i for each $x_i \in X$, and a source vertex s. Let V^O be an arbitrary fixed order on the vertices $\mathcal{V}$. There are m hyperedges in $\mathcal{E} = \{E_1, \ldots, E_m\}$, one for each set in $\mathcal{S}$. For a set $S_i \in \mathcal{S}$, let us take the vertices that correspond to the elements in S_i and order them according to V^O to obtain $(v_{i_1}, v_{i_2}, \ldots, v_{i_r})$. Then the sequence of the hyperedge E_i corresponding to S_i is simply $(s, v_{i_1}, v_{i_2}, \ldots, v_{i_r})$. The constructed sequence hypergraph is acyclic, as all the edges of its hyperedges agree with V^O.

It remains to show that from a minimum s-rooted spanning hypergraph T for I' a minimum set cover for I of the same size can be constructed. By definition, T is the smallest subset of $\mathcal{E}$ that allows to reach from s any other vertex in $\mathcal{V}$. Moreover, since $\mathcal{H}$ is acyclic and each hyperedge starts at s, for each vertex $v \in \mathcal{V}$, it is enough to choose one of the hyperedges in $\mathcal{E}$ (containing v) to reach v from s. This, together with the one-to-one correspondence between the hyperedges in $\mathcal{E}$ and sets in $\mathcal{S}$, provides us with a prescription for a minimum set cover for I of size $|T|$. On the other hand, we can map any solution of the original minimum set cover instance to a minimum s-rooted spanning hypergraph of the same size. □

Definition 8 (Strongly Connected Component). *Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a sequence hypergraph. We say that a set $C \subseteq \mathcal{E}$ forms a* strongly connected component *if for every two vertices $u, v \in \mathcal{V}'$, with $\mathcal{V}'$ being all the vertices of $\mathcal{V}$ present in C, the set C contains a uv-hyperpath. We say that the vertices in $\mathcal{V}'$ are* covered *by C.*

Clearly, we can decide in polynomial time whether a given set of hyperedges C forms a strongly connected component as follows. Consider the underlying graph G of $\mathcal{H}$ induced by the set of sequence hyperedges C and find a maximum strongly connected component there. If this component spans the whole G, then C is a strongly connected component in $\mathcal{H}$.

Theorem 13. *Given a sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, it is* NP*-hard to find a* minimum *number of hyperedges that form a strongly connected component C so that (a) C is any non-empty set, or (b)* all *the vertices in $\mathcal{V}$ are covered by C.*

Proof. Variant (b) is clearly NP-hard by a reduction from the Hamiltonian cycle problem: Consider a standard directed graph $G = (V, E)$ and view it as a sequence hypergraph, where each sequence hyperedge is just an edge of G. Then, by finding a strongly connected component that covers all the vertices of V and uses the minimum number of sequence hyperedges (i.e., normal edges) we can solve the Hamiltonian cycle problem in G (either the component consists of $|V|$ edges or more).

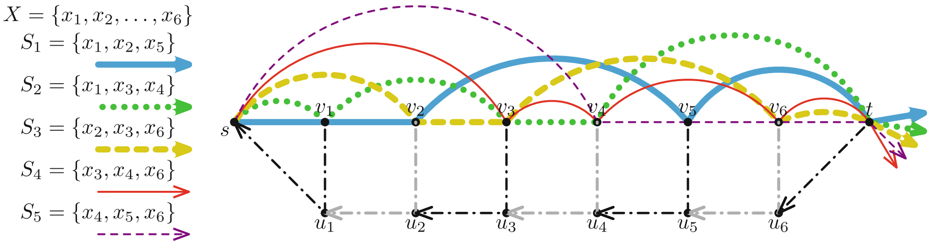

$X = \{x_1, x_2, \ldots, x_6\}$
$S_1 = \{x_1, x_2, x_5\}$
$S_2 = \{x_1, x_3, x_4\}$
$S_3 = \{x_2, x_3, x_6\}$
$S_4 = \{x_3, x_4, x_6\}$
$S_5 = \{x_4, x_5, x_6\}$
s
t
v_1
v_2
v_3
v_4
v_5
v_6
u_1
u_2
u_3
u_4
u_5
u_6

Fig. 12. For a given sequence hypergraph, it is NP-hard to find a minimum non-empty set of sequence hyperedges that forms a strongly connected component.

We show NP-hardness of variant (a) by a reduction from the APX-hard set cover problem [10]. Let $I = (X, \mathcal{S})$ be an instance of the set cover problem, given by a ground set $X = \{x_1, \ldots, x_n\}$, and a family of its subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$.

From I we construct a sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ as follows (see Fig. 12 for an example). The set of vertices $\mathcal{V} = \{s, t, v_1 \ldots, v_n, u_1 \ldots, u_n\}$ contains one vertex v_i and one vertex u_i for each $x_i \in X$, a source vertex s, and a target vertex t. Let X^O be an arbitrary fixed order on the elements of X. The set of sequence hyperedges $\mathcal{E}$ consists of $m + n + 1$ hyperedges. There are n *element hyperedges*: for each element x_i in X (i-th element of X in the order X^O), there is a hyperedge (v_i, u_i, u_{i-1}), the hyperedge (v_1, u_1, s) corresponding to x_1 ends at s instead. There is one *target hyperedge* (t, u_n). Finally, there are m *set hyperedges*, one for each set in $\mathcal{S}$ defined as follows. For every set $S_i \in \mathcal{S}$, we take the vertices from $\{v_1, \ldots, v_n\}$ that correspond to the elements in S_i and order them according to X^O to obtain $(v_{i_1}, v_{i_2}, \ldots, v_{i_r})$. Then the sequence of the hyperedge E_i corresponding to S_i is simply $(s, v_{i_1}, v_{i_2}, \ldots, v_{i_r}, t)$.

It remains to show that a minimum set cover for I can be constructed from a minimum non-empty set of sequence hyperedges of $\mathcal{H}$ that form a strongly connected component. Let O be such a non-empty set of hyperedges. First observe that if O contains any set hyperedge, then O must contain *all* the element hyperedges and the target hyperedge. This follows from the fact that if O contains a set hyperedge, then s and t are covered by O, and O must contain a ts-hyperpath, but the only ts-hyperpath clearly consists of all element hyperedges plus the target hyperedge. Clearly, if O contains an element hyperedge or a target hyperedge, then either some v_i or t is covered by O, and thus O must contain a hyperpath that leads to v_i or t, respectively. However, the only sequence hyperedges that can reach v_i or t are the set hyperedges. Therefore, O always contains at least one set hyperedge and all the element hyperedges and the target hyperedge. However, this implies that *all* the vertices have to be covered by O, which, in particular, implies that O must contain an sv_i-hyperpath for each v_i. Therefore, O consists of all element hyperedges, the target hyperedge, and the smallest subset of the set hyperedges that allows to reach from s any vertex $v_i \in \mathcal{V}$. In other words, for each vertex $v_i \in \mathcal{V}$, O must contain a set hyperedge that can reach v_i from s. This, together with the one-to-one correspondence between the set hyperedges in $\mathcal{E}$ and sets in $\mathcal{S}$, provides us with a prescription for a minimum set cover for I of size $|O| - n - 1$. On the other hand, using the same mapping in the opposite direction, we can map any solution of the original minimum set cover instance to a minimum non-empty set of sequence hyperedges of $\mathcal{H}$ that form a strongly connected component. □

Theorem 14. *Given a sequence hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, finding a* maximum *number of hyperedges that form a strongly connected component C so that (a) C is any non-empty set, or (b)* all *the vertices in $\mathcal{V}$ are covered by C, is polynomial-time solvable.*

Proof. As discussed above, for a given set of hyperedges, we can decide in polynomial time whether it forms a strongly connected component or not. Thus, in particular, we can check whether the set $\mathcal{E}$ forms a strongly connected component. In case it does, variant (b) is solved trivially by taking *all* the hyperedges in $\mathcal{E}$. Otherwise, this variant has no solution, because there is no strongly connected component covering all vertices in $\mathcal{V}$.

To solve variant (a), we start with the set of all hyperedges $\mathcal{E}$ and during the process, we iteratively remove those hyperedges that cannot be part of a feasible solution. Iteratively, we find strongly connected components in the underlying graph induced by the current set of hyperedges and remove all the sequence hyperedges that occur in two or more components (as these cannot be a part of a single strongly connected component), and repeat. Once no more hyperedges can be removed, we reached a situation, where each of the remaining hyperedges is in exactly one strongly connected component. Thus, the solution is defined by the component that contains the maximum number of hyperedges. □

References

1. Ausiello, G., Franciosa, P.G., Frigioni, D.: Directed hypergraphs: problems, algorithmic results, and a novel decremental approach. ICTCS 2001. LNCS, vol. 2202, pp. 312–328. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45446-2_20
2. Ausiello, G., Giaccio, R., Italiano, G.F., Nanni, U.: Optimal traversal of directed hypergraphs. Technical report (1992)

3. Berry, J., Dean, N., Goldberg, M., Shannon, G., Skiena, S.: Graph drawing and manipulation with *LINK*. In: DiBattista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 425–437. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63938-1_87
4. Böhmová, K., Mihalák, M., Pröger, T., Sacomoto, G., Sagot, M.-F.: Computing and listing *st*-paths in public transportation networks. In: Kulikov, A.S., Woeginger, G.J. (eds.) CSR 2016. LNCS, vol. 9691, pp. 102–116. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-34171-2_8
5. Broersma, H., Li, X., Woeginger, G., Zhang, S.: Paths and cycles in colored graphs. Australas. J. Comb. **31**, 299–311 (2005)
6. Carr, R.D., Doddi, S., Konjevod, G., Marathe, M.V.: On the red-blue set cover problem. In: SODA 2000, vol. 9, pp. 345–353. Citeseer (2000)
7. Coudert, D., Datta, P., Pérennes, S., Rivano, H., Voge, M.E.: Shared risk resource group complexity and approximability issues. Parallel Process. Lett. **17**(02), 169–184 (2007)
8. Coudert, D., Pérennes, S., Rivano, H., Voge, M.E.: Combinatorial optimization in networks with shared risk link groups. Research report, INRIA, October 2015
9. Crescenzi, P., Kann, V., Halldórsson, M., Karpinski, M., Woeginger, G.: A compendium of NP optimization problems (1997). http://www.nada.kth.se/~viggo/problemlist/compendium.html
10. Dinur, I., Steurer, D.: Analytical approach to parallel repetition. In: STOC 2014, pp. 624–633. ACM, New York (2014)
11. Elias, P., Feinstein, A., Shannon, C.E.: A note on the maximum flow through a network. IRE Trans. Inf. Theory **2**(4), 117–119 (1956)
12. Erdős, P.L., Frankl, P., Katona, G.O.: Intersecting sperner families and their convex hulls. Combinatorica **4**(1), 21–34 (1984)
13. Fellows, M.R., Guo, J., Kanj, I.A.: The parameterized complexity of some minimum label problems. In: Paul, C., Habib, M. (eds.) WG 2009. LNCS, vol. 5911, pp. 88–99. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11409-0_8
14. Gallo, G., Longo, G., Pallottino, S., Nguyen, S.: Directed hypergraphs and applications. Discrete Appl. Math. **42**(2), 177–201 (1993)
15. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
16. Goldberg, P.W., McCabe, A.: Shortest paths with bundles and non-additive weights is hard. In: Spirakis, P.G., Serna, M. (eds.) CIAC 2013. LNCS, vol. 7878, pp. 264–275. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38233-8_22
17. Hassin, R., Monnot, J., Segev, D.: Approximation algorithms and hardness results for labeled connectivity problems. J. Comb. Optim. **14**(4), 437–453 (2007)
18. Hu, J.Q.: Diverse routing in optical mesh networks. IEEE Trans. Commun. **51**(3), 489–494 (2003)
19. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within 2-epsilon. J. Comput. Syst. Sci. **74**(3), 335–349 (2008)
20. Krumke, S.O., Wirth, H.C.: On the minimum label spanning tree problem. Inf. Process. Lett. **66**(2), 81–85 (1998)
21. Wachman, G., Khardon, R.: Learning from interpretations: a rooted kernel for ordered hypergraphs. In: Proceedings of the 24th International Conference on Machine Learning, pp. 943–950. ACM (2007)
22. Yuan, S., Varma, S., Jue, J.P.: Minimum-color path problems for reliability in mesh networks. In: INFOCOM 2005, vol. 4, pp. 2658–2669. IEEE (2005)
23. Zhang, P., Cai, J.Y., Tang, L.Q., Zhao, W.B.: Approximation and hardness results for label cut and related problems. J. Comb. Optim. **21**(2), 192–208 (2011)

Relative Worst-Order Analysis: A Survey

Joan Boyar(✉), Lene M. Favrholdt, and Kim S. Larsen

University of Southern Denmark, Odense, Denmark
{joan,lenem,kslarsen}@imada.sdu.dk

Abstract. Relative worst-order analysis is a technique for assessing the relative quality of online algorithms. We survey the most important results obtained with this technique and compare it with other quality measures.

1 Introduction

Online problems are optimization problems where the input arrives one request at a time, and each request must be processed without knowledge of future requests. The investigation of online algorithms was largely initiated by the introduction of competitive analysis by Sleator and Tarjan [61]. They introduced the method as a general analysis technique, inspired by approximation algorithms. The term "competitive" is from Karlin et al. [51] who named the worst-case ratio of the performance of the online to the offline algorithm the "competitive ratio". Many years earlier, Graham carried out what is now viewed as an example of a competitive analysis [44].

The over-all goal of a theoretical quality measure is to predict behavior of algorithms in practice. In that respect, competitive analysis works well in some cases, but, as pointed out by the inventors [61] and others, fails to discriminate between good and bad algorithms in other cases. Ever since its introduction, researchers have worked on improving the measure, defining variants, or defining measures based on other concepts to improve on the situation. Relative worst-order analysis (RWOA), a technique for assessing the relative quality of online algorithms, is one of the most thoroughly tested such proposals.

RWOA was originally defined by Boyar and Favrholdt [17], and the definitions were extended together with Larsen [20]. As for all quality measures, an important issue is to be able to separate algorithms, i.e., determine which of two algorithms is the best. RWOA has been shown to be applicable to a wide variety of problems and provide separations, not obtainable using competitive analysis, corresponding better to experimental results or intuition in many cases.

In this survey, we motivate and define RWOA, outline the background for its introduction, survey the most important results, and compare it to other measures.

Supported in part by the Independent Research Fund Denmark, Natural Sciences, grants DFF-7014-00041.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 216–230, 2018.
https://doi.org/10.1007/978-3-319-98355-4_13

2 Relative Worst-Order Analysis

As a motivation for RWOA, consider the following desirable property of a quality measure for online algorithms: For a given problem P and two algorithms $\mathbb{A}$ and $\mathbb{B}$ for P, if $\mathbb{A}$ performs at least as well as $\mathbb{B}$ on every possible request sequence and better on many, then the quality measure indicates that $\mathbb{A}$ is better than $\mathbb{B}$. We consider an example of such a situation for the paging problem.

2.1 A Motivating Example

In the paging problem, there is a cache with k pages and a larger, slow memory with $N > k$ pages. The request sequence consists of page numbers in $\{1, \ldots, N\}$. When a page is requested, if it is not among the at most k pages in cache, there is a fault, and the missing page must be brought into cache. If the cache is full, this means that some page must be evicted from the cache. The goal is to minimize the number of faults. Clearly, the only thing we can control algorithmically is the eviction strategy.

We consider two paging algorithms, LRU (Least-Recently-Used) and FWF (Flush-When-Full). On a fault with a full cache, LRU always evicts its least recently used page from cache. FWF, on the other hand, evicts everything from cache in this situation. It is easy to see that, if run on the same sequence, whenever LRU faults, FWF also faults, so LRU performs at least as well as FWF on every sequence. LRU usually faults less than FWF [64]. It is well known that LRU and FWF both have competitive ratio k, so competitive analysis does not distinguish between them, and there are relatively few measures which do. RWOA, however, is one such measure [20]. In Subsect. 3.1, we consider LRU and FWF in greater detail to give a concrete example of RWOA.

2.2 Background and Informal Description

Table 1 gives informal "definitions" of the relative worst-order ratio and related measures. The ratios shown in the table capture the general ideas, although they do not reflect that the measures are asymptotic measures. We discuss the measures below, ending with a formal definition of the relative worst-order ratio.

RWOA compares two online algorithms directly, rather than indirectly by first comparing both to an optimal offline algorithm. When differentiating between online algorithms is the goal, performing a direct comparison between the algorithms can be an advantage; first comparing both to an optimal offline algorithm and then comparing the results, as many performance measures including competitive analysis do, can lead to a loss of information. This appears to be at least part of the problem when comparing LRU to FWF with competitive analysis, which finds them equally bad. Measures comparing directly, such as RWOA, bijective and average analysis [5], and relative interval analysis [37], would generally indicate correctly that LRU is the better algorithm.

Table 1. Simplified "definitions" of measures

Measure	Value
Competitive ratio	$\mathrm{CR}_{\mathbb{A}} = \sup_{I} \frac{\mathbb{A}(I)}{\textsc{Opt}(I)}$
Max/max ratio	$\mathrm{MR}_{\mathbb{A}} = \frac{\max_{\lvert I\rvert=n} \mathbb{A}(I)}{\max_{\lvert I'\rvert=n} \textsc{Opt}(I')}$
Random-order ratio	$\mathrm{RR}_{\mathbb{A}} = \sup_{I} \frac{E_{\pi}\left[\mathbb{A}(\pi(I))\right]}{\textsc{Opt}(I)}$
Relative worst-order ratio	$\mathrm{WR}_{\mathbb{A},\mathbb{B}} = \sup_{I} \frac{\sup_{\pi} \left\{\mathbb{A}(\pi(I))\right\}}{\sup_{\pi'} \left\{\mathbb{B}(\pi'(I))\right\}}$

Up to permutations of the request sequences, if an algorithm is always at least as good and sometimes better than another, RWOA separates them. RWOA compares two algorithms on their respective worst orderings of sequences having the same content. This is different from competitive analysis where an algorithm and Opt are compared on the same sequence. When comparing algorithms directly, using exactly the same sequences will tend to produce the result that many algorithms are not comparable, because one algorithm does well on one type of sequence, while the other does well on another type. In addition, comparing on possibly different sequences can make it harder for the adversary to produce unwanted, pathological sequences which may occur seldom in practice, but skew the theoretical results. Instead, with RWOA, online algorithms are compared directly to each other on their respective worst permutations of the request sequences. This comparison in RWOA combines some of the desirable properties of the max/max ratio [9] and the random-order ratio [52].

The Max/Max Ratio. With the max/max ratio defined by Ben-David and Borodin, an algorithm is compared to Opt on its and Opt's respective worst-case sequences of the same length. Since Opt's worst sequence of any given length is the same, regardless of which algorithm it is being compared to, comparing two online algorithms directly gives the same result as dividing their max/max ratios. Thus, the max/max ratio allows direct comparison of two online algorithms, to some extent, without the intermediate comparison to Opt. The max/max ratio can only provide interesting results when the length of an input sequence yields a bound on the cost/profit of an optimal solution.

In the paper [9] introducing the max/max ratio, the k-server problem is analyzed. This is the problem where k servers are placed in a metric space, and the input is a sequence of requests to points in that space. At each request, a server must be moved to the requested point if there is not already a server at the point. The objective is to minimize the total distance the servers are moved. It is demonstrated that, for k-server on a bounded metric space, the max/max ratio can provide more optimistic and detailed results than competitive analysis.

Unfortunately, there is still the loss of information as generally occurs with the indirect comparison to Opt, and the max/max ratio does not distinguish between LRU and FWF, or actually between any two deterministic online paging algorithms.

However, the possibility of directly comparing online algorithms and comparing them on their respective worst-case sequences from some partition of the space of request sequences was inspirational. RWOA uses a more fine-grained partition than partitioning with respect to the sequence length. The idea for the specific partition used stems from the random-order ratio.

The Random-Order Ratio. The random-order ratio was introduced in [52] by Kenyon (now Mathieu). The appeal of this quality measure is that it allows considering some randomness of the input sequences without specifying a complete probability distribution. It was introduced in connection with bin packing, i.e., the problem of packing items of sizes between 0 and 1 into as few bins of size 1 as possible. For an algorithm $\mathbb{A}$ for this minimization problem, the random-order ratio is the maximum ratio, over all multi-sets of items, of the expected performance, over all permutations of the multi-set, of $\mathbb{A}$ compared with an optimal solution; see also Table 1. If, for all possible multi-sets of items, any permutation of these items is equally likely, this ratio gives a meaningful worst-case measure of how well an algorithm can perform.

In the paper introducing the random-order ratio, it was shown that for bin packing, the random-order ratio of Best-Fit lies between 1.08 and 1.5. In contrast, the competitive ratio of Best-Fit is 1.7 [49].

Random-order analysis has also been applied to other problems, e.g., knapsack [7], bipartite matching [35,43], scheduling [42,59], bin covering [29,41], and facility location [56]. However, the analysis is often rather challenging, and in [32], a simplified version of the random-order ratio is used for bin packing.

2.3 Definitions

Let I be a request sequence of length n for an online problem P. If π is a permutation on n elements, then $\pi(I)$ denotes I permuted by π.

If P is a minimization problem, $\mathbb{A}(I)$ denotes the cost of the algorithm $\mathbb{A}$ on the sequence I, and

$$\mathbb{A}_W(I) = \max_{\pi} \mathbb{A}(\pi(I)),$$

where π ranges over the set of all permutations of n elements.

If P is a maximization problem, $\mathbb{A}(I)$ denotes the profit of the algorithm $\mathbb{A}$ on the sequence I, and

$$\mathbb{A}_W(I) = \min_{\pi} \mathbb{A}(\pi(I)).$$

Informally, RWOA compares two algorithms, $\mathbb{A}$ and $\mathbb{B}$, by partitioning the set of request sequences as follows: Sequences are in the same part of the partition if and only if they are permutations of each other. The relative worst-order ratio is defined for algorithms $\mathbb{A}$ and $\mathbb{B}$, whenever one algorithm performs at least as well as the other on every part of the partition, i.e., whenever $\mathbb{A}_W(I) \leq \mathbb{B}_W(I)$, for all request sequences I, or $\mathbb{A}_W(I) \geq \mathbb{B}_W(I)$, for all request sequences I (in the definition below, this corresponds to $c_u(\mathbb{A},\mathbb{B}) \leq 1$ or $c_l(\mathbb{A},\mathbb{B}) \geq 1$). In this case, to compute the relative worst-order ratio of $\mathbb{A}$ to $\mathbb{B}$, we compute a bound ($c_l(\mathbb{A},\mathbb{B})$ or $c_u(\mathbb{A},\mathbb{B})$) on the ratio of how the two algorithms perform on their respective worst permutations of some sequence. Note that the two algorithms may have different worst permutations for the same sequence.

We now state the formal definition:

Definition 1. *For any pair of algorithms $\mathbb{A}$ and $\mathbb{B}$, we define*

$$\begin{aligned} c_l(\mathbb{A},\mathbb{B}) &= \sup\,\{c \mid \exists b\,\forall I\colon \mathbb{A}_W(I) \geq c\,\mathbb{B}_W(I) - b\} \text{ and} \\ c_u(\mathbb{A},\mathbb{B}) &= \inf\,\{c \mid \exists b\,\forall I\colon \mathbb{A}_W(I) \leq c\,\mathbb{B}_W(I) + b\}. \end{aligned}$$

If $c_l(\mathbb{A},\mathbb{B}) \geq 1$ or $c_u(\mathbb{A},\mathbb{B}) \leq 1$, the algorithms are said to be comparable *and the* relative worst-order ratio $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ *of algorithm $\mathbb{A}$ to algorithm $\mathbb{B}$ is defined as*

$$\mathrm{WR}_{\mathbb{A},\mathbb{B}} = \begin{cases} c_u(\mathbb{A},\mathbb{B}), & \text{if } c_l(\mathbb{A},\mathbb{B}) \geq 1, \text{ and} \\ c_l(\mathbb{A},\mathbb{B}), & \text{if } c_u(\mathbb{A},\mathbb{B}) \leq 1. \end{cases}$$

Otherwise, $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ *is undefined.*

For a minimization (maximization) problem, the algorithms $\mathbb{A}$ and $\mathbb{B}$ are said to be comparable in $\mathbb{A}$'s favor *if* $\mathrm{WR}_{\mathbb{A},\mathbb{B}} < 1$ *(*$\mathrm{WR}_{\mathbb{A},\mathbb{B}} > 1$*). Similarly, the algorithms are said to be* comparable in $\mathbb{B}$'s favor*, if* $\mathrm{WR}_{\mathbb{A},\mathbb{B}} > 1$ *(*$\mathrm{WR}_{\mathbb{A},\mathbb{B}} < 1$*).*

Note that the ratio $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ can be larger than or smaller than one depending on whether the problem is a minimization problem or a maximization problem and which of $\mathbb{A}$ and $\mathbb{B}$ is the better algorithm. Table 2 indicates the result in each case. Instead of saying that two algorithms, $\mathbb{A}$ and $\mathbb{B}$, are comparable in $\mathbb{A}$'s favor, one would often just say that $\mathbb{A}$ is better than $\mathbb{B}$ according to RWOA.

Table 2. Relative worst-order ratio interpretation, depending on whether the problem is a minimization or a maximization problem.

Result	Minimization	Maximization
$\mathbb{A}$ better than $\mathbb{B}$	<1	>1
$\mathbb{B}$ better than $\mathbb{A}$	>1	<1

For quality measures evaluating algorithms by comparing them to each other directly, it is particularly important to be transitive: If $\mathbb{A}$ and $\mathbb{B}$ are comparable in $\mathbb{A}$'s favor and $\mathbb{B}$ and $\mathbb{C}$ are comparable in $\mathbb{B}$'s favor, then $\mathbb{A}$ and $\mathbb{C}$ are comparable in $\mathbb{A}$'s favor. When this transitivity holds, to prove that a new algorithm is better than all previously known algorithms, one only has to prove that it is better than the best among them. This holds for RWOA [17].

3 Paging

In this section, we survey the most important RWOA results for paging and explain how they differ from the results obtained with competitive analysis. As a relatively simple, concrete example of RWOA, we first explain how to obtain the separation of LRU and FWF [20] mentioned in Subsect. 2.1.

3.1 LRU vs. FWF

The first step in computing the relative worst-order ratio, $\mathrm{WR}_{\mathrm{FWF,LRU}}$, is to show that LRU and FWF are comparable. Consider any request sequence I for paging with cache size k. For any request r to a page p in I, if LRU faults on r, either p has never been requested before or there have been at least k different requests to distinct pages other than p since the last request to p. In the case where p has never been requested before, any online algorithm faults on r. If there have been at least k requests to distinct pages other than p since the last request to p, FWF has flushed since that last request to p, so p is no longer in its cache and FWF faults, too. Thus, for any request sequence I, $\mathrm{FWF}(I) \geq \mathrm{LRU}(I)$. Consider LRU's worst ordering, I_{LRU}, of a sequence I. Since FWF's performance on its worst ordering of any sequence is at least as bad as its performance on the sequence itself, $\mathrm{FWF}_W(I_{\mathrm{LRU}}) \geq \mathrm{FWF}(I_{\mathrm{LRU}}) \geq \mathrm{LRU}(I_{\mathrm{LRU}}) = \mathrm{LRU}_W(I_{\mathrm{LRU}})$. Thus, $c_l(\mathrm{FWF}, \mathrm{LRU}) \geq 1$.

As a remark, in general, to prove that one algorithm is at least as good as another on their respective worst orderings of all sequences, one usually starts with an arbitrary sequence and its worst ordering for the better algorithm. Then, that sequence is gradually permuted, starting at the beginning, so that the poorer algorithm does at least as badly on the permutation being created.

The second step is to show the separation, giving a lower bound on the term $c_u(\mathrm{FWF}, \mathrm{LRU})$. We assume that the cache is initially empty. Consider the sequence $I_s = \langle 1, 2, \ldots, k, k+1, k, \ldots, 2\rangle^s$, where FWF faults on all $2ks$ requests. LRU only faults on $2s + k - 1$ requests in all, the first k requests and every request to 1 or $k+1$ after that, but we need to consider how many times LRU faults on its worst ordering of I_s.

It is proven in [20] that, for any sequence I, there is a worst ordering of I for LRU that has all faults before all hits (requests which are not faults). The idea is to consider any worst order of I for LRU and move requests which are hits, but are followed by a fault towards the end of the sequence without decreasing the number of faults. Since LRU needs k distinct requests between two requests to the same page in order to fault, with only $k+1$ distinct pages in all, the faults at the beginning must be a cyclic repetition of the $k+1$ pages. Thus, a worst ordering of I_s for LRU is $I'_s = \langle 2, 3, \ldots, k, k+1, 1\rangle^s, \langle 2, \ldots, k\rangle^s$, and $\mathrm{LRU}(I'_s) = (k+1)s + k - 1$. This means that, asymptotically, $c_u(\mathrm{FWF}, \mathrm{LRU}) \geq \frac{2k}{k+1}$. We now know that $\mathrm{WR}_{\mathrm{FWF,LRU}} \geq \frac{2k}{k+1}$, showing that FWF and LRU are comparable in LRU's favor, which is the most interesting piece of information.

However, one can prove that this is the exact result. In the third step, we prove that $c_u(\mathrm{FWF}, \mathrm{LRU})$ cannot be larger than $\frac{2k}{k+1}$, asymptotically. In fact, this is shown in [20] by proving the more general result that, for any *marking* algorithm [13], $\mathbb{M}$, and for any request sequence I, $\mathbb{M}_W(I) \leq \frac{2k}{k+1}\text{LRU}_W(I) + k$. A marking algorithm is defined with respect to *k-phases*, a partitioning of the request sequence. Starting at the beginning of I, the first phase ends with the request immediately preceding the $(k+1)$st distinct page, and succeeding phases are also longest intervals containing at most k distinct pages. An algorithm is a marking algorithm if, assuming we mark a page each time it is requested and start with no pages marked at the beginning of each phase, the algorithm never evicts a marked page. As an example, FWF is a marking algorithm. Now, consider any sequence, I, with m k-phases. A marking algorithm $\mathbb{M}$ faults at most km times on I. Any two consecutive k-phases in I contain at least $k+1$ pages, so there must be a permutation of the sequence where LRU faults at least $k+1$ times on the requests of each of the $\lfloor \frac{m}{2} \rfloor$ consecutive pairs of k-phases in I. This gives the desired asymptotic upper bound, showing that $\text{WR}_{\text{FWF,LRU}} = \frac{2k}{k+1}$.

3.2 Other Paging Algorithms

Like LRU and FWF, the algorithm FIFO also has competitive ratio k [12]. FIFO simply evicts the first page that entered the cache, regardless of its use while in cache. In experiments, both LRU and FIFO are consistently much better than FWF. LRU and FIFO are both *conservative* algorithms [64], meaning that on any sequence of requests to at most k different pages, each of them faults at most k times. This means that, according to RWOA, they are equally good and both are better than FWF, since for any pair of conservative paging algorithms, $\mathbb{A}$ and $\mathbb{B}$, $\text{WR}_{\mathbb{A},\mathbb{B}} = 1$ and $\text{WR}_{\text{FWF},\mathbb{A}} = \frac{2k}{k+1}$ [20].

With a quality measure that separates FWF and LRU, an obvious question to ask is: Is there a paging algorithm which is better than LRU according to RWOA? The answer to this is "yes". LRU-2 [58], which was proposed for database disk buffering, is the algorithm which evicts the page with the earliest second-to-last request. LRU-2 and LRU are $(1 + \frac{1}{2k+2}, \frac{k+1}{2})$-related. This concept was introduced in [20], expressing that $c_u(\text{LRU-2}, \text{LRU}) = 1 + \frac{1}{2k+2}$ and $c_u(\text{LRU}, \text{LRU-2}) = \frac{k+1}{2}$ (see Definition 1 for a definition of c_u). Thus, the algorithms are *asymptotically comparable* in LRU-2's favor [14].

In addition, a new algorithm, RLRU (Retrospective LRU), was defined in [20] and shown to be better than LRU according to RWOA. Experiments, simply comparing the number of page faults on the same input sequences, have shown that RLRU is consistently slightly better than LRU [57]. RLRU is a phase-based algorithm. When considering a request, it determines whether Opt would have had the page in cache given the sequence seen so far (this is efficiently computable), and uses that information in a marking procedure.

Interestingly, LRU-2 has competitive ratio $2k$ and RLRU has competitive ratio $k+1$, so both are worse than LRU according to competitive analysis.

Also for paging, considering LRU and LRU(ℓ), which is LRU adapted to use look-ahead ℓ (the next ℓ requests after the current one), evicting a least recently used page *not* occurring in the look-ahead, both algorithms have competitive ratio k, though look-ahead helps significantly in practice. Using RWOA, $\mathrm{WR}_{\mathrm{LRU},\mathrm{LRU}(\ell)} = \min\{k, \ell + 1\}$, so LRU($\ell$) is better [20].

4 Other Online Problems

In this section, we give further examples of problems and algorithms where RWOA gives results that are qualitatively different from those obtained with competitive analysis. We consider various problems, including list accessing, bin packing, bin coloring, and scheduling.

List accessing [4, 61] is a classic problem in data structures, focusing on maintaining an optimal ordering in a linked list. In online algorithms, it also has the rôle of a theoretical benchmark problem, together with paging and a few other problems, on which many researchers evaluate new techniques or quality measures.

The problem is defined as follows: A list of items is given and requests are to items in the list. Treating a request requires accessing the item, and the cost of that access is the index of the item, starting with one. After the access, the item can be moved to any location closer to the front of the list at no cost. In addition, any two consecutive items may be transposed at a cost of one. The objective is to minimize the total cost of processing the input sequence.

We consider three list accessing algorithms: On a request to an item x, the algorithm Move-To-Front (MTF) [55] moves x to the front of the list, whereas the algorithm Transpose (Trans) just swaps x with its predecessor. The third algorithm, Frequency-Count (FC), keeps the list sorted by the number of times each item has been requested.

For list accessing [61], letting l denote the length of the list, the algorithm Move-To-Front has strict competitive ratio $2 - \frac{2}{l+1}$ [47] (referring to personal communication, Irani credits Karp and Raghavan with the lower bound). In contrast, Frequency-Count and Transpose both have competitive ratio $\Omega(l)$ [12]. Extensive experiments demonstrate that MTF and FC are approximately equally good, whereas Trans is much worse [8, 10]. Using RWOA, MTF and FC are equally good, whereas both $\mathrm{WR}_{\mathrm{Trans},\mathrm{MTF}} \in \Omega(l)$ and $\mathrm{WR}_{\mathrm{Trans},\mathrm{FC}} \in \Omega(l)$, so Trans is much worse [38].

For bin packing, both Worst-Fit (WF), which places an item in a bin with largest available space (but never opens a new bin unless it has to), and Next-Fit (NF), which closes its current bin whenever an item does not fit (and never considers that bin again), have competitive ratio 2 [48]. However, WF is at least as good as NF on every sequence and sometimes much better [16]. Using RWOA, $\mathrm{WR}_{\mathrm{NF},\mathrm{WF}} = 2$, so WF is the better algorithm.

Bin coloring is a variant of bin packing, where items are unit-sized and each have a color. The goal is to minimize the maximum number of colors in any bin, under the restriction that only a certain number, q, of bins are allowed to be open at any time and a bin is not closed until it is full. Consider the algorithms One-Bin, which never has more than one bin open, and Greedy-Fit, which always keeps q open bins, placing an item in a bin already having that color, if possible, and otherwise in a bin with fewest colors. We claim that Greedy-Fit is obviously the better algorithm, but if the bin size is larger than approximately q^3, One-Bin has a better competitive ratio than Greedy-Fit [54]. However, according to RWOA, Greedy-Fit is better [40].

For Scheduling on two related machines to minimize makespan (the time when all jobs are completed), the algorithm Fast, which only uses the fast machine, is $\frac{s}{s+1}$-competitive, where s is the speed ratio of the two machines. If s is larger than the golden ratio, this is the best possible competitive ratio. However, the algorithm Post-Greedy, which schedules each job on the machine where it would finish first, is never worse than Fast and sometimes better. This is reflected in the relative worst-order ratio, since $\mathrm{WR}_{\textsc{Fast},\textsc{Post-Greedy}} = \frac{s+1}{s}$ [39].

In addition to these examples, it is widely believed and consistent with experiments that for bin packing problems, First-Fit algorithms perform better than Worst-Fit algorithms, and that processing larger items first is better than processing smaller items first. For the problem examples below, competitive analysis cannot distinguish between the algorithms, that is, they have the same competitive ratio, whereas using RWOA, we get the separation in the right direction. The examples are the following: For dual bin packing (the variant of bin packing where there is a fixed number of bins, the aim is to pack as many items as possible, and all bins are considered open from the beginning), First-Fit is better than Worst-Fit [17]. For grid scheduling (a variant of bin packing where the items are given from the beginning and variable-sized bins arrive online), First-Fit-Decreasing is better than First-Fit-Increasing [18]. For seat reservation (the problem where a train with a certain number of seats travels from station 1 to some station $k \in \mathbb{Z}^+$, requests to travel from some station i to a station $j > i$ arrive online, and the aim is to maximize either the number of passengers or the total distance traveled), First-Fit and Best-Fit are better than Worst-Fit [28] with regards to both objective functions.

5 Approaches to Understanding Online Computation

In this section, we discuss other means of analyzing and thereby gaining insight into online computation. This includes other performance measures and advice complexity.

5.1 Other Performance Measures

Other than competitive analysis, many alternative measures have been introduced with the aim of getting a better or more refined picture of the (relative) quality of online algorithms.

In chronological order, the main contributions are the following: online/online ratio [45], statistical adversary [60], loose competitive ratio [64], max/max ratio [9], access graphs (incorporating locality of reference) [13], random-order ratio [52], accommodating ratio [24], extra resource analysis [50], diffuse adversary

[53], accommodating function [27], smoothed analysis [62], working set (incorporating locality of reference) [2], relative worst-order analysis [17,20], bijective and average analysis [5], relative interval analysis [37], bijective ratio [6], and online-bounded analysis [15].

We are not defining all of these measures here, but we give some insight into the strengths and weaknesses of selected measures in the following. We start with a discussion of work directly targeted at performance measure comparison.

Comparisons of Performance Measures. A systematic comparison of performance measures for online algorithms was initiated in [23], comparing some measures which are applicable to many types of problems. To make this feasible, a particularly simple problem was chosen: the 2-server problem on a line with three points, one point farther away from the middle point than the other.

A well known algorithm, Double Coverage (DC), is 2-competitive and best possible for this problem [30] according to competitive analysis. A lazy version of this, LDC, is at least as good as DC on every sequence and often better. Investigating which measures can make this distinction, LDC was found to be better than DC by bijective analysis and RWOA, but equivalent to DC according to competitive analysis, the max/max ratio, and the random-order ratio. The first proof, for any problem, of an algorithm being best possible under RWOA established this for LDC.

Greedy performs unboundedly worse than LDC on certain sequences, so ideally a performance measure would not find Greedy to be superior to LDC. According to the max/max ratio and bijective analysis, Greedy is the better algorithm, but not according to competitive analysis, random-order analysis, or RWOA.

Further systematic comparisons of performance measures were made in [25,26], again comparing algorithms on relatively simple problems. The paper [25] considered competitive analysis, bijective analysis, average analysis, relative interval analysis, random-order analysis, and RWOA. There were differences between the measures, but the most clear conclusions were that bijective analysis found all algorithms incomparable and average analysis preferred an intuitively poorer algorithm.

Notable omissions from the investigations above are extra resource analysis [50] and the accommodating function [27], both focusing on resources, which play a major rôle in most online problems. Both measures have been applied successfully to a range of problems, giving additional insight; extra resource analysis (also referred to as resource augmentation) has been used extensively. They can both be viewed as extensions of competitive analysis, explaining observed behavior of algorithms by expressing ratios as functions of resource availability.

5.2 Advice Complexity

As a means of analyzing problems, as opposed to algorithms for those problems, *advice complexity* was proposed [11,36,46]. The "no knowledge about the future" property of online algorithms is relaxed, and it is assumed that some bits of advice are available; such knowledge is available in many situations. One asks how many bits of advice are necessary and sufficient to obtain a given competitive ratio, or indeed optimality. For a survey on advice complexity, see [19].

6 Applicability

Competitive analysis has been used for decades and sophisticated, supplementary analysis techniques have been developed to make proofs more manageable, or with the purpose of capturing more fine-grained properties of algorithms.

We discuss two of the most prominent examples of these supplementary techniques: *list factoring* for analyzing list accessing and *access graphs* for modeling locality of reference for paging. Both techniques have been shown to work with RWOA. As far as we know, list factoring has not been established as applicable to any other alternative to competitive analysis. Access graphs have also been studied for relative interval analysis [22] with less convincing results.

6.1 List Factoring for Analyzing List Accessing

The idea behind *list factoring* is to reduce the analysis to lists of two elements, thereby making the analysis much more manageable. The technique was first introduced by Bentley and McGeoch [10] and later extended and improved [1,3,47,63]. In order to use this technique, one uses the *partial cost model*, where the cost of each request is one less than in the standard (full) cost model (the access to the item itself is not counted). The list factoring technique is applicable for algorithms where, in treating any request sequence I, one gets the same result by counting only the costs of passing through x or y when searching for y or x (denoted $\mathbb{A}_{xy}(I)$), as one would get if the original list contained only x and y and all requests different from those were deleted from the request sequence, denoted I_{xy}. If this is the case, that is $\mathbb{A}_{xy}(I) = \mathbb{A}(I_{xy})$ for all I, then $\mathbb{A}$ is said to have the *pairwise property*, and it is not hard to prove that then $\mathbb{A}(I) = \sum_{x \neq y} \mathbb{A}(I_{xy})$. Thus, we can reduce the analysis of $\mathbb{A}$ to an analysis of lists of length two. The results obtained also apply in the full cost model if the algorithms are *cost independent*, meaning that their decisions are independent of the costs of the operations.

Since the cost measure is different, some adaption is required to get this to work for RWOA:

We now say that $\mathbb{A}$ has the *worst-order projection property* if and only if, for all sequences I, there exists a worst ordering $\pi_{\mathbb{A}}(I)$ of I with respect to $\mathbb{A}$, such that for all pairs $\{a, b\} \subseteq L$ $(a \neq b)$, $\pi_{\mathbb{A}}(I)_{ab}$ is a worst ordering of I_{ab} with respect to $\mathbb{A}$ on the initial list L_{ab}.

The results on Move-To-Front, Frequency-Count, and Transpose, reported on in Subsect. 3.2, as well as results on Timestamp [1], were obtained [38] using this tool.

6.2 Access Graphs for Modeling Locality of Reference for Paging

Locality of reference refers to the observed behavior of certain sequences from real life, where requests seem to be far from uniformly distributed, but rather exhibit some form of locality; for instance with repetitions of pages appearing in close proximity [33,34]. Performance measures that are worst-case over all possible sequences will usually not reflect this, so algorithms exploiting locality of reference are not deemed better using the theoretical tools, though they may be superior in practice. This has further been underpinned by the following result [5] on bijective analysis: For the class of demand paging algorithms (algorithms that never evict a page unless necessary), for any two positive integers $m, n \in \mathbb{N}$, all algorithms have the same number of input sequences of length n that result in exactly m faults.

One attempt at formalizing locality of reference, making it amenable to theoretical analysis, was made in [13], where *access graphs* were introduced. An access graph is an undirected graph with vertices representing pages and edges indicating that the two pages being connected could be accessed immediately after each other. In the performance analysis of an algorithm, only sequences respecting the graph are considered, i.e., any two distinct, consecutive requests must be to the same page or to neighbors in the graph.

Under this restriction on inputs, [13,31] were able to show that, according to competitive analysis, LRU is strictly better than FIFO on some access graphs and never worse on any graph. Thus, they were the first to obtain a separation, consistent with empirical results.

Using RWOA, [21] proved that on the primary building blocks of access graphs, paths and cycles, LRU is strictly better than FIFO.

7 Open Problems and Future Work

For problems where competitive analysis deems many algorithms best possible or gives counter-intuitive results, comparing algorithms with RWOA can often provide additional information. Such comparisons can be surprisingly easy, since it is often possible to use parts of previous results when applying RWOA.

Often the exploration for new algorithms for a given problem ends when an algorithm is proven to have a best possible competitive ratio. Using RWOA to continue the search for better algorithms after competitive analysis fails to provide satisfactory answers can lead to interesting discoveries. As an example, the paging algorithm RLRU was designed in an effort to find an algorithm that could outperform LRU with respect to RWOA.

Also for the paging problem, RLRU and LRU-2 are both known to be better than LRU according to RWOA. It was conjectured [14] that LRU-2 is comparable to RLRU in LRU-2's favor. This is still unresolved. It would be even more interesting to find a new algorithm better than both. It might also be interesting to apply RWOA to an algorithm from the class of OnOpt algorithms from [57].

For bin packing, it would be interesting to know whether Best-Fit is better than First-Fit, according to RWOA.

Acknowledgment. The authors would like to thank an anonymous referee for many constructive suggestions.

References

1. Albers, S.: Improved randomized on-line algorithms for the list update problem. SIAM J. Comput. **27**(3), 682–693 (1998)
2. Albers, S., Favrholdt, L.M., Giel, O.: On paging with locality of reference. J. Comput. Syst. Sci. **70**(2), 145–175 (2005)
3. Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. Inf. Process. Lett. **56**, 135–139 (1995)
4. Albers, S., Westbrook, J.: Self-organizing data structures. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms. LNCS, vol. 1442, pp. 13–51. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0029563
5. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 229–237 (2007)
6. Angelopoulos, S., Renault, M.P., Schweitzer, P.: Stochastic dominance and the bijective ratio of online algorithms. arXiv arXiv:1607.06132 [cs.DS] (2016)
7. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: A knapsack secretary problem with applications. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) APPROX/RANDOM-2007. LNCS, vol. 4627, pp. 16–28. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74208-1_2
8. Bachrach, R., El-Yaniv, R.: Online list accessing algorithms and their applications: recent empirical evidence. In: 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 53–62 (1997)
9. Ben-David, S., Borodin, A.: A new measure for the study of on-line algorithms. Algorithmica **11**(1), 73–91 (1994)
10. Bentley, J.L., McGeoch, C.C.: Amortized analyses of self-organizing sequential search heuristics. Commun. ACM **28**, 404–411 (1985)
11. Böckenhauer, H.-J., Komm, D., Královic, R., Královic, R., Mömke, T.: Online algorithms with advice: the tape model. Inf. Comput. **254**, 59–83 (2017)
12. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
13. Borodin, A., Irani, S., Raghavan, P., Schieber, B.: Competitive paging with locality of reference. J. Comput. Syst. Sci. **50**(2), 244–258 (1995)
14. Boyar, J., Ehmsen, M.R., Kohrt, J.S., Larsen, K.S.: A theoretical comparison of LRU and LRU-K. Acta Inf. **47**(7–8), 359–374 (2010)
15. Boyar, J., Epstein, L., Favrholdt, L.M., Larsen, K.S., Levin, A.: Online-bounded analysis. J. Sched. **21**(4), 429–441 (2018)
16. Boyar, J., Epstein, L., Levin, A.: Tight results for Next Fit and Worst Fit with resource augmentation. Theor. Comput. Sci. **411**(26–28), 2572–2580 (2010)
17. Boyar, J., Favrholdt, L.M.: The relative worst order ratio for on-line algorithms. ACM Trans. Algorithms **3**(2), 24 (2007). Article 22
18. Boyar, J., Favrholdt, L.M.: Scheduling jobs on grid processors. Algorithmica **57**(4), 819–847 (2010)
19. Boyar, J., Favrholdt, L.M., Kudahl, C., Larsen, K.S., Mikkelsen, J.W.: Online algorithms with advice: a survey. ACM Comput. Surv. **50**(2), 19:1–19:34 (2017)
20. Boyar, J., Favrholdt, L.M., Larsen, K.S.: The relative worst order ratio applied to paging. J. Comput. Syst. Sci. **73**(5), 818–843 (2007)

21. Boyar, J., Gupta, S., Larsen, K.S.: Access graphs results for LRU versus FIFO under relative worst order analysis. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 328–339. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31155-0_29
22. Boyar, J., Gupta, S., Larsen, K.S.: Relative interval analysis of paging algorithms on access graphs. Theor. Comput. Sci. **568**, 28–48 (2015)
23. Boyar, J., Irani, S., Larsen, K.S.: A comparison of performance measures for online algorithms. Algorithmica **72**(4), 969–994 (2015)
24. Boyar, J., Larsen, K.S.: The seat reservation problem. Algorithmica **25**(4), 403–417 (1999)
25. Boyar, J., Larsen, K.S., Maiti, A.: A comparison of performance measures via online search. Theor. Comput. Sci. **532**, 2–13 (2014)
26. Boyar, J., Larsen, K.S., Maiti, A.: The frequent items problem in online streaming under various performance measures. Int. J. Found. Comput. Sci. **26**(4), 413–440 (2015)
27. Boyar, J., Larsen, K.S., Nielsen, M.N.: The accommodating function: a generalization of the competitive ratio. SIAM J. Comput. **31**(1), 233–258 (2001)
28. Boyar, J., Medvedev, P.: The relative worst order ratio applied to seat reservation. ACM Trans. Algorithms **4**(4), 22 (2008). Article 48
29. Christ, M.G., Favrholdt, L.M., Larsen, K.S.: Online bin covering: expectations vs guarantees. Theor. Comput. Sci. **556**, 71–84 (2014)
30. Chrobak, M., Karloff, H.J., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. **4**(2), 172–181 (1991)
31. Chrobak, M., Noga, J.: LRU is better than FIFO. Algorithmica **23**(2), 180–185 (1999)
32. Coffmand Jr., E.G., Csirik, J., Rónyai, L., Zsbán, A.: Random-order bin packing. Discrete Appl. Math. **156**, 2810–2816 (2008)
33. Denning, P.J.: The working set model for program behaviour. Commun. ACM **11**(5), 323–333 (1968)
34. Denning, P.J.: Working sets past and present. IEEE Trans. Softw. Eng. **6**(1), 64–84 (1980)
35. Devanur, N.R., Hayes, T.P.: The adwords problem: online keyword matching with budgeted bidders under random permutations. In: 10th ACM Conference on Electronic Commerce (EC), pp. 71–78 (2009)
36. Dobrev, S., Kralović, R., Pardubskă, D.: Measuring the problem-relevant information in input. RAIRO - Theor. Inf. Appl. **43**(3), 585–613 (2009)
37. Dorrigiv, R., López-Ortiz, A., Munro, J.I.: On the relative dominance of paging algorithms. Theor. Comput. Sci. **410**, 3694–3701 (2009)
38. Ehmsen, M.R., Kohrt, J.S., Larsen, K.S.: List factoring and relative worst order analysis. Algorithmica **66**(2), 287–309 (2013)
39. Epstein, L., Favrholdt, L.M., Kohrt, J.S.: Separating scheduling algorithms with the relative worst order ratio. J. Comb. Optim. **12**(4), 362–385 (2006)
40. Epstein, L., Favrholdt, L.M., Kohrt, J.S.: Comparing online algorithms for bin packing problems. J. Sched. **15**(1), 13–21 (2012)
41. Fischer, C., Röglin, H.: Probabilistic analysis of the dual Next-Fit algorithm for bin covering. In: Kranakis, E., Navarro, G., Chávez, E. (eds.) LATIN 2016. LNCS, vol. 9644, pp. 469–482. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49529-2_35
42. Göbel, O., Kesselheim, T., Tönnis, A.: Online appointment scheduling in the random order model. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 680–692. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_57

43. Goel, G., Mehta, A.: Online budgeted matching in random input models with applications to adwords. In: 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 982–991 (2008)
44. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**(2), 416–429 (1969)
45. Gyárfás, A., Lehel, J.: First fit and on-line chromatic number of families of graphs. Ars Comb. **29**(C), 168–176 (1990)
46. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15155-2_3
47. Irani, S.: Two results on the list update problem. Inf. Process. Lett. **38**(6), 301–306 (1991)
48. Johnson, D.S.: Fast algorithms for bin packing. J. Comput. Syst. Sci. **8**, 272–314 (1974)
49. Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham, R.L.: Worst-case performance bound for simple one-dimensional packing algorithms. SIAM J. Comput. **3**, 299–325 (1974)
50. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM **47**, 617–643 (2000)
51. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica **3**, 79–119 (1988)
52. Kenyon, C.: Best-fit bin-packing with random order. In: 7th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 359–364 (1996)
53. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. SIAM J. Comput. **30**(1), 300–317 (2000)
54. Krumke, S.O., de Paepe, W., Rambau, J., Stougie, L.: Bincoloring. Theor. Comput. Sci. **407**(1–3), 231–241 (2008)
55. McCabe, J.: On serial files with relocatable records. Oper. Res. **13**(4), 609–618 (1965)
56. Meyerson, A.: Online facility location. In: 42nd IEEE Symposium on Foundations of Computer Science (FOCS), pp. 426–433 (2001)
57. Moruz, G., Negoescu, A.: Outperforming LRU via competitive analysis on parametrized inputs for paging. In: 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1669–1680 (2012)
58. O'Neil, E.J., O'Neil, P.E., Weikum, G.: The LRU-K page replacement algorithm for database disk buffering. In: ACM SIGMOD International Conference on Management of Data, pp. 297–306 (1993)
59. Osborn, C.J., Torng, E.: List's worst-average-case or WAC ratio. J. Sched. **11**, 213–215 (2008)
60. Raghavan, P.: A statistical adversary for on-line algorithms. In: On-Line Algorithms, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, vol. 7, pp. 79–83. American Mathematical Society (1992)
61. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)
62. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. J. ACM **51**(3), 385–463 (2004)
63. Teia, B.: A lower bound for randomized list update algorithms. Inf. Process. Lett. **47**, 5–9 (1993)
64. Young, N.E.: The k-server dual and loose competitiveness for paging. Algorithmica **11**, 525–541 (1994)

Length-Weighted Disjoint Path Allocation

Advice and Parametrization

Elisabet Burjons, Fabian Frei(✉), Jasmin Smula, and David Wehner

Department of Computer Science, ETH Zurich, Zürich, Switzerland
{eburjons,fabian.frei,david.wehner}@inf.ethz.ch

Abstract. We modify one of the foundational online problems, *Disjoint Path Allocation*, to include weighted requests. We provide a comprehensive competitive analysis, incorporating the viewpoints of both advice complexity and parametrized complexity. Our bounds feature a consistent parametrization and closely trace the trade-off between advice complexity and competitiveness.

Jointly dedicated to an Unceasingly Requested Algorithmist on his three-score Jubilee.

1 Motivation

Imagine an incredibly intelligent and industrious individual. Take the reasonably random example of a professor in computer science at a renowned university in the heart of Europe. Said professor has to meticulously manage his or her time schedule—we will arbitrarily stick with "his" for the purpose of this paper. He is invited to an influential school to give its teachers a brief talk on the education in his field, illuminating computer science's core principals and providing tried and tested tools to guide our future generations. As always, the organizers wish for an immediate confirmation whether he will embrace this offer. Undoubtedly, he would love to do so; nevertheless, he needs to check whether he is already booked at the requested time.[1]

Even if the booking remains a viable possibility, however, he is of course prudent enough to not accept any invitation precipitately. After all, it might interfere with another opportunity yet to come—envision an extensive exposition of his educational efforts and an ensuing exemplary execution for the government's leading officials. Such a feasible future request might fill the available time window even better, thereby minimizing his idle time. As a conscientious person valuing reliability, going back on a made commitment is no option to him, limiting the flexibility even further. In order to cope successfully with all the diverse and demanding duties, a good guideline for accepting or rejecting requests is thus indispensable.

We try to help our professor in finding and analyzing the best acceptance strategies.

[1] This is the most probable case. Inexplicably, he still finds time to mentor his doctoral students.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 231–256, 2018.
https://doi.org/10.1007/978-3-319-98355-4_14

2 Introduction and Definitions

In this section, we introduce the model of our problem, the necessary formal framework, and the tools used for its analysis.

2.1 Length-Weighted Disjoint Path Allocation

The task at hand can be described as follows. First, we are given a time span of length $\ell \in \mathbb{N}$. It is measured in a reasonable time unit, assumed to be indivisible. Certainly, we should not be micro-managing single seconds. The time span can be visualized as a path of length ℓ, as it might appear in a day planner. Switching from the motivating model to a more tangible example, suppose there is a meeting room that can be booked, initially available over the entire time span. Inevitably, an unknown number of reservation requests will start pouring in. Every request corresponds to a *subpath* and demands immediate *acceptance* or *rejection*. Once taken, a decision cannot be revoked. Our goal is to *maximize* the room's occupancy, the overall time it is in use.

This is of course an *optimization problem*, but also a typical *online problem*. Such problems appear abundantly in all areas of computer science. They are characterized by the fact that a solving algorithm $\mathcal{A}$ is forced to produce parts of the output without complete knowledge of the input.

We now define our problem of *Length-Weighted Disjoint Path Allocation* on paths, LWDPA for short, more formally. An instance I consists of a path P of length ℓ and n requests r_i, which are just subpaths of P. An algorithm either *accepts* or *rejects* each request. It cannot accept a request that *overlaps* a previously accepted request, where two requests are said to *overlap* if they share at least one edge. In this case, we say that they *block* each other. The requests accepted by an algorithm $\mathcal{A}$ on an instance I constitute the *solution* $\mathcal{A}(I)$ of $\mathcal{A}$ on I. A solution is thus a selection of *non-overlapping* requests. For a fixed solution, we refer to the leaves of the accepted requests, that is, the starting and ending points of the chosen subpaths, as *transition points*.

The *gain* of a solution to I computed by $\mathcal{A}$ is the total length of all accepted requests, denoted by $\text{gain}(\mathcal{A}(I))$. A solution is *optimal* if it has the maximum possible gain. An *optimal algorithm* Opt produces an optimal solution $\text{Opt}(I)$ with gain $\text{gain}(\text{Opt}(I))$ on every instance I. We refer to the accepted requests of a fixed optimal solution as the *optimal requests*.

In accordance with the conventional way of defining related problems—see Subsect. 2.2—we assume that all requests are unique; in other words, no subpath is offered more than once. Moreover, we assume that the algorithm is not given the number n of requests beforehand and also does not recognize the last request as such. This is a severe constraint for any algorithm.

2.2 Connection with Classic Disjoint Path Allocation

Among all researched online problems, the problem of *Disjoint Path Allocation* (DPA) is one of the earliest. It has been thoroughly analyzed in many different settings [1–3,8]. The difference to our *Length-Weighted Disjoint Path Allocation* (LWDPA) lies in a single but significant detail: For DPA, the gain of a solution is defined as the number of requests it contains; for LWDPA, on the other hand, we weight the requests by their length. Thus, the gain is the total length of all accepted requests.

Both DPA and LWDPA can be altered to allow preemption, that is, the removal of provisionally accepted requests from the solution. The preemptive model has been the focus of the work by Kováčová [11]. Nevertheless, it contains a few results for the non-preemptive case. Her Theorems 8, 9, 12 and 13 correspond to the unparametrized special case of our optimality results in Subsect. 3.1, namely Theorems 1 to 4. Theorem 18 gives an upper bound that is asymptotically improved upon in Theorem 7 by choosing $b > 0$.

DPA is traditionally motivated by a network wherein permanent communication channels between pairs of nodes are established. It is unknown what the connections are used for; consequently, the only goal is to maximize their number. In contrast, our motivating example of LWDPA illustrates why maximizing the number of covered edges will be preferred instead under the right circumstances.

2.3 Competitive Analysis

Competitive analysis was introduced by Sleator and Tarjan as a tool to measure the performance of online algorithms [12]. The basic idea is to assess the quality of the solution computed by an online algorithm by comparing it to an optimal solution that is calculated *offline*, that is, by an algorithm with prior knowledge of the whole input. Note that no limit on the computational complexity is imposed; enforcing online decisions is restrictive enough. An offline algorithm can, therefore, always deliver an optimal solution, if necessary by applying brute force.

In the present case of a maximization problem, the following two notions enable us to express the mentioned comparison succinctly:

Definition 1 (Strict Competitiveness). *$\mathcal{A}$ is* strictly c-competitive *if*

$$\forall I\colon\ \frac{\text{gain}(\text{OPT}(I))}{\text{gain}(\mathcal{A}(I))} \leq c.$$

The best attainable strict competitiveness is 1. This is only achieved if $\mathcal{A}$ always produces an optimal solution. The poorer $\mathcal{A}$'s handling of any worst-case instance, the larger a c we need to choose.

Definition 2 (Strict Competitive Ratio). *The strict competitive ratio of an online algorithm $\mathcal{A}$ is the infimum over all c that satisfy the condition in Definition 1.*

A thorough introduction to online computation and competitive analysis can be found in the textbook by Borodin and El-Yaniv [4].

2.4 Advice Complexity

The competitive analysis of online algorithms compares them to offline algorithms. One might say that this sets the bar far too high. Often, an online algorithm has no chance to come even near the performance of an offline algorithm.

Here, the concept of *advice complexity* comes into play, offering a more fine-grained classification of online problems. The aim is to answer the question of how much information about the complete input is required for an online algorithm to guarantee any given competitiveness. Arguably the most fundamental question in this regard is how much knowledge about the input suffices for an online algorithm to ensure an optimal solution. With less information it is generally more difficult to solve a problem well; hence, a trade-off between the necessary amount of information and the achievable competitive ratio occurs.

We allow for any kind of information and measure it in bits. We model this by an oracle with infinite computational power that delivers, in dependence of the given instance, an infinite binary string of *advice bits*. This string is provided to the online algorithm along with the instance in the form of an infinite tape. The online algorithm may read a prefix of arbitrary length from the tape and adjust its behavior accordingly. We refer to the read prefix as the *advice string* and call its length the *advice complexity* of the algorithm on the given instance.

An online algorithm that reads at most b advice bits can also be interpreted as a collection of 2^b classic deterministic algorithms, one for each advice string. Conversely, we need $\log B$ advice bits[2] to simulate B distinct deterministic algorithms.[3] This viewpoint will come in handy in our proofs.

Across all possible online algorithms, there is a minimum number of advice bits that must be read to process any problem instance optimally. This minimal advice complexity can be interpreted as a fundamental property of the problem, its *information content*.

The concept of advice complexity was introduced by Dobrev et al. [6] and subsequently refined by Hromkovič et al. [9], Böckenhauer et al. [3], and Emek et al. [7]. In particular the work by Hromkovič et al. [9] elaborated on the intriguing idea of advice complexity as a tool to measure a problem's information content. Advice complexity has since been successfully applied to numerous online problems. For a recent and well-written introduction to both online problems and the concept of advice, we refer to Komm [10].

2.5 Parametrization

For many online problems, bounds are naturally given as functions of n, the number of requests. For DPA as well as LWDPA, however, there is the alternative

[2] Throughout this paper, log denotes the logarithm to base 2.

[3] Note that even $\lceil \log B \rceil$ advice bits are necessary. However, we will often omit ceiling and floor brackets and assume divisibility in our proofs for the sake of better readability.

to express results in terms of the path length ℓ instead. We present results for both choices.

Our analysis of the LWDPA problem is consistently parametrized by $k_{\min}$ and $k_{\max}$, a guaranteed lower and upper bound, respectively, on the length of the requests in the instances. These two parameters are known to both the algorithm and the oracle. We denote the ratio between these two bounds by $\varphi := k_{\max}/k_{\min}$. Naturally, we have $1 \leq k_{\min} \leq k_{\max} \leq \ell$. When we restrict the instances to requests of a single length, DPA and LWDPA coincide. This corresponds to the particular case of $k_{\min} = k_{\max}$, for which good bounds are already proven in the bachelor's thesis by Dietiker [5]; we therefore assume $k_{\min} < k_{\max}$. Our parametrized results are a true generalization in the sense that setting $k_{\min} = 1$ and $k_{\max} = \ell$ yields meaningful bounds on the strict competitive ratio in the classic sense.

Why would we want to parametrize the LWDPA problem in this way? We get a good intuition by going back to the motivating example of a meeting room that is available for booking over a time span of length ℓ. Blocks of 5 minutes could be a reasonable time unit, providing sufficient flexibility without an excessive number of options. Since nobody properly enjoys concise meetings, we might want to enforce a minimum reservation length of half an hour. With our parametrization, we can model this by setting $k_{\min}$ to 6, yielding a minimum of $k_{\min} \cdot 5 = 30$ minutes. One might object that we could have just as well stretched the time unit to half an hour, with no need for introducing new parameters. This, however, would leave no option between 30 minutes and a full hour, thereby preventing a reasonable booking time for, say, a 45-minute exercise class. Imposing upper bounds on the reservation length via $k_{\max}$, on the other hand, could make sense in order to avoid permanent reservations and give everybody a booking opportunity.

Altogether, we see that the proposed parametrization in terms of $k_{\min}$ and $k_{\max}$ is well justified and will help us to deal better with boundary conditions of this sort. Note that $k_{\min}$ and $k_{\max}$ are only bounds on the request lengths known to the algorithm and not necessarily the actual lengths of the shortest and longest requests. In the example, we know that every meeting will last at least half an hour; however, this does not mean we can count on the appearance of a booking request for a 30-minute meeting.

3 Results

This section presents all of our results for LWDPA in consistent parametrization. It is structured as follows.

In Subsect. 3.1, we give upper and lower bounds on the amount of advice bits needed to achieve optimality. In Subsect. 3.2, we prove a tight bound on the strict competitive ratio of the problem without advice.

Subsection 3.3 bridges the gap between these two extremes; we present results that trace the trade-off between strict competitiveness and advice complexity. The first one, Theorem 6, is an upper bound witnessed by an intelligent greedy algorithm. It performs exceptionally well with only few advice bits. The performance deteriorates quickly with increasing advice complexity, however. Theorem 7 cures this shortcoming with an advice-assisted preprocessing; it is well-suited for situations where much advice is available. The last two theorems give the complementing lower bounds. Theorem 8 is a good match to Theorem 6; it features a hard instance set that is tailor-made against the greedy approach. Theorem 9, on the other hand, is a rather technical but all the more flexible result, yielding multiple useful corollaries.

The conclusion in Sect. 4 recapitulates the results and discusses what light they shed on the advice complexity behavior of lwDPA.

3.1 Optimality

This subsection answers the question of how much advice an online algorithm needs in order to produce an optimal solution to every instance. Theorems 1 and 3 give an answer that depends on n, whereas Theorems 2 and 4 address the question in terms of the path length ℓ. It is worth noting that we have two closely matching pairs of upper and lower bounds and even a perfect match in the unparametrized case of $k_{\min} = 1$ and $k_{\max} = \ell$.

Theorem 1. *Optimality can be achieved with n advice bits.*

Proof. The oracle chooses an arbitrary but fixed optimal solution. Then, the oracle specifies for every request whether it is part of this solution or not, using n advice bits. Whenever the algorithm receives a request, it reads the next advice bit from the tape and accepts or rejects accordingly. This strategy does not depend on the values of $k_{\min}$ and $k_{\max}$. $\square$

Theorem 2. *Optimality can be achieved with $(\ell - 1)/k_{\min} \cdot (1 + 2\log(k_{\min}))$ advice bits.*

Proof. Before we describe how the oracle determines the advice given to the algorithm $\mathcal{A}$, let us consider the subpaths of P consisting of $k_{\min}$ vertices. We first show that in any optimal solution to an instance I, any such subpath can contain at most two transition points. For the sake of contradiction, fix some subpath S of length $k_{\min} - 1$ and some optimal solution $\textsc{Opt}(I)$ and assume that S contains at least three transition points of $\textsc{Opt}(I)$. Then we can pick three consecutive transition points from S and call them (from left to right) u, v, and w. Since v is the middle transition point, $\textsc{Opt}(I)$ must contain at least one of the requests (u, v) and (v, w). However, since S only contains $k_{\min}$ vertices, the subpaths (u, v) and (v, w) both have length at most $k_{\min} - 1$ and are thus shorter than the minimum guaranteed request length. Hence, they can neither be part of the instance I nor $\textsc{Opt}(I)$, yielding the contradiction.

As depicted in Fig. 1, we now divide the $\ell - 1$ inner vertices of the path P into $(\ell - 1)/k_{\min}$ segments containing $k_{\min}$ vertices each. For the given input sequence I, the oracle chooses some arbitrary but fixed optimal solution $\textsc{Opt}(I)$ and uses the advice bits to indicate the positions of the transition points

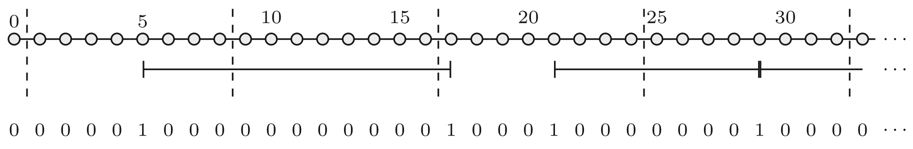


Fig. 1. An example of a possible optimal solution $\textsc{Opt}(I)$. The dashed lines indicate the segments containing $k_{\min}$ vertices each. The bit string underneath indicates all transition points of $\textsc{Opt}(I)$, enabling $\mathcal{A}$ to act optimally on I. The algorithm can deduce this string by reading only $1 + 2\log k_{\min}$ advice bits per segment.

of $\textsc{Opt}(I)$ in each segment. To calculate the number of advice bits necessary, let us count the number of cases the oracle has to distinguish. Taking into account that the number of transition points in a segment might be two, one, or zero, the number of different possible combinations that have to be distinguished is

$$\binom{k_{\min}}{2} + k_{\min} + 1 = \frac{k_{\min}(k_{\min} - 1) + 2k_{\min} + 2}{2} = \frac{k_{\min}^2 + k_{\min} + 2}{2},$$

which can be bounded from above by $2k_{\min}^2$. The number of advice bits necessary to encode that many different possible combinations of transition points per segment is

$$\frac{\ell - 1}{k_{\min}} \cdot \log\bigl(2k_{\min}^2\bigr) = \frac{\ell - 1}{k_{\min}} \cdot (1 + 2\log(k_{\min})).$$

The algorithm $\mathcal{A}$ reads exactly this many advice bits. (It can do so even before the first request arrives.) From this information, $\mathcal{A}$ can derive all transition points of $\textsc{Opt}(I)$ except the first and last vertex of P, which can always be treated as transition points. Now it only has to accept every presented request whose starting and ending points coincide with two successive transition points, and reject all others. This way, $\mathcal{A}$ never accepts any requests that block any requests from $\textsc{Opt}(I)$. Hence, $\mathcal{A}$ accepts all requests from $\textsc{Opt}(I)$ and must be optimal itself. $\square$

For these two upper bounds, it was sufficient to provide an algorithm with the given advice complexity that performs optimally on all instances. For the corresponding lower bounds, we now take the adversary's position and show that there is always an instance that the algorithm will solve suboptimally, unless given a certain number of advice bits.

Theorem 3. *At least $n - 1 - \lfloor (n-1)/(k_{\max} - k_{\min} + 1) \rfloor$ advice bits are needed for optimality.*

Proof. We describe a set $\mathcal{I}$ of instances such that no deterministic algorithm can solve two of them optimally. This implies that an optimal algorithm must be able to distinguish them all, so $\log(|\mathcal{I}|)$ advice bits are required.

To give a lower bound in n, we can choose the path length as large as necessary. We fix it to be $\ell := nk_{\min}$ for all instances from $\mathcal{I}$. Every instance is presented in k blocks that consist of $i_1, \dots, i_k$ requests, respectively; hence $i_1 + \dots + i_k = n$. The first block has i_1 requests of lengths $k_{\min}, \dots, k_{\min} + i_1 - 1$; they are all aligned on the left endpoint and presented in increasing order. All following blocks are constructed in the same way: Block j has i_j requests of lengths $k_{\min}, \dots, k_{\min} + i_j - 1$, aligned as far as possible to the left, without overlapping the previous block. See Fig. 2 for an example. Every new request protrudes further to the right: by 1 if it stays in the same block, and by $k_{\min}$ if it starts a new one. Thus, the path length $\ell = n \cdot k_{\min}$ is indeed just large enough to accommodate all of them.

The optimal solution, obtained by accepting the last request of each block and highlighted in Fig. 2, is unique. Hence, already a single mistake prevents optimality. We prove now that a deterministic algorithm cannot handle any two such instances $I \neq I'$ optimally, but will make a mistake on one of them. The requests for I and I' are identical up to some point. Let r_j be the first request that differs between them. Like every request, r_j either starts a new block or continues an existing one. Only in the former case is r_{j-1} the last request in a block and has to be accepted. Since the deterministic algorithm cannot distinguish the two instances before r_j is presented, it is bound to make a mistake on one of them.

It remains to give a good estimate on the number of instances in $\mathcal{I}$. For this, we consider how these instances can be constructed step by step, adding the n requests one after another. The first request in the first block is identical for all instances; it has length $k_{\min}$. In each of the following $n-1$ steps we can either start a new block with a request of length $k_{\min}$ or—if the preceding request is shorter than $k_{\max}$—extend the current block downward by a request that is one unit longer. The second option is excluded only if the preceding request has the maximum length $k_{\max}$. How often can this situation, namely a request of length $k_{\max}$ preceding another one, occur at most? There are $n-1$ requests preceding another one. Also, any block featuring a request of length $k_{\max}$ comprises $k_{\max} - k_{\min} + 1$ requests. Therefore, the answer is

$$d := \left\lfloor \frac{n-1}{k_{\max} - k_{\min} + 1} \right\rfloor.$$

Thus, we always have at least $n-1-d$ binary choices during the construction of an instance and the number of different instances is bounded from below by $|\mathcal{I}| \geq 2^{n-1-d}$. As a result, the required number of advice bits is at least $\log |\mathcal{I}| \geq n-1-d$. □

Theorem 4. *At least $\ell/k_{\min} - 1 - \lfloor (\ell/k_{\min} - 1)/(k_{\max} - k_{\min} + 1) \rfloor$ advice bits are needed for optimality.*

Proof. The approach from the proof of Theorem 3 works here as well. In fact, we can use the exact same set $\mathcal{I}$ of hard instances. To obtain a lower bound that now depends on ℓ, we are free to choose the number n of requests conveniently. Since we had $\ell = nk_{\min}$ for all instances from $\mathcal{I}$ before, we now set $n := \ell/k_{\min}$. Using this to substitute n in Theorem 3 directly yields the desired result. □

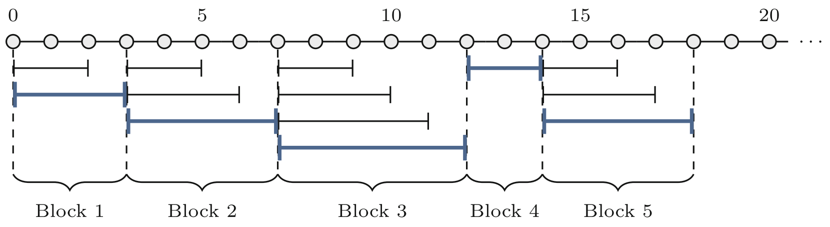


Fig. 2. An instance $I \in \mathcal{I}$ with $k_{\min} = 2$, $k_{\max} = 5$, $n = 13$ and $k = 5$. Requests are presented from top to bottom and from left to right. The highlighted optimal solution contains the last request of each block. Each block is adjacent to the previous one.

3.2 No Advice

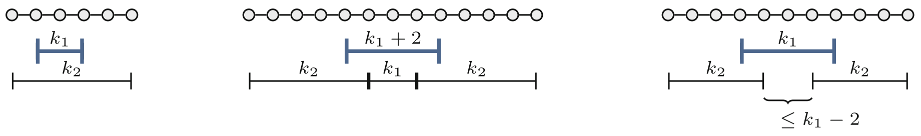
Having explored the boundaries on advice complexity for optimality, we now turn to the opposite end of the spectrum: What is the best strict competitive ratio achievable by a classic online algorithm operating without advice? First, we analyze the strict competitive ratio on only a subset of the requests, for which even better lower and upper bounds on the request lengths might apply. In Lemma 1, we give a case-based upper bound for this scenario, and we extract a more general bound that encompasses all cases in Corollary 1. Although the advantage of considering only subsets of requests may not be obvious at first, this will prove to be useful for Theorem 6 in the next section. Additionally, we can directly derive from Lemma 1 a greedy algorithm that is given in Theorem 5, together with a perfectly matching lower bound.

Lemma 1. *Let I be some input sequence for* LWDPA *and let $I' \subseteq I$ be a subset of its requests. Moreover, let k_1 and k_2 be lower and upper bounds, respectively, on the lengths of all requests from I', with $k_{\min} \le k_1 \le k_2 \le k_{\max}$. Then, the strict competitive ratio of* Greedy*—the algorithm that accepts every request that is not blocked yet—on the requests from I' is*

$$c \le \begin{cases} k_2 & \text{if } k_1 = 1, \\ (2k_2 + k_1)/(k_1 + 2) & \text{if } k_1 \ge 2 \text{ and } k_1 + 1 < k_2 < k_1^2/4, \\ 2k_2/k_1 & \text{otherwise.} \end{cases}$$

Proof. The worst case for the greedy algorithm occurs in different situations, depending on the lengths k_1 and k_2. We distinguish several cases depending on the length of an accepted request in comparison to the length of the subpath which the blocked requests could have covered otherwise. All relevant cases are depicted in Fig. 3.

If $k_1 = 1$, only the cases depicted in Fig. 3a and 3b can occur. In the case corresponding to Fig. 3a, Greedy accepts one request of length $k_1 = 1$, which later blocks an optimal request of length k_2, leading to a strict competitive ratio of k_2. In the case corresponding to Fig. 3b, Greedy accepts a request of length $k_1 + 2$, which blocks three requests appearing later: One request of



(a) Case 1. **(b)** Case 2. **(c)** Case 3.

Fig. 3. The highlighted requests are those accepted by GREEDY. The three figures show which requests can be presented afterwards such that the worst case for GREEDY occurs, depending on k_1 and k_2. In case 1, a request of length k_1 is accepted, blocking an optimal request of length k_2. This situation can always occur, independently of the values k_1 and k_2. Case 2 is only possible if $k_2 \geq k_1 + 2$. Here, an accepted request of length $k_1 + 2$ can block one request of length k_1 and two requests of length k_2. In case 3, two requests of length k_2 are blocked by one accepted request of length k_1. This can only happen if $k_1 \geq 2$.

length $k_1 = 1$ and two requests of length k_2, yielding a strict competitive ratio of $(2k_2 + 1)/3 \leq k_2$. The strict competitive ratio in the case $k_1 = 1$ is thus $c \leq k_2$.

If $k_1 \geq 2$, we further distinguish two subcases. If $k_2 \leq k_1 + 1$, the case depicted in Fig. 3b cannot occur. We can quickly convince ourselves that, under this assumption, the worst case realized in the situation of Fig. 3c is where GREEDY accepts one request of length k_1 which blocks two requests of length k_2. This gives us a strict competitive ratio of $c \leq 2k_2/k_1$ in this case.

If $k_1 \geq 2$ and $k_2 \geq k_1 + 2$, the worst case occurs when requests are presented according to Fig. 3b or 3c. (Requests may also be presented according to Fig. 3a; however, requests arriving according to Fig. 3c always ensure an even worse performance of the greedy algorithm.) We have to distinguish two further subcases; $k_2 \geq k_1^2/4$ and $k_2 < k_1^2/4$. In the former case, the worst-case strict competitive ratio is $2k_2/k_1$, as

$$\frac{2k_2}{k_1} = \frac{2k_2(k_1+2)}{k_1(k_1+2)} = \frac{2k_2k_1 + 4k_2}{k_1(k_1+2)} \geq \frac{2k_2k_1 + k_1^2}{k_1(k_1+2)} = \frac{2k_2 + k_1}{k_1 + 2}.$$

Analogously, in the case that $k_2 < k_1^2/4$, the strict competitive ratio obtained in the worst case is $(2k_2 + k_1)/(k_1 + 2)$. □

Corollary 1. *Let I be some input sequence for* LWDPA *and let $I' \subseteq I$ be a subset of its requests. Moreover, let k_1 and k_2 be lower and upper bounds, respectively, on the lengths of all requests from I', with $k_{\min} \leq k_1 \leq k_2 \leq k_{\max}$. Then, the strict competitive ratio of* GREEDY *on the requests from I' is bounded from above by $2k_2/k_1 + 1$.*

Proof. We only have to consider the three different cases from Lemma 1 and make sure the strict competitive ratio is bounded from above by $2k_2/k_1 + 1$ in each case. It is obvious that the claim holds for the first and the last case. For the second case, we make the following transformation.

$$\frac{2k_2+k_1}{k_1+2} < \frac{2k_2+k_1}{k_1} = \frac{2k_2}{k_1} + 1. \qquad \square$$

Theorem 5. *There is an online algorithm without advice, namely* Greedy, *that has a strict competitive ratio of c, and no online algorithm without advice can have a strict competitive ratio better than c, with*

$$c = \begin{cases} k_{\max} & \textit{if } k_{\min} = 1, \\ (2k_{\max}+k_{\min})/(k_{\min}+2) & \textit{if } k_{\min} \geq 2 \textit{ and } k_{\min}+1 < k_{\max} < k_{\min}^2/4, \\ 2k_{\max}/k_{\min} & \textit{otherwise.} \end{cases}$$

Proof. The first part of the statement follows directly from Lemma 1 by setting $I' := I$, $k_1 := k_{\min}$, and $k_2 := k_{\max}$.

For the second part, we make the following considerations. Since we cannot use any advice bits, we only have one deterministic algorithm available. Thus, let A be some arbitrary but fixed deterministic algorithm for lwDPA. We show that for any combination of $k_{\min}$ and $k_{\max}$, there is a hard set of two input sequences for A that leads to the strict competitive ratio mentioned in the theorem. The hard input sequences we consider are derived from the worst-case scenarios depicted in Fig. 3. If $k_{\min} = 1$, we consider the input consisting of only one request of length 1 and the input consisting of this request and a second one of length $k_{\max}$ underneath it as depicted in Fig. 3a. The algorithm A can either accept the first request, leading to a strict competitive ratio of $k_{\max}$ on the second of these two inputs, or reject it, leading to a strict competitive ratio of 1/0 on the first of the two inputs. Overall, A cannot have a strict competitive ratio better than $k_{\max}$.

Similarly, we can construct hard input sequences for the case that $k_{\min} \geq 2$ and $k_{\min}+1 < k_{\max} < k_{\min}^2/4$ according to the scenario depicted in Fig. 3b, and those for the remaining case according to Fig. 3c.

In all three cases, we choose the length of the path just long enough for all requests to fit in; this is $\ell := k_{\max}$ in the first case, $\ell := 2k_{\max}+k_{\min}$ in the second, and $\ell := 2k_{\max}+k_{\min}-2$ in the third case. $\square$

3.3 Trade-Off

Having covered the two extreme cases, classic online algorithms and optimal online algorithms with advice, we now we now strive to find the best possible behavior in between, balancing out advice complexity and strict competitiveness. Theorem 6 proves an upper bound by describing a class-based greedy algorithm. It is a great choice when only few advice bits are given, but less so with increasing advice complexity. Corollary 2 shows that it cannot undercut a strict competitive ratio logarithmic in $\varphi := k_{\max}/k_{\min}$, even when reading arbitrarily many advice bits. This problem is remedied by Theorem 7, which incorporates the approach of Theorem 6, but also makes efficient use of ample available advice by performing a sort of preprocessing.

Finally, we prove two lower bounds. Theorem 8 is evidently a counterpart to Theorem 6, with the same order of magnitude for a constant number of advice bits. Theorem 9, on the other hand, states a very versatile, albeit quite technical result. The two Corollaries 5 and 6 bring it in a more applicable form. The first one improves on the lower bound of Theorem 8 in the case of a strict competitive ratio logarithmic in φ; the second one yields the missing lower bound for even larger advice complexities, complementing Theorem 7 and Corollary 3.

Theorem 6. *There is an online algorithm reading b bits of advice that has a strict competitive ratio of $c \leq 2^b \cdot \left(2 \cdot \sqrt[2^b]{\varphi} + 1\right)$.*

Proof. The following is a variation of the randomized strategy *Classify and Randomly Select* for DPA, first proposed by Awerbuch et al. [1], adapted to LWDPA in our advice setting.

We divide the requests of the input sequence I into 2^b classes according to their lengths. Let class C_i contain all requests of length k_i with

$$k_{\min} \cdot \sqrt[2^b]{\varphi^{i-1}} \leq k_i < k_{\min} \cdot \sqrt[2^b]{\varphi^{i}}$$

for all i with $1 \leq i \leq 2^b$, and let C_{2^b} additionally contain all requests of length $k_{\max}$. This way, each request is contained in exactly one class since $k_{\min} \cdot \sqrt[2^b]{\varphi^0} = k_{\min}$ and $k_{\min} \cdot \sqrt[2^b]{\varphi^{2^b}} = k_{\min} \cdot \varphi = k_{\max}$. Furthermore, let A_i be the deterministic algorithm that accepts all requests from class C_i greedily and rejects all others. The oracle specifies the algorithm $A^* \in \{A_1, A_2, \ldots, A_{2^b}\}$ that has the largest gain on I among all algorithms A_i. Let us denote this maximum gain by $a^* := \text{gain}(A^*(I))$. To specify A^*, the oracle uses b advice bits.

Now let us consider some arbitrary but fixed optimal solution $\text{OPT}(I)$. Plugging $k_1 := k_{\min} \cdot \sqrt[2^b]{\varphi^{i-1}}$ and $k_2 := k_{\min} \cdot \sqrt[2^b]{\varphi^{i}}$ into Corollary 1, we can deduce that for all i, $\text{OPT}(I)$ has a gain of at most

$$a^* \cdot \left(\frac{2k_2}{k_1} + 1\right) = a^* \cdot \left(\frac{2k_{\min} \cdot \sqrt[2^b]{\varphi^{i}}}{k_{\min} \cdot \sqrt[2^b]{\varphi^{i-1}}} + 1\right) = a^* \cdot \left(2 \cdot \sqrt[2^b]{\varphi} + 1\right)$$

on the requests from C_i.

As the overall gain of $\text{OPT}(I)$ can be at most the sum of the gains on all 2^b classes, we obtain

$$\text{gain}(\text{OPT}(I)) \leq 2^b \cdot a^* \cdot \left(2 \cdot \sqrt[2^b]{\varphi} + 1\right),$$

and hence the strict competitive ratio of A^* on I is

$$\frac{\text{gain}(\text{OPT}(I))}{\text{gain}(A^*(I))} \leq \frac{2^b \cdot a^* \cdot \left(2 \cdot \sqrt[2^b]{\varphi} + 1\right)}{a^*} = 2^b \cdot \left(2 \cdot \sqrt[2^b]{\varphi} + 1\right).$$

□

As mentioned above, the algorithm of Theorem 6 is most efficient for little advice. For no advice at all, $b = 0$, we get $c \leq 2\varphi + 1$; this coincides with the upper bound that we get from Corollary 1 for the adviceless greedy algorithm

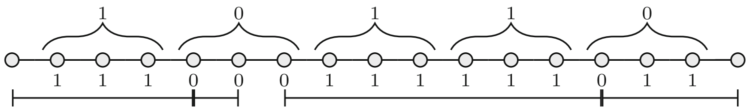


Fig. 4. An example with path length $\ell = 16$. The requests are those of the fixed optimal solution on the bases of which the advice is given. There are $\ell - 1 = 15$ inner vertices, gathered into groups of size s. The top part of the figure gives the advice string that results from groups of size $s = 3$. Underneath, the advice string for $s = 1$ is given. From this string, we can easily deduce the advice string for general $s \geq 1$ by conjoining the bits groupwise.

on the complete input sequence I by plugging in $k_1 := k_{\min}$ and $k_2 := k_{\max}$. For a single bit of advice, $b = 1$, the bound reads $c \leq 2 \cdot (2\sqrt{\varphi} + 1)$. This is a marked improvement for large φ, whose influence is dampened significantly. We pay with an additional factor of $2^b = 2$, however. This problem exacerbates quickly with increasing advice complexity. In fact, there is a threshold beyond which more advice even worsens the bound. The standard method for finding extrema reveals it to be $b = \log(\ln \varphi / (1 + W(1/(2\mathrm{e}))))$, where $W(x)$ is the Lambert W function. For this value, Theorem 6 yields its lowest upper bound on the strict competitive ratio: There is a strictly $(4.41 \cdot \log \varphi)$-competitive online algorithm that reads $\log \log \varphi - 0.73$ advice bits. The following corollary provides a more tangible bound in the same order of magnitude.

Corollary 2. *There is a strictly* $(5 \log \varphi)$*-competitive online algorithm that reads* $\log \log \varphi$ *advice bits.*

Suppose now that a rather large amount of advice is available, φ or $\ell/2$ bits for example. Only a tiny fraction of advice can be put to good use in the approach of Theorem 6; most of it would be wasted on the algorithm. Theorem 7 shows how to use this otherwise superfluous advice in a kind of preprocessing. The underlying idea is that the oracle first gives information about transition points, similarly to the optimal algorithm of Theorem 2. If the advice does not suffice to communicate their locations exactly, we reveal them only approximately in a way that is useful nonetheless. In essence, this first bulk of advice allows us to predict all long optimal requests with reasonable accuracy and ensure that enough of them will be accepted. The remaining advice can be spent in the spirit of Theorem 6 on the remaining requests, with a virtual upper bound of $2s - 1$ taking the place of $k_{\max}$.

Theorem 7. *For any* $s \geq 1$ *and any* $b \geq 0$*, there is an online algorithm that is strictly* $\max\{6 - 8/(s+1), 2^b(2 \cdot \sqrt[2^b]{(2s-1)/k_{\min}} + 1)\}$*-competitive and uses* $(\ell - 1)/s + b$ *advice bits.*

Proof. As usual, the oracle fixes an optimal solution; let $r_1, \ldots, r_k$ be its requests. Any request r_i is a subpath with two leaves plus a set R_i of inner vertices. This set is empty if and only if the request has length 1. Since two requests $r_i \neq r_j$

of a solution cannot overlap, any two sets $R_i \neq R_j$ are separated by at least one vertex; we will use this below. The given path of length ℓ contains exactly $\ell - 1$ inner vertices. We gather these into groups G_i, each of size s, as illustrated in Fig. 4 for $s = 3$. For every group G_i, the oracle communicates to the algorithm by a single bit b_i whether the group's vertices are enclosed within a single optimal request, that is, $b_i = 1 \iff \exists j \in \{1, \ldots, k\} \colon G_i \subseteq R_j$. This requires one bit per group, hence $(\ell - 1)/s$ bits in total.

How does this information help us? We begin with the case $s = 1$. Here, the advice reveals the exact location of all vertices of all R_i. Remember that R_i is the set of inner vertices of r_i. Since these sets are pairwise separated, we can directly infer the exact position of all requests r_i containing any inner vertex, that is, all optimal requests of length 2 or greater. Hence, the algorithm can *reserve* these subpaths and accept the fitting requests whenever they appear. The requests of length 1, in contrast, need to be accepted exactly if the *reservations* allow for it. Indeed, either they are blocked by longer optimal requests and can therefore not be part of the chosen solution; or they can and must be included in the solution by its optimality. The algorithm will thus reconstruct an optimal solution, using only $\ell - 1$ advice bits.

Now to the general case $s \geq 1$. The advice bit of a group G_i is still 1 if and only if there is a request r_j that encloses all its vertices, that is, $G_i \subseteq R_j$. If this is true for two neighboring groups, they must be contained within the same request, that is, $G_i, G_{i+1} \subseteq R_j$, since the R_j are pairwise disjoint. The groups are also disjoint, so $|G_i| + |G_{i+1}| \leq |R_j|$. By the same reasoning, we see that any substring B of the advice string that consists of $|B|$ consecutive ones stands for $|B|$ groups whose $|B| \cdot s$ vertices are inner vertices of a single optimal request r_j; hence $|B| \cdot s \leq |R_j|$.

Denote the set of all maximal substrings of ones by $\mathcal{B}$. For any $B \in \mathcal{B}$, we can state an upper limit of $|R_j| \leq |B|s + 2(s - 1)$. Otherwise, the substring would not be maximal: The upper bound is attained if and only if R_j overlaps the vertices of the groups corresponding to B by $s - 1$ vertices on each side. If the overlap were any larger on either side, then the respective neighboring bit of the substring B would also have been set to 1 instead of 0, contradicting the maximality.

This insight about R_j translates directly to r_j, the optimal request associated with B; just add the two leaves: For the upper limit, r_j covers exactly $|B| + 2$ groups, those of B plus one additional on each side; its length is therefore $(|B| + 2)s - 1$. The lower limit is attained if r_j only contains the two leaves in addition to the inner vertices of the $|B|$ groups. In this case, the length of r_j is $|B|s + 1$. Thus, the optimal solution's gain associated with block B stays in the range $[|B|s + 1, (|B| + 2)s - 1]$.

Now, we might hope for our algorithm to accept at least one request within the described boundaries for each substring; this would result in a lower bound of $\sum_{B \in \mathcal{B}} |B|s + 1 \leq \text{gain}'(\mathcal{A}(I))$, where gain$'$ denotes the gain from requests that cover at least one group. Since there exists at least one fitting request for each block, namely r_j, the algorithm could just wait for the first one. Due to the mentioned potential overlap on either side, it may happen, however, that an accepted request for one substring blocks all suited requests of the next substring. In fact, this is only possible for two neighboring substrings that are separated by a single 0. This might be the case for all substrings, however. So we forego our unrealistic hopes on every second substring $B \in \mathcal{B}$ and focus on the others. For those, we can now reserve subpaths of size $(|B|+2)s-1$, large enough to ensure an accepted request within the given range.

Among the two possible choices of alternate substrings of ones, the algorithm picks that with the larger grand total of ones. This way, the actual lower bound gets cut down to no less than half of what we have calculated above,

$$\frac{1}{2}\sum_{B\in\mathcal{B}} |B|s+1 \leq \text{gain}'(\mathcal{A}(I)).$$

For the optimal solution, we have $\text{gain}'(\textsc{Opt}(I)) \leq \sum_{B\in\mathcal{B}}((|B|+2)s-1)$. The strict competitive ratio on the reserved subpaths can therefore be bounded by 6:

$$\frac{\text{gain}'(\textsc{Opt}(I))}{\text{gain}'(\mathcal{A}(I))} \leq 2+2\frac{\sum_{B\in\mathcal{B}}\Big(2s-2\Big)}{\sum_{B\in\mathcal{B}}\Big(|B|s+1\Big)} \leq 2+2\frac{2s+2-4}{s+1} = 6-\frac{8}{s+1}.$$

We have not yet accounted for the gain on the unreserved subpaths. For this part, all optimal requests went undetected by any advice bit. Hence, they have a length of $2s-1$ or less, as seen by setting $|B|=0$ in the range established above. The algorithm can thus greedily accept all requests with lengths up to $2s-1$ and be strictly $(2s-1)$-competitive on this remaining part. Employing the more sophisticated greedy approach of Theorem 6, we achieve a strict competitive ratio of $2^b(2\cdot\sqrt[2^b]{(2s-1)/k_{\min}}+1)$ by granting b additional advice bits. This way, the algorithm uses $(\ell-1)/s+b$ advice bits overall and guarantees

$$\frac{\text{gain}(\textsc{Opt}(I))}{\text{gain}(\mathcal{A}(I))} \leq \max\Big\{6-\tfrac{8}{s+1}, 2^b\Big(2\cdot\sqrt[2^b]{(2s-1)/k_{\min}}+1\Big)\Big\}. \qquad \square$$

The following corollary extracts a more accessible result from Theorem 7 that will later be complemented by Corollary 6.

Corollary 3. *For any constant $k \geq 6$, the number of advice bits sufficient to achieve a strict competitive ratio of $c \leq k$ is at most*

$$2\cdot\frac{\ell-1}{2^{k/5}\cdot k_{\min}+1}+\log\left(\frac{k}{5}\right) \in \mathcal{O}\Big(\frac{\ell}{k_{\min}}\Big).$$

Proof. We plug the values $s := (2^{k/5}\cdot k_{\min}+1)/2$ and $b := \log(k/5)$ into Theorem 7 and obtain that there exists an algorithm with a strict competitive ratio of at most

$$c = \max\left\{6-\frac{8}{s+1}, \tfrac{k}{5}\Big(2\cdot\sqrt[k/5]{2^{k/5}}+1\Big)\right\} \leq \max\left\{6, \tfrac{k}{5}(2\cdot 2+1)\right\} = k$$

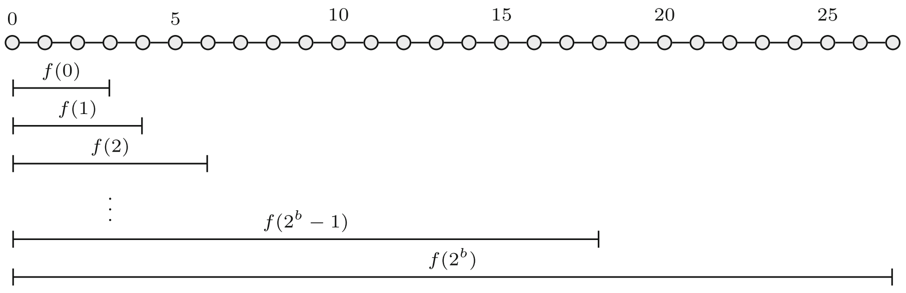


Fig. 5. The 2^b+1 requests $r_0,\dots,r_{2^b}$ of $I_{2^b}\in\mathcal{I}$.

and an advice complexity of

$$b = \frac{(\ell-1)\cdot 2}{2^{k/5}\cdot k_{\min}+1} + \log\left(\frac{k}{5}\right) \in \mathcal{O}\left(\frac{\ell}{k_{\min}}\right). \qquad \square$$

The following lower bound is a good match for Theorem 6 in the case of a constant number of advice bits.

Theorem 8. *No online algorithm reading at most b advice bits can have a strict competitive ratio better than $\sqrt[2^b]{\varphi}$.*

Proof. Let $f(i) := k_{\min}\cdot\sqrt[2^b]{\varphi^i}$ for all i with $0\le i\le 2^b$. Note that $f(0)=k_{\min}$ and $f(2^b)=\varphi\cdot k_{\min}=k_{\max}$. For any number b of advice bits, we consider the following set $\mathcal{I}=\{I_0,\dots,I_{2^b}\}$ of instances. Instance I_{2^b} presents 2^b+1 left-aligned requests $r_0,\dots,r_{2^b}$ of increasing length $f(i)$, as illustrated in Fig. 5. Instance $I_i\in\mathcal{I}$ presents only the first $i+1$ of these requests, namely $r_0,\dots,r_i$.

With b advice bits, we only have 2^b deterministic algorithms available, and thus at least two out of the 2^b+1 instances from $\mathcal{I}$ must be processed by the same deterministic algorithm. Hence, let A denote a deterministic algorithm that processes at least two instances from $\mathcal{I}$ and let I_i and I_j with $i<j$ be two of the instances processed by A. The gain of A on any instance I_s is

$$\text{gain}(A(I_s)) = \begin{cases} f(t) & \text{if } A \text{ accepts a request } r_t \text{ with } t\le s, \text{ and} \\ 0 & \text{otherwise.}\end{cases}$$

The optimal gain for any I_s is $f(s)$, of course. The two instances I_i and I_j are identical for the first $i+1$ requests; hence, A must treat them the same way up to this point. We distinguish two cases and show that A performs badly on at least one of the instances I_i or I_j in both cases:

First, if A does not accept a request $r_t\in\{r_0,\dots,r_i\}$, then its gain on I_i is $\text{gain}(A(I_i))=0$ and the strict competitive ratio is unbounded. Second, if A does accept a request $r_t\in\{r_0,\dots,r_i\}$, then its gain on I_j is $\text{gain}(A(I_j))=f(t)\le f(i)$, which leads to a strict competitive ratio of

$$c=\frac{f(j)}{f(t)}\ge\frac{f(j)}{f(i)}\ge\frac{f(i+1)}{f(i)}=\frac{\sqrt[2^b]{\varphi^{i+1}}\cdot k_{\min}}{\sqrt[2^b]{\varphi^i}\cdot k_{\min}}=\sqrt[2^b]{\varphi}.$$

This means that no matter which algorithm A is used for two instances from $\mathcal{I}$, on one of these two instances the strict competitive ratio cannot be better than $\sqrt[2b]{\varphi}$. $\square$

The following proof of Theorem 9 adapts a method from Gebauer et al. [8] that has been used to analyze the unweighted and non-parametrized disjoint path allocation problem on paths. First, we need a result that is proven in the mentioned paper. It is a slight variant of a bound for the tail of the hypergeometric distribution.

Fact 1 (Corollary 1 from Gebauer et al. [8]). *Let X be a discrete random variable with a hypergeometric distribution with parameters M, N, and n, and let $t \geq 0$. Then, for every $M' \leq M$, we have*

$$\Pr\left(X \leq n \cdot \frac{M'}{N} - tn\right) \leq \mathrm{e}^{-2t^2 n} .$$

We construct a set $\mathcal{I}$ of instances which is hard for many deterministic algorithms at once and thereby show that many deterministic algorithms are necessary to achieve a good strict competitive ratio on all instances from $\mathcal{I}$. For the time being, we choose three parameters x, y, and z with $x, y, z \in \mathbb{N}_{\geq 1}$ and construct the set $\mathcal{I}$ subject to those parameters to obtain a result as general as possible. The parameters are chosen in such a way that the following requirements are satisfied.

$$x \leq \log \varphi, \tag{1}$$

$$y \leq \log\left(\ell/k_{\min}\right) - 2x, \qquad \text{and} \tag{2}$$

$$z \leq \sqrt{2^y/\ln x}. \tag{3}$$

The general result we obtain eventually, depending on the variables x and z, is the following.

Theorem 9. *Let $x, z \in \mathbb{N}_{\geq 1}$. For any $x \leq \log \varphi$ and $z \leq \sqrt{\ell/(k_{\min} \cdot 4^x \cdot \ln x)}$, any online algorithm for* LWDPA *needs to read at least*

$$b = \frac{\ell}{k_{\min}} \cdot \frac{\log \mathrm{e}}{z^2 \cdot 4^x} - \log x$$

advice bits to obtain a strict competitive ratio of

$$c \leq \frac{x+2}{2 \cdot \left(1 + \frac{x}{z}\right)}.$$

Proving this will be the goal of our subsequent considerations. Afterwards, we will also see how to choose reasonable values for the parameters to get less general but more expressive statements. More specifically, we will derive two corollaries from Theorem 9, which will give us almost matching lower bounds to two of the upper bounds proven above. Before we prove Theorem 9, however, we will have to make the necessary arrangements.

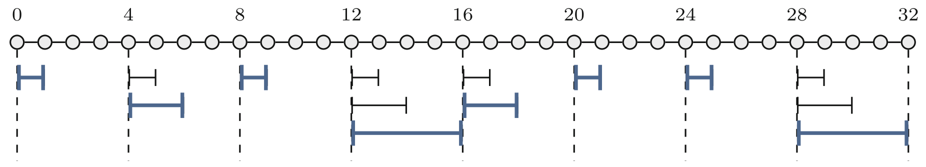


Fig. 6. An example for $\ell = 32$, $k_{\min} = 1$, $x = 2$, and $y = 1$. The dashed lines indicate the segments. There are $2^{x+y} = 8$ segments of width $2^2 \cdot k_{\min} = 4$ each, and requests appear in $x + 1 = 3$ phases; 8 requests of length 1 in phase 0, then 4 of length 2 in phase 1, and 2 of length 4 in phase 2. Good requests are marked by thick lines.

Let us start with a description of how to construct the hard instance set $\mathcal{I}$. In each instance, requests are presented in $x + 1$ phases, starting with phase 0. We divide the path into 2^{x+y} segments of width $2^x \cdot k_{\min}$ each, and all requests presented during the whole computation are always aligned with the left border of the respective segment. Phase 0 contains 2^{x+y} requests of length $2^0 \cdot k_{\min}$ each, one in each segment. In each phase i with $1 \leq i \leq x$, we choose exactly half of the segments that contained requests in phase $i-1$ and present a request of twice the size in those segments. Let us call a request *good* if it has no further requests underneath and *bad* otherwise. Figure 6 shows an example.

Let us define $h := x + y$. We make the following observation.

Observation 1. *For each i with $0 \leq i \leq x$, phase i contains 2^{h-i} requests of length $2^i \cdot k_{\min}$ each.*

We have to check two points concerning the presentation of requests: First, whether the path is long enough to hold all segments, and second, whether all requests have lengths between $k_{\min}$ and $k_{\max}$. Our requirement (2) ensures the first point, as

$$2^{x+y} \cdot 2^x \cdot k_{\min} = 2^{2x+y} \cdot k_{\min} \leq 2^{\log\left(\frac{\ell}{k_{\min}}\right)} \cdot k_{\min} = \ell.$$

Furthermore, according to Observation 1, all requests are between $2^0 \cdot k_{\min}$ and $2^x \cdot k_{\min}$ in length. So all requests are large enough, and requirement (1) ensures that no requests are larger than $k_{\max}$, since

$$2^x \cdot k_{\min} \leq 2^{\log \varphi} \cdot k_{\min} = \varphi \cdot k_{\min} = k_{\max}.$$

For all instances constructed in the way described before, the gain of their respective optimal solutions is the same, as stated in the following observation.

Observation 2. *For each instance $I \in \mathcal{I}$, the gain of the optimal solution is*

$$\text{gain}(\textsc{Opt}(I)) = (x + 2) \cdot k_{\min} \cdot 2^{h-1}.$$

Proof. The optimal solution contains exactly all good requests. For each phase, these are exactly half of the requests presented, except for the last phase, in which all requests are good and hence contained in the optimal solution. Thus, using Observation 1, the gain of the optimal solution on each instance I is

$$\text{gain}(\textsc{Opt}(I)) = \left(\sum_{i=0}^{x-1} \frac{2^{h-i} \cdot 2^i k_{\min}}{2}\right) + 2^{h-x} \cdot 2^x k_{\min} = (x+2)k_{\min} \cdot 2^{h-1}.$$

□

The set $\mathcal{I}$ can be represented by a tree in which each vertex corresponds to a subset of instances. The depth of this tree is x (i.e., it has $x+1$ levels); the root on level 0 represents all instances from $\mathcal{I}$, and every vertex on level i represents the set of all instances with the same particular set of requests in phases $0, \ldots, i$. Every leaf is located on level x and represents a single instance from $\mathcal{I}$. The same tree representation of $\mathcal{I}$ has also been used by Gebauer et al. [8]. However, it is worth mentioning that in our case, the vertices of the tree have a different number of children since we construct the instances from $\mathcal{I}$ in a different way.

Consider some vertex v on level i and the corresponding set of instances $\mathcal{I}_v$. Now consider any deterministic algorithm A, given an arbitrary instance from $\mathcal{I}_v$ as input. Then A is in the same state during the whole computation up to and including phase i, independently of the exact instance from $\mathcal{I}_v$ that A is given, meaning that it sees and accepts the exact same requests up until the end of phase i.

From now on, let A be arbitrary but fixed. For a given vertex v on level i, let $a(i)$ be the number of accepted requests of A on any instance from $\mathcal{I}_v$ during phase i, and let $g(i)$ be the gain of A achieved during phase i. Obviously, due to Observation 1,

$$g(i) = a(i) \cdot 2^i \cdot k_{\min}. \tag{4}$$

Moreover, let us define $\hat{g}(i)$ to be the gain that A achieves in all phases up to and including phase i and, in accordance with this definition, $\hat{g}(-1)$ to be the total gain of requests accepted before the start of computation. Hence,

$$\hat{g}(i) := \sum_{j=0}^{i} g(j) \quad \text{and} \quad \hat{g}(-1) := 0. \tag{5}$$

Now let us define the sets of requests $B(i)$ and $B^+(i)$ as follows. Let $B(i)$ be the set of requests from phase i that are already blocked at the beginning of phase i. Let $B^+(i)$ be the set of requests from phase i that are blocked at the beginning of phase $i+1$, including those that were accepted during phase i and are now blocking themselves.

We call a phase i *bad* for A if $B(i)$ contains at least

$$d_i := \frac{1}{2^i} \cdot \left(\frac{\hat{g}(i-1)}{k_{\min}} - \frac{i}{z} \cdot 2^h\right) \tag{6}$$

requests and *good* otherwise. Moreover, let us call a vertex v on level i *bad* for A if, when given an instance from $\mathcal{I}_v$ as its input, phase i is bad for A, and otherwise let us call v *good* for A. Since each instance corresponds to a vertex on level $x+1$, an instance is bad for A if and only if phase $x+1$ is bad for A.

Observation 3. *If at least d_{i+1} requests from $B^+(i)$ are bad, then phase $i+1$ is bad for A.*

Proof. As defined before, $B^+(i)$ is the set of requests from phase i that are blocked at the beginning of phase $i+1$. If at least d_{i+1} requests of those are bad, then at least d_{i+1} requests are presented in phase $i+1$ that are already blocked at the beginning of phase $i+1$; this matches the definition of a bad phase. □

Lemma 2. *For each bad vertex, the fraction of bad vertices among its children is at least*

$$1 - \mathrm{e}^{-2^y/z^2}.$$

Proof. Let v be a bad vertex on level i. Since v is bad, phase i must be bad for A when A is given any instance from $\mathcal{I}_v$ as input. Thus, $B(i)$ must contain at least d_i requests. As A accepts $a(i)$ requests in phase i, the set $B^+(i)$ contains at least $d_i + a(i)$ requests. According to Observation 3, if at least d_{i+1} requests from $B^+(i)$ are bad, phase $i+1$ is bad. Hence, a sufficient condition for a child w of v to be bad is that at least d_{i+1} requests from $B^+(i)$ are bad when A is given an instance from $\mathcal{I}_w$ as input. Thus, we have the following scenario. There are $N := 2^{h-i}$ requests in phase i; out of these, $M \geq M' := d_i + a(i)$ are blocked at the beginning of phase $i+1$ and hence contained in $B^+(i)$. Each child w of v corresponds to the set of instances in which the same set of $n := 2^{h-i-1}$ requests from phase i are bad. This in turn corresponds to the following. We have an urn containing N balls, out of which $M \geq M'$ are black; we draw n balls without replacement, and we are interested in the probability that the number of black balls drawn is at least d_{i+1}.

Let X_i be the random variable counting the number of bad requests from $B^+(i)$ in this scenario. X_i has a hypergeometric distribution with parameters M, N, and n, and we are interested in $\Pr(X_i \geq d_{i+1})$.

We can bound the probability

$$\Pr\left(X_i \leq n \cdot \frac{M'}{N} - tn\right) = \Pr\left(X_i \leq \tfrac{d_i + a(i) - t \cdot 2^{h-i}}{2}\right)$$

from above for any $t \geq 0$ by using Fact 1. We obtain

$$\Pr\left(X_i \leq \frac{d_i + a(i) - t \cdot 2^{h-i}}{2}\right) \leq \mathrm{e}^{-2t^2 \cdot n} = \mathrm{e}^{-2t^2 \cdot 2^{h-i-1}} = \mathrm{e}^{-t^2 \cdot 2^{h-i}}. \quad (7)$$

Using (5) and (6), we get

$$d_{i+1} = \frac{1}{2^{i+1}}\left(\frac{\hat{g}(i)}{k_{\min}} - \frac{i+1}{z} \cdot 2^h\right) = \frac{1}{2^{i+1}}\left(\frac{\hat{g}(i-1)}{k_{\min}} + \frac{g(i)}{k_{\min}} - \frac{i}{z} \cdot 2^h - \frac{1}{z} \cdot 2^h\right).$$

Doing further transformations using (4) and again (6), we obtain

$$d_{i+1} = \frac{\frac{1}{2^i} \cdot \left(\frac{\hat{g}(i-1)}{k_{\min}} - \frac{i}{z} \cdot 2^h + 2^i \cdot a(i) - \frac{1}{z} \cdot 2^h\right)}{2} = \frac{d_i + a(i) - \frac{1}{z} \cdot 2^{h-i}}{2}.$$

Combining this result with (7) and choosing $t := 1/z$ yields

$$\Pr(X_i \geq d_{i+1}) \geq 1 - \Pr\left(X_i \leq \frac{d_i + a(i) - \frac{1}{z} \cdot 2^{h-i}}{2}\right) \geq 1 - \mathrm{e}^{-2^{h-i}/z^2},$$

and since $h - i \geq y$ for all i with $0 \leq i \leq x$, we finally obtain that for all such i,

$$\Pr(X_i \geq d_{i+1}) \geq 1 - \mathrm{e}^{-2^y/z^2}. \qquad \square$$

From the fraction of bad vertices among the children of bad vertices, we can now draw conclusions concerning the fraction of bad vertices on each level.

Lemma 3. *The fraction of bad vertices on level i is at least*

$$\left(1 - \mathrm{e}^{-2^y/z^2}\right)^i.$$

Proof. We prove the claim by induction on i. For the base case, we consider the root, the only vertex on level 0. The root is bad if phase 0 is bad for A, and phase 0 is bad for A if at least d_0 requests from phase 0 are already blocked at the beginning of phase 0. This is obviously the case as $d_0 = 0$ according to (5) and (6). Hence, the root is a bad vertex and thus the fraction of bad vertices on level 0 is 1, which is at least

$$\left(1 - \mathrm{e}^{-2^y/z^2}\right)^0 = 1.$$

For the induction step, let us assume that the claim holds for level $i - 1$. Let us define $v(i)$ to be the number of all vertices on level i and $b(i)$ the number of bad vertices among those. Moreover, let $c(i)$ denote the number of children of each vertex on level i. As a final definition, let $f(i)$ be the fraction of bad vertices among the children of each bad vertex on level i. Lemma 2 states that $f(i) \geq 1 - \mathrm{e}^{-2^y/z^2}$ for all i, and by the induction hypothesis it holds that $b(i-1)/v(i-1) \geq (1 - \mathrm{e}^{-2^y/z^2})^{i-1}$. Using Lemma 2, the induction hypothesis, and the fact that $b(i) \geq b(i-1) \cdot c(i-1) \cdot f(i-1)$ and $v(i) = v(i-1) \cdot c(i-1)$, the fraction $b(i)/v(i)$ of bad vertices on level i can be bounded from below by

$$\frac{b(i-1) \cdot f(i-1)}{v(i-1)} \geq \left(1 - \mathrm{e}^{-2^y/z^2}\right)^{i-1} \cdot \left(1 - \mathrm{e}^{-2^y/z^2}\right) = \left(1 - \mathrm{e}^{-2^y/z^2}\right)^i. \qquad \square$$

From this we can conclude that for each deterministic online algorithm, a large number of instances must be bad.

Corollary 4. *For any deterministic online algorithm A, the fraction of instances of $\mathcal{I}$ which are bad for A is at least* $(1-\mathrm{e}^{-2^y/z^2})^x$.

Proof. This follows directly from Lemma 3 and the fact that each single instance from $\mathcal{I}$ corresponds to a vertex on level x. □

Now we want to show that any deterministic algorithm can only accept few requests on each bad instance.

Lemma 4. *Let A be an arbitrary but fixed deterministic online algorithm for* LWDPA*, and let $I \in \mathcal{I}$ be a bad instance for A. Then the gain of A on I is*

$$\mathrm{gain}(A(I)) = \hat{g}(x) \leq 2^h \cdot k_{\min} \cdot \Big(1 + \frac{x}{z}\Big).$$

Proof. According to the definition of bad vertices and instances, an instance (corresponding to a vertex on level x) is bad if there are at least d_x requests from phase x blocked at the beginning of phase x. In this phase, A is presented 2^{h-x} requests of length $2^x \cdot k_{\min}$ due to Observation 1, so the gain of A in phase x is

$$\begin{aligned} g(x) &\leq 2^x \cdot k_{\min} \cdot \big(2^{h-x} - d_x\big) \\ &= 2^x \cdot k_{\min} \cdot \left(\frac{2^h}{2^x} + \frac{1}{2^x} \cdot \left(\frac{x}{z} \cdot 2^h - \frac{\hat{g}(x-1)}{k_{\min}}\right)\right) \\ &= 2^h \cdot k_{\min} \cdot \Big(1 + \frac{x}{z}\Big) - \hat{g}(x-1). \end{aligned}$$

For the total gain at the end of A's computation, we obtain

$$\hat{g}(x) = g(x) + \hat{g}(x-1) \leq 2^h \cdot k_{\min} \cdot \Big(1 + \frac{x}{z}\Big).$$

□

Now we can bound the strict competitive ratio of A on any bad instance from below.

Lemma 5. *The strict competitive ratio of any deterministic online algorithm on any bad instance from $\mathcal{I}$ is*

$$c \geq \frac{x+2}{2 \cdot \big(1 + \frac{x}{z}\big)}.$$

Proof. Let A be some arbitrary but fixed deterministic online algorithm for LWDPA and let $I \in \mathcal{I}$ be a bad instance for A. Using Lemma 4 and Observation 2, we obtain

$$c = \frac{\mathrm{gain}(\mathrm{OPT}(I))}{\mathrm{gain}(A(I))} \geq \frac{(x+2) \cdot k_{\min} \cdot 2^{h-1}}{2^h \cdot k_{\min} \cdot \big(1 + \frac{x}{z}\big)} = \frac{x+2}{2 \cdot \big(1 + \frac{x}{z}\big)}.$$ □

From this we can conclude that any online algorithm for LWDPA needs to read a large number of advice bits to obtain a good strict competitive ratio. Hence, we are finally ready to prove Theorem 9.

Proof (of Theorem 9). We interpret an online algorithm that reads b advice bits in the usual way as a set of deterministic algorithms $\{A_1, \ldots, A_{2^b}\}$. According to Lemma 5, each such algorithm A_i can have a strict competitive ratio of at most c only on instances which are good, and according to Corollary 4, for each A_i the fraction of good instances from $\mathcal{I}$ is at most

$$1 - \left(1 - \mathrm{e}^{-2^y/z^2}\right)^x \leq \frac{x}{\mathrm{e}^{2^y/z^2}}, \tag{8}$$

where we used Bernoulli's Inequality. Requirement (3) from the very beginning makes sure that (8) always yields a reasonable result, i.e., that the right-hand side of (8) never becomes larger than 1, since

$$\frac{x}{\mathrm{e}^{2^y/z^2}} \leq \frac{x}{\mathrm{e}^{2^y/(2^y/\ln x)}} = \frac{x}{\mathrm{e}^{\ln x}} = 1.$$

Thus, the number of deterministic algorithms that are necessary to guarantee a strict competitive ratio of c on all instances from $\mathcal{I}$ is at least $(\mathrm{e}^{2^y/z^2})/x$. Hence, the number of necessary advice bits is

$$b \geq \log\left(\frac{\mathrm{e}^{2^y/z^2}}{x}\right) = \frac{2^y}{z^2} \cdot \log \mathrm{e} - \log x.$$

Now we are almost done. As our final step, we observe that the variable y that appears in the formula for b does not have any influence on c. Since the number of necessary advice bits grows with y, we can set it to the maximum possible value according to requirement (2). Thus, we set $y := \log(\ell/k_{\min}) - 2x = \log\left(\ell/\left(k_{\min} \cdot 2^{2x}\right)\right)$, and as a result, we finally obtain that the number of necessary advice bits is

$$b \geq \frac{2^{\log\left(\frac{\ell}{k_{\min} \cdot 4^x}\right)}}{z^2} \cdot \log \mathrm{e} - \log x = \frac{\ell}{k_{\min}} \cdot \frac{\log \mathrm{e}}{z^2 \cdot 4^x} - \log x.$$

Plugging the value that we just fixed for y into requirement (3) yields the range of $z \leq \sqrt{\ell/(k_{\min} \cdot 4^x \cdot \ln x)}$ given in the theorem. □

By plugging in values for x and z into this theorem, we can derive two results which nicely complement Corollaries 2 and 3, respectively. The first one states that many advice bits are necessary to obtain a strict competitive ratio better than $1/4 \cdot \log \varphi$.

Corollary 5. *Let δ be an arbitrary constant with $0 < \delta < 1/2$ and let $\varepsilon > 2\delta$. Any online algorithm for* LWDPA *that achieves a strict competitive ratio of $\delta/2 \cdot \log \varphi$ needs to read at least $\omega\left(\varphi^{1-\varepsilon} \cdot \ell/k_{\max}\right)$ advice bits when $\varphi \in \omega(1)$.*

Proof. Let $x := \delta \log(k_{\max}/k_{\min}) = \delta \log \varphi$ and $z := \log^2 \varphi$. Then, $4^x = 2^{2x} = 2^{2\delta \log \varphi} = \varphi^{2\delta}$. We obtain

$$\frac{\frac{\ell}{k_{\min}}}{z^2 \cdot 4^x} = \frac{\frac{\ell}{k_{\max}} \cdot \frac{k_{\max}}{k_{\min}}}{\log^4 \varphi \cdot \varphi^{2\delta}} = \frac{\ell}{k_{\max}} \cdot \frac{\varphi}{\log^4 \varphi \cdot \varphi^{2\delta}} = \frac{\ell}{k_{\max}} \cdot \frac{\varphi^{1-2\delta}}{\log^4 \varphi}.$$

The number of advice bits necessary to achieve a strict competitive ratio of c is

$$b \geq \frac{\ell}{k_{\min}} \cdot \frac{\log \mathrm{e}}{z^2 \cdot 4^x} - \log x = \frac{\varphi^{1-2\delta}}{\log^4 \varphi} \cdot \frac{\ell}{k_{\max}} \cdot \log \mathrm{e} - \log \delta - \log \log \varphi.$$

For any constant $\varepsilon > 2\delta$, we have $(\varphi^{1-2\delta}/\log^4 \varphi) \in \omega\big(\varphi^{1-\varepsilon}\big)$ if $\varphi \in \omega(1)$. Therefore, the number of advice bits necessary to obtain a strict competitive ratio of c is in $\omega\big(\varphi^{1-\varepsilon} \cdot \ell/k_{\max}\big)$, for any constant $\varepsilon > 2\delta$. The value of c obtained by plugging in the values for x and z as chosen above is

$$c \geq \frac{x+2}{2(1+\frac{x}{z})} \geq \frac{\delta \log \varphi + 1}{2(1+\frac{\delta}{\log \varphi})} = \frac{(\delta \log \varphi + 1) \log \varphi}{2(\log \varphi + \delta)} = \frac{\delta \log \varphi}{2} \cdot \frac{\log \varphi + \frac{1}{\delta}}{\log \varphi + \delta}.$$

Using this together with $\delta < 1/\delta$, we can bound c from below by $(\delta \log \varphi)/2$. □

Combining this with Corollary 2, we observe a surprising fact. While $\log \log \varphi$ advice bits are sufficient to obtain a strict competitive ratio of $5 \log \varphi$, we need $\omega\big(\varphi^{1-\varepsilon} \cdot \ell/k_{\max}\big)$ advice bits to decrease the strict competitive ratio by only another constant factor. The next corollary complements Corollary 3, stating that many advice bits are necessary to obtain a constant strict competitive ratio. The two bounds match up to a constant factor.

Corollary 6. *Let $k_{\min} \in o(\ell)$. For any constant $k \leq (\log \varphi + 2)/4$, the number of advice bits necessary to achieve a strict competitive ratio of $c \leq k$ is at least*

$$\frac{\ell}{k_{\min}} \cdot \frac{\log \mathrm{e}}{(4k-2)^2 \cdot 4^{4k-2}} - \log(4k-2) \in \Omega\left(\frac{\ell}{k_{\min}}\right).$$

Proof. We set $x := z := 4k - 2$ and plug these values into Theorem 9. To be able to apply this theorem, it must hold that $x \leq \log \varphi$ and $z \leq \sqrt{\ell/(k_{\min} \cdot 4^x \cdot \ln x)}$. The first requirement is trivially fulfilled as $x = 4k - 2 \leq 4 \cdot (\log \varphi + 2)/4 - 2 = \log \varphi$. The second one is also satisfied since $k_{\min} \in o(\ell)$ and hence $\sqrt{\ell/(k_{\min} \cdot 4^x \cdot \ln x)} \in \omega(1)$, while $z \in \mathcal{O}(1)$. Due to Theorem 9, any algorithm needs to read at least

$$b \geq \frac{\ell}{k_{\min}} \cdot \frac{\log \mathrm{e}}{(4k-2)^2 \cdot 4^{4k-2}} - \log(4k-2) \in \Omega\left(\frac{\ell}{k_{\min}}\right)$$

advice bits to achieve a strict competitive ratio of

$$c \leq \frac{x+2}{2 \cdot (1+\frac{x}{z})} = \frac{4k}{2 \cdot (1+1)} = k.$$

□

Thus, combining Corollaries 3 and 6, we obtain that $\Theta(\ell/k_{\min})$ advice bits are necessary and sufficient to achieve any constant strict competitive ratio.

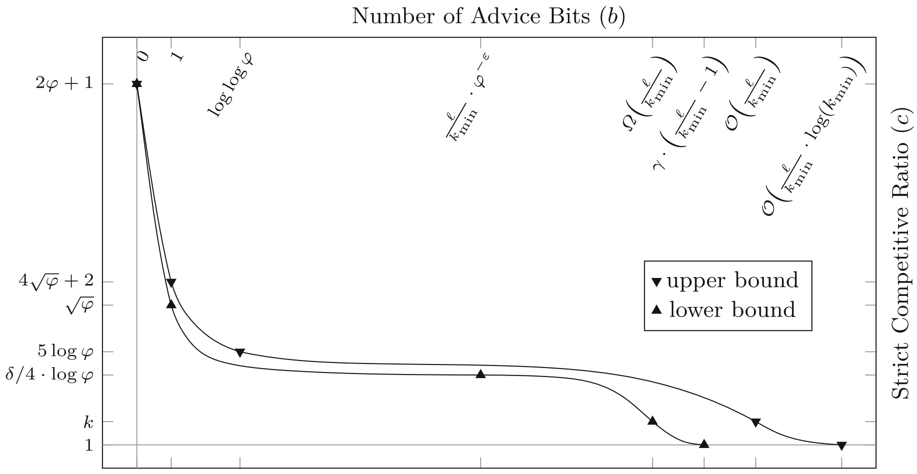


Fig. 7. A schematic plot of the results proven in this paper.

4 Conclusion

In this paper, we thoroughly studied the length-weighted disjoint path allocation problem on paths parametrized by given lower and upper bounds on the request lengths $k_{\min}$ and $k_{\max}$.

We examined the trade-off between the number of advice bits an online algorithm reads and the strict competitive ratio it can achieve. The strict competitive ratios we considered cover the entire possible range; from optimality to the worst-case strict competitive ratio when no advice is given at all. We described several algorithms for LWDPA and were able to give nearly or perfectly matching lower bounds in most cases. Our results imply a very interesting threshold behavior of the LWDPA problem.

As depicted schematically in Fig. 7, we showed the following. When an online algorithm for LWDPA is not given any advice at all, the strict competitive ratio cannot be better than $2\varphi + 1$, where $\varphi = k_{\max}/k_{\min}$, and a simple greedy approach achieves this ratio. With any constant number b of advice bits, the achievable strict competitive ratio lies between $\sqrt[2^b]{\varphi}$ and $2^b \cdot (2 \cdot \sqrt[2^b]{\varphi} + 1)$, which is matching up to a constant factor. When the algorithm is allowed to read $\log\log\varphi$ advice bits, a relatively good strict competitive ratio can be obtained already, namely $5 \log \varphi$. However, to decrease it further by only a constant factor to $\delta/4 \cdot \log \varphi$ for any $\delta < 1$, we showed that $\ell/k_{\min} \cdot \varphi^{-\varepsilon}$ advice bits are necessary, for any $\varepsilon > \delta$. We proved that $\Theta(\ell/k_{\min})$ advice bits are necessary and sufficient to achieve any constant strict competitive ratio c with $6 \leq c \leq (\log\varphi + 2)/4$. Finally, in order to be optimal, n or $(\ell - 1)/k_{\min} \cdot (1 + 2\log k_{\min})$ advice bits are sufficient; the number of advice bits necessary to be optimal is at least $\gamma \cdot (n-1)$ or $\gamma \cdot (\ell/k_{\min} - 1)$ for some constant γ that depends on both $k_{\max}$ and $k_{\min}$, but is always at least $1/2$ and approaches 1 with growing difference between $k_{\min}$

and $k_{\max}$. Thus, there is only a factor of $\mathcal{O}(\log k_{\min})$ between the lower and upper bounds for optimality in ℓ and a constant factor between those in n; moreover, in the unparametrized case, the bounds in ℓ are perfectly matching and those in n match up to an additive constant of 1.

Acknowledgements. We thank Hans-Joachim Böckenhauer and Dennis Komm for valuable discussions.

References

1. Awerbuch, B., Bartal, Y., Fiat, A., Rosén, A.: Competitive non-preemptive call control. In: Proceedings of SODA 1994, pp. 312–320. SIAM (1994)
2. Barhum, K., et al.: On the power of advice and randomization for the disjoint path allocation problem. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 89–101. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04298-5_9
3. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the advice complexity of online problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10631-6_35
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
5. Dietiker, S.: The advice complexity of the online disjoint path allocation problem. Bachelor thesis, ETH Zürich (2013)
6. Dobrev, S., Královič, R., Pardubská, D.: Measuring the problem-relevant information in input. Theor. Inform. Appl. (RAIRO) **43**(3), 585–613 (2009)
7. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theoret. Comput. Sci. **412**(24), 2642–2656 (2011)
8. Gebauer, H., Komm, D., Královič, R., Královič, R., Smula, J.: Disjoint path allocation with sublinear advice. In: Xu, D., Du, D., Du, D. (eds.) COCOON 2015. LNCS, vol. 9198, pp. 417–429. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21398-9_33
9. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15155-2_3
10. Komm, D.: An Introduction to Online Computation: Determinism, Randomization, Advice. Springer, Switzerland (2016). https://doi.org/10.1007/978-3-319-42749-2
11. Kováčová, I.: Advice complexity of disjoint path allocation. Theor. Inform. Appl. (RAIRO) **50**(2), 171–191 (2016)
12. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)

Universal Hashing via Integer Arithmetic Without Primes, Revisited

Martin Dietzfelbinger(✉)

Department of Computer Science and Automation,
Technische Universität Ilmenau, Ilmenau, Germany
martin.dietzfelbinger@tu-ilmenau.de

Abstract. Let $u, m \geq 1$ be arbitrary integers and let $r \geq (u-1)m$ be any multiple of m. Consider the multiset ("*class*") $\mathcal{H}^{\text{lin}}_{u,m,r} = \{h_{a,b} \mid 0 \leq a, b < r\}$ of *hash functions* from $U = \{0, \ldots, u-1\}$ to $M = \{0, \ldots, m-1\}$, where $h_{a,b}(x) = \lfloor((ax+b) \bmod r)/(r/m)\rfloor$, for $x \in U$. In a STACS paper of 1996 it was shown that $\mathcal{H}^{\text{lin}}_{u,m,r}$ is $\frac{5}{4}$-approximately 2-wise independent and (error-free) 2-wise independent if in addition r is a power of a prime number. Here, we revisit this result. We prove slightly stronger bounds (in part also shown by Woelfel (1999) with different methods) and we discuss applications that have appeared in the meantime, e.g., complexity lower bounds for integer multiplication, fine-grained complexity inside the class P, and the usefulness and limitations of these simple hash classes in combination with other applications of hashing.

Dedicated to Juraj Hromkovič on the occasion of his 60th birthday.

1 Introduction

Let $u, m \geq 1$ be arbitrary integers and let $r \geq (u-1)m$ be any multiple of m. Consider the "*class*" $\mathcal{H}^{\text{lin}}_{u,m,r} = \{h_{a,b} \mid 0 \leq a, b < r\}$ of *hash functions* from $U = [u] = \{0, \ldots, u-1\}$ to $M = [m] = \{0, \ldots, m-1\}$, where

$$h_{a,b}(x) = \lfloor((ax+b) \bmod r)/(r/m)\rfloor\,, \text{ for } x \in U.$$

In a STACS paper [10] of 1996 it was shown that $\mathcal{H}^{\text{lin}}_{u,m,r}$ is $\frac{5}{4}$-approximately 2-wise independent and (error-free) 2-wise independent if in addition r is a power of a prime number. In the present paper, we revisit the contribution of 1996, describe improvements of the results and discuss ramifications and applications that have appeared in the meantime.

1.1 Background: Two-Wise Independence and Universality

Two-wise independent random variables and hash functions have a multitude of applications. We mention just two: universal hashing for use in data structures

© Springer Nature Switzerland AG 2018
H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 257–279, 2018.
https://doi.org/10.1007/978-3-319-98355-4_15

[7,13,18,26,29,40] and amplifying randomness and derandomization [2,8,11,23, 28]. For a classical survey of applications, see [24].

Given sets U (the *universe*, usually finite) and M (the *range*, finite), we call a function $h\colon U \to M$ a *hash function*. Elements of U are called *keys*. We consider "*classes*" (multisets) $\mathcal{H}$ of hash functions. Such a class is called 1-*universal* if for h chosen uniformly at random from $\mathcal{H}$ the collision probability is bounded by $1/|M|$, i.e., if we have

$$\mathbf{Pr}(h(x_1) = h(x_2)) \leq \frac{1}{|M|}, \text{ for } x_1, x_2 \in U \text{ distinct.} \tag{1}$$

Class $\mathcal{H}$ is called 2-*wise independent* if for h chosen at random from $\mathcal{H}$ we have that for arbitrary distinct $x_1, x_2 \in U$ the random variables $h(x_1)$ and $h(x_2)$ are uniformly and independently distributed in M, more precisely:

$$\mathbf{Pr}(h(x_1) = i_1 \wedge h(x_2) = i_2) = \frac{1}{|M|^2}, \text{ for } x_1, x_2 \in U \text{ distinct, } i_1, i_2 \in M. \tag{2}$$

For applications, it is usually sufficient if (1) and (2) hold up to some relative error, i.e., if we require that $\mathbf{Pr}(h(x_1) = h(x_2)) \leq c/|M|$ in (1) for some $c \geq 1$ (such a class is called c-*universal*, which corresponds to the original definition of the term in [7]) or $(2-c)/|M|^2 \leq \mathbf{Pr}(h(x_1) = i_1 \wedge h(x_2) = i_2) \leq c/|M|^2$ (2) for some $c \in [1, 2)$ (then we say the class is c-*approximately* 2-*wise independent*).

By 1996, simple methods for constructing such classes had been known for quite some time. We list examples of such constructions.[1]

(a) $U = [p]$, $M = [m]$ for a prime p and some $m \leq p$, $h \in \mathcal{H}$ is given by $h(x) = ((ax + b) \bmod p) \bmod m$, for $a, b \in [p]$. This class is c-approximately 2-wise independent for $c = (\lceil p/m \rceil\, m/p)^2 \leq 1 + 3m/p$. For $a, b \in [p]$, $a \neq 0$, it is 1-universal [7,18,40].

(b) $U = \mathbb{F}$ is a finite field of prime characteristic p, $M = \mathbb{Z}_p^\mu$ for some μ, $\xi\colon \mathbb{F} \to M$ is some $\mathbb{Z}_p$-linear projection, $h \in \mathcal{H}$ is given by $h(x) = \xi(ax + b)$, for $a, b \in \mathbb{F}$. This class is 2-wise independent.

(c) $U = \mathbb{F}^\mu$ for a finite field $\mathbb{F}$, $M = \mathbb{F}^\nu$, $h \in \mathcal{H}$ given by $h(x) = A \cdot x + b$, for $A \in \mathbb{F}^{\nu \times \mu}$, $b \in M = \mathbb{F}^\nu$. This class is 2-wise independent.

(d) $U = \mathbb{F}^\mu$ for a finite field $\mathbb{F}$, $M = \mathbb{F}^\nu$, $h \in \mathcal{H}$ given by $h(x) = a \circ x + b$, for $a \in \mathbb{F}^{\mu+\nu-1}$, $b \in M = \mathbb{F}^\nu$, where $\circ$ denotes convolution. This class is 2-wise independent [25].

We note that for implementing such classes we either need prime numbers or representations of the arithmetic in finite fields that have size $|U|$ or at least size $|M|$, or, for (c) or (d), we have to carry out vector-matrix multiplication or a convolution over some finite field $\mathbb{F}$, the most natural case being $\mathbb{F}_2 = \{0, 1\}$. The main purpose of [10] was to provide a construction using only plain integer

[1] For a natural number n, we denote the set $\{0, 1, \ldots, n-1\}$ by $[n]$; for a prime number p, we denote by $\mathbb{Z}_p$ the field of size p, with ground set $[p]$.

arithmetic without the need for prime numbers to obtain two-wise independent hash families.

A simple first step in this direction had been made in [12]. There, for $k \geq \mu \geq 1$ and sets $U = [2^k]$ and $M = [2^\mu]$ the class $\mathcal{H}^{\text{mult}}_{2^k,2^\mu}$ of *multiplicative functions*

$$h_a \colon U \ni x \mapsto \left\lfloor (ax \bmod 2^k)/2^{k-\mu} \right\rfloor \in M$$

for $0 < a < 2^k$ odd was studied. This class is *not* 2-wise independent; hash values $h(x)$ for a key x are not even uniformly distributed. However, it is 2-universal. Its big advantage is that functions from the class can be evaluated very efficiently on a computer—one only needs one multiplication and one or two shift operations. The construction in [10] is a variant of this class.

1.2 Definitions and Properties of the Class

Notation. The universe is $U = [u]$, the range is $M = [m]$, for positive integers u and m. We will calculate in the ring $\mathbb{Z}_r$ of integers with operations modulo r, where $r \geq 2$ is a suitable integer. In order to keep the notation simple, we will identify the ground set of $\mathbb{Z}_r$ with $[r]$. Arithmetic operations modulo r are denoted by $\cdot_r$, $+_r$, $-_r$. If a positive integer x divides an integer y, we write $x \mid y$, otherwise $x \nmid y$.

Definition 1. *Let u, m, and $r \geq m$ be given, where $m \mid r$. Let $k = r/m$.*

(a) *For $a, b \in \mathbb{Z}_r$ define*

$$g_{a,b}(x) = a \cdot_r x +_r b, \text{ for } x \in U, \text{ and}$$
$$h_{a,b}(x) = \lfloor g_{a,b}(x)/k \rfloor, \text{ for } x \in U.$$

(b) *The class of* linear hash functions *from U to M modulo r is the multiset*

$$\mathcal{H}^{\text{lin}}_{u,m,r} = \{h_{a,b} \mid a, b \in \mathbb{Z}_r\}.$$

This class has size r^2; representing (choosing) a function from the class requires $2\lceil \log r \rceil$ (random) bits. Following [10], Woelfel [41,42] gave several related constructions of substantially smaller subclasses that exhibit behaviour similar to that of $\mathcal{H}^{\text{lin}}_{u,m,r}$.

The basic result of [10] and of this paper is that $\mathcal{H}^{\text{lin}}_{u,m,r}$ is approximately 2-wise independent if r is large enough. (This will be proved in Sect. 3.)

Theorem 1. (i) *If $m, u \geq 2$ and $r \geq (u-1)m$ is a multiple of m, then $\mathcal{H}^{\text{lin}}_{u,m,r}$ is $\frac{9}{8}$-approximately 2-wise independent. More precisely, for $i \in M$, $x \in U$ we have*

$$\mathbf{Pr}(h(x) = i) = \frac{1}{m},$$

and for arbitrary $i_1, i_2 \in M$ and distinct $x_1, x_2 \in U$ we have

$$\frac{2-c}{m^2} \leq \frac{1}{cm^2} \leq \mathbf{Pr}(h(x_1) = i_1 \wedge h(x_2) = i_2) \leq \frac{c}{m^2},$$

where $c = c(u, m, r) = (4z^2 + 4z + 1)/(4z^2 + 4z)$, for some $z \geq \lfloor r/((u-1)m) \rfloor$.

(ii) *If r and m are powers of a prime p, and $r \geq um/p$, then $\mathcal{H}^{\text{lin}}_{u,m,r}$ is 2-wise independent. (The most natural value for p is 2.)*

Section 2 contains technical preparations involving "gap matrices", which form the basis for the proof of the main theorem in Sect. 3. In Subsect. 4.1 the main result is extended to keys that are represented as sequences of numbers, a standard variation of hash classes, which leads to a more efficient evaluation for very long keys. In Subsect. 4.3 we show that the approach is sufficient for constructing sequences in $M = [m]$ that are sufficiently close to two-independence for carrying out two-independent sampling in the sense of [8]. As an example, we show that for $r \geq u^{3/2} \cdot m$ the sequence

$$\lfloor (ax + b) \bmod r)/(r/m) \rfloor, \ 0 \leq x < u,$$

where $a, b \in [r]$ are chosen uniformly at random, is suitable. Subsection 4.4 deals with the problem of "collapsing the universe": Given a subset $S \subseteq U$ of n keys from $U = [u]$, transform the long keys $x \in U$ into ones (x') of length $O(\log \log u + \log n)$ such that this transformation is one-to-one on S. We give a construction that uses just a linear hash function from $\mathcal{H}^{\text{lin}}_{u,m,r}$ to achieve the length $\log r = O(\log \log u + \log n)$.

Finally, Sect. 5 discusses various applications of and observations about the hash class $\mathcal{H}^{\text{lin}}_{u,m,r}$ that have appeared in the literature since it was first described in 1996.

2 Gap Matrices

This section provides bounds on the number of 1's in 0–1 matrices with a certain shape.

Definition 2. *Let $\gamma, k, \ell \geq 1$ be integers. A $k \times \ell$ matrix $A = (a_{ij})_{0 \leq i < k, 0 \leq j < \ell}$ with entries from $\{0, 1\}$ is called a* gap-γ (diagonal) matrix *if the 1's are arranged in diagonals of (horizontal/vertical) distance γ, i.e., if there is some $t \in [\gamma]$ such that*

$$a_{ij} = 1 \iff j - i \equiv t \pmod{\gamma}, \ \textit{for } 0 \leq i < k, 0 \leq j < \ell.$$

Figure 1 shows examples of gap-γ square matrices. By N_A we denote the number of 1's in a 0–1 matrix A. In a $k \times \ell$ matrix we expect N_A to be about $k\ell/\gamma$, if $\gamma \leq k, \ell$. Even in $k \times k$ matrices there will be deviations if $\gamma \nmid k$, as demonstrated in Fig. 1. We provide bounds on the relative deviation $N_A/(k\ell/\gamma)$, for $k \times \ell$ matrices A.

Proposition 1. *Assume A is a gap-γ matrix of dimensions $k \times \ell$ with $k, \ell \geq \gamma$. Then we have the following:*

(a) *If $\gamma \mid k$ or $\gamma \mid \ell$, then $N_A = k\ell/\gamma$.*

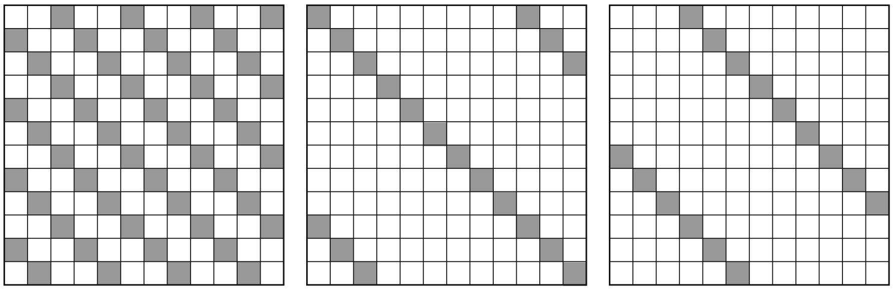


Fig. 1. Some $k \times k$ gap-γ matrices, with $k = 12$. 1's are represented as grey squares, 0's as white ones. *Left*: $\gamma = 3$, which divides k. There are exactly k^2/γ 1's. *Middle*: $\gamma = 9$, with $18 = \frac{9}{8}k^2/\gamma$ many 1's. *Right*: $\gamma = 9$, with $15 = \frac{15}{16}k^2/\gamma$ many 1's.

(b) *If* $z = \lfloor k/\gamma \rfloor = \lfloor \ell/\gamma \rfloor$, *then*

$$\frac{1}{c_z} \le \frac{N_A}{k^2/\gamma} \le c_z, \text{ for } c_z = 1 + \frac{1}{4z(z+1)}.$$

(c) *If* $y = \lfloor k/\gamma \rfloor \le z = \lfloor \ell/\gamma \rfloor$, *then*

$$2 - c_{y,z} \le \frac{N_A}{k\ell/\gamma} \le c_{y,z}, \text{ for } c_{y,z} = 1 + \frac{1}{4y(z+1)}.$$

Remark 1. (i) Note that $1/c_z = 4z(z+1)/(4z(z+1)+1) = 1 - 1/(2z+1)^2$ and $2 - c_{y,z} = 1 - 1/(4y(z+1))$. In (b), the upper bound is $\le \frac{9}{8}$, the lower bound $\ge \frac{8}{9}$. Asymptotically, the upper and lower bounds in (b) and (c) are $1 \pm O(\gamma^2/k^2)$ and $1 \pm O(\gamma^2/(k\ell))$, respectively.

(ii) A bound $|N_A/(k\ell/\gamma) - 1| \le \gamma^2/(4k\ell)$ for $k \times \ell$ gap-γ matrices was proved in [10, Lemma 8(c)]. It depends on the concrete values of k, ℓ, and γ if that bound or the one from Proposition 1(b) and (c) is stronger. For γ slightly smaller than $k = \ell$ (hence $y = z = 1$) the new bound is smaller by essentially a factor of 2.

(iii) The bounds in (b) are optimal. (See Remark 2 below.)

Proof (of Proposition 1). (a) If $\gamma = \ell$, each row contains exactly one 1, and hence $N_A = k$. (For an example see Fig. 2.) If $\gamma \mid \ell$, we may partition A into ℓ/γ many disjoint $k \times \gamma$ gap-γ matrices. So $N_A = (\ell/\gamma) \cdot k = k\ell/\gamma$. The cases $\gamma = k$ and $\gamma \mid k$ are similar.

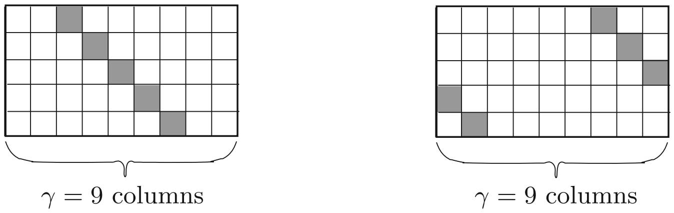


Fig. 2. A gap-γ matrix with $k = 5, \gamma = \ell = 9$ contains k many 1's.

(b) We can assume w.l.o.g. that $k \leq \ell$. (Otherwise consider the transpose of A.) Because of (a), we may assume that $\gamma \nmid k$ and $\gamma \nmid \ell$. Let $k' = k \bmod \gamma$ and $\ell' = \ell \bmod \gamma$. With $\delta = k'/\gamma$ and $\varepsilon = \ell'/\gamma$ we have $0 < \delta \leq \varepsilon < 1$ and $k = (z+\delta)\gamma$ and $\ell = (z+\varepsilon)\gamma$.

We partition A into four matrices, splitting after the $z\gamma$'th column and $z\gamma$'th row. We get a $(k-k') \times (\ell - \ell')$ submatrix A_1, a $(k-k') \times \ell'$ submatrix A_2, a $k' \times (\ell - \ell')$ submatrix A_3, and a $k' \times \ell'$ submatrix A_4. Since γ divides $k-k'$ and $\ell - \ell'$, part (a) gives that

$$N_{A_1} + N_{A_2} + N_{A_3} = \frac{(k-k')(\ell-\ell') + (k-k')\ell' + k'(\ell-\ell')}{\gamma} = \frac{k\ell - k'\ell'}{\gamma}. \quad (3)$$

So

$$\frac{N_A}{k\ell/\gamma} - 1 = \frac{N_{A_4}}{k\ell/\gamma} - \frac{k'\ell'}{k\ell} = \frac{N_{A_4} - k'\ell'/\gamma}{k\ell/\gamma}. \quad (4)$$

We need to bound this error term.

Upper Bound. Since no row in A_4 contains more than one 1, we have $N_{A_4} \leq k'$. Hence

$$\frac{N_{A_4} - k'\ell'/\gamma}{k\ell/\gamma} \leq \frac{k' - k'\ell'/\gamma}{k\ell/\gamma} = \frac{\delta\gamma - \delta\varepsilon\gamma}{(z+\delta)(z+\varepsilon)\gamma} = \frac{\delta(1-\varepsilon)}{(z+\delta)(z+\varepsilon)}.$$

Since $\delta \leq \varepsilon$, this implies that $(N_{A_4} - k'\ell'/\gamma)/(k\ell/\gamma) \leq \delta(1-\delta)/(z+\delta)^2$. Standard methods yield that the last fraction is maximal for $\delta = \delta_z = z/(1+2z)$; the corresponding maximal value is $\delta_z(1-\delta_z)/(z+\delta_z)^2 = 1/(4z(1+z))$. With (4) we get $N_A/(k\ell/\gamma) \leq 1 + 1/(4z(1+z))$, which is the claimed upper bound.

Lower Bound. According to (4), we must show that $(k'\ell'/\gamma - N_{A_4})/(k\ell/\gamma) \leq 1/(2z+1)^2$.
Case 1: $\delta + \varepsilon \leq 1$, or equivalently $k' + \ell' \leq \gamma$. — In this case we have

$$\frac{k'\ell'/\gamma - N_{A_4}}{k\ell/\gamma} \leq \frac{k'\ell'}{k\ell} = \frac{\delta\varepsilon}{(z+\delta)(z+\varepsilon)}. \quad (5)$$

We observe that the last expression cannot be maximal if $\delta < \varepsilon$. (Assume $\delta < \varepsilon$. Define a mapping

$$\begin{aligned} D\colon [0, \varepsilon - \delta] \ni \zeta \mapsto \ln\left(\frac{(\delta+\zeta)(\varepsilon-\zeta)}{(z+(\delta+\zeta))(z+(\varepsilon-\zeta))}\right) \\ = \ln(\delta+\zeta) + \ln(\varepsilon-\zeta) - \ln(z+\delta+\zeta) - \ln(z+\varepsilon-\zeta). \end{aligned}$$

Since

$$\frac{d}{d\zeta} D(\zeta)\Big|_{\zeta=0} = \frac{1}{\delta} - \frac{1}{\varepsilon} - \frac{1}{z+\delta} + \frac{1}{z+\varepsilon} = \frac{z}{\delta(z+\delta)} - \frac{z}{\varepsilon(z+\varepsilon)} > 0,$$

the value $\delta\varepsilon/((z+\delta)(z+\varepsilon))$ cannot be maximal.) Thus we only have to bound $\delta^2/(z+\delta)^2$ over all $\delta \leq \frac{1}{2}$. Clearly, this maximum is $\left(\frac{1}{2}\right)^2/\left(z+\frac{1}{2}\right)^2 = 1/(2z+1)^2$.

Case 2: $\delta + \varepsilon > 1$, or equivalently $k' + \ell' > \gamma$. — In this case A_4 contains at least $k' + \ell' - \gamma = (\delta + \varepsilon - 1)\gamma$ many 1's. Hence

$$\frac{k'\ell'/\gamma - N_{A_4}}{k\ell/\gamma} \le \frac{\delta\varepsilon\gamma - (\delta+\varepsilon-1)\gamma}{(z+\delta)(z+\varepsilon)\gamma} = \frac{(1-\delta)(1-\varepsilon)}{(z+\delta)(z+\varepsilon)}. \tag{6}$$

As above, we observe that the last expression cannot be maximal for $\delta < \varepsilon$. (Define $D\colon [0, \varepsilon - \delta] \ni \zeta \mapsto \ln(1-(\delta+\zeta)) + \ln(1-(\varepsilon-\zeta)) - \ln(z+(\delta+\zeta)) - \ln(z+(\varepsilon-\zeta))$. Then

$$\begin{aligned}\frac{d}{d\zeta}D(\zeta)\Big|_{\zeta=0} &= -\frac{1}{1-\delta} + \frac{1}{1-\varepsilon} - \frac{1}{z+\delta} + \frac{1}{z+\varepsilon}\\ &= \frac{z+1}{(1-\varepsilon)(z+\varepsilon)} - \frac{z+1}{(1-\delta)(z+\delta)}.\end{aligned}$$

This is positive, since the function $\tau \mapsto (1-\tau)(z+\tau)$ is decreasing for $\tau \in (0,1)$.) Thus we only have to bound $(1-\delta)^2/(z+\delta)^2$ over all $\delta > \frac{1}{2}$. This expression is bounded by $\left(\frac{1}{2}\right)^2/\left(z+\frac{1}{2}\right)^2 = 1/(2z+1)^2$.

(c) We now turn to matrices that are not necessarily almost square: Let A be a $k \times \ell$ gap-γ matrix with $y = \lfloor k/\gamma \rfloor \le z = \lfloor \ell/\gamma \rfloor$. Since the other cases have been covered already, we may assume that $y < z$ and that $\gamma \nmid k$ and $\gamma \nmid \ell$. Let $\delta = k/\gamma - y$ and $\varepsilon = \ell/\gamma - z$. Then $0 < \delta, \varepsilon < 1$. We extend A to an almost square gap-γ matrix $\bar{A}$ by adding $(z-y)\gamma$ rows at the bottom. Let $\bar{k} = (z+\delta)\gamma$ be the number of rows in $\bar{A}$. Applying part (b) yields

$$\frac{4z(z+1)}{(2z+1)^2} \le \frac{N_{\bar{A}}}{\bar{k}\ell/\gamma} \le 1 + \frac{1}{4z(z+1)}. \tag{7}$$

Further, by Proposition 1(a), we have $N_{\bar{A}} - N_A = (z-y)\ell$.

Upper Bound. We have

$$\begin{aligned}\frac{N_A}{k\ell/\gamma} &\le \left(1 + \frac{1}{4z(z+1)}\right)\frac{\bar{k}}{k} - \frac{(z-y)\ell}{k\ell/\gamma}\\ &= 1 + \frac{1}{4z(z+1)} + \left(1 + \frac{1}{4z(z+1)}\right)\frac{(z-y)\gamma}{(y+\delta)\gamma} - \frac{z-y}{y+\delta}\\ &= 1 + \frac{1}{4z(z+1)} + \frac{z-y}{4z(z+1)(y+\delta)}\\ &\le 1 + \frac{1}{4z(z+1)}\left(1 + \frac{z-y}{y}\right) = 1 + \frac{1}{4y(z+1)}.\end{aligned}$$

Lower Bound. By (7) we have

$$\begin{aligned}\frac{N_A}{k\ell/\gamma} &\ge \frac{4z(z+1)}{(2z+1)^2}\cdot\frac{\bar{k}}{k} - \frac{(z-y)\ell}{k\ell/\gamma}\\ &= 1 - \frac{1}{(2z+1)^2} + \left(1 - \frac{1}{(2z+1)^2}\right)\frac{z-y}{y+\delta} - \frac{z-y}{y+\delta}\\ &= 1 - \frac{1}{(2z+1)^2}\left(1 + \frac{z-y}{y+\delta}\right) \ge 1 - \frac{1}{4z(z+1)}\cdot\frac{z}{y} = 1 - \frac{1}{4y(z+1)}.\end{aligned}$$

□

Remark 2. There are arbitrarily large $k \times k$ matrices for which the bounds in Proposition 1(b) are optimal. For the *upper bound* we first consider $z = 1$. In the middle $k \times k$ matrix of Fig. 1 we have $\gamma = 9$ and $k = 12$, and $N_A = 18 = \frac{9}{8}k^2/\gamma$ many 1's. In the same way one gets $12h \times 12h$ gap-$9h$ matrices for arbitrary $h \geq 1$ that realize excess ratio $c_1 = \frac{9}{8}$. For arbitrary z, we define a $k \times k$ matrix for $k = 2z(z+1)$, as follows. We choose $\gamma = 2z + 1$ and obtain $k' = k - \gamma z = z$. If we place 1's on the main diagonal, the $k' \times k'$ submatrix in the lower right corner has k' many 1's; hence the total number of 1's is $(k^2 - k'^2)/\gamma + k'$. One easily checks that $((k^2 - k'^2)/\gamma + k')/(k^2/\gamma)$ evaluates to $1 + 1/(4z(1+z)) = c_z$. For the *lower bound* we first consider $k = 9$ and $\gamma = 6$, hence $z = 1$. We obtain a matrix A with the minimum number of 1's by placing them in the diagonals running from $(0, 3)$ to $(5, 8)$ and from $(3, 0)$ to $(8, 5)$, which gives $N_A = 12$, hence $N_A/(k^2/\gamma) = 12/(9^2/6) = \frac{8}{9} = 1/c_1$. Again, it is easy to find larger matrices with ratio $1/c_1$ and examples for arbitrary z with ratio $1/c_z$.

3 Proof of Universality Properties

In this section we show that $\mathcal{H}^{\text{lin}}_{u,m,r}$ is approximately two-wise independent in general and two-wise independent if r is a prime power.

We will assume throughout this section that $U = [u] = \{0, \ldots, u-1\}$ for some $u \geq 2$ and $M = [m] = \{0, \ldots, m-1\}$ for some $m \geq 2$, and that $r = km$ is a multiple of m.

As preparation for the proof, we provide some observations and lemmas.

Fact 1. *Let $z \in \mathbb{Z}_r$ and let $\gamma = \gcd(z, r)$. Then for arbitrary $t \in \mathbb{Z}_r$ the following holds:*

$$|\{x \in \mathbb{Z}_r \mid x \cdot_r z = t\}| = \begin{cases} \gamma & \text{if } \gamma \mid t; \\ 0 & \text{otherwise.} \end{cases}$$

Proof. Using the fact ("Bézout's Lemma") that we may write $\gamma = \alpha r + \beta z$ for certain $\alpha, \beta \in \mathbb{Z}$, one sees that the range $m_z(\mathbb{Z}_r)$ of the mapping $m_z \colon \mathbb{Z}_r \ni x \mapsto x \cdot_r z \in \mathbb{Z}_r$ is (γ), the set of all multiples of γ in $\mathbb{Z}_r$. The fundamental homomorphism theorem from group theory applied to the group homomorphism m_z (with respect to the additive group structure $(\mathbb{Z}_r, +_r, 0)$) yields that $\left|m_z^{-1}(t)\right|$ is the same for all $t \in m_z(\mathbb{Z}_r) = (\gamma)$. Since $|(\gamma)| = r/\gamma$, the claim follows. $\square$

Recall that $h(x) = \lfloor g(x)/k \rfloor$, where $g(x) = g_{a,b}(x) = a \cdot_r x +_r b$, for $a, b \in \mathbb{Z}_r$ chosen uniformly at random. We make some simple observations on the distributions of $g(x)$ and $h(x)$.

Lemma 1. *For each $x \in U$ the following holds:*

(a) *$g(x)$ is independent of a and uniformly distributed in $\mathbb{Z}_r$;*
(b) *$h(x)$ is independent of a and uniformly distributed in M.*

Proof. (a) For $s \in \mathbb{Z}_r$ and $\alpha \in \mathbb{Z}_r$ fixed we calculate:

$$\mathbf{Pr}(g(x) = s \mid a = \alpha) = \mathbf{Pr}(b = s - \alpha x \mid a = \alpha) = \frac{1}{r}.$$

Thus $g(x)$ and a are independent. Since the events $\{a = \alpha\}$, $\alpha \in \mathbb{Z}_r$, partition the probability space, we also get $\mathbf{Pr}(g(x) = s) = 1/r$.
(b) This follows immediately from (a), since k divides r, and hence the operation $s \mapsto \lfloor s/k \rfloor$ maps exactly k elements of $\mathbb{Z}_r$ to each element i of M. □

From here on, assume that $x_1, x_2 \in U$ with $x_2 < x_1$ are fixed. Let

$$z = x_1 - x_2 \ (= x_1 -_r x_2) \quad \text{and} \quad \gamma = \gcd(z, r).$$

Then $0 < z < u$ and $1 \le \gamma < u$. Now we can describe the joint distribution of $g(x_1)$ and $g(x_2)$ (see Fig. 3 below).

Lemma 2. (a) *For arbitrary $t \in \mathbb{Z}_r$ we have*

$$\mathbf{Pr}(g(x_1) -_r g(x_2) = t) = \begin{cases} \gamma/r & \textit{if } \gamma \mid t; \\ 0 & \textit{otherwise.} \end{cases}$$

(b) *For arbitrary $s_1, s_2 \in \mathbb{Z}_r$ we have*

$$\mathbf{Pr}(g(x_1) = s_1 \wedge g(x_2) = s_2) = \begin{cases} \gamma/r^2 & \textit{if } \gamma \mid (s_1 -_r s_2); \\ 0 & \textit{otherwise.} \end{cases}$$

Proof. (a) Clearly, for $g = g_{a,b}$, we have

$$g(x_1) -_r g(x_2) = a \cdot_r (x_1 -_r x_2) = a \cdot_r z.$$

Thus, the claim follows immediately from Fact 1.
(b) Since $g(x_1)$ and a are independent by Lemma 1(a) and $g(x_1) -_r g(x_2) = a \cdot_r z$ is a function of a, the random variables $g(x_1)$ and $g(x_1) -_r g(x_2)$ are independent. Thus, using part (a) we may calculate:

$$\begin{aligned} &\mathbf{Pr}(g(x_1) = s_1 \wedge g(x_2) = s_2) \\ &= \mathbf{Pr}(g(x_1) = s_1 \wedge g(x_1) -_r g(x_2) = s_1 -_r s_2) \\ &= \mathbf{Pr}(g(x_1) = s_1) \cdot \mathbf{Pr}(g(x_1) -_r g(x_2) = s_1 -_r s_2) \\ &\overset{(*)}{=} \begin{cases} (1/r) \cdot (\gamma/r) = \gamma/r^2 & \text{if } \gamma \text{ divides } s_1 -_r s_2; \\ (1/r) \cdot 0 = 0 & \text{otherwise.} \end{cases} \end{aligned}$$

(For $(*)$ we use both (a) and Lemma 1(a).) Thus (b) is proved. □

Finally we are ready to formulate and prove the central theorem about our hash functions. Let

$$\Gamma_{u,m,r} = \max(\{0\} \cup \{\gamma \in \{1, \ldots, u-1\} \mid \gamma \text{ divides } r \wedge \gamma \nmid k\}). \tag{8}$$

(The definition from [10] is changed in the clever way suggested in [41] to take the special case into account where all $\gamma \le u - 1$ that divide r also divide k.) Note that $\Gamma_{u,m,r} \le u - 1$.

Observation 1. *(a) If r is a power of a prime p and $r \geq um/p$, then $\Gamma_{u,m,r} = 0$. (b) If $r \geq (u-1)m$, then $\Gamma_{u,m,r} \leq k$.*

Proof. (a) Since $r = mk$, the numbers m and k are also powers of p. Now if $\gamma < u$ divides r, it is a power of p itself, strictly smaller than $u \leq rp/m$. So γ divides $r/m = k$. This shows that $\Gamma_{u,m,r} = 0$. (b) This is trivial. □

Even if no assumption about r is made, we will always be able to make sure that $\Gamma_{u,m,r} \leq k$, so that the following is well-defined. With $\Gamma = \Gamma_{u,m,r}$, let

$$c_{u,m,r} = \begin{cases} 1 & \text{if } \Gamma = 0, \\ 1 + \dfrac{1}{4\lfloor k/\Gamma \rfloor (\lfloor k/\Gamma \rfloor + 1)} & \text{if } \Gamma > 0. \end{cases}$$

Note that $1 \leq c_{u,m,r} \leq \frac{9}{8}$ and that $c_{u,m,r} = 1 + O(u^2/k^2)$.

Theorem 2 (Main Theorem). *Let h be chosen at random from $\mathcal{H}^{\text{lin}}_{u,m,r}$, where $m \mid r$. Then the following holds, for arbitrary $i_1, i_2 \in M$ and distinct $x_1, x_2 \in U$:*
(a) *If r is a power of a prime p and $r \geq um/p$, then*

$$\mathbf{Pr}(h(x_1) = i_1 \wedge h(x_2) = i_2) = \frac{1}{m^2},$$

i.e., in this case $\mathcal{H}^{\text{lin}}_{u,m,r}$ is 2-wise independent.
(b) *If $r \geq (u-1)m$, then $\mathbf{Pr}(h(x_1) = i_1) = 1/m$ and*

$$\frac{2 - c_{u,m,r}}{m^2} \leq \frac{1}{c_{u,m,r}} \cdot \frac{1}{m^2} \;\leq\; \mathbf{Pr}(h(x_1) = i_1 \wedge h(x_2) = i_2) \;\leq\; \frac{c_{u,m,r}}{m^2},$$

i.e., in this case $\mathcal{H}^{\text{lin}}_{u,m,r}$ is $c_{u,m,r}$-approximately 2-wise independent.

Remark 3. A weaker version of the upper bound in (b), a similar lower bound, and (a) were shown in [10]. Woelfel [41] was the first to prove the upper bound in (b), with a different method.

Proof. Our aim is to find upper and lower bounds on

$$Q = Q(x_1, x_2, i_1, i_2) = \frac{\mathbf{Pr}(h(x_1) = i_1 \wedge h(x_2) = i_2)}{1/m^2}.$$

From the definition of $h_{a,b}$ it is immediate that with $I_i = \{ki, ki+1, \ldots, k(i+1)-1\}$, for $i \in M$, we have

$$\mathbf{Pr}(h(x_1) = i_1 \wedge h(x_2) = i_2) = \mathbf{Pr}(g(x_1) \in I_{i_1} \wedge g(x_2) \in I_{i_2}).$$

As usual, let $\gamma = \gcd(x_1 - x_2, r)$. Adding the equation given in Lemma 2(b) for all $s_1 \in I_{i_1}, s_2 \in I_{i_2}$, we obtain

$$\mathbf{Pr}(h(x_1) = i_1 \wedge h(x_2) = i_2) = |S_{i_1,i_2}| \cdot \gamma/r^2,$$

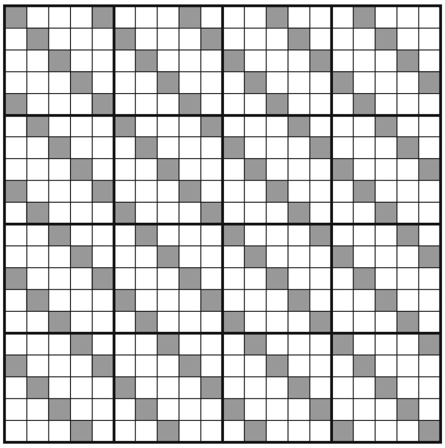


Fig. 3. The distribution of $(g(x_1), g(x_2))$ with $r = 20$, $\gamma = \gcd(x_1 - x_2, r) = 4$, $m = 4$, hence $k = 5$. We see $m^2 = 16$ many gap-4 matrices of dimension 5×5. Each grey position is attained with probability $\frac{4}{400} = \frac{1}{100}$. Some submatrices have probability $\frac{6}{100} < \frac{1}{16}$, some have probability $\frac{7}{100} > \frac{1}{16}$.

where $S_{i_1,i_2} = \{(s_1, s_2) \in I_{i_1} \times I_{i_2} \mid \gamma \text{ divides } s_1 -_r s_2\}$, hence

$$Q = \frac{|S_{i_1,i_2}| \cdot \gamma \cdot m^2}{r^2} = \frac{|S_{i_1,i_2}|}{k^2/\gamma}. \tag{9}$$

We can regard (the characteristic function of) S_{i_1,i_2} as being represented by a $k \times k$ matrix with row indices s_1 from I_{i_1} and column indices s_2 from I_{i_2}, with the entry for (s_1, s_2) being 1 if γ divides $s_1 -_r s_2$ and 0 otherwise. (Figure 3 shows all these matrices in one picture, for an example.) Clearly this is a gap-γ matrix as considered in Sect. 2, with shifted row and column numberings. So we can apply the estimates from there.

Case 1: $\gamma \mid k$. — In this case the quotient in (9) equals 1, by Proposition 1(a). Since the hypothesis of (a) implies that γ divides k (as seen in Observation 1), we have proved (a).

Case 2: $\gamma \nmid k$. — With Proposition 1(b) we may estimate:

$$\left(1 + \frac{1}{4\lfloor k/\gamma \rfloor (\lfloor k/\gamma \rfloor + 1)}\right)^{-1} \le Q \le 1 + \frac{1}{4\lfloor k/\gamma \rfloor (\lfloor k/\gamma \rfloor + 1)}. \tag{10}$$

We have $\gamma \le \Gamma$, hence $\lfloor k/\Gamma \rfloor (\lfloor k/\Gamma \rfloor + 1) \le \lfloor k/\gamma \rfloor (\lfloor k/\gamma \rfloor + 1)$, so (10) implies part (b). $\square$

Remark 4. One can utilize the construction in Remark 2 of gap-γ matrices that realize the upper and lower bounds for the number of 1's to find u, m, and r for which the bounds in Theorem 2 are optimal. The details are left to the reader.

3.1 Generalization: r Not Divisible by m

We have proven the central universality properties of $\mathcal{H}^{\mathrm{lin}}_{u,m,r}$. However, we always had to assume that m, the size of the range, is a divisor of r, which may not always be convenient. For example, we might wish to use as r some power of 2, because computer arithmetic in $\mathbb{Z}_r$ is particularly efficient in this case, but on the other hand wish to use some m that is not a power of 2. Another such case arises when we want to choose r randomly from some range and do not have control over the divisors of r. (See Subsect. 4.4 for such a situation.) For this reason we note here that the methods developed so far make it possible to construct approximately 2-wise independent classes also with a ring $\mathbb{Z}_r = \{0, \ldots, r-1\}$ where r is not a multiple of m.

Partition $\mathbb{Z}_r$ into m intervals $I_0, \ldots, I_{m-1}$, where $I_0, \ldots, I_{(r \bmod m)-1}$ have length $\overline{k} = \lceil r/m \rceil$ and $I_{r \bmod m}, \ldots, I_{m-1}$ have length $\underline{k} = \lfloor r/m \rfloor$. Let $h_{a,b}(x) = i$ if $g_{a,b}(x) \in I_i$. Arithmetically, this is expressed as follows: Precompute $\overline{k}$ and $\underline{k}$, as well as $B = r \bmod m$ and $T = B \cdot \overline{k}$. Then let

$$h_{a,b}(x) = \begin{cases} \lfloor g_{a,b}(x)/\overline{k} \rfloor & \text{if } g_{a,b}(x) < T; \\ B + \lfloor (g_{a,b}(x) - T)/\underline{k} \rfloor & \text{otherwise.} \end{cases}$$

The resulting class serves all hash values with almost the same probability in the case $r \gg m$, since

$$\mathbf{Pr}(h(x) = i) = \begin{cases} \lceil r/m \rceil / r \le (1 + 1/\lfloor r/m \rfloor)/m, & \text{for } 0 \le i < B; \\ \lfloor r/m \rfloor / r \ge (1 - 1/\lceil r/m \rceil)/m, & \text{for } B \le i < m. \end{cases}$$

The class of all $h_{a,b}$, $a, b \in [r]$, is again called $\mathcal{H}^{\mathrm{lin}}_{u,m,r}$. This class satisfies approximate independence conditions, which can be proved in analogy to Theorem 2(b), using Proposition 1(b) for $\ell \in \{k-1, k, k+1\}$. (Assume $\gamma = \gcd(x_1 - x_2, r)$ as in the proof of Theorem 2. Following that proof, we will have $|I_{i_1}|, |I_{i_2}| \in \{\underline{k}, \overline{k}\}$. Since the error-free formula from Proposition 1(a) applies if at least one of the numbers $|I_{i_1}|$ and $|I_{i_2}|$ is divisible by γ, the quotients $y = \lfloor |I_{i_1}| / \gamma \rfloor$ and $z = \lfloor |I_{i_2}| / \gamma \rfloor$ can be assumed to be the same.) The reader is invited to work out the details of the proof of the following statement.

Proposition 2. *Assume $r \ge (u-1)m$, and consider the generalized version of class $\mathcal{H}^{\mathrm{lin}}_{u,m,r}$ as just described. Let $x_1, x_2 \in U = [u]$ be distinct and $i_1, i_2 \in M = [m]$ be arbitrary. Then we have, with $c_{u,m,r}$ as defined for Theorem 2:*

$$2 - c_{u,m,r} \;\le\; \frac{\mathbf{Pr}(h(x_1) = i_1 \wedge h(x_2) = i_2)}{\mathbf{Pr}(h(x_1) = i_1)\,\mathbf{Pr}(h(x_2) = i_2)} \;\le\; c_{u,m,r}\,.$$

□

4 Variations of the Construction

4.1 Vectors as Keys and Range Elements

If keys are very long, it may be inconvenient in practice to treat them as integers in a universe $U = [u]$, since one has to carry out long integer multiplication. How can one deal with longer keys, e.g., given as vectors in a universe $U' = U^\ell$, for $U = [u]$, when the range is $M = [m]$? It is well known that 2-independence can be ensured by just using 2-independent hashing on the ℓ components separately and taking the sum of the results modulo m. In [10] it was sketched how to proceed just utilizing the ring $\mathbb{Z}_r$ with $r \geq (u-1)m$ and $k = r/m$, as before. An advantage is that fewer random bits are needed and that, as observed by Woelfel [41,42], a parsimonious extension to a range that is also a set of vectors is possible. A hash function is specified by a coefficient vector $\boldsymbol{a} = (a_0, \ldots, a_{\ell-1}) \in \mathbb{Z}_r^\ell$ and $b \in \mathbb{Z}_r$. The hash function $h_{\boldsymbol{a},b}$ is defined by

$$h_{\boldsymbol{a},b}((\xi_0, \ldots, \xi_{\ell-1})) = \left\lfloor \Big(\sum_{0 \leq \lambda < \ell}^{(r)} a_\lambda \cdot_r \xi_\lambda +_r b \Big) \Big/ k \right\rfloor,$$

for $\boldsymbol{x} = (\xi_0, \ldots, \xi_{\ell-1}) \in [u]^\ell$. (The superscript "$(r)$" indicates summation in $\mathbb{Z}_r$.) The resulting class exhibits the same universality properties as $\mathcal{H}^{\mathrm{lin}}_{u,m,r}$, as stated in Theorem 2. The proof is very similar to that of Theorem 2. The idea is simple. Two different keys $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \in U^\ell$ must have one component in which they differ, $\xi_0^{(1)} \neq \xi_0^{(2)}$, say. One fixes $a_1, \ldots, a_{\ell-1}$ and studies the joint distribution of the pair

$$\mathbb{Z}_r^2 \ni (a_0, b) \mapsto \Big\lfloor \Big(a_0 \cdot_r \xi_0^{(i)} +_r b +_r \underbrace{\sum_{1 \leq \lambda < \ell}^{(r)} a_\lambda \cdot_r \xi_\lambda^{(i)}}_{=C_i} \Big) / k \Big\rfloor, \quad i = 1, 2,$$

of random variables. The values C_1 and C_2 are regarded as constant. The analysis is practically identical to that in the proof of Theorem 2.

Example: Assume we work on a computer with fast 64-bit arithmetic, the keys are bit strings of arbitrary, fixed length given as a sequence of words, and the range is $M = [2^\mu]$ for some $\mu \leq 32$. Then we may let $u = 2^{32}$ and $r = 2^{64}$. A key x is split into 32-bit pieces $\xi_0, \xi_1, \ldots, \xi_{\ell-1}$ for a suitable ℓ. A hash function is represented as a sequence $(a_0, \ldots, a_{\ell-1}, b)$ of 64-bit integers $a_0, \ldots, a_{\ell-1}, b$. The modulo r operation is for free, since the standard hardware carries out multiplication and addition modulo 2^{64}. The final division by $2^{64-\mu}$ is done by a shift.

This construction is helpful if the range M is not too big— the ring size r must be bigger than $(u-1)m$ or um/p in the case that r is a power of a prime number p. Woelfel [41,42] gave a more general, very elegant construction to remove this restriction. It resembles the convolution construction over finite fields studied in [25], but is made to work over the ring $\mathbb{Z}_r$. Assume the range is a set $M' = M^\varrho = [m]^\varrho$ of vectors. Of course, we could use ϱ many independent hash functions with range M to build one with range M^ϱ. However, then the number $(\ell+1)\varrho$ of required random coefficients from $\mathbb{Z}_r$ grows with the product of ℓ and ϱ.

To save random bits, and storage space, Woelfel proposed using convolution (or "polynomial multiplication") over $\mathbb{Z}_r$. A hash function is given by vectors

$\boldsymbol{a} = (a_0, \ldots, a_{\ell+\varrho-2})$ and $\boldsymbol{b} = (b_0, \ldots, b_{\varrho-1})$ with coefficients from $\mathbb{Z}_r$ For a key $\boldsymbol{x} = (\xi_0, \ldots, \xi_{\ell-1}) \in U^\ell$ and $0 \le \kappa < \varrho$ component κ of the hash value $h(\boldsymbol{x})$ is given by

$$(h_{\boldsymbol{a},\boldsymbol{b}}((\xi_0, \ldots, \xi_{\ell-1})))_\kappa = \left\lfloor \Big(\sum_{0\le\lambda<\ell}^{(r)} a_{\kappa+\lambda} \cdot_r \xi_\lambda +_r b_\kappa \Big) \Big/ k \right\rfloor .$$

The resulting class of hash functions from $U' = U^\ell$ to $M' = M^\varrho$ is called $\mathcal{H}^{\mathrm{conv},\ell,\varrho}_{u,m,r}$. In the "clean" case where r is a power of a prime number p and $r \ge um/p$ one can show that $\mathcal{H}^{\mathrm{conv},\ell,\varrho}_{u,m,r}$ is 2-wise independent. If we only have that m divides r and $r \ge (u-1)m$, then $\mathcal{H}^{\mathrm{conv},\ell,\varrho}_{u,m,r}$ is $(c_{u,m,r})^\varrho$-approximately 2-wise independent, for $c_{u,m,r}$ the approximation constant for class $\mathcal{H}^{\mathrm{lin}}_{u,m,r}$ as in Theorem 2. (For details, see [41,42]. There it is also discussed how the parameter spaces for the coefficients may be reduced.)

4.2 Higher Independence

In [10] it was proved that polynomials over $\mathbb{Z}_r$ can be used to obtain (approximately) d-wise independent classes for arbitrary fixed $d \ge 2$. Here, we give the relevant definitions and state the theorem, and refer the reader to [10] for the proof. Unfortunately, the construction seems to be mainly of theoretical interest. While it achieves higher independence without prime numbers or finite fields being involved (and without tabulation [37], which requires longer tables of random entries), the big disadvantage of the construction is that r has to be quite large, which makes for slow evaluation. —As before, we assume that a universe $U = [u]$ and a range $M = [m]$ are given.

Definition 3. *Let $d \ge 2$. A class $\mathcal{H}$ of hash functions from U to M is called* d-wise independent *if for arbitrary distinct keys $x_0, \ldots, x_{d-1} \in U$ and arbitrary $i_0, \ldots, i_{d-1} \in M$ we have*

$$\mathbf{Pr}(h(x_s) = i_s, \text{ for } 0 \le s < d) = \frac{1}{m^d},$$

for h chosen uniformly at random from $\mathcal{H}$. For $c \ge 1$, such a class is called c-approximately d-wise independent if for each key x the hash value $h(x)$ is uniformly distributed in M and if for distinct $x_0, \ldots, x_{d-1} \in U$ and arbitrary $i_0, \ldots, i_{d-1} \in M$ we have

$$\frac{2-c}{m^d} \le \mathbf{Pr}(h(x_s) = i_s, \text{ for } 0 \le s < d) \le \frac{c}{m^d}.$$

As before, we fix $r = km$ for some positive integer k, and consider the ring $\mathbb{Z}_r = [r]$ with arithmetic modulo r. Further, fix $d \ge 2$.

Definition 4. *For $\boldsymbol{a} = (a_0, \ldots, a_{d-1}) \in \mathbb{Z}_r^d$ define (arithmetic in $\mathbb{Z}_r$)*

$$g_{\boldsymbol{a}}(x) = \sum_{0\le\mu<d} a_\mu x^\mu, \text{ for } x \in U,$$

and

$$h_{\boldsymbol{a}}(x) = \lfloor g_{\boldsymbol{a}}(x)/(r/m) \rfloor, \text{ for } x \in U.$$

Further, let

$$\mathcal{H}^{\text{deg-}d}_{u,m,r} = \{h_{\boldsymbol{a}} \mid \boldsymbol{a} \in \mathbb{Z}_r^d\}.$$

Theorem 3. *If $r \geq m \cdot (u-1)^{\binom{d}{2}}$, then $\mathcal{H}^{\text{deg-}d}_{u,m,r}$ is c-approximately d-wise independent for $c = c(u,m,r,d) = (1 - m(u-1)^{\binom{d}{2}}/r)^{-d}$. More precisely, $h(x)$ is uniformly distributed in $[m]$, for each $x \in U$, and for arbitrary distinct elements $x_0, \ldots, x_{d-1} \in U$ and arbitrary values $i_0, \ldots, i_{d-1} \in M$ we have:*

(a) $\left(1 - \dfrac{m(u-1)^{\binom{d}{2}}}{r}\right)^d \leq \dfrac{\mathbf{Pr}(h(x_\lambda) = i_\lambda, 0 \leq \lambda < l)}{1/m^d} \leq \left(1 + \dfrac{m(u-1)^{\binom{d}{2}}}{r}\right)^d$;

(b) *if u, m, and r are powers of the same prime number $p \geq 2$, and $r \geq m(u/p)^{\binom{d}{2}}$, we even have d-wise independence, i.e.,*

$$\mathbf{Pr}(h(x_\lambda) = i_\lambda, \text{ for } 0 \leq \lambda < l) = \frac{1}{m^d}.$$

□

4.3 Two-Wise Independent Sampling

Here we describe how the function classes described before can be used for two-independent sampling in the sense of Chor and Goldreich [8]. There one has a finite set $M = [m]$ and a set $A \subseteq M$ of which one wants to find an arbitrary element. The only operation available regarding A is a membership test "is $x \in A$?" for $x \in M$. The most obvious search method is random sampling (keep choosing random elements of M until an element of A is found); however, this method has the disadvantage that in expectation $(1/\varrho)\log|M|$ random bits are needed, where $\varrho = |A|/|M|$ is the density of A in M (which sometimes is small). In order to save random bits, one employs a 2-wise independent sequence $X_0, \ldots, X_{u-1}$ of random variables, uniformly distributed in M. These elements are tested one after the other, until an element of A is found ("success"). In [8], Chor and Goldreich use finite fields based on prime numbers for constructing such sequences. They show that the probability that the first j trials all fail is bounded by $1/(j\varrho)$. If $\mathcal{H}$ is an arbitrary 2-wise independent class of hash functions from $U = [u]$ to M, one can use $X_i = h(i)$ for $0 \leq i < u$. The classes studied in the present paper can be used immediately if m and u are powers of the same prime number p and $r \geq um$. This requires $2\log r$ random bits for the coefficients a and b.

We wish to show with a slightly refined analysis that the classes $\mathcal{H}^{\text{lin}}_{u,m,r}$ can be used for arbitrary $M = [m]$ and $U = [u]$ if one chooses r sufficiently large. Let h be chosen uniformly at random from $\mathcal{H}^{\text{lin}}_{u,m,r}$. We let

$$X_i = h(i), \text{ for } 0 \leq i < u.$$

For given $A \subseteq M$ we let

$$Y_i = \begin{cases} 1 & \text{if } X_i \in A, \\ 0 & \text{otherwise,} \end{cases}$$

for $0 \le i < u$. Since X_i is uniformly distributed, we have $\mathbf{E}(Y_i) = 1/\varrho$.

For $0 \le j \le u$, the number of successes one encounters in $X_0, \ldots, X_{j-1}$ is

$$Z_j = \sum_{0 \le i < j} Y_i .$$

Clearly, $\mathbf{E}(Z_j) = j\varrho$. The crux of Chor and Goldreich's method is to note that from the two-wise independence of the Y_i's it follows that $\mathbf{Var}(Z_j) \le \mathbf{E}(Z_j)$. The Chebychev-Cantelli inequality $\mathbf{Pr}(X \le \mathbf{E}(X) - t) \le \mathbf{Var}(X) / (\mathbf{Var}(X) + t^2)$ then implies that

$$\begin{aligned} \mathbf{Pr}(X_i \notin A \text{ for all } i,\ 0 \le i < j) &= \mathbf{Pr}(Z_j = 0) \le \mathbf{Pr}(Z_j \le \mathbf{E}(Z_j) - \mathbf{E}(Z_j)) \\ \le \frac{\mathbf{Var}(Z_j)}{\mathbf{Var}(Z_j) + \mathbf{E}(Z_j)^2} &= \frac{1}{1 + \mathbf{E}(Z_j)^2 / \mathbf{Var}(Z_j)} \le \frac{1}{1 + \mathbf{E}(Z_j)} = \frac{1}{1 + j\varrho} . \end{aligned}$$

This is the desired bound on the failure probability of 2-wise independent sampling.

We show that for all sufficiently large r the required bound $\mathbf{Var}(Z_j) \le \mathbf{E}(Z_j)$ is true even though the random variables $X_0, \ldots, X_{u-1}$ are only approximately 2-wise independent.

Lemma 3. *If $r \ge u^{3/2} m$, then $\mathbf{Var}(Z_j) \le \mathbf{E}(Z_j)$, for $0 \le j < u$.*

Proof. We calculate:

$$\mathbf{Var}(Z_j) = \sum_{0 \le i < j} \mathbf{Var}(Y_i) + \sum_{\substack{0 \le i, i' < j \\ i \ne i'}} \mathbf{Cov}(Y_i, Y_{i'}) . \tag{11}$$

Clearly, $\mathbf{Var}(Y_i) = \varrho(1 - \varrho)$ for all i, $0 \le i < j$. We analyze a summand of the second sum in (11). Fix $i \ne i'$. For $s \in M$, $t \in M$, let

$$\chi_s(t) = \begin{cases} 1 & \text{if } t = s, \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$\begin{aligned} \mathbf{Cov}(Y_i, Y_{i'}) &= \mathbf{E}((Y_i - \varrho)(Y_{i'} - \varrho)) \\ &= \mathbf{E}\left(\left(\sum_{s \in A} (\chi_s(X_i) - 1/m) \right) \left(\sum_{t \in A} (\chi_t(X_{i'}) - 1/m) \right) \right) \\ &= \sum_{s, t \in A} \mathbf{Cov}(\chi_s(X_i), \chi_t(X_{i'})) . \end{aligned} \tag{12}$$

For $s, t \in A$ arbitrary we have

$$\begin{aligned}\mathbf{Cov}(\chi_s(X_i), \chi_t(X_{i'})) &= \mathbf{E}(\chi_s(X_i) \cdot \chi_t(X_{i'})) - \mathbf{E}(\chi_s(X_i)) \cdot \mathbf{E}(\chi_t(X_{i'})) \\ &= \mathbf{Pr}(X_i = s \wedge X_{i'} = t) - 1/m^2 \leq (c_{u,m,r} - 1)/m^2,\end{aligned}$$

by Theorem 2(b). Since $r \geq (u-1)m$, we have $c_{u,m,r} - 1 \leq 1/(4 \lfloor r/((u-1)m) \rfloor \cdot (\lfloor r/((u-1)m) \rfloor + 1)) \leq (um/r)^2$, and we get by summing up according to (11) and (12):

$$\begin{aligned}\mathbf{Var}(Z_j) &\leq j\varrho(1-\varrho) + j^2 |A|^2 (c_{u,m,r} - 1)/m^2 \\ &= j\varrho(1-\varrho) + j^2\varrho^2(c_{u,m,r} - 1) \\ &\leq j\varrho(1-\varrho) + j^2\varrho^2 \cdot (um/r)^2 \\ &\leq j\varrho(1-\varrho) + j\varrho^2(u^{3/2}m/r)^2.\end{aligned}$$

By the assumption $r \geq u^{3/2}m$ we get $\mathbf{Var}(Z_j) \leq j\varrho(1-\varrho) + j\varrho^2 = j\varrho = \mathbf{E}(Z_j)$, as desired. $\square$

4.4 "Collapsing the Universe" Without Finite Fields

Assume $U = [u]$ is very large, meaning that if we represent keys as bit strings, they are very long. Very often, in applications like dictionaries, one wishes to replace a key x by a shorter key x' in a range $U' = [u']$, where a "collapse function" $h\colon U \to U'$ maps x to x'. For this to work, h must be one-to-one on the set $S \subseteq U$ of "relevant keys". With the results from Sect. 4 it is easy to find a good collapsing function with a description size of $O(\log u)$ bits. Collapse functions with smaller description sizes can be constructed deterministically [19, 33], or, usually much simpler, in a randomized way [11,35]. In the latter case the description size of h is $O(\log n + \log\log u)$ bits, and it requires choosing a random prime of this bitlength or knowing a finite field with elements of this bitlength. Here we demonstrate that class $\mathcal{H}^{\text{lin}}_{u,m,r}$ can be used to build collapse functions with such a small description size on the basis of modular arithmetic alone, without the need for prime numbers or finite field arithmetic.

Technically, assume $S = \{x_0, \ldots, x_{n-1}\} \subseteq U = [u]$ is given. We let $m = \lceil n^3 \log u \rceil$. The size r of the ring $\mathbb{Z}_r$ is chosen uniformly at random from $[m^2, 2m^2)$. Then $\lfloor r/m \rfloor \in [m, 2m)$. (Since usually r is not a multiple of m, we need to use the modified functions from Subsect. 3.1.) For the hash class $\mathcal{H}^{\text{lin}}_{u,m,r}$ to function in a 2-wise independent way on S, we only require, to make the proof of Theorem 2 work, that r is S-*good* in the following sense:

$$\Gamma_S = \max\{\gcd(x_j - x_i, r) \mid x_i, x_j \in S \text{ distinct}\} \leq r/m.$$

If this is the case, we will have $\mathbf{Pr}(h(x_i) = h(x_j)) = O(1/m) = O(1/n^3)$ for all distinct $x_i, x_j \in S$, and hence $\mathbf{Pr}(h \text{ is not one-to-one on } S) = O(1/n)$. Clearly, for r being S-good it is sufficient that

$$\gcd\Bigl(\prod_{0 \leq i < j < n} (x_j - x_i),\ r\Bigr) \leq r/m.$$

The following lemma implies that among the numbers in $[m^2, 2m^2)$ a constant fraction is S-good.

Lemma 4. *Let Y be a sufficiently large integer, and let $L \geq \ln Y$. Then (at least) a constant fraction of the numbers r in $[L^2, 2L^2)$ satisfy* $\gcd(r, Y) \leq r/L$.

Proof. Let

$$A = A_{L,Y} = \{r \in [L^2, 2L^2) \mid \gcd(r, Y) > r/L\}.$$

Then $A \subseteq B \cup C_L$, where

$$B = B_{L,Y} = \{r \in [L^2, 2L^2) \mid \gcd(r, Y)\, \text{has a prime factor}\, p > L\}$$

and

$$C_L = \{r \in [L^2, 2L^2) \mid p \leq L\, \text{for all prime factors}\, p\, \text{of}\, r\}.$$

Indeed, if $r \in [L^2, 2L^2) - (B \cup C_L)$, then r has a prime factor $p > L$ that does not divide Y, hence $\gcd(r, Y) \leq r/p < r/L$. We must estimate the sizes of B and C_L. First note that Y has at most $\ln Y / \ln L$ prime factors larger than L. Each such factor divides at most $L^2/L = L$ elements of $[L^2, 2L^2)$; thus, $|B| \leq L \cdot \ln Y / \ln L \leq L^2 / \ln L$.

In order to deal with C_L, we give a lower bound on the size of the complementary set $D_L = \{r \in [L^2, 2L^2) \mid r\, \text{has a prime factor larger than}\, L\}$. It is well known that $|D_L| = (\ln 2 - O(1/\log L)) \cdot L^2$ (for $L \to \infty$), the reason being the following: For each prime number $p \in (L, L^2]$ the set of multiples of p in $(L^2, 2L^2)$ has size at least $\lfloor L^2/p \rfloor \geq L^2/p - 1$. If we just add these figures, numbers $r \in [L^2, 2L^2)$ with two distinct prime factors $p_1, p_2 > L$ are counted twice. Note that in this situation we must have $p_1, p_2 < 2L$ and $r = p_1 p_2$; hence, by the prime number theorem, there are only $O((L/\log L)^2) = O(L/(\log L)^2)$ many such numbers r. The prime number theorem also entails that there are only $O(L^2/\log L)$ many primes in $(L, 2L^2]$. So we obtain:

$$|D_L| \geq \sum_{\substack{L<p\leq L^2 \\ p\,\text{prime}}} \left(\frac{L^2}{p} - 1\right) - O\left(\frac{L}{(\log L)^2}\right) = L^2 \cdot \sum_{\substack{L<p\leq L^2 \\ p\,\text{prime}}} \frac{1}{p} - O(L^2/\log L).$$

A well-known theorem from analytic number theory [20, p. 351, Theorem 427] says that $\sum_{p \leq x, p\,\text{prime}} 1/p = \ln\ln x + B_1 + E(x)$, where B_1 is a constant and $E(x) = O(1/\log x)$. It follows that

$$\sum_{\substack{L<p\leq L^2 \\ p\,\text{prime}}} \frac{1}{p} = \ln\ln L^2 - \ln\ln L - O(1/\log L) = \ln 2 - O(1/\log L)\,.$$

Summing up, we get that

$$|A| \leq |B| + |C_L| = |B| + (L^2 - |D_L|) = (1 - \ln 2) \cdot L^2 + O(L^2/\log L),$$

i.e., 69% of the r in $[L^2, 2L^2)$ satisfy $\gcd(r, Y) \leq r/L$, asymptotically. $\square$

We apply Lemma 4 for $L = m$ and $Y = \prod_{0\le i<j<n}(x_j - x_i)$. Since $m = n^3 \ln u > \ln Y$, the assumptions are satisfied, and we conclude that asymptotically at least 69% of the r in $[m^2, 2m^2)$ are S-good.

Remark 5. In the context of algorithms that offer higher reliability for the running time, like the real-time dictionary from [14], the probability bounds provided by Lemma 4 are too weak. At present, all *reliable* collapse functions that only use $O(\log\log u + \log n)$ random bits involve the use of prime numbers.

5 Extensions and Applications

Smaller Pure Arithmetic Classes. Starting from the construction of Sect. 3, Woelfel [41,42] succeeded in giving smaller hash classes with similar universality properties, thus reducing the space, the number of random bits, and possibly the evaluation time. In particular, he observed that it is sufficient to choose b at random from $k \cdot [m] = \{ik \mid 0 \le i < m\}$ to achieve the results in Sect. 3. Moreover, he showed that if r is a power of some prime p, one can have u as large as r, and one still gets a 1-universal class of hash functions from $[u]$ to $[m]$ with $m = r/k$, if one chooses a from $1 + p[r/p]$ and b from $p^{\lceil k/2\rceil} \cdot [p^{\lfloor k/2\rfloor}]$ uniformly at random. This class is both very efficient and small. Woelfel's work contains many more constructions for different universality concepts.

Lower Bounds on the Complexity of Multiplication. In [25], it is shown that 2-wise independent classes contain functions that are difficult to compute in several respects (time-space tradeoff $T \cdot S = \Omega(\log u \cdot \log m)$; quadratic $A \cdot T^2$ bounds for VLSI implementation; bounds for CREW PRAMs, boolean formulas, and constant-depth circuits). All these bounds transfer to integer multiplication in binary notation, by choosing r as a power of 2. Woelfel [43] and Bollig and Woelfel [6] used linear classes for proving lower bounds on the complexity of multiplication on several versions of branching programs, notably OBDDs. The central observation was that the universality properties imply that there are functions of the form $x \mapsto \lfloor((ax + b) \bmod r)/m\rfloor$ with moderately large r and $U = [u]$ only of size m^2 that are surjective. Bollig, Waack, and Woelfel [5] used a similar approach for lower bounds for multiplication in more general branching programs. Using similar hash classes, Sauerhoff and Woelfel [34] prove a time-space tradeoff for unrestricted, deterministic multi-way branching programs that compute the middle bit of integer multiplication.

Error-Correction Properties. For the purpose of constructing deterministic dictionaries, Miltersen [27] and Hagerup, Miltersen, and Pagh [19] utilized error-correction properties of 2-wise independent classes, in particular of the class from [12] and the classes studied in the present paper.

Limitations for Cuckoo Hashing and Other Applications. Pătraşcu and Thorup [31] proved that for min-wise hashing the linear classes fail badly. Dietzfelbinger and Schellbach [15,16] showed that using linear functions $h(x) =$

$((ax+b) \bmod p) \bmod m$ (for $p = u$ prime) and $h(x) = \lfloor((ax+b) \bmod r)/(r/m)\rfloor$ as in this paper in the naive way for cuckoo hashing will lead to failure of the data structure. On the other hand, Dietzfelbinger and Woelfel [17] showed how to run cuckoo hashing [32] with polynomials of constant degree in combination with lookup tables of random numbers as hash functions. With Aumüller these authors showed [3] how to modify the construction so that 1-universal classes, 2-wise independent classes, and tables are sufficient, so that now cuckoo hashing can be run with the linear classes from the present paper alone.

3SUM and Fine-Grained Complexity Inside P. 3SUM over an interval $[u]$ of integers, for sets of size n, is the following problem: Given three sets $A, B, C \subseteq [u]$, all of size n, decide whether there are $x \in A$, $y \in B$, $z \in C$ such that $x + y = z$. Depending on the model of computation, this problem is thought to have different complexities. The obvious deterministic algorithm (involving sorting and repeated merging) takes time $\Theta(n^2)$. If one specifies the model in more detail, one may consider the word RAM (random access machine with word length w). Here, randomization, bit-level parallelism, and other tricks make it possible to solve the problem faster, see, e.g., the seminal paper by Baran, Demaine, and Pătraşcu [4]. One tool in this faster (randomized) algorithm is universal hashing. The authors utilize a 1-universal hash class of functions from $[u]$ to $[m]$ that is "almost affine", meaning that for every hash function h from the class there is a constant c_h such that for $x, y, z \in [u]$ with $x+y=z$ we have $h(x) +_m h(y) \in \{h(x+y) +_m c_h, h(x+y) +_m c_h +_m 1\}$. (The offset c_h is known from affine functions in vector spaces.) It is not hard to see (see Wang [39], who also considers k-SUM) that our class $\mathcal{H}^{\text{lin}}_{u,m,r}$ has this property. A second property that is needed is 1-universality, from which it follows that the number of keys mapped to overfull buckets is close to its expectation. The same kind of hash function was used in many works on low-level complexity, e.g., [1,22,30,38], in arguments that prove conditional lower bounds for dynamic data structures and string and graph problems.[2]

Upper Bound on Bucket Size. Knudsen [21] showed that the expected bucket sizes created by $\mathcal{H}^{\text{lin}}_{u,m,r}$ on a set S of size $|S| = m$ is $O(m^{1/3})$.

Efficiency. Thorup [36] and Dahlgaard et al. [9] experimentally explore the efficiency of different hash classes. Our class $\mathcal{H}^{\text{lin}}_{u,m,r}$, for u, m, r powers of 2, turns out to be very fast, in particular for values that combine well with the word length of the computer. One should beware, however, that the class must be used only for purposes where its suitability has been proved.

Acknowledgment. The author is grateful to Carl Pomerance for helping with the proof of Lemma 4 and to a reviewer for a careful reading of the manuscript and helpful suggestions.

[2] It should be noted that although the class of functions $x \mapsto (ax \bmod p) \bmod m$ for primes p is not suitable, the class of functions $x \mapsto (ax \bmod p)/\lfloor p/m\rfloor$ is also almost affine and $(1+\frac{1}{m})$-universal, so it could also have been used for this application.

References

1. Abboud, A., Williams, V.V., Weimann, O.: Consequences of faster alignment of sequences. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 39–51. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_4
2. Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple constructions of almost k-wise independent random variables. Random Struct. Algorithms **3**, 289–304 (1992). https://doi.org/10.1002/rsa.3240030308
3. Aumüller, M., Dietzfelbinger, M., Woelfel, P.: Explicit and efficient hash families suffice for cuckoo hashing with a stash. Algorithmica **70**(3), 428–456 (2014). https://doi.org/10.1007/s00453-013-9840-x
4. Baran, I., Demaine, E.D., Pătraşcu, M.: Subquadratic algorithms for 3SUM. Algorithmica **50**(4), 584–596 (2008). https://doi.org/10.1007/s00453-007-9036-3
5. Bollig, B., Waack, S., Woelfel, P.: Parity graph-driven read-once branching programs and an exponential lower bound for integer multiplication. In: Baeza-Yates, R., Montanari, U., Santoro, N. (eds.) Foundations of Information Technology in the Era of Network and Mobile Computing. ITIFIP, vol. 96, pp. 83–94. Springer, Boston, MA (2002). https://doi.org/10.1007/978-0-387-35608-2_8
6. Bollig, B., Woelfel, P.: A read-once branching program lower bound of $\Omega(2^{n/4})$ for integer multiplication using universal hashing. In: Proceedings of 33rd ACM STOC, pp. 419–424 (2001). https://doi.org/10.1145/380752.380835
7. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. J. Comp. Syst. Sci. **18**, 143–154 (1979). https://doi.org/10.1016/0022-0000(79)90044-8
8. Chor, B., Goldreich, O.: On the power of two-point based sampling. J. Complex. **5**, 96–106 (1989). https://doi.org/10.1016/0885-064X(89)90015-0
9. Dahlgaard, S., Knudsen, M.B.T., Thorup, M.: Practical hash functions for similarity estimation and dimensionality reduction. In: Proceedings of NIPS 30, pp. 6618–6628 (2017). http://papers.nips.cc/paper/7239-practical-hash-functions-for-similarity-estimation-and-dimensionality-reduction.pdf
10. Dietzfelbinger, M.: Universal hashing and k-wise independent random variables via integer arithmetic without primes. In: Puech, C., Reischuk, R. (eds.) STACS 1996. LNCS, vol. 1046, pp. 567–580. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-60922-9_46
11. Dietzfelbinger, M., Gil, J., Matias, Y., Pippenger, N.: Polynomial hash functions are reliable. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 235–246. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55719-9_77
12. Dietzfelbinger, M., Hagerup, T., Katajainen, J., Penttonen, M.: A reliable randomized algorithm for the closest-pair problem. J. Algorithms **25**(1), 19–51 (1997). https://doi.org/10.1006/jagm.1997.0873
13. Dietzfelbinger, M., Karlin, A., Mehlhorn, K., Meyer auf der Heide, F., Rohnert, H., Tarjan, R.E.: Dynamic perfect hashing: upper and lower bounds. SIAM J. Comput. **23**(4), 738–761 (1994). https://doi.org/10.1137/S0097539791194094
14. Dietzfelbinger, M., Meyer auf der Heide, F.: Dynamic hashing in real time. In: Buchmann, J., Ganzinger, H., Paul, W.J. (eds.) Informatik: Festschrift zum 60. Geburtstag von Günter Hotz, Teubner-Texte zur Informatik, vol. 1, pp. 95–119. B. G. Teubner, Stuttgart-Leipzig (1992). https://doi.org/10.1007/978-3-322-95233-2_7
15. Dietzfelbinger, M., Schellbach, U.: On risks of using cuckoo hashing with simple universal hash classes. In: Proceedings of 20th ACM-SIAM SODA, pp. 795–804 (2009). https://doi.org/10.1137/1.9781611973068.87

16. Dietzfelbinger, M., Schellbach, U.: Weaknesses of cuckoo hashing with a simple universal hash class: the case of large universes. In: Nielsen, M., Kučera, A., Miltersen, P.B., Palamidessi, C., Tůma, P., Valencia, F. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 217–228. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-95891-8_22
17. Dietzfelbinger, M., Woelfel, P.: Almost random graphs with simple hash functions. In: Proceedings of 35th ACM STOC, pp. 629–638 (2003). https://doi.org/10.1145/780542.780634
18. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. J. Assoc. Comput. Mach. **31**(3), 538–544 (1984). https://doi.org/10.1145/828.1884
19. Hagerup, T., Miltersen, P.B., Pagh, R.: Deterministic dictionaries. J. Algorithms **41**(1), 69–85 (2001). https://doi.org/10.1006/jagm.2001.1171
20. Hardy, G.H., Wright, E.M.: An Introduction to the Theory of Numbers. Oxford Science Publications, Clarendon Press, Oxford (1994)
21. Knudsen, M.B.T.: Linear hashing is awesome. CoRR arXiv:1706.02783 (2017). Technical report version of Knudsen, M.B.T.: Linear hashing is awesome. In: Proceedings of 57th IEEE FOCS, pp. 345–352 (2016). https://doi.org/10.1109/FOCS.2016.45
22. Kopelowitz, T., Pettie, S., Porat, E.: Higher lower bounds from the 3SUM conjecture. In: Proceedings of 27th ACM-SIAM SODA, pp. 1272–1287 (2016). https://doi.org/10.1137/1.9781611974331.ch89
23. Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput. **15**, 1036–1053 (1986). https://doi.org/10.1137/0215074
24. Luby, M., Wigderson, A.: Pairwise independence and derandomization. In: Foundations and Trends in Theoretical Computer Science, vol. 1, no. 4 (2005). https://doi.org/10.1561/0400000009
25. Mansour, Y., Nisan, N., Tiwari, P.: The computational complexity of universal hashing. Theor. Comput. Sci. **107**, 121–133 (1993). https://doi.org/10.1016/0304-3975(93)90257-T
26. Mehlhorn, K., Vishkin, U.: Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. Acta Inform. **21**, 339–374 (1984). https://doi.org/10.1007/BF00264615
27. Miltersen, P.B.: Error correcting codes, perfect hashing circuits, and deterministic dynamic dictionaries. In: Proceedings of 9th ACM-SIAM SODA, pp. 556–563 (1998). https://dl.acm.org/citation.cfm?id=314613.314845
28. Nisan, N.: Pseudorandom generators for space-bounded computations. Combinatorica **12**(4), 449–461 (1992). https://doi.org/10.1007/BF01305237
29. Pagh, R.: Hash and displace: efficient evaluation of minimal perfect hash functions. In: Dehne, F., Sack, J.-R., Gupta, A., Tamassia, R. (eds.) WADS 1999. LNCS, vol. 1663, pp. 49–54. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48447-7_5
30. Pătraşcu, M.: Towards polynomial lower bounds for dynamic problems. In: Proceedings of 42nd ACM STOC, pp. 603–610 (2010). https://doi.org/10.1145/1806689.1806772
31. Pătraşcu, M., Thorup, M.: On the k-independence required by linear probing and minwise independence. ACM Trans. Algorithms **12**(1), 8:1–8:27 (2016). https://doi.org/10.1145/2716317
32. Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms **51**(2), 122–144 (2004). https://doi.org/10.1016/j.jalgor.2003.12.002

33. Ružić, M.: Making deterministic signatures quickly. ACM Trans. Algorithms **5**(3), 26:1–26:26 (2009). https://doi.org/10.1145/1541885.1541887
34. Sauerhoff, M., Woelfel, P.: Time-space tradeoff lower bounds for integer multiplication and graphs of arithmetic functions. In: Proceedings of 35th ACM STOC, pp. 186–195 (2003). https://doi.org/10.1145/780542.780571
35. Siegel, A.: On universal classes of extremely random constant-time hash functions. SIAM J. Comput. **33**(3), 505–543 (2004). https://doi.org/10.1137/S0097539701386216
36. Thorup, M.: Even strongly universal hashing is pretty fast. In: Proceedings of 11th ACM-SIAM SODA, pp. 496–497 (2000). https://dl.acm.org/citation.cfm?id=338219.338597
37. Thorup, M., Zhang, Y.: Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. SIAM J. Comput. **41**(2), 293–331 (2012). https://doi.org/10.1137/100800774
38. Williams, V.V., Williams, R.: Finding, minimizing, and counting weighted subgraphs. SIAM J. Comput. **42**(3), 831–854 (2013). https://doi.org/10.1137/09076619X
39. Wang, J.R.: Space-efficient randomized algorithms for K-SUM. In: Schulz, A.S., Wagner, D. (eds.) ESA 2014. LNCS, vol. 8737, pp. 810–829. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44777-2_67
40. Wegman, M.N., Carter, J.L.: New classes and applications of hash functions. In: Proceedings of 20th IEEE FOCS, pp. 175–182 (1979). https://doi.org/10.1109/SFCS.1979.26
41. Woelfel, P.: Efficient strongly universal and optimally universal hashing. In: Kutyłowski, M., Pacholski, L., Wierzbicki, T. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 262–272. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48340-3_24
42. Woelfel, P.: Klassen universeller Hashfunktionen mit ganzzahliger Arithmetik. Diploma thesis, University of Dortmund (2000). (In German)
43. Woelfel, P.: New bounds on the OBDD-Size of integer multiplication via universal hashing. In: Ferreira, A., Reichel, H. (eds.) STACS 2001. LNCS, vol. 2010, pp. 563–574. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44693-1_49

Small Complexity Gaps for Comparison-Based Sorting

Shogo Ehara[1], Kazuo Iwama[1](✉), and Junichi Teruyama[2]

[1] Kyoto University, Kyoto, Japan
ehafib@gmail.com, iwama@kuis.kyoto-u.ac.jp
[2] Kwansei Gakuin University, Nishinomiya, Japan
junichi.teruyama@gmail.com

Abstract. Our problem is the *average* complexity on the number of comparisons for sorting algorithms. Its information-theoretic lower bound is $n \lg n - 1.4426n + O(\log n)$. Since it is easy to achieve the first $n \lg n$ term, so our focus is on the (negative) constant factor of the linear term. In their WADS2017 paper, Iwama and Teruyama obtained a new upper bound for this constant, which improves the existing gap by some 25%. However, since its main purpose is to have a closed formula for analysis, their algorithm itself, *(1,2)Insertion*, seems to have several possibilities for improvement. This paper focuses on such possibilities, and to this goal, we introduce a general scheme, *GMS*, for merge-based sorting algorithms, which includes, as a special case, (1,2)Insertion, the standard Merge Sort, the BinaryInsertion Sort, etc. We obtain a lower bound of the complexity for this model and three upper bounds for three specific GMSs. We cannot obtain closed forms for the quantity but can obtain an exact average for the number of comparisons up to $n = 65$ efficiently. The major finding is that inserting one or two elements to an already sorted sequence, which is the fundamental strategy of (1,2)Insertion, is quite powerful; we conjecture that more complicated merges, i.e., merging two sequences one of which is longer than two, would not help to seek a better performance.

1 Introduction

Sorting has probably the longest history of research and education among major topics in computer science. It is not only a popular routine that has appeared most frequently in all kinds of programs but also a research target that has received intensive studies from various different angles. One of the remarkable facts claiming its popularity as a research target is that a sorting algorithm which is purely aimed at its theoretical performance with almost nothing in its practical merit, due to Ford and Johnson [5], already appeared as early as the 1950's. As is well known, we met a lot of similar examples of "theoretically biased" algorithms later in the 70's and 80's.

Recall that a majority of existing sorting algorithms, including Bubble Sort, Quick Sort, Heap Sort, Merge Sort and Insertion Sort, are so-called comparison-based sort, in which the input can be accessed only through a comparison of two (positions of the) numbers and only the 1-bit information of which is larger is available. Thus the number of comparisons needed to obtain a sorted sequence is a natural complexity measure which can be scaled more accurately than other measures such as computation time. Note that any sorting algorithm for n elements can be described as a binary (due to two different outcomes in the comparison) decision tree having $n!$ leaves corresponding to all different permutations of the n elements. The number of comparisons to obtain one of them is the number of nodes on the path from the root to the leaf corresponding to the sequence. Therefore we have an obvious lower bound, called the information-theoretic lower bound, for the number of comparisons: Any sorting algorithm needs

$$\lceil \lg n! \rceil \approx n \lg n - 1.4426n + O(\log n)$$

comparisons in the worst case.

If we are interested in only the major term, $n \lg n$, it is not hard to achieve it. For instance, consider the BinaryInsertion Sort that increases the length of the sorted sequence one by one using binary insertion of new items. We have $n-1$ insertion steps and each of them consists of at most $\lceil \lg n \rceil$ comparisons and less ones for many of the entire steps. Thus our interest naturally comes to the constant factor for the linear term in n. Unfortunately, however, our knowledge on its quantity is quite limited in spite of the problem's long history; it is at most -0.91 for Merge Sort [8,16] and the current best one is -1.32 for the MergeInsertion sort due to [5] mentioned above. We have had no progress for almost six decades!

Compared to the worst-case complexity, the *average-case complexity* seems more tractable (its information-theoretic lower bound is the same as the worst-case). In fact we have a number of better values for the constant; -1.26 for Merge sort [8], -1.38 for BinaryInsertion sort, and most recently -1.3999 for MergeInsertion sort [4]. Notice that 1.3999 is some 96.98% of 1.4426, but there still exists a gap and seeking the exact bound for this fundamental problem should be an important research goal.

In [7], Iwama and Teruyama narrowed this small but still existing gap between 1.3999 and 1.4426 by some 25%. Note that the BinaryInsertion Sort, based on such a simple idea of repeating a binary insertion of a new item into a sorted sequence of length i, has a fairly good performance as shown above. Here the performance of a single binary insertion itself is optimal because it constitutes an optimal decision tree, D_i, whose path length differs at most by one. However, this optimality quickly disappears in the decision tree, $D_{i-1,i}$, of two consecutive insertions, one to a sequence of length $i-1$ and the next to that of length i. Namely $D_{i-1,i}$ is the tree obtained by replacing each leaf of D_{i-1} by D_i, and its path length differs by two unless $i-1$ or i is a power of two. Thus if i is close to a power of two, most of the paths still have the same length and the imbalance may not be very serious for the average complexity. However, if i is far from a power of two, only one half of the paths have the same length, a quarter has a length smaller by one and a quarter has a length longer by one, in the worst case.

This observation leads to the following natural idea for a possible improvement. If the current i is close to a power of two, we just use a binary insertion. If i is far from a power of two, then we use what we call a "two-element merge," or *2Merge*. 2Merge merges a two-element sequence (a_1, a_2), $a_1 < a_2$, with a sorted sequence Y of length $i-2$ to obtain a sorted sequence of length i. Of course we would like to see that the decision tree for 2Merge is close to optimal or at least looks better than the above $D_{i-1,i}$. This is in fact possible and is the essence of the improvement obtained by [7].

Thus [7] achieves -1.4034 by combining 2Merge and the binary insertion, and -1.4106 by further combining them with MergeInsertion. Note that the main purpose of [7] is to obtain a good theoretical performance by closed formulas, for which we need to avoid complication of algorithms as much as possible. Hence there remain several questions about the optimality of this new algorithm as well as its possible extensions. For instance, one would be quickly interested in the possibility of adding 3Merge and 4Merge and so on to the algorithm.

The main purpose of this paper is to answer these questions. To do so, we introduce a sorting scheme called *Generalized Merge Sort* or *GMS*. Recall that Mergesort cuts the given input numbers into two parts, the front part X and the rear part Y, sorts X and Y recursively, and merges the resulting sorted sequences X' and Y'. Here the standard Mergesort has the practice that (i) the size of X and Y is the same and (ii) when merging (sorted) X' and Y', the order of comparisons of two numbers, one from X' and the other from Y', is regular, left to right in both sequences. This seems ok for its compact implementation and several practical merits, but should limit its theoretical performance. Our GMS relaxes (i) and (ii), namely, the size of X and Y may be arbitrarily different and we can design the order of comparing two numbers in X and Y freely. Now the new sorting algorithm in [7], called *(1,2)Insertion*, is a GMS where $|X|$ is one or two and the order of the pair-wise comparisons is that of the standard binary search if $|X|$ is one, and carefully designed if $|X|$ is two but still based on a kind of binary search (in order to make analysis tractable).

Thus, (1,2)Insertion lacks two kinds of generality in terms of GMS. One is that (i) $|X|$ must be one or two and the other is that (ii) the comparison sequence when X is of length two is probably not optimal. To investigate the effect of (ii) we want to design a GMS whose $|X|$ is still one or two but the comparison sequence is optimal. This is indeed possible: For a given (unsorted) sequence of n numbers, a GMS first determines the cut position between X and Y and after recursively sorting X and Y, then it merges X and Y using its own order of pair-wise comparisons. Our GMS, called *Opt(1, 2)GMS*, can select a better cut position, 1 or 2, and if it is 1, the insertion use the standard binary search that is optimal and if it is 2, *Opt(1, 2)GMS* can find an optimal comparison sequence for the merge. These selections are done by constructing (implicitly) the entire decision tree optimal for the input length n.

To investigate (i), we want to design a GMS that can select any cut position between X and Y. But unfortunately, there is no obvious way of finding an optimal comparison sequence for merging X and Y *in polynomial time*. (If we can spend exponential time, obtaining the absolutely optimal decision tree is possible by exhaustive search. In the case of above *Opt(1, 2)GMS*, "an implicit" construction of the optimal tree is possible in polynomial time thanks to the length of X.) Allowing exponential time is clearly unfair, so our idea is to abandon the absolute optimality or to seek an approximated optimality using heuristics and randomness.

For an input size n our GMS, *RandGMS*, first checks all $n-1$ cut positions. For each cut position i, $|X| = i$ and $|Y| = n - i$, we further check all possible (roughly n^2) pair-wise comparisons of a in X' and b in Y' (recall we have no specific knowledge about X' or Y' other than they are sorted): For each a and b, we construct two posets (partially ordered sets) due to $a < b$ and $b < a$. Let P_1 be the poset for the former and P_2 for the latter. We then count the number of linear extensions for P_1 and P_2 (in polynomial time), and the pair a and b whose P_1 and P_2 have most balanced numbers of linear extensions is selected (the heuristics part). Then we move to one of the posets at random and recurse until the linear extension becomes unique. The number of recursion steps is the number of comparisons of this randomly selected path. By repeating this random traverse a large constant number of times we can estimate the average depth of the approximately optimal decision tree. We use this quantity plus the average number of comparisons for X and Y as the quality factor of the cut position, select the best cut position, recursively call GMS for X and Y, and merge X' and Y' using the pair-wise comparison order described above. The time complexity is apparently polynomial, but it is far larger than $n \log n$, as large as $O(n^5)$, a possible improvement of which would be our obvious target for future research.

We have obtained an average number of comparisons for these GMSs for up to $n = 65$. Note that this is not a simulation of the GMS using benchmark inputs but our performance evaluation program computes an "exact complexity" or the true average of comparison numbers for $n!$ input sequences in polynomial time. Through this numerical experiment, we have several findings, for instance, it turned out that cut positions 1 and 2 are probably enough and cut positions 3 and larger are unlikely to help for a better performance. Details are given in Sect. 5 with Table 1 summarizing several data for the number of comparisons.

Related Work. The worst-case complexity for the number of comparisons has also been a popular research topic. The second and third columns of Table 1 in Sect. 5 shows what is known up to now for small n's [15]. Here, $C(n)$ is the information-theoretic lower bound above mentioned. $S(n)$ is the lower bound for comparison-based sorting algorithms which were obtained by exhaustive search using computers up to $n = 15$ [12–14,18]. Thus it is known that the information-theoretic lower bound cannot be achieved by any comparison-based sorting for $12 \leq n \leq 15$. It is also known that the Ford-Johnson algorithm [1,5] achieves a matching upper bound for $1 \leq n \leq 15$. So as far as comparison-based sorting is concerned, our knowledge is complete for $1 \leq n \leq 15$. However, $n = 16$ (sorting of only 16 numbers!) is still open, namely we have a gap of one between the lower and upper bounds for the minimum number of comparisons. It is also known that Ford-Johnson is not optimal for some values of n larger than 16, for instance $n = 47$ [9–11].

2 Lower Bounds

A GMS is given as Algorithm 1. *GCut* computes the best cut position from solely n, the length of the unsorted sequence, without using any comparison (we conjecture this restriction does not lose generality, but do not have a formal proof). *GMerge* merges two sorted sequences (see the next section for details).

Algorithm 1. *GMS*(S)

```
Input: A sequence $S$ of integers
Output: The sorted sequence $S'$ for $S$
1 $n := |S|$
2 if $n = 1$ then
3     return $S$
4 end
5 $n_1 = GCut(n)$
6 return $GMerge(GMS(S(1, n_1), GMS(S(n_1 + 1, n))$
```

Let $T(n)$ be the (average) number of comparisons of *GMS*(S) for $n = |S|$. According to our assumption that no comparisons are used in *GCut*, $T(n)$ is the sum of $T(n_1)$, $T(n - n_1)$ and the number of comparisons executed in *GMerge*. Note that *GMerge* outputs a sorted sequence from two sorted sequences X and Y of length n_1 and $n-n_1$, respectively. So for fixed X and Y, there are $\binom{n}{n_1}$ different sequences as the output of *GMerge*. It then turns out that the optimal decision tree (if any) for *GMerge* that minimizes the (both average-case and worst-case) number of comparisons, is the optimal binary tree having $\binom{n}{n_1}$ leaves. Thus we have

$$T(n) \geq \min_{1 \leq i \leq \lfloor n/2 \rfloor} \left(T(i) + T(n-i) + AvDpth\left(\binom{n}{i} \right) \right),$$

where $AvDpth(M)$ is the average path length of the optimal binary tree having M leaves. The lower bound of $T(n)$ that is given by the same formula with replacing $\geq$ with $=$ can be efficiently computed from small to large n's, its value up to $n = 65$ is given in Table 1.

3 Upper Bounds

3.1 Partially Ordered Sets

In this section, a partially ordered set (poset) plays an important role. Our posets in this paper have a special form, namely as shown in Fig. 1, they consist of two totally ordered sets and several (ordered) edges between them. The two ordered

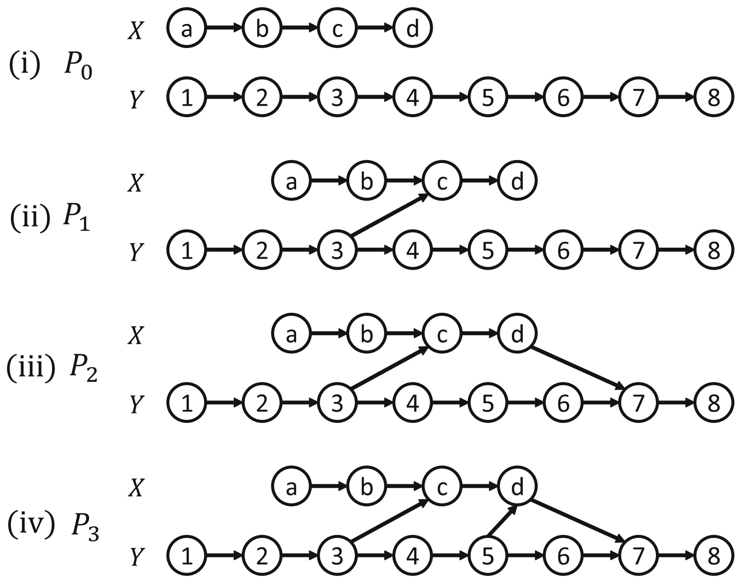


Fig. 1. Partially ordered sets

sets correspond to two sorted sequences X and Y and edges between them are due to the results of pair-wise comparisons executed in the (generalized) merge procedure. The poset, say P_0, in Fig. 1 (i) has X of length 4 and Y of length 8 with no comparison between them yet. Suppose we compare 3 and c and the result is $3 < c$. Then the poset changes to P_1 in (ii). If we further compare d and 7 (the result is $d < 7$), and then d and 5 (the result is $d > 5$), the poset changes to P_2 in (iii) and then P_3 in (iv). Now let us consider our knowledge about the right position of c in Y. In P_0, we have no information about that, namely $-\infty < c < +\infty$. In P_1, its range is narrowed to $3 < c < +\infty$. Since $c < d$, we also have $3 < d < +\infty$. Thus we increase our knowledge on the position of each item in X; we have $-\infty < a < 7$, $-\infty < b < 7$, $3 < c < 7$ and $5 < d < 7$ in P_3.

We denote this information by $PI(P)$ (*poset information*) for a poset P and describe it as a tuple of eight (two times the length of X) items $(x_1, x_2, \ldots, x_8)$ where x_i is an item in Y or $\pm\infty$. For the poset P_3, $PI(P_3) = (-\infty, 7, -\infty, 7, 3, 7, 5, 7)$, which is simply a combination of inequalities $3 < c$ and $3 < d$ for edge 3 to c, $5 < d$ for edge 5 to d, and $a < 7, b < 7, c < 7$ and $d < 7$ for edge d to 7. Thus it is straightforward to obtain $PI(P)$ from a poset P in polynomial time; its formal algorithm may be omitted.

Let us see the importance of poset information: Suppose that the length of Y is n in general but the length of X is small, say 4 as in Fig. 1. Then the number of different posets can be exponential since we have up to n^4 different pairs for the pair-wise comparison between X and Y and many combinations of different edge directions of them can result in posets. For instance we have 2^3 patterns by considering all different directions of the three edges between X and Y in Fig. 1 (iv) and six of them do not include a cycle and they are posets. However, the number of different poset informations is obviously $(n+1)^8$. Thus we can use a typical DP technique when handling this type of posets if X or Y is short.

A *linear extension* of a poset P is a total order of all the items in P which does not violate any inequality in P. Due to our restriction, a linear extension of our poset is a "shuffle" of X and Y having no violation of inequalities given by the edges between X and Y. The number of different linear extensions of poset P is denoted by $L(P)$. If $PI(P_1) = PI(P_2)$, then the sets of linear extensions for P_1 and P_2 are obviously identical, and we have $L(P_1) = L(P_2)$, too.

3.2 Optimal (1,2)-Cut

A specific GMS is obtained by giving specific *GCut* and *GMerge*. We consider three different GMSs in this paper and the first one, *Opt(1, 2)GMS*, is a modified version of (1,2)Insertion in [7]. Our design is based on the construction of an optimal binary decision tree, $D(n)$, for the input size n under the condition that we follow the GMS scheme and the cut position is only 1 or 2 ($|X|$ =1 or 2), meaning *Opt(1, 2)GMS* is optimal under that condition. Suppose that we have already obtained $D(n-2)$ and $D(n-1)$. Then $D(n)$ can be obtained as follows: Let a_1 and a_2 be the first and the second element of the input (assume $a_1 < a_2$), respectively. Then $D(n)$ should be either (1) $D(n-1)$ followed by the optimal decision tree for inserting a_1 into a sorted sequence of length $n-1$ or (2) $D(n-2)$ followed by the optimal decision tree for merging (a_1, a_2) and a sorted sequence of length $n-2$. Of course $D(n)$ is the better one in terms of the average number of comparisons.

Since this decision tree has $n!$ leaves, we cannot construct it explicitly. Instead we compute, in polynomial time, the average number of comparisons, $\overline{D}(n)$, of tree $D(n)$ in a bottom-up fashion. Suppose that we have already computed $\overline{D}(n-2)$ and $\overline{D}(n-1)$. Then if we know the average number of comparisons for the merge part of (1) and (2), we can obtain $\overline{D}(n)$. The optimal decision tree for the insertion in (1) is obviously that of the standard binary search and we can compute its average number of comparisons easily. By adding this number and $\overline{D}(n-1)$, we have the average complexity of option (1).

To obtain the same quantity for option (2), we first need to design an optimal pair-wise comparison sequence for merging $X = (a_1, a_2)$ and Y of length $n-2$. This algorithm already appeared in the literature [17], which however seems quite complicated and slow. Our idea in this paper is as follows: The decision tree for this merging process is a binary tree whose node V is a poset $P(V)$. The poset $P(R)$ of the root node R consists of only two total orders corresponding to X and Y. Suppose that we first compare a and b (a is a_1 or a_2 and b is some item in Y) in the merge. Then the two sons of R, U and W, corresponding to the two outcomes of the comparison, have posets $P(U) = P(R) \cup \{a < b\}$ and $P(W) = P(R) \cup \{b < a\}$, respectively. The tree grows similarly (see Fig. 2).

In order to make this decision tree optimal, we need to select the best pair a and b among roughly $2n$ candidates (recall that $|X| = 2$ and $|Y| = n-2$) at

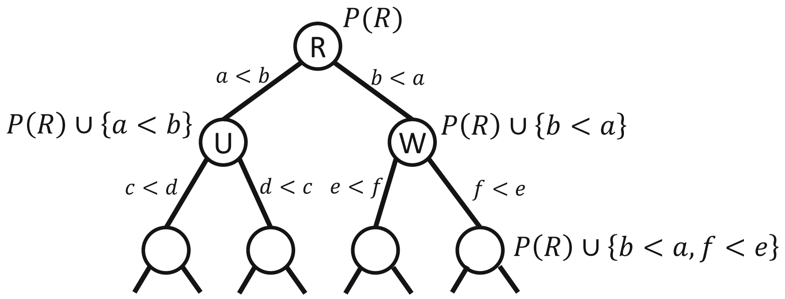


Fig. 2. Decision trees for merge

node R, the best pair c and d at node U and so on. To do this, we associate two numbers s and t with each node V of the tree; $s(V)$ is the number of leaves and $t(V)$ is the total path length of the subtree whose root is V. Suppose that we have already calculated $s(U), t(U), s(W)$, and $t(W)$. Then it turns out that we can compute the two values of the root R as

$$s(R) = s(U) + s(W) \text{ and } t(R) = s(U) + s(W) + t(U) + t(W).$$

Thus we can choose a and b so that $t(R)$ becomes minimum. (since the number of leaves is all the same for any pair, the minimum total path length means the minimum average length). We can obviously do this by computing s and t values of the posets appearing in the decision tree in a bottom-up fashion. Note that for each leaf V (= a poset having a unique total order) $s(V) = 1$ and $t(V) = 0$.

This is again impossible to do in polynomial time since there are exponentially many different posets even for X of size 2. Fortunately, we can bypass this problem by replacing a poset owned by each node with a poset information. As mentioned in Subsect. 3.1, the number of all different poset informations is polynomial. Now we can obtain the average complexity of option (2) by adding $t(R)/s(R)$ and $\overline{D}(n-2)$. Thus we are done with *Opt(1, 2)Cut*, which just returns the better option. It is important to know that *Opt(1, 2)Cut* needs to compute $\overline{D}(i)$ for $1 \leq i \leq n$, since these values are exactly what we want for claiming the performance of the algorithm for a specific n. We indeed use *Opt(1, 2)Cut* for the evaluation of our GMSs in Sect. 5. *Opt(1, 2)Merge* just follows the comparison sequence computed by *Opt(1, 2)Cut*.

The pseudo codes are given as Algorithms 2 to 4. Val2M(P) computes the s and t values of the poset information, where we denote those values by using a subscript. $P \wedge b < a$ means the poset information for the poset P', such that $PI(P') = P$, with the added edge from b to a. The code is described recursively, but we have to run it in a DP fashion or in a bottom-up fashion to achieve polynomial time. Recall that there are only polynomially many poset informations. OPT(1,2)Cut(n) computes the array $\overline{D}[n]$ recursively. The following routines are easy and omitted: ValBS($n-1$) returns the average complexity of inserting an element into a sorted sequence of length $n-1$ by the standard binary search.

BinaryInsertion actually does this insertion. Also, OPT2Merge merges X of length 2 and Y of length $n-2$ following the comparison sequence determined by Val2M.

Algorithm 2. OPT(1,2)GMS(S)

```
Input: A sequence $S$
Output: Sorted sequence of $S$
1 if $|S| \leq 3$ then
2     return Sorted $S$ (use any reasonable sort algorithm)
3 end
4 if OPT(1,2)Cut($n$) = 1 then
5     return BinaryInsertion($S[1]$, OPT(1,2)GMS($S[2,\ldots,n]$))
6 end
7 else
8     return
      OPT2Merge(OPT(1,2)GMS($S[1,2]$), OPT(1,2)GMS($S[3,\ldots,n]$))
9 end
```

Algorithm 3. OPT(1,2)Cut(n)

```
Input: Integer $n \geq 4$
Output: Cut position $c = 1$ or 2, and value $v = \overline{D}[n]$
1  $\overline{D}[1] := 0$; $\overline{D}[2] := 1$; $\overline{D}[3] := 8/3$;
2  $\overline{D}[n-1] :=$ OPT(1,2)Cut$_v(n-1)$; $\overline{D}[n-2] :=$OPT(1,2)Cut$_v(n-2)$
3  $M :=$Val2M$_t(P_0(2,n-2))/$Val2M$_s(P_0(2,n-2))$
4  if $\overline{D}[n-1]$+ValBS$(n-1) < \overline{D}[n-2] + M$ then
5      return $c = 1$ and $\overline{D}[n] := \overline{D}[n-1]$+ValBS$(n-1)$
6  end
7  else
8      return $c = 2$ and $\overline{D}[n] := \overline{D}[n-2] + 1 + M$
9      ($P_0(2,n-2)$ is the poset information for the poset including only two total
       orders of length 2 and $n-2$.)
10 end
```

Algorithm 4. Val2M(P)

```
Input: Poset information $P$ for a poset including $X$, $Y$ and arrows between them
Output: Values $s$ and $t$
1 if $P$ has a unique linear extension then
2     return $s := 1$, $t := 0$
3 end
4 Among all $(a,b)$'s s.t. $a \in X$, $b \in Y$ and not yet compared, find $(a,b)$ s.t.
  Val2M$_t(P \wedge a < b)$+Val2M$_t(P \wedge b < a)$ becomes minimum
5 Let such $(a,b)$ be $(a_0,b_0)$
6 return $s :=$ Val2M$_s(P \wedge a_0 < b_0)$+Val2M$_s(P \wedge b_0 < a_0)$ and
  $t := s+$ Val2M$_t(P \wedge a_0 < b_0)$+Val2M$_t(P \wedge b_0 < a_0)$
```

3.3 Random Estimation

Our second GMS, *RandGMS*, has *RandCut* and *RandMerge* and allows us to remove the restriction of *Opt(1, 2)GMS* for which we considered only cut positions 1 and 2. Namely, *RandCut* can cut the input at any position. This is definitely an improvement over *Opt(1, 2)GMS*, but in turn we have to abandon the optimality of the decision trees as is shown in a moment. Our obvious goal is to investigate what this trade-off looks like.

The basic structure of *RandCut* is the same as *Opt(1, 2)Cut*. Again let $D(n), D(n-1), \ldots, D(1)$ be the decision tree of our *RandGMS* for the input size $n, n-1, \ldots, 1$. As with the previous case, $D(n)$ is obtained from $D(n-1), \ldots, D(1)$. Namely, for each cut i, there is an optimal decision tree, $M(n, i)$, for merging X of length i and Y of length $n-i$. Then the best decision tree for cut i is the concatenation (each leaf is replaced by the next tree) of $D(i)$, $D(n-i)$ and $M(n, i)$. To have $D(n)$, we simply select the best i in terms of the average complexity.

Again we do not compute the trees themselves but only their average complexity, $\overline{D}(n), \overline{D}(n-1), \ldots, \overline{D}(1) = 0$, in a bottom-up fashion. Suppose that we have obtained $\overline{D}(1)$ through $\overline{D}(n-1)$. Now we want to compute $\overline{D}(n)$ for the input of length n and to do so, we want to obtain a "good" cut position i. This can be done if we can evaluate $M(n, i)$ for each i namely if we can compute $\overline{M}(n, i)$ that is the average number of comparisons of $M(n, i)$. All we have to do then is to take the best $i = i_0$ that minimizes $\overline{D}(n) = \overline{M}(n, i_0) + \overline{D}(i_0) + \overline{D}(n-i_0)$. The tree $M(n, i)$ looks like the one in Fig. 2; in each node one should select the best pair to be compared and branch to the two sons due to the outcome. Again the number of different posets is exponential, but what differs from the previous section is that it does not help to replace each poset with poset information; there are exponentially many poset informations, too, because the length of X is arbitrary.

Thus we cannot do the bottom-up computation in polynomial time. What we do instead is to use an approximated value for $\overline{M}(n, i)$ or to "estimate" the true value by randomly traversing $M(n, i)$. (We abuse the same notation $\overline{M}(n, i)$ for the approximated value.) We first need to fix $M(n, i)$ by determining the pair of a in X and b in Y to be compared in the first step, by which we can determine two sons U for $a < b$ and W for $b < a$ of the root R (see Fig. 2 again). Here we would want to select the best pair minimizing the average complexity, but it is impossible as mentioned above. Instead, we use a heuristic that $s(U)$ and $s(W)$ should be balanced. (Recall that $s(U)$ and $s(W)$ the number of leaves of subtrees with root U and W, respectively.) Here $s(U)$ is equal to the number of linear extensions, $L(P(U))$, of its poset $P(U)$. This rule is the same in all the nodes of the tree. Fortunately the number of linear extensions for our type of poset can be computed in polynomial time as explained in Sect. 4, so each traverse of the path takes polynomial time.

Thus we decide the first pair as the one which achieves the most size-balanced subtrees. Then we go to one of the two sons at random, where we go to U with probability $L(P(U))/L(P(R))$ and to W with $L(P(W))/L(P(R))$ (both are close to 1/2 but might not be exactly 1/2). Note that $L(P(R)) = L(P(U)) + L(P(W))$.

Suppose that the random selection was U. Then we do the same thing, i.e., we select c and d to be compared next as the pair giving us the most balanced linear extensions for the two sons. We continue this until getting to a leaf and obtain the path length that is equal to the number of comparisons along this traversal of the decision tree $M(n, i)$. By repeating this traversal a (large) constant number of times, we define $\overline{Q}(n, i)$ as the average of the path length in those repeated traversals. Thus our $M(n, i)$ may not be optimal any longer and furthermore $\overline{Q}(n, i)$ is the value estimated from a small fraction of its paths.

We do this for all i's and select the best i_0 that minimizes $D^*(n) = \overline{Q}(n, i_0) + D^*(i_0) + D^*(n - i_0)$. Since $\overline{Q}(n, i_0)$ is not exact, $D^*(n)$ is not, either. That is why we use new notations $D^*(n)$ here instead of $\overline{D}(n)$. *RandCut* just returns this best i_0. Note that this $D^*(n)$ is used only for selecting cut positions and the sequence of comparisons for merge. As mentioned later, our performance evaluation is done using not $D^*(n)$ but $\overline{D}(n)$. See Algorithms 5 to 7 for the pseudo codes.

RandMerge simply follows the comparison sequence computed in *RandCut*, namely it always uses the pair-wise comparison that produces the most size-balanced subtrees. Once the best pair is determined, then it actually makes that comparison and goes to one of the subtrees depending on its outcome.

3.4 Combination of *Opt(1, 2)GMS* and *RandGMS*

Our $\overline{Q}(n, i)$ in *RandGMS* was an approximated value, but if $i = 1$ or 2, we can use the exact value as described in Subsect. 3.1. Our third GMS, *OptRandGMS*, replaces this part of *RandGMS* by the same part of *Opt(1, 2)GMS*. Details may be omitted.

4 Enumeration of Linear Extensions

Enumeration of $L(P)$ for a poset P is #P-hard in general [2]. However, if the width of the poset is bounded by a constant, it can be computed in polynomial time [3]. Since all posets in this paper are even more restricted, we can use the following simple approach for its enumeration. (Similar ideas are somewhat popular for different purposes but we are not aware of any literature on its application to counting linear extensions.)

Figure 3 (i) shows a poset satisfying the restriction in this paper. We use a 2D mesh structure as illustrated in (iii) to count linear extensions. Ignore the arrow from 3 to c in the poset for a while. Then the poset has only two total orders and the corresponding mesh has complete rows and columns corresponding to items of one total order and corresponding to items of the other, respectively. A linear extension has a one-to-one correspondence to a path from the left-top corner to the right-bottom corner in the mesh. The path must follow the traversing rule, namely it can go from left to right or downward but cannot from right to left or upward. For instance, the path given by the bold line corresponds to the linear extension $1a2bcd34e56$. Now we revive the edge from 3 to c. Then this path violates this arrow because c comes before 3. Suppressing such an illegal path is easy: it suffices to delete the edge from $2b$ to $2c$ in the mesh. The reason is that if the path comes to $2b$, then it has already passed 1 and 2 but not 3. So if the path next goes to c, then 3 must appear after that, violating the order specified by the edge from 3 to c.

Algorithm 5. RandGMS(S)

```
Input: A sequence $S$
Output: Sorted sequence of $S$
1  if $|S| \leq 3$ then
2      return Sorted $S$ (use any reasonable sort algorithm)
3  end
4  $n_1$ :=RandCut($n$)
5  return RandMerge(RandGMS($S[1, \ldots, n_1]$), RandGMS($S[n_1 + 1, \ldots, n]$))
```

Algorithm 6. RandCut(n)

```
Input: Integer $n \geq 4$
Output: Cut position $c$ and value $v = D^*[n]$
1  $D^*[1] := 0$; $D^*[2] := 1$; $D^*[3] := 8/3$;
2  for $i = 1$ to $n - 1$ do
3      Repeat Traverse($P_0(i, n - i)$) $R_0$ times and let $\overline{Q}[i]$ := the average of the
       returned values. $R_0$ is a big constant.
4      $QD[i] := \overline{Q}[i] + D^*[i] + D^*[n - i]$
5  end
6  $c$ =argmin($QD[i]$)
7  return $c$ and $D^*[n] := QD[c]$
```

Algorithm 7. Traverse(P)

```
Input: Poset information $P$ for a poset including $X$, $Y$ and arrows between them
Output: Path length $l$
1  if $P$ has a unique linear extension then
2      return $l := 0$
3  end
4  Among all $(a, b)$'s s.t. $a \in X$, $b \in Y$ and not yet compared, find $(a, b)$ s.t.
   $|e_1$ :=LExt($P \wedge a < b$) $- e_2$ :=LExt($P \wedge b < a$)$|$ becomes minimum. (LExt($P$)
   computes $L(P)$).
5  Let such $(a, b)$ be $(a_0, b_0)$
6  $r$ := a random value in $[0, 1]$
7  if $r \leq e_1/(e_1 + e_2)$ then
8      return Traverse($P \wedge a_0 < b_0$) + 1
9  end
10 else
11     return Traverse($P \wedge b_0 < a_0$) + 1
12 end
```

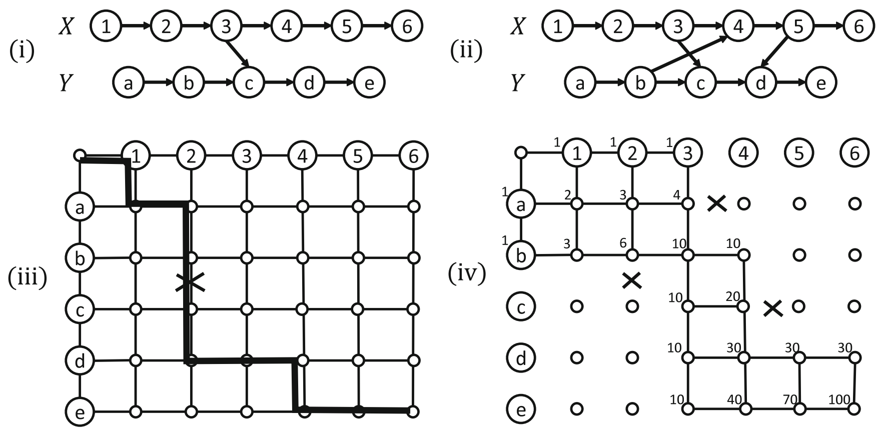


Fig. 3. Enumeration of linear extensions

For the same reason, we have to remove edges from b to c and from $1b$ to $1c$. Then due to the traversal rule, all the edges located below those removed edges should be removed, too. Thus the mesh changes as shown in the left-bottom area of (iv). We have more edges in the poset in (ii) and more edges are removed in the mesh as in (iv). Counting the number of legal linear extensions is straightforward, too. In the mesh. we put number 1 to each of the uppermost vertices and to each of leftmost vertices. For an inner vertex, we assign the value that is the sum of the value of its left-adjacent one and its upper-adjacent one. The final answer, the number of linear extensions, is the value assigned to the right-bottom corner. The algorithm obviously runs in quadratic time in the size of the poset. Details may be omitted.

5 Performance Evaluation for Small n

Table 1 shows several lower and upper bounds for the average number of comparisons, up to $n = 65$. As mentioned in Sect. 3, our three algorithms run in polynomial time. For instance, Opt(1,2)GMS computes the values of $\overline{D}[n]$, which is also the data that our performance–evaluation program wants to have and can be obviously obtained by simulating a part of Opt(1,2)GMS that computes $\overline{D}[n]$. Thus our evaluation programs also run pretty fast. The main reason for stopping at $n = 65$ is the limit of the largest integer that can be handled in the program. We did not have enough time to fix this problem, but we are sure we can in the near future. We will then be able to reach at least up to $n = 100$.

The first column $C(n)$ is the information-theoretic lower bound given in Sect. 1, which is probably not achieved by any sorting algorithm. $S(n)$ and $F(n)$ are the worst-case lower and upper bounds also described in Sect. 1. $T(n)$ is the (average-case) lower bound for GMS (see Sect. 2). The next two columns, "Opt(1,2)" and "Rand," are upper bounds for our GMSs Opt(1,2)GMS and

RandGMS. As is explained later, the data for our third one, OptRandGMS, is the same as that of Opt(1,2)GMS and omitted. Here “cut” is the cut position of the input selected by the algorithm. Recall that Opt(1,2) shows exact values, $\overline{D}[n]$ in the algorithm, namely the true average for n! different sequences. Recall that RandGMS uses a random estimation of the best cut position, but once the position is determined, the average complexity due to that decision tree can be computed exactly (thus the values in the table are different from the values in the estimated values, $D^*[n]$, in the algorithm). This needs exponential time in general, but fortunately in our case, the best estimated cut position is all 1 or 2, so this counting can be done in polynomial time, too. The next column, IT1, is the upper bound for the (1,2)Insertion in [7]. The last column, IT2, is the current best-possible upper bound that will be explained later. Here are some observations.

(1) What is most interesting and a bit surprising is that no cut position larger than 2 wins in Rand, and this is why OptRandGMS and Opt(1,2)GMS have the same data. Furthermore, in most n, the second best cut is still 1 or 2 (2 if the best is 1 and vice versa). Only a few cases where the second best is 4 exist: for instance, the estimated complexity (i.e., the values of $D^*[n]$) for cut 4 comes in the second place, next to cut 2, for $n = 12, 14, 21, 23, 25, 44, 46, 48$. We did not compute the real complexity (it takes much more time) for cut 4, but since the estimated values are considerably worse than those for cut 2, we guess that the exact values are also considerably worse. We can probably make a mathematical analysis to claim this interesting property by calculating the error bound of the random traversal, which should be a future research goal.

(2) Our main motivation of this paper is the numerical evaluation of (1,2)Insertion in [7]. Good news is that cut 1 and cut 2 are much more important than other cut positions, so the main idea of [7], adding cut 2, should have played an important role. It is often hard to discuss a difference of numerical data, the quick impression of IT1 data is not very bad.

(3) One demerit of random estimation is an accumulation of errors. As shown in Table 1, the cut position of Rand becomes different from that of Opt(1,2) at $n = 11$ for the first time and after that the Rand performance is always worse although the cut positions are the same.

(4) MergeInsertion [5] is surprisingly fast for some values of n. So, what if we combine this algorithm with Opt(1,2)GMS? Namely in each step we extend the selection from “cut 1 or cut 2” to “cut 1 or cut 2 or a direct execution of MergeInsertion.” The result, IT2, is extremely good, even better than the $T(n)$ lower bound (this is not a contradiction, since the new algorithm is not a GMS any longer). Note that we have no efficient evaluation algorithm for MergeInsertion, so we did it just by simulation using as many sample inputs as possible. We believe this specific algorithm is the current best one in terms of the average complexity of comparisons.

Table 1. First column is the input length $n \in \{2, \ldots, 35\}$, the second column the value of $C(n)$ (the information-theoretic lower bound), the third column the worst-case lower bound, the forth column the upper bound of Ford-Johnson, the fifth column the value of $T(n)$ (the average-case lower bound for any GMSs), the 6th column the cut position realizing that lower bound. From the 7th to the 10th columns, the upper bounds for our GMSs and its cut position: the 7th and the 8th columns for OPT(1,2), the 9th and the 10th columns for Rand. the 10th column (IT1) the upper bound for $(1,2)$*Insertion∗* [7] and the 11th column its cut position, The 12th column (IT2) the upper bound for the combined algorithm of MergeInsertion and $(1,2)$*Insertion∗* [7].

n	$C(n)$	$S(n)$	$F(n)$	$T(n)$	cut	$Opt(1,2)$	cut	*Rand*	cut	IT1	cut	IT2
2	1.000	1	1	1.000	1					1	-	
3	2.585	3	3	2.667	1	2.667	1	2.667	1	2.667	1	2.667
4	4.585	5	5	4.667	1	4.667	1	4.667	2	4.667	1	4.667
5	6.907	7	7	7.067	1	7.067	1	7.067	1	7.067	1	6.933
6	9.492	10	10	9.600	2	9.667	2	9.667	2	9.733	2	9.600
7	12.299	13	13	12.457	1	12.524	1	12.524	1	12.590	1	12.457
8	15.299	16	16	15.457	1	15.524	1	15.524	1	15.590	1	15.457
9	18.469	19	19	18.679	1	18.746	1	18.746	1	18.813	1	18.584
10	21.791	22	22	22.035	2	22.102	2	22.102	2	22.213	2	21.851
11	25.250	26	26	25.516	2	25.619	2	25.647	1	25.758	1	25.396
12	28.835	30 [18]	30	29.089	4	29.238	2	29.253	2	29.364	2	29.063
13	32.536	34 [12]	34	32.859	1	32.991	2	33.019	2	33.133	1	32.832
14	36.343	38 [13]	38	36.679	4	36.831	2	36.847	2	36.980	2	36.678
15	40.250	42 [14]	42	40.612	1	40.765	1	40.780	1	40.913	1	40.612
16	44.250	45/46	46	44.612	1	44.765	1	44.780	1	44.913	1	44.612
17	48.338	49/50	50	48.730	1	48.882	1	48.897	1	49.031	1	48.729
18	52.508	53/54	54	52.924	6	53.105	1	53.120	2	53.266	2	52.872
19	56.755	57/58	58	57.222	3	57.385	2	57.400	2	57.582	1	57.073
20	61.077	62	62	61.506	8	61.757	2	61.772	2	61.934	2	61.340
21	65.470	66	66	65.982	1	66.185	2	66.215	2	66.410	1	65.630
22	69.929	71 [13]	71	70.398	2	70.697	2	70.742	2	70.891	2	70.176
23	74.453	-	76	74.970	2	75.260	2	75.302	2	75.500	1	74.784
24	79.038	-	81	79.543	2	79.896	2	79.968	1	80.094	2	79.379
25	83.682	-	86	84.248	8	84.577	2	84.632	2	84.814	1	84.099
26	88.382	-	91	88.967	2	89.327	2	89.401	1	89.531	2	88.816
27	93.137	-	96	93.770	4	94.118	2	94.173	2	94.346	1	93.630
28	97.944	-	101	98.598	12	98.972	2	99.030	1	99.184	2	98.469
29	102.802	-	106	103.467	6	103.867	2	103.927	1	104.081	1	103.366
30	107.709	-	111	108.377	6	108.800	1	108.860	1	109.014	1	108.299
31	112.663	-	116	113.345	1	113.768	1	113.828	1	113.982	1	113.267
32	117.663	-	121	118.345	1	118.768	1	118.828	1	118.982	1	118.267
33	122.708	-	126	123.405	1	123.829	1	123.888	1	124.042	1	123.327
34	127.795	-	131	128.519	2	128.946	1	129.006	1	129.160	1	128.445

(*continued*)

Table 1. (*continued*)

n	$C(n)$	$S(n)$	$F(n)$	$T(n)$	cut	$Opt(1,2)$	cut	*Rand*	cut	IT1	cut	IT2
35	132.924	-	136	133.684	2	134.118	1	134.177	1	134.332	1	133.616
36	138.094	-	141	138.894	2	139.329	2	139.400	1	139.546	2	138.831
37	143.304	-	146	144.087	4	144.582	2	144.665	2	144.816	1	144.020
38	148.552	-	151	149.403	1	149.872	2	149.943	2	150.094	2	149.221
39	153.837	-	156	154.705	2	155.200	2	155.302	1	155.452	1	154.435
40	159.159	-	161	160.018	20	160.561	2	160.652	2	160.792	2	159.702
41	164.517	-	166	165.419	6	165.962	2	166.086	2	166.231	1	164.960
42	169.909	-	171	170.829	2	171.395	2	171.514	2	171.637	2	170.251
43	175.335	-	177	176.285	2	176.867	2	177.025	1	177.148	1	175.762
44	180.795	-	183	181.746	2	182.368	2	182.509	2	182.618	2	181.307
45	186.286	-	189	187.250	2	187.907	2	188.086	1	188.195	1	186.885
46	191.810	-	195	192.768	2	193.475	2	193.637	2	193.729	2	192.418
47	197.365	-	201	198.356	2	199.077	2	199.275	1	199.367	1	198.057
48	202.950	-	207	203.952	2	204.704	2	204.893	2	204.959	2	203.649
49	208.564	-	213	209.614	2	210.365	2	210.587	1	210.653	1	209.343
50	214.208	-	219	215.280	2	216.050	2	216.263	2	216.307	2	214.996
51	219.881	-	225	220.973	4	221.767	2	222.005	2	222.052	1	220.742
52	225.581	-	231	226.682	4	227.509	2	227.743	2	227.768	2	226.457
53	231.309	-	237	232.475	1	233.282	2	233.528	2	233.560	1	232.250
54	237.064	-	243	238.193	8	239.078	2	239.311	2	239.339	2	238.028
55	242.845	-	249	244.029	1	244.903	2	245.154	2	245.175	1	243.865
56	248.653	-	255	249.847	6	250.748	2	250.993	2	251.015	2	249.704
57	254.485	-	261	255.722	9	256.625	2	256.870	1	256.892	1	255.581
58	260.343	-	267	261.608	2	262.522	2	262.766	1	262.788	1	261.478
59	266.226	-	273	267.480	22	268.437	1	268.682	1	268.704	1	267.393
60	272.133	-	279	273.414	1	274.370	1	274.615	1	274.637	1	273.326
61	278.064	-	285	279.361	2	280.321	1	280.566	1	280.588	1	279.277
62	284.018	-	291	285.329	1	286.289	1	286.534	1	286.556	1	285.245
63	289.995	-	297	291.313	2	292.273	1	292.518	1	292.540	1	291.229
64	295.995	-	303	297.313	1	298.273	1	298.518	1	298.540	1	297.229
65	302.018	-	309	303.343	1	304.304	1	304.548	1	304.570	1	303.260

References

1. Ayala-Rincón, M., De Abreu, B.T., De Siqueira, J.: A variant of the Ford-Johnson algorithm that is more space efficient. Inf. Process. Lett. **102**(5), 201–207 (2007)
2. Brightwell, G., Winkler, P.: Counting linear extensions. Order **8**(3), 225–242 (1991)
3. Cooper, J.: Linear Extension Counting is Fixed-Parameter Tractable in Essential Width. AMS Southeastern Sectional Fall 2013. www.math.louisville.edu/~biro/ams/cooper.pdf

4. Edelkamp, S., Weiß, A.: QuickXsort: efficient sorting with $n \log n - 1.399n + o(n)$ comparisons on average. In: CSR 2014, pp. 139–152 (2014)
5. Ford, L.R., Johnson, S.M.: A tournament problem. Am. Math. Mon. **66**(5), 387–389 (1959)
6. Hwang, F.K., Lin, S.: Optimal merging of 2 elements with n elements. Acta Inform. **1**, 145–158 (1971)
7. Iwama, K., Teruyama, J.: Improved average complexity for comparison-based sorting. Algorithms and Data Structures. LNCS, vol. 10389, pp. 485–496. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62127-2_41
8. Knuth, D.E.: The Art of Computer Programming. Sorting and Searching, vol. 3, 2nd edn. Addison Wesley Longman Publishing Co., Inc., Redwood City (1998)
9. Manacher, G.K.: The Ford-Johnson algorithm is not optimal. J. Assoc. Comput. Mach. **26**, 441–456 (1979)
10. Manacher, G.K., Bui, T.D., Mai, T.: Optimum combinations of sorting and merging. J. Assoc. Comput. Mach. **36**, 290–334 (1989)
11. Mönting, J.S.: Merging of 4 or 5 elements with n elements. Theor. Comput. Sci. **14**, 19–37 (1981)
12. Peczarski, M.: Sorting 13 elements requires 34 comparisons. In: Möhring, R., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 785–794. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45749-6_68
13. Peczarski, M.: New results in minimum-comparison sorting. Algorithmica **40**(2), 133–145 (2004)
14. Peczarski, M.: The Ford-Johnson algorithm still unbeaten for less than 47 elements. Inf. Process. Lett. **101**(3), 126–128 (2007)
15. Peczarski, M.: Towards Optimal Sorting of 16 Elements. arXiv preprint arXiv:1108.0866 (2011)
16. Steinhaus, H.: Mathematical Snapshots, pp. 37–40, New Nork (1950)
17. Thanh, M., Alagar, V.S., Bui, T.D.: Optimal expected-time algorithms for merging. J. Algorithms **7**(3), 341–357 (1986)
18. Wells, M.: Applications of a language for computing in combinatorics. In: Proceedings of 1965 IFIP Congress, North-Holland, Amsterdam, pp. 497–498 (1966)

Online Matching in Regular Bipartite Graphs with Randomized Adversary

Josep Fàbrega and Xavier Muñoz(✉)

Mathematics Department, Universitat Politècnica de Catalunya, Barcelona, Spain
{josep.fabrega,xavier.munoz}@upc.edu

Abstract. This paper deals with the problem of finding matchings in bipartite regular graphs in the online model with randomized adversary. We determine the competitive ratio for 2-regular graphs for any number of random decisions made by the adversary. We also give upper and lower bounds for other values of the degree.

1 Introduction

An online algorithm solves a computational problem piecewise. Its input arrives as a sequence and each piece of the sequence is a request. The algorithm is required to compute a partial output after each time step, before the next request arrives, i.e., the partial output only depends on the previous requests. This partial output may not be changed. Commonly, competitive analysis is used for analyzing the output quality of online algorithms.

1.1 Different Models for Online Algorithms Used in This Paper

Competitive analysis was introduced in 1985 by Sleator and Tarjan [17] and is a powerful tool to measure the performance of online algorithms. For an overview on competitive analysis and online algorithms, we refer to the standard literature [3,8,10,13].

Let us consider an online maximization problem $\mathtt{p}$, and a deterministic algorithm $\mathtt{A}$ that solves that problem. Let $\mathtt{I}$ be an adversarial instance for the problem $\mathtt{p}$ presented piecewise to $\mathtt{A}$. We name $\mathrm{Cost}_{\mathtt{p}}(\mathtt{A},\mathtt{I})$ the cost of the solution computed by $\mathtt{A}$ on the instance $\mathtt{I}$ of $\mathtt{p}$. We measure the performance of $\mathtt{A}$ on $\mathtt{I}$ comparing its cost to the cost of an algorithm performing optimally on $\mathtt{I}$. We name such an algorithm $\mathtt{Opt}$ and the cost of its solution $\mathrm{Cost}_{\mathtt{p}}(\mathtt{Opt},\mathtt{I})$. The *strict competitive ratio* of an algorithm $\mathtt{A}$ on the instance $\mathtt{I}$ is then

$$\mathbf{r}_{\mathtt{p}}(\mathtt{A},\mathtt{I}) = \frac{\mathrm{Cost}_{\mathtt{p}}(\mathtt{Opt},\mathtt{I})}{\mathrm{Cost}_{\mathtt{p}}(\mathtt{A},\mathtt{I})}.$$

Observe that, as $\mathtt{p}$ is a maximization problem, the cost of $\mathtt{A}$ is always going to be smaller than the cost of $\mathtt{Opt}$ on any instance. This means that $\mathbf{r}_{\mathtt{p}}(\mathtt{A},\mathtt{I}) \geq 1$.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 297–310, 2018.
https://doi.org/10.1007/978-3-319-98355-4_17

Now, the competitive ratio of an algorithm $\mathtt{A}$ for any possible instance $\mathtt{I} \in \mathcal{I}$ is defined in a worst-case manner as

$$\mathbf{r}_{\mathtt{p}}(\mathtt{A}) = \sup_{\mathtt{I} \in \mathcal{I}}(\mathbf{r}_{\mathtt{p}}(\mathtt{A}, \mathtt{I})).$$

Finally, we define the competitive ratio of a problem $\mathtt{p}$ as the best case, i.e.,

$$\mathbf{r}_{\mathtt{p}} = \inf_{\mathtt{A} \in \mathcal{A}}(\mathbf{r}_{\mathtt{p}}(\mathtt{A}))$$

where $\mathcal{A}$ is the set of deterministic online algorithms solving problem $\mathtt{p}$.

Let $\mathtt{RA}$ be a randomized online algorithm that solves $\mathtt{p}$. It can be modeled as a set $\mathtt{RA} = \{\mathtt{A}_1, \mathtt{A}_2, \ldots, \mathtt{A}_k\}$ of deterministic algorithms one of which will be picked uniformly at random. The competitive ratio of a randomized algorithm $\mathtt{RA}$ on the instance $\mathtt{I}$ is defined as

$$\bar{\mathbf{r}}_{\mathtt{p}}(\mathtt{RA},\mathtt{I}) = \frac{1}{\mathbb{E}(\frac{1}{\mathbf{r}_{\mathtt{p}}(\mathtt{RA},\mathtt{I})})},$$

and the competitive ratio of a randomized algorithm on $\mathtt{p}$ is defined analogously as

$$\bar{\mathbf{r}}_{\mathtt{p}}(\mathtt{RA}) = \sup_{\mathtt{I} \in \mathcal{I}}\{\bar{\mathbf{r}}_{\mathtt{p}}(\mathtt{RA}, \mathtt{I})\}$$

and

$$\bar{\mathbf{r}}_{\mathtt{p}} = \inf_{\mathtt{RA} \in \mathcal{RA}}\{\bar{\mathbf{r}}_{\mathtt{p}}(\mathtt{RA})\},$$

respectively.

The definition of the competitive ratio for a randomized algorithm may sound a little odd. The reason of such a definition is that the competitive ratio is typically defined for minimization problems instead of maximization problems. In case of minimization the competitive ratio is defined as $\mathbf{r}_{\mathtt{p}}(\mathtt{A}, \mathtt{I}) = \mathrm{Cost}_{\mathtt{p}}(\mathtt{A}, \mathtt{I})/\mathrm{Cost}_{\mathtt{p}}(\mathtt{Opt}, \mathtt{I})$ (i.e., the inverse of the competitive ratio for maximization). In order to have similar behaviors in minimization and maximization problems with randomization, it is more convenient to use that definition.

The randomized adversary model we use here was introduced by Hromkovič (see [18]) and further investigated by Burjons et al. [4] and Dobrev et al. [7].

In this model, the adversary has access to s random bits, and uses them to pick uniformly at random an instance from the set $\mathtt{RI} = \{\mathtt{I}_1, \mathtt{I}_2, \ldots, \mathtt{I}_{2^s}\}$ for the problem $\mathtt{p}$. The online algorithm $\mathtt{A}$ is deterministic and knows in advance all the information from the adversary by means of an oracle, except the outcome of the random bits used by the adversary. This means that the algorithm knows the full set of instances $\mathtt{RI}$ but does not know which of the instances will be chosen by the adversary. The competitive ratio of $\mathtt{A}$ with a randomized adversary is defined as

$$\mathbf{r}^*_{\mathtt{p}}(\mathtt{A}, \mathtt{RI}) = \frac{1}{\mathbb{E}(\frac{1}{\mathbf{r}_{\mathtt{p}}(\mathtt{A},\mathtt{RI})})} = \frac{2^s}{\sum_i \frac{1}{\mathbf{r}_{\mathtt{p}}(\mathtt{A},\mathtt{I}_i)}}.$$

Analogously to the previous models, the competitive ratio for a specific algorithm $\mathtt{A}$ in the class of instance sets $\mathcal{RI}_s$ using no more than s random bits and the competitive ratio for the problem p in the same set of instances can be defined as

$$\mathbf{r}^*_{\mathtt{p}}(\mathtt{A}, s) = \sup_{\mathtt{RI} \in \mathcal{RI}_s} \{\mathbf{r}^*_{\mathtt{p}}(\mathtt{A}, \mathtt{RI})\}$$

and

$$\mathbf{r}^*_{\mathtt{p}}(s) = \inf_{\mathtt{A} \in \mathcal{A}} \{\mathbf{r}^*_{\mathtt{p}}(\mathtt{A}, s)\},$$

respectively.

Using the introduced notation, Yao's principle for online algorithms reads as follows:

Theorem 1 (Yao [19]). *Let* p *be an online maximization problem. Then, for any* $s \in \mathbb{N}$,

$$\mathbf{r}^*_{\mathtt{p}}(s) \leq \bar{\mathbf{r}}_{\mathtt{p}}.$$

This theorem states that the competitive ratio of a problem p solved using any randomized algorithm against instances of finite length is bounded from below by the competitive ratio of a problem with a randomized adversary using any finite number of random bits s to select amongst finite instances.

In order to simplify the expressions and the formulae throughout the paper, let us define the *inverse competitive ratio* ρ as

$$\rho_{\mathtt{p}}(\mathtt{A}, \mathtt{I}) = \frac{1}{\mathbf{r}_{\mathtt{p}}(\mathtt{A}, \mathtt{I})}.$$

It is easy to check that, with this parameter,

$$\rho^*_{\mathtt{p}}(\mathtt{A}, \mathtt{RI}) = \mathbb{E}(\rho_{\mathtt{p}}(\mathtt{A}, \mathtt{RI})) = \frac{\sum_i \rho_{\mathtt{p}}(\mathtt{A}, \mathtt{I}_i)}{2^s}.$$

1.2 Online Matching in Regular Graphs with Randomized Adversary

In the online bipartite matching problem, we are given a bipartite graph, $G = (U \cup W, E)$. The online algorithm processes the input as follows. The vertices in U are *known*; the vertices in W arrive *one at a time*, together with their *list of adjacent vertices*, and each edge must be matched (or not matched) exactly at the time it arrives. The goal is to obtain a matching as close as possible to the optimum one.

First results on this problem are due to the seminal paper by Karp, Vazirani, and Vazirani [12]. Since then, several variants of the problem have been studied, some of them motivated by the following applications: adwords [14,16], matching in metric space [15], weighted matching in metric spaces [11], matching on the real line [9].

The competitive ratio of deterministic algorithms for the online problem is 2 [12], as any algorithm that always matches a vertex if possible constructs a maximal matching. Also, given any deterministic algorithm, it is easy to construct an input that forces the algorithm to find a matching of size no more than half of the optimum. It is also proven in [12] that choosing an edge uniformly at random every time a vertex arrives does not provide significant improvement. Indeed, the expected size of the matching is bounded by $\frac{n}{2} + O(\log n)$.

Several randomized algorithms for online bipartite matching have been proposed. The first known is an algorithm called RANKING [12]. It initially ranks the known vertices, so that, each time a new vertex arrives, it is matched to the highest ranked available vertex that is adjacent to it. The competitive ratio is proven to be $1 + \frac{1}{e-1}$. In [2], the authors revisit the proof and give a simpler one.

The online matching problem for the case of regular bipartite graphs has recently been studied in the deterministic model [1,16] and also in the advice complexity model [1].

In this paper, we deal with the online bipartite matching problem in regular graphs under the randomized adversary model.

Recall that any regular bipartite graph, $G(U \cup W, E)$, has a perfect matching (see [6, Chap. 2]), and therefore an optimum matching will contain $n = |U| = |W|$ edges.

Along this paper, in order to count the size of a matching, we may sometimes refer to a *matched vertex* $u \in U$ standing for a matched edge incident to u.

2 Online Bipartite Matchings in 2-Regular Bipartite Graphs

In this section, we determine the competitive ratio for the online matching problem for the case of 2-regular graphs with any number of random bits used by the adversary.

Proposition 1. *Let μ_2 be the problem of finding a matching in a 2-regular bipartite graph. Then, for any number of random bits s, $\mathbf{r}^*_{\mu_2}(s) \geq 8/7$.*

Proof. Take first $s = 1$ and consider the set $\mathcal{RI}$ constituted by the following two bipartite graphs with eight vertices (see Fig. 1), $G_1 = (U \cup W, E_1)$ and $G_2 = (U \cup W, E_2)$, where $U = \{u_1, u_2, u_3, u_4\}$, $W = \{w_1, w_2, w_3, w_4\}$,

$$E_1 = \{u_1w_1, u_1w_3, u_2w_1, u_2w_4, u_3w_2, u_3w_3, u_4w_2, u_4w_4\}$$

and

$$E_2 = \{u_1w_1, u_1w_3, u_2w_1, u_2w_4, u_3w_2, u_3w_4, u_4w_2, u_4w_3\}.$$

The adversary selects at random one of the two graphs and presents the vertices of W in the order w_1, w_2, w_3, w_4. Notice that, before presenting vertex w_3, both instances are indistinguishable.

It is easy to check that, for any choice of the algorithm in the first two rounds, there will always be exactly one round, either in G_1 or in G_2, in which no edge can be selected. Therefore, the competitive ratio of the algorithm is 4/3 for one of the instances and 1 for the other. This proves that, for any online algorithm A,

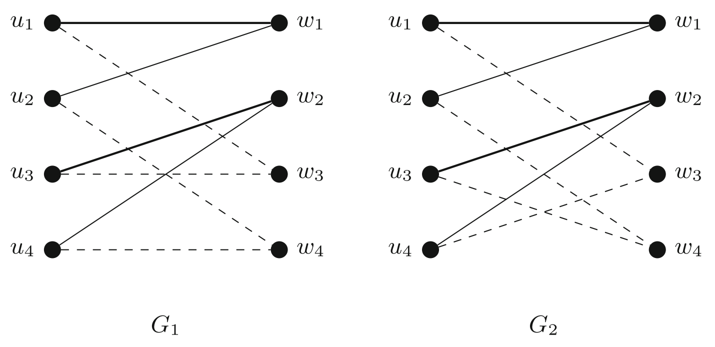


Fig. 1. The two adversarial instances in the set $\mathcal{RI}$ in the lower bound for $s = 1$ with the two first edges selected.

$$\mathbf{r}^*_{\mu_2}(\mathtt{A}, s = 1) \geq \mathbf{r}^*_{\mu_2}(s = 1) = \frac{2}{3/4 + 1} = \frac{8}{7},$$

as claimed.

For a general value of the number of random bits, $s \in \mathbb{N}$, let $\mathcal{RI}'$ be a set of size 2^s from which the adversary chooses the instance to be presented, and assume that $\mathcal{RI}'$ is formed by 2^{s-1} copies of G_1 and 2^{s-1} copies of G_2. Then,

$$\mathbf{r}^*_{\mu_2}(\mathtt{A}, s) \geq \mathbf{r}^*_{\mu_2}(s = 1) \geq \frac{8}{7},$$

which finishes the proof. □

Proposition 2. *The competitive ratio for the bipartite matching problem for 2-regular graphs, μ_2, in the randomized algorithm model is $\bar{\mathbf{r}}_{\mu_2} \leq \frac{8}{7}$.*

Proof. Let us consider an instance $\mathtt{I}(G)$ presented by the adversary, with $G = (U \cup W, E)$. Every new vertex $w \in W$ shown by the adversary appears together with two edges u_1w and u_2w and the algorithm selects one of them for the matching whenever it is possible. We can assume that the algorithm is greedy, since not choosing an edge when it is possible will never produce a better solution than choosing a wrong edge (a wrongly chosen edge will produce at most one error).

To make the proof clearer, we transform the problem as follows: The instance $\mathtt{I}(G)$ will be represented by $\mathtt{I}'(G')$ with $G' = (U, E')$. Every new vertex $w \in W$ appearing in the original problem together with edges $u_1w, u_2w \in E(G)$ will be represented by a new edge $u_1u_2 \in E'(G')$. It is clear that, if G is a 2-regular bipartite graph, G' will be a set of disjoint cycles.

The algorithm choosing one of the edges $u_1w, u_2w \in E(G)$ when vertex $w \in G$ appears is translated to the assignment of a direction to the edge $u_1u_2 \in E'(G')$. The new arc will be adjacent to $u_1 \in U(G')$ if the selected edge in the original problem is $u_1w \in E(G)$ (see Fig. 2). The algorithm should assign a direction to every new edge, in such a way that the newly built directed graph maintains an in-degree not larger than 1.

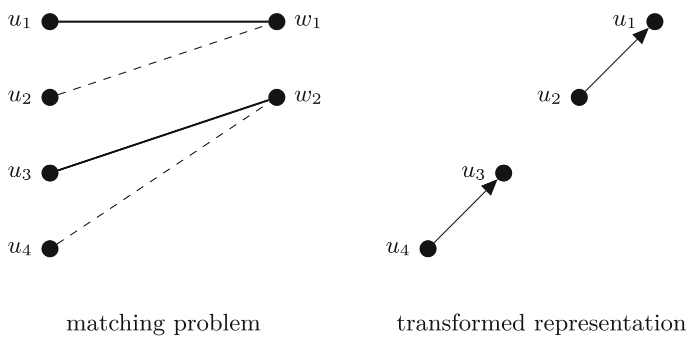


Fig. 2. Translation of the matching problem for $\delta = 2$

Notice that, if the adversary shows a new edge incident to an already shown path, the algorithm can always assign to that edge the direction of the path and no error will be produced. Therefore, we can assume that the adversary will always present disjoint edges as long as it is possible, and this will be the worst-case adversary.

We can further assume (for the sake of simplicity in the analysis) that the adversary will consist of two phases: In the first one, a set of disjoint edges will be presented. In the second phase, the edges presented will be incident to vertices that have already appeared in the first phase, closing cycles of even length, say $2m$.

Any randomized algorithm RA will choose at random the directions of the edges in the first phase. In the second phase, no decision has to be taken: If an edge u_1u_2 appears in the second phase, either u_1 and u_2 have already in-degree 1, and no direction can be assigned (producing an error), or at least one of them has in-degree 0, and it is possible to assign the direction towards that vertex.

Therefore, all mistakes are produced during the random choices in the first phase. This problem can be translated again into the following game: Given a cycle C_m, let the algorithm choose at random the direction of each edge in the cycle. The number of errors is the number of vertices with in-degree 2. That number of errors is equal to the number of edges missing in the matching in a bipartite graph with $2m$ vertices in each vertex set.

We claim that the expected number of vertices with in-degree 2 in a digraph obtained by orienting uniformly at random the edges of the cycle C_m is m/4: Indeed, let $v_1, \ldots, v_m$ be the vertices of C_m and let N be the number of vertices with in-degree 2 in a random orientation of its edges. Then $N = \sum_{i=1}^{m} I_i$, where I_i is the indicator random variable of the event $\delta^+(v_i) = 2$. Clearly, $\mathbb{E}(I_i) = \mathbb{P}(I_i = 1) = 1/4$. Hence, $\mathbb{E}(N) = \sum_{i=1}^{m} \mathbb{E}(I_i) = m/4$, as claimed.

The back-translation of this result into the original matching problem states that the number of missed edges in expectation in the matching is m/4 out of $2m$ possible, and therefore $\bar{\rho}_{\mu_2}(\texttt{RA}) = \frac{2m-m/4}{2m} = \frac{7}{8}$. Finally, $\bar{\mathbf{r}}_{\mu_2} \leq \frac{1}{\bar{\rho}_{\mu_2}(\texttt{RA})} = \frac{8}{7}$. □

As a consequence of Propositions 1 and 2 and Theorem 1, we conclude:

Theorem 2. *The competitive ratio for the bipartite matching problem μ_2 for 2-regular graphs with randomized adversary for any number of random bits s is given by $\mathbf{r}^*_{\mu_2}(s) = 8/7$.*

3 Adversary with One Random Bit for Other Values of δ

3.1 Lower Bounds on the Competitive Ratio

Proposition 3. *Let μ_3 be the problem of finding a matching in a 3-regular bipartite graph with a randomized adversary. Then $\mathbf{r}^*_{\mu_3}(s = 1) \geq 18/17$.*

Proof. We will prove that there exists an adversary using one random bit for which, for any online algorithm, $\mathbf{r}^*_{\mu_3}(\mathtt{A}, s = 1) \geq 18/17$. In order to define such an adversary, let us use a result borrowed from the theory of combinatorial designs. A *balanced incomplete block design* (BIBD for short) is a pair $(V, \mathcal{B})$ where V is a v-set of points, and $\mathcal{B}$ is a collection of b k-subsets of V (blocks) such that each element of V is contained in exactly r blocks and any 2-subset of V is contained in exactly λ blocks. The numbers v, b, r, k, and λ are parameters of the BIBD. A *parallel class* or *resolution class* in a design is a set of blocks that partition the point set. A BIBD is said to be *resolvable* if its blocks can be partitioned into parallel classes. The notation RBIBD(v, k, λ) is commonly used for a resolvable balanced incomplete block design (see [5] for more information on resolvable designs).

Let us consider the resolvable design RBIBD$(9, 3, 1)$ given by

$$\begin{array}{cccc}
I & II & III & IV \\
\{1,2,3\} & \{1,4,7\} & \{1,5,9\} & \{1,6,8\} \\
\{4,5,6\} & \{2,5,8\} & \{2,6,7\} & \{2,4,9\} \\
\{7,8,9\} & \{3,6,9\} & \{3,4,8\} & \{3,5,7\}.
\end{array}$$

Observe that each column is a parallel class, i.e., a partition of the set $\{1, \ldots, 9\}$.

Let the adversary present one instance out of two possible bipartite graphs $G_0 = (U \cup W, E_0)$ and $G_1 = (U \cup W, E_1)$, with $U = \{1, \ldots, 9\}$ and $W = \{w_1, \ldots, w_9\}$. The edge sets E_0 and E_1 are given by the blocks in the design shown above as follows: Let the blocks in the parallel classes I and II be the neighbors of vertices $w_1, \ldots, w_6$, respectively, in both graphs, and let the neighbors of vertices w_7, w_8 and w_9 be the blocks in class III in G_0, and the blocks in class IV in G_1. Let the adversary present the vertices $w_1, \ldots, w_9$ in increasing order.

In order to analyze the performance of this algorithm, let us consider two phases:

- **Phase I.** During this phase, vertices $w_1, \ldots, w_6$ are presented. At the end of this phase, 6 edges incident to different vertices in U have been selected for the matching. Notice that both instances are indistinguishable at this point.

– **Phase II.** During this phase, the adversary presents vertices w_7, w_8 and w_9. At this point, the algorithm knows which of both instances was selected by the adversary, and the algorithm should select three edges incident to the three vertices in U not used during the first phase. Let U_m be the set of those three edges.

Let us show that it is not possible to match all vertices in U_m during the second phase in both instances. Let us recall that U_m cannot be a block in classes I or II (if it were, no edge would have been selected from that block). Therefore there are at least two vertices in U_m that are not in any block in classes I or II. Because of the properties of block designs, every pair of elements belongs to exactly λ blocks (in our case $\lambda = 1$), and therefore these two vertices belong to the same block either in class III or IV. Due to the pigeonhole principle, it will not be possible to select all three vertices in the instance corresponding to the class where the pair is in the same block. Therefore, at least one edge is missed in one of the instances. The bound for the inverse competitive ratio follows straightforwardly. □

Notice that this adversary is a generalization of the adversary shown for $\delta = 2$ (the adversary in the proof for $\delta = 2$ is actually an RBIBD$(4, 2, 1)$).

The next proposition shows a generalization of the adversary for $\delta = 2$ to adversaries for any even degree.

Proposition 4. *Let $\mu_{2\delta}$ be the problem of finding a matching in a 2δ-regular bipartite graph. Then $\mathbf{r}^*_{\mu_{2\delta}}(s = 1) \geq \frac{8\delta}{8\delta-1}$.*

Proof. Let us consider the graphs G_1 and G_2 used in the case $\delta = 2$ (Fig. 1) and let us construct the Cartesian product $G'_i = G_i \times \{1, \ldots, \delta\}$, i.e., $V(G'_i) = \{(v, j) \mid v \in G_i \text{ and } j \in \{1 \ldots \delta\}\}$ and (v_a, j_1) is adjacent to (v_b, j_2) in G'_i whenever $v_a v_b$ is an edge in G_i. Two instances are given by presenting the vertices of the graphs G'_1 and G'_2 in the order

$$(w_1, 1), \ldots, (w_1, \delta), (w_2, 1), \ldots, (w_2, \delta), (w_3, 1), \ldots, (w_3, \delta), (w_4, 1), \ldots, (w_4, \delta)\,.$$

It is a simple exercise to check that the claimed bound holds by using the same reasoning as in the proof of Proposition 1. □

3.2 Upper Bounds on the Competitive Ratio

Proposition 5. *Let μ_δ be the problem of finding a matching in a δ-regular bipartite graph with a randomized adversary using a single random bit. Then $\mathbf{r}^*_{\mu_\delta}(s = 1) \leq \frac{4\delta}{3\delta+1}$.*

Proof. We prove that, for any adversary using one random bit, there exists an online algorithm $\mathtt{A}$ solving μ_δ for a δ-regular bipartite graph, for which the inverse competitive ratio satisfies $\rho_{\mathtt{A}}(s = 1) \geq \frac{3\delta+1}{4\delta}$.

Since the adversary is using one random bit, there are two possible instances I_0, I_1 that can each be presented with probability $\frac{1}{2}$. Let the algorithm start

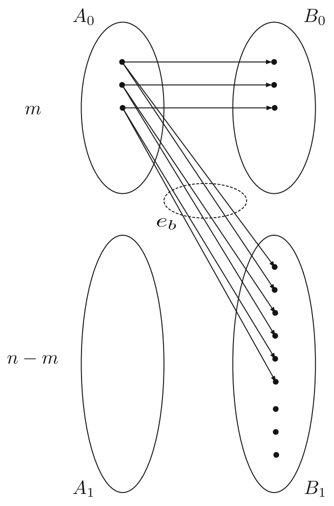


Fig. 3. Counting the number of edges when an error is detected

taking edges in the matching assuming it is one of them (let us suppose it is I_0). If the presented instance is I_0, then the algorithm will achieve a perfect matching. If the presented instance is I_1, let us assume that the algorithm will realize the wrong choice after m vertices have appeared. Since no error was detected before, it can be guaranteed that the algorithm will have selected m edges in the matching at this point.

Let $G_1 = (U \cup W, E)$ be the graph presented by the adversary in instance I_1. Let $A_0 \subset W$ be the subset of vertices in W that have been shown at time step m, and let $B_0 \subset U$ the subset of vertices in U that have been chosen in the matching. $A_1 \subset W$ and $B_1 \subset U$ contain the rest of the vertices of the graph. Let e_b be the number of edges that have already appeared between sets A_0 and B_1. This situation is illustrated in Fig. 3.

Let us point out the following facts:

- $e_b \leq (n-m)(\delta-1)$, since the vertices of B_1 have degree δ, but there is at least one edge per vertex which is not incident to A_0. Otherwise an error would have been detected before.
- $e_b \leq (\delta-1)m$, since the number of edges going from A_0 to B_0 is at least m.
- The number of edges going from A_1 to B_0 is also e_b.

At this point, the algorithm knows exactly what is the graph that will be presented by the adversary, and therefore the algorithm can proceed as in an off-line model (since the algorithm discovered that the instance was not I_0, and then it must be I_1).

Let h be the number of edges that the algorithm may miss to put in the matching. It is easy to see that $h \leq e_b/\delta$: Take the subgraph induced by the vertex sets A_1 and B_1 and add e_b edges so that it becomes δ-regular. Since it is a δ-regular bipartite graph, it can be edge-colored with δ colors. This coloring is equivalent to the partition of the edge set into δ perfect matchings. Therefore, at least one of those matchings will contain at most e_b/δ edges among the added ones.

Thus, $h \leq \frac{e_b}{\delta} \leq \min\{\frac{\delta-1}{\delta}m, \frac{\delta-1}{\delta}(n-m)\}$. The maximum value of that minimum is obtained when $m = \frac{n}{2}$. For that value of m, we have $h = n - m$, and therefore the number of edges in the matching is $n - h$.

Finally, the inverse competitive ratio in expectation is $\rho \geq \frac{1+\frac{\delta+1}{2\delta}}{2} = \frac{3\delta+1}{4\delta}$. $\square$

4 Adversaries with a Large Number of Random Bits

Let us consider the set $\mathtt{RI}_{k\delta}$ formed by the instances $\mathtt{I}_i(G_i)$ in which $G_i = (U \cup W, E)$ is a δ-regular bipartite graph with $n = |U| = |W| = k\delta$. For any instance $\mathtt{I}_i(G_i)$, let the adversary present the vertices $w \in W$ in δ phases. In phase i, $1 \leq i \leq \delta$, a subset of k vertices F_i, $F_i \subset W$, is presented in such a way that $\Gamma(w) \cap \Gamma(w') = \emptyset$, for any two distinct vertices $w, w' \in F_i$, where $\Gamma(v)$ stands for the set of neighbors of vertex v.

In this section, we want to compute the competitive ratio $\mathbf{r}^*_{\mathtt{RI}_{k\delta}}$ for the randomized adversary given by the set $\mathtt{RI}_{k\delta}$. Notice that, with such instance set, all deterministic greedy algorithms will have the same behavior, since at each time step all edges will look the same and any deterministic choice will have the same probability of being a good choice.

Theorem 3. *The competitive ratio for the randomized adversary given by the set* $\mathtt{RI}_{k\delta}$ *for large values of* k *is* $\mathbf{r}^*_{\mathtt{RI}_{k\delta}} = \frac{\delta}{\delta - g(\delta)}$ *with* $g(\delta) = \sum_{i=1}^{\delta} \left(\frac{i-1}{\delta}\right)^{\delta}$.

Proof. Let $G = (U \cup W, E)$ be the graph presented by the adversary. Let us recall that G is a δ-regular bipartite graph with $n = |U| = |W| = k\delta$. Let the adversary present the vertices $w \in W$ in δ phases. In phase i, $1 \leq i \leq \delta$, a subset of k vertices F_i, $F_i \subset W$, is presented in such a way that $\Gamma(w) \cap \Gamma(w') = \emptyset$, for any two distinct vertices $w, w' \in F_i$.

For every $w \in F_i$, the algorithm tries to match w with a vertex of $\Gamma(w)$. From the point of view of the algorithm the vertices of F_i are selected at random, and each vertex $w \in F_i$ with at least one available neighbor in $\Gamma(w)$ will produce a new matched vertex in U that will not be used in the following phases.

Let us say that a vertex of U is marked if it has already been used by the algorithm to construct the matching. In phase i the algorithm will mark $k - N_i$ vertices of U, with N_i being the random variable that counts the number of vertices $w \in F_i$ such that all the vertices of $\Gamma(w)$ are already marked. Notice that $N_1 = 0$ because k vertices are always marked in the first phase. Moreover, since the total number of vertices of U that have been marked previous to phase i is $\sum_{j=1}^{i-1}(k - N_j)$, $1 \leq i \leq \delta$, we have

$$N_i \leq \left\lfloor \frac{\sum_{j=1}^{i-1}(k - N_j)}{\delta} \right\rfloor . \tag{1}$$

Moreover, $N_i = \sum_{w \in F_i} I_w$, where I_w is the indicator random variable of the event that all the vertices of $\Gamma(w)$ have been previously marked. If $M_i = \sum_{j=1}^{i-1} N_j$, $1 \le i \le \delta$, then

$$\mathbb{P}\left(I_w = 1 \mid M_i = m_i\right) = \frac{\binom{(i-1)k-m_i}{\delta}}{\binom{k\delta}{\delta}}. \tag{2}$$

(For $i = 1$ one has $M_1 = 0$ and $\mathbb{P}\left(I_w = 1\right) = 0$.) Therefore, the expected number of edges that will be missed during phase i, conditioned by that during previous phases (1 to $i-1$) the number of missed edges is m_i, is given by

$$\begin{aligned}\mathbb{E}\left(N_i \mid M_i = m_i\right) &= \sum_{w \in F_i} \mathbb{E}\left(I_w \mid M_i = m_i\right) \\ &= \sum_{w \in F_i} \mathbb{P}\left(I_w = 1 \mid M_i = m_i\right) = k\, \frac{\binom{(i-1)k-m_i}{\delta}}{\binom{k\delta}{\delta}}. \end{aligned} \tag{3}$$

Notice that, for $2 \le i \le \delta$, the binomial coefficient $\binom{(i-1)k-m_i}{\delta}$ will give a positive contribution only if $m_i \le (i-1)k - \delta$, and that, for large values of k, the probability $\mathbb{P}\left(I_w = 1 \mid M_i = m_i\right)$ is asymptotically $\left((i-1)/\delta\right)^{\delta}$, because

$$\frac{\binom{(i-1)k-m_i}{\delta}}{\binom{k\delta}{\delta}} = \prod_{j=1}^{\delta} \frac{(i-1)k - m_i - j + 1}{k\delta - j + 1} = \left(\frac{i-1}{\delta}\right)^{\delta} \cdot \prod_{j=1}^{\delta} \frac{1 - \frac{m_i + j - 1}{(i-1)k}}{1 - \frac{j-1}{k\delta}}. \tag{4}$$

When the δ phases of the algorithm have been accomplished, the total number of matched vertices is $\sum_{i=1}^{\delta}(k - N_i)$ and the inverse competitive ratio can be expressed as

$$\rho = \frac{\sum_{i=1}^{\delta}(k - N_i)}{k\delta} = 1 - \frac{1}{k\delta}\sum_{i=1}^{\delta} N_i.$$

The inverse competitive ratio ρ is a random variable and we are interested in estimating its expected value

$$\mathbb{E}(\rho) = 1 - \frac{1}{k\delta}\sum_{i=1}^{\delta} \mathbb{E}(N_i).$$

A more detailed analysis of expressions (2) to (4) allows us to write (for $2 \le i \le \delta$)

$$\left(1 - \frac{m_i + \delta - 1}{(i-1)k}\right)^{\delta} \le \frac{\mathbb{P}\left(I_w = 1 \mid M_i = m_i\right)}{((i-1)/\delta)^{\delta}} \le \left(\frac{1 - m_i/((i-1)k)}{1 - (\delta-1)/(k\delta)}\right)^{\delta}$$

and

$$\left(1 - \frac{m_i + \delta - 1}{(i-1)k}\right)^{\delta} \le \frac{\mathbb{E}\left(N_i \mid M_i = m_i\right)}{k((i-1)/\delta)^{\delta}} \le \left(\frac{1 - m_i/((i-1)k)}{1 - (\delta-1)/(k\delta)}\right)^{\delta}.$$

Hence, the expression

$$\frac{1}{k\delta}\sum_{i=1}^{\delta}\mathbb{E}\left(N_i \mid M_1=0, M_2=m_2,\ldots M_\delta=m_\delta\right)=\frac{1}{k\delta}\sum_{i=1}^{\delta}\mathbb{E}\left(N_i \mid M_i=m_i\right)$$

is bounded from below by

$$a(k,\delta)=\frac{1}{\delta}\sum_{i=2}^{\delta}\left(\frac{i-1}{\delta}\right)^{\delta}\left(1-\frac{m_i+\delta-1}{(i-1)k}\right)^{\delta}$$

and bounded from above by

$$b(k,\delta)=\frac{1}{\delta}\sum_{i=2}^{\delta}\left(\frac{i-1}{\delta}\right)^{d}\left(\frac{1-m_i/((i-1)k)}{1-(\delta-1)/(k\delta)}\right)^{\delta}.$$

The expected inverse competitive ratio can be calculated as

$$\mathbb{E}\left(\rho\right)=1-\frac{1}{k\delta}\sum_{i=1}^{\delta}\mathbb{E}\left(N_i\right)=1-\mathbb{E}\left(\frac{1}{k\delta}\sum_{i=1}^{\delta}\mathbb{E}\left(N_i \mid M_i\right)\right).$$

Therefore,

$$l(k,\delta)\le\mathbb{E}\left(\rho\right)\le u(k,\delta)$$

where

$$l(k,\delta)=1-\frac{1}{\delta}\sum_{i=2}^{\delta}\left(\frac{i-1}{\delta}\right)^{\delta}\mathbb{E}\left(\left(\frac{1-M_i/((i-1)k)}{1-(\delta-1)/(k\delta)}\right)^{\delta}\right)$$

and

$$u(k,\delta)=1-\frac{1}{\delta}\sum_{i=2}^{\delta}\left(\frac{i-1}{\delta}\right)^{\delta}\mathbb{E}\left(\left(1-\frac{M_i+\delta-1}{(i-1)k}\right)^{\delta}\right).$$

From (1), we deduce that

$$M_i=\sum_{j=1}^{i-1}N_j\le\sum_{j=1}^{i-1}\frac{(j-1)k}{\delta}=\frac{(i-1)(i-2)}{2\delta}.$$

Therefore, $u(k,\delta)\le u'(k,\delta)$ where

$$u'(k,\delta)=1-\frac{1}{\delta}\sum_{i=2}^{\delta}\left(\frac{i-1}{\delta}\right)^{\delta}\left(1-\frac{(i-1)(i-2)/2\delta+\delta-1}{(i-1)k}\right)^{\delta}$$

and

$$u'(k,\delta)\longrightarrow 1-\frac{1}{\delta}\sum_{i=1}^{\delta}\left(\frac{i-1}{\delta}\right)^{\delta}\text{ as } k\longrightarrow\infty.$$

Similarly, since $M_i \geq 0$, we have $l(k,\delta) \geq l'(k,\delta)$ where

$$l'(k,\delta) = 1 - \frac{1}{\delta}\sum_{i=2}^{\delta}\left(\frac{i-1}{\delta}\right)^{\delta}\left(\frac{1}{1-(\delta-1)/(k\delta)}\right)^{\delta},$$

and, also,

$$l'(k,\delta) \longrightarrow 1 - \frac{1}{\delta}\sum_{i=1}^{\delta}\left(\frac{i-1}{\delta}\right)^{\delta} \text{ as } k \longrightarrow \infty.$$

Finally, since $l'(k,\delta) \leq \mathbb{E}(\rho) \leq u'(k,\delta)$, we conclude that

$$\lim_{k\to\infty} \mathbb{E}(\rho) = 1 - \frac{g(\delta)}{\delta}$$

with

$$g(\delta) = \sum_{i=1}^{\delta}\left(\frac{i-1}{\delta}\right)^{\delta}.$$

The value for the competitive ratio $\mathbf{r}^*_{\mathtt{RI}_{k\delta}}$ is obtained by taking into account that $\mathbf{r}^*_{\mathtt{RI}_{k\delta}} = \frac{1}{\mathbb{E}(\rho)}$. □

Theorem 3 shows the competitive ratio for the online bipartite matching problem for regular graphs with a given randomized adversary choosing between $\mathrm{O}\left(n!^2\right)$ instances. This is therefore a lower bound on the competitive ratio of the matching problem for regular bipartite graphs for any randomized adversary.

It can be shown that $g(m)$ is a monotonically increasing function and that

$$\lim_{m\to\infty} g(m) = \lim_{m\to\infty}\sum_{i=1}^{m}\left(\frac{i-1}{m}\right)^{m} = \frac{1}{e-1}.$$

5 Conclusions

We have studied the online bipartite matching problem for regular graphs with a randomized adversary. We have determined the competitive ratio for the case $\delta = 2$ for any number of random bits. This turns out to be an interesting example of a problem in which one random bit used by the adversary makes the problem as hard as for many random bits. It is also an example in which the competitive ratio for the randomized adversary model is equal to the ratio for the randomized algorithm model. We also gave upper and lower bounds for other values of the degree for adversaries using one random bit.

Finally, we studied the case for adversaries with a large number of random bits and found the competitive ratio for a given class of instances. We were not able to find an upper bound for the competitive ratio in the general case, but we are convinced that the exact value for the competitive ratio should not be far from the given result.

References

1. Barrière, L., Muñoz, X., Fuchs, J., Unger, W.: Online matching in regular bipartite graphs. Parallel Process. Lett. **28**(2) (2018). https://doi.org/10.1142/S0129626418500081
2. Birnbaum, B., Mathieu, C.: On-line bipartite matching made simple. ACM SIGACT News **39**(1), 80 (2008)
3. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
4. Burjons, E., Hromkovič, J., Muñoz, X., Unger, W.: Online graph coloring with advice and randomized adversary (extended abstract). In: Freivalds, R.M., Engels, G., Catania, B. (eds.) SOFSEM 2016. LNCS, vol. 9587, pp. 229–240. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49192-8_19
5. Colbourn, Ch., Dinitz, J.: Handbook on Combinatorial Designs, 2nd edn. Chapman & Hall/CRC, London (2007)
6. Diestel, R.: Graph Theory, 3rd edn. Springer, Heidelberg (2005)
7. Dobrev, S., Hromkovič, J., Komm, D., Královič, R., Královič, R., Mömke, T.: The complexity of paging against a probabilistic adversary. In: Freivalds, R.M., Engels, G., Catania, B. (eds.) SOFSEM 2016. LNCS, vol. 9587, pp. 265–276. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49192-8_22
8. Fiat, A., Woeginger, G.J. (eds.): Online Algorithms – The State of the Art. LNCS, vol. 1442. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0029561
9. Fuchs, B., Hochstättler, W., Kern, W.: Online matching on a line. Electron. Notes Discret. Math. **13**, 49–51 (2003)
10. Irani, S., Karlin, A.R.: On online computation. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-Hard Problems, Chap. 13, pp. 521–564. PWS Publishing Company, Boston (1997)
11. Kalyanasundaram, B., Pruhs, K.: Online weighted matching. J. Algorithms **14**(3), 478–488 (1993)
12. Karp, R.M., Vazirani, U.V., Vazirani, V.V.: An optimal algorithm for on-line bipartite matching. In: Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC), pp. 352–358 (1990)
13. Komm, D.: An Introduction to Online Computation - Determinism, Randomization, Advice. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-42749-2
14. Mehta, A., Saberi, A., Vazirani, U.V., Vazirani, V.V.: AdWords and generalized online matching. J. ACM **54**(5), 1–19 (2007)
15. Meyerson, A., Nanavati, A., Poplawskiavi, L.: Randomized online algorithms for minimum metric bipartite matching. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 954–959 (2006)
16. Naor, J., Wajc, D.: Near-optimum online ad allocation for targeted advertising. In: Proceedings of the Sixteenth ACM Conference on Economics and Computation (EC 2015), pp. 131–148 (2015)
17. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)
18. Smula, J.: Information content of online problems - advice versus determinism and randomization. Ph.D. thesis, ETH Zürich (2015)
19. Yao, A.: Probabilistic computations: toward a unified measure of complexity. In: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 222–227 (1977)

A Dynamic Distributed Data Structure for Top-k and k-Select Queries

Björn Feldkord, Manuel Malatyali, and Friedhelm Meyer auf der Heide(✉)

Heinz Nixdorf Institute and Computer Science Department,
Paderborn University, Paderborn, Germany
{bjoern.feldkord,manuel.malatyali,fmadh}@upb.de

Abstract. We consider a scenario where n sensor nodes observe streams of data. The nodes are connected to a central server whose task it is to compute some function over all data items observed by the nodes. Extending the capabilities of the distributed monitoring model from [8], we allow, in addition to sending messages from the sensor nodes to the server, also broadcasts from the server to the sensor nodes (see for example [9]).

In this paper, we address the problem of answering Top-k queries (report the k largest data items currently observed) and approximate k-Select queries (report an element with rank close to k). We present a communication-efficient dynamic data structure that supports these queries under updates of the data items arriving at the sensor nodes.

1 Introduction

Consider a sensor network which is a system comprising of a huge amount of nodes. Each node continuously observes its environment and measures information (e.g., temperature, pollution or similar parameters). We are interested in aggregations describing the current observations at a central server. To keep the server's information up to date, the server and the nodes can communicate with each other. In sensor networks, however, the amount of such communication is particularly crucial, as communication has the largest impact to energy consumption, which is limited due to battery capacities [11]. Therefore, algorithms aim at minimizing the (total) communication required for computing the respective aggregation function at the server.

We consider the following idea to potentially lower the communication used. Computations of the same aggregate should *reuse* parts of previous computations. We realize this by introducing a data structure which, at every point in time, keeps track of an approximation of a data item with rank k. These approximations can be exploited by the protocols for a Top-k or k-Select computation to significantly decrease the communication and interestingly also the time bounds, making this approach a very powerful tool.

This work was partially supported by the German Research Foundation (DFG) within the Priority Program "Algorithms for Big Data" (SPP 1736).

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 311–329, 2018.
https://doi.org/10.1007/978-3-319-98355-4_18

1.1 The Distributed Monitoring Model with Broadcast Channel (DMBC-Model)

We consider the distributed monitoring model introduced by Cormode, Muthukrishnan, and Yi in [8], in which there are n distributed nodes, each uniquely identified by an ID from the set $\{1, \ldots, n\}$, connected to a single server. Each node observes a stream of data items over time, i.e., at any discrete time step t node i observes a data item d_i^t. We assume that the data items have a total order and denote by *rank(d)* the position of data item d in the sorted ordering. The server is asked to, given a query at time t, compute an output $f(t)$ which depends on the data items d_i^t with $i = 1, \ldots, n$ observed across all distributed streams.

To be able to compute the output, the nodes and the server have to communicate with each other. The distributed monitoring model introduced by Cormode, Muthukrishnan, and Yi in [8] allows exchanging single cast messages. The extension we use is the Distributed Monitoring Model with a Broadcast Channel (DMBC-Model) (proposed in [7] and exploited in [3,4,9,10]) which allows, in addition, to broadcast messages from the server to all nodes. Both types of communication are instantaneous and have unit cost per message. That is, sending a single message to one specific node incurs cost of one and so does one broadcast message. Each message has a size of $\mathcal{O}(\mathcal{B} + \log n)$ bits, where $\mathcal{B}$ denotes the number of bits needed to encode a data item. A message will usually, besides a constant number of control bits, consist of a data item, and a node ID.

Between any two time steps we allow a communication protocol to take place, which may use a polylogarithmic number of rounds. The optimization goal is the minimization of the communication complexity, given by the total number of exchanged messages, required to answer the posed requests or setup/update the data structure.

1.2 Related Work

Cormode, Muthukrishnan, and Yi introduce the Continuous Monitoring Model [8] with an emphasis on systems consisting of n nodes generating or observing *distributed* data streams and a designated coordinator. In this model the coordinator is asked to continuously compute a function, i.e., to compute a new output with respect to all observations made up to that point. The objective is to aim at minimizing the total communication between the nodes and the coordinator. We enhance the continuous monitoring model (as proposed by Cormode, Muthukrishnan, and Yi in [7]) by a broadcast channel. Note that we are not strictly continuous in the sense that we introduce a dynamic data structure which only computes a function if there is a query for it. However, there is still a continuous aspect: In every time step, our data structure maintains elements close to all possible ranks in order to quickly answer queries (cf. [5,6,15,16]).

An interesting area of problems within this model are threshold functions: The coordinator has to decide whether the function value (based on all observations) has reached a given threshold τ. For well-structured functions (e.g., count-distinct or the sum-problem) asymptotically optimal bounds are known [7,8,13]. Functions which do not provide such structures (e.g., the entropy [1]), turn out to require much more communication volume.

A related problem is a variant of the distributed Top-k monitoring problem considered by Babcock and Olston [2]: There is a set of objects $\{O_1, \ldots, O_n\}$ given, in which each object has a numeric value. The stream of data items updates these numeric values (of the given objects). Babcock and Olston have shown by an empirical evaluation that the amount of communication is by an order of magnitude lower than that of a naive approach.

Furthermore, the Top-k computation is also considered in [9]. The proposed protocol needs a number of $\mathcal{O}(k \cdot \log N)$ messages and $\mathcal{O}(k \cdot \log N)$ rounds, where $N > n$ denotes an upper bound on the number of nodes. In this paper we improve both the number of messages and communication rounds to $k + \log n + 2$ and $\mathcal{O}(k + \log N)$ respectively. The techniques used in this paper are fundamentally based on [4] applying the idea of an inorder treewalk in a distributed searchtree and analyzing using a mixed distribution.

A model related to our (sub-)problem of finding the k-th largest values, and exploiting a broadcast channel, is investigated by the shout-echo model in [12,14]. A communication round is defined as a broadcast by a single node, which is replied by all remaining nodes. The objective is to minimize the number of communication rounds, which differs from ours.

1.3 Contribution of the Paper

We present a distributed data structure for the DMBC-Model with the following properties: In each step t, each client i receives a data item d_i^t as above. For ease of description let $s_1^t, \ldots, s_n^t$ be the *sorted* version of the data items $d_1^t, \ldots, d_n^t$ received at time step t. Our data structure supports the following operations:

TOP-k: Output $\{s_1^t, \ldots, s_k^t\}$
STRONG SELECT: Output $d \in \{s_{(1-\varepsilon)k}^t, \ldots, s_{(1+\varepsilon)k}^t\}$
WEAK SELECT: Output d with $s_{k \cdot \log^{c_1} n}^t \leq d \leq s_{k \cdot \log^{c_2} n}^t$, with $c_1, c_2 > 1$

Our data structure gives the following *performance guarantees*:

– The expected amortized total communication cost for an update (amortized over all updates of the data items received by clients) is $\mathcal{O}(1/\text{polylog}\, n)$, the number of rounds is $\mathcal{O}(\log n)$.
– WEAK SELECT does not need any communication. The output is correct with probability at least $1 - 1/\text{polylog}\, n$.
– The expected total communication cost for a STRONG SELECT operation is bounded by $\mathcal{O}(1/\varepsilon^2 \log 1/\delta + \log^2 \log n)$, the expected number of rounds is $\mathcal{O}(\log\log \frac{n}{k})$. The output is correct with probability at least $1 - \delta$.
– The expected total communication cost for TOP-k is $\mathcal{O}(k + \log\log n)$, the expected number of rounds is $\mathcal{O}(\log\log n)$. The output is always correct.

2 Outline of the Data Structure

Our data structure maintains an information *Sketch*(t) about the data items received at time t in the server, at every time t.

As above, let $s_1^t, \ldots, s_n^t$ be the *sorted* version of the data items $d_1^t, \ldots, d_n^t$ received at time step t. Fix sufficiently large constants $c_1, c_2, c > 1$. We call *Sketch*(t) correct if it consists of a set of data items $\{d_1, \ldots, d_{\log n}\}$ such that, for each $k = 1, \ldots, n$, there exists a d_j such that $s_{k \cdot \log^{c_1} n}^t \leq d_j \leq s_{k \cdot \log^{c_2} n}^t$ holds. *INIT* denotes the process of computing *Sketch*(t), where the input $d_1^t, \ldots, d_n^t$ of step t is given to the n sensor nodes.

Observation 1. *Consider a time step t at which the Init operation is called. A correct Sketch(t) is also a correct Sketch(t'), for a $t' > t$, if at most $\log^c(n)$ values of the clients are updated during the time interval $(t, t']$.*

This observation holds, because in the worst case the rank of a fixed data item, facing $\log^c n$ updates, can change by at most $\log^c n$. Since we allow the data item to be upper bounded by $s_{k \cdot \log^{c_2} n}^t$ simply observe that this still holds after $\log^c n$ updates, for sufficiently large choices of constants. On the other hand, to prevent that the data item d_j gets not smaller than s_k^t, the data structure computes a data item $d_j > s_{k \cdot \log^{c_1} n}^t$. Note that the constants c_1 and c_2 depend on c. However, if the constants are (beforehand) chosen large enough, this ensures that after $\log^c n$ updates *Sketch*(t) is also a *Sketch*(t').

Lemma 1. *INIT is executed correctly with probability at least $1 - 1/\text{polylog}\, n$. It needs expected total communication of $\mathcal{O}(\log n)$ and $\mathcal{O}(\log n)$ rounds.*

We present the INIT algorithm and the necessary technical basis to prove this lemma in Sect. 3. We prove that the algorithm computes a *Sketch*(t) correctly in Theorem 1 and present the performance guarantees in Theorem 2.

The next operation *UPD* denotes the process of updating *Sketch*(t), in response to the updates of data items received in step t.

Lemma 2. *UPD can be done using expected amortized (w.r.t. number of updates of data items in the nodes) total communication of $\mathcal{O}(1/\text{polylog}\, n)$, the amortized number of rounds is constant (assuming each update is processed at a different time step). For every step t, the computed Sketch(t) is correct with probability at least $1 - 1/\text{polylog}\, n$.*

The UPD algorithm is presented in Sect. 4. In this section, we shortly argue its correctness in Lemma 7 and show communication bounds in Lemma 8.

By definition of a correct *Sketch*(t), the following observation holds.

Observation 2. *Given a correct Sketch(t), WEAK SELECT can be executed without any communication. It is correct with probability $1 - 1/\text{polylog}\, n$.*

We present this observation shortly in Sect. 5.

Lemma 3. *Given a correct Sketch(t), STRONG SELECT can be correctly computed with probability $1-\delta$. It needs $\mathcal{O}(1/\varepsilon^2 \log 1/\delta + \log^2 \log n)$ communication and $\mathcal{O}(\log \log n)$ communication rounds.*

This result is considered in Sect. 6. The algorithm is based on three phases which are analyzed independently. The main result of this section is presented in Theorem 3.

Lemma 4. *Given a correct Sketch(t), TOP-k can be computed using expected total communication of $\mathcal{O}(k + \log \log n)$ and $\mathcal{O}(k + \log \log n)$ communication rounds. The output is always correct.*

The TOP-k algorithm is presented in Sect. 7. On the total communication is argued in Lemma 13 and the number of rounds in Lemma 14.

The lemmata above imply the performance guarantees formulated in the previous section.

3 Initialization of the Data Structure

We start the presentation of our results with the goal to prove the first lemma. We propose the algorithm INIT which computes the *Sketch*(t) at a time step t. Since (a variation of) this algorithm is reused in later sections, we describe a procedure CFS (ConstantFactorSelect) with different parameters (see Algorithm 1).

CFS. The high-level idea of CFS is as follows: Initially each node is defined to be active. The protocol samples a node uniformly at random and broadcasts its value. All nodes with a larger data item deactivate themselves. This process is repeated until the remaining nodes are sampled with probability 1.

However, since the server does not know which nodes remain active, a sample cannot be chosen directly. Instead, we let the nodes proceed a random process such that the server can probe each node with a certain outcome based on the random process. We consider this random process in more detail: Each node i chooses a height h_i from a geometric distribution, i.e., the number of coin flips with success probability p until one coin flip was successful. (Observe that based on the definition of p the expected maximal height $\max_i h_i$ varies. This fact can be used to trade-off between the expected number of messages and the number of rounds the algorithm uses.)

Intuitively speaking, we build a distributed (not binary) searchtree where the heights are chosen randomly and the algorithm follows the path to the sensor node observing the minimum value. The $\log n$ nodes on this path yield an approximation of the data items with respect to their ranks. We will exploit this fact and show that a data item at a specific level can be used to approximate a given rank successfully.

Algorithm 1. INIT()

```
CFS($\phi, h_{\max}, k$)                                        [ConstantFactorSelect]

 1. Each node $i$ defines a random variable $h_i$, i.i.d. drawn from
    a geometric distribution with $p = (1 - \phi)$, and
    redefines $h_i := \min\{h_i, h_{\max}\}$.
 2. Server defines $d_{\min} := \infty$, keeps a set $S := \emptyset$, and initializes $cnt := 0$.
 3. if $k = 1$ then $h_{\min} := 1$.
 4. else $h_{\min} := \lfloor \log_{1/\phi}(7k) \rfloor + 1$.     (let $\alpha := \log_{1/\phi}(7k) - \lfloor \log_{1/\phi}(7k) \rfloor$)
 5. for $h := h_{\max}$ to $h_{\min}$ do
 6.    Server probes all nodes $i$ with $d_i < d_{\min}$ and $h_i = h$.
 7.    Let $r_1 < r_2 < \ldots < r_j$ be the responses, ordered by their values.
 8.    The tuple $(r_1, h)$ is added to $S$.
 9.    if $h > h_{\min} > 1$ then Server redefines $d_{\min} := r_1$ else $d_{\min} := r_{(1/\phi)^\alpha}$.
10. output $d_{\min}$

INIT()
 1. call CFS ($\phi := \frac{1}{2}, h_{\max} = \log n, k = 1$).
```

Initialize. We only need a simple variant of the CFS protocol as follows: The INIT operation defines $p := 1/2$, the success probability of each coin flip. That is, each sensor node has a height of 2 in expectation. Thus, observe that the expected maximum of n nodes is $h_{\max} = \log n$. For each height h the server keeps the smallest response of the sensor nodes in the data structure.

3.1 Initialize Computes *Sketch*(*t*)

Recall that the data structure is asked to answer each request for a data item of rank k by a data item d. We group a set of requests with different ranks which we answer with the same data item d. To this end, we divide the ranks $1, \ldots, n$ into classes $C_1, \ldots, C_m$, where m is chosen sufficiently large such that each data item belongs to a class. The exact number of classes is based on a constant which is defined by the analysis, however, note that $m = \mathcal{O}(\log n)$ holds.

We define a representative for each class which is the response for a request of any rank in the next-smaller class. Furthermore, the height of a class represents the expected maximum height found within this class, such that our representative will have a height value within the noted bounds. In the following we use the constant $\kappa > 1$ chosen sufficiently large which represents the constants in the bounds on the precision and the success probability. Furthermore, let $\mathscr{H} := \log \log n$ to ease the notation. The idea of classes is captured in the following definition:

Definition 1 (Classes). *A* Class C_ℓ^t *consists of all data items* d_j^t *fulfilling* $rank(d_j^t) \in [\log^{6\ell\kappa}(n), \log^{6(\ell+1)\kappa}(n))$. *We define* $h_{\min}(C_\ell) := (6\ell\kappa + 1\kappa)\mathscr{H}$ *and* $h_{\max}(C_\ell) := (6\ell\kappa + 7\kappa)\mathscr{H}$. *The height of the class* C_ℓ^t *is given by* $h(C_\ell^t) := (h_{\min}(C_\ell^t), h_{\max}(C_\ell^t)]$.

By abuse of notation we introduce $d_i^t \in C_\ell^t$ which shortens $rank(d_i^t) \in C_\ell^t$. Furthermore let $class(d)$ be the class where the data item d belongs to, i.e., for a given d, $class(d)$ gives the class C_ℓ^t such that $d \in C_\ell^t$ holds.

Definition 2 (Inner Classes). *We denote by an* inner class $I_{\ell,\tau}^t$ *(where* $\tau \in \{0,1,2\}$ *holds) the set of data items* d_i^t *with a rank between* $\log^{6\ell\kappa+2\kappa}(n)$ *and* $\log^{6\ell\kappa+4\kappa}(n)$*. The height of* $I_{\ell,\tau}^t$ *is* $h(I_{\ell,\tau}^t) = ((6\ell\kappa + (2\tau+1)\kappa)\mathscr{H}, (6\ell\kappa + (2\tau + 3)\kappa)\mathscr{H}]$.

We omit the time step t in our notation whenever it is clear from the context.

Definition 3 (Well-Shaped). *The data items in an inner class* $I_{\ell,\tau}$ *are* well-shaped *if for each data item* $d_i \in I_{\ell,\tau}$ *it holds* $h_i \leq (6\ell\kappa + (2\tau+3)\kappa)\mathscr{H}$.

We start by analyzing the outcome of the INIT operation. That is, we show that a class is well-shaped with sufficiently large probability in Lemma 5 and argue that the data structure has one representative in Theorem 1, afterwards.

Lemma 5. *After an execution of INIT, the inner class* $I_{\ell,\tau}$ *is well-shaped with probability at least* $1 - \log^{-\kappa}(n)$.

Proof. Recall that for a fixed data item d_i and sensor node i the probability for $h_i > h$ is ϕ^h. Fix an inner class $I_{\ell,\tau}$ and consider the data items $d_i \in I_{\ell,\tau}$. We upper bound the probability that there is a data item with a height of at least h with $h := (6\ell\kappa + (2\tau+3)\kappa)\mathscr{H}$ by applying the union bound as follows:

$$\begin{aligned}\Pr[\exists d_i \in C_{\ell,\tau} \mid h_i > h] &\leq \left(\log^{6\ell\kappa+(2\tau+2)\kappa}(n) - \log^{\ell 8\kappa+2\tau\kappa}(n)\right) \cdot \phi^h \\ &\leq \log^{6\ell\kappa+(2\tau+2)\kappa}(n) \cdot \log^{-(6\ell\kappa+(2\tau+3)\kappa)}(n) \\ &\leq \log^{-\kappa}(n)\end{aligned}$$

□

Lemma 6. *Consider the inner class* $I_{\ell,1}$*. There is a data item* $d_i \in I_{\ell,1}$ *with* $h_i > (6\ell\kappa + 3\kappa)\mathscr{H}$ *with high probability.*

Proof. Here we simply upper bound the probability that each data item in the inner class has a height of at most h as follows:

$$\begin{aligned}\Pr[\forall d_i \in I_{\ell,1} \mid h_i \leq (6\ell\kappa+3\kappa)\mathscr{H}] &\leq \left(1 - 2^{-(6\ell\kappa+3\kappa)\mathscr{H}}\right)^{|I_{\ell,1}|} \\ &\leq \left(1 - \log^{-(6\ell\kappa+3\kappa)} n\right)^{\log^{6\ell\kappa+4\kappa} n - \log^{6\ell\kappa+2\kappa} n} \\ &\leq \left(\frac{1}{e}\right)^{\frac{1}{2}\log^{\kappa} n} \leq n^{-\frac{1}{2}\log(e)\log^{\kappa-1}(n)} \leq n^{-c},\end{aligned}$$

for some constant c. □

We can now prove the first part of Lemma 1, i.e., that INIT computes a correct *Sketch*(t). Technically, we show a result which is more restricted than the stated precision of Lemma 1, as follows:

Theorem 1. *After execution of INIT there exists, for each rank k, a data item in Sketch(t) with rank between $k \cdot \log^{2\kappa}(n)$ and $k \cdot \log^{10\kappa}(n)$ with probability at least $1 - \log^{-\kappa+2}(n)$.*

Proof. First consider a fixed inner class $I_{\ell,\tau}$ for a fixed $\ell \in \mathbb{N}$ and $\tau \in \{0, 1, 2\}$. Based on Lemma 5 we can show that the distribution of the random heights is well-shaped with probability at least $1 - \log^{-\kappa}(n)$. Now, with high probability there is a data item with such a height for sufficiently large κ and n due to Lemma 6. These observations together show that there is a data item d identified and stored in DS with probability at least $1 - \log^{-\kappa+1}(n)$.

Furthermore, note that the number of inner classes is upper bounded by $\log n$. The argument stated above applied to each class leads to the result that for each inner class there exists a data item in the data structure, and each inner class is well-shaped with probability at least $1 - \log^{\kappa-2}(n)$ (by simply applying the union bound). □

3.2 Communication Bounds

In the following we show the second part of Lemma 1, i.e., that the number of messages used by INIT is upper bounded by $\mathcal{O}(\log n)$ and the same bound holds for the number of rounds. We start by analyzing the bound on the total communication.

We show an upper bound on the communication used by the CFS protocol analyzing the expected value of a mixed distribution: Intuitively speaking, consider the path from the root to the maximum in a non-binary searchtree. For each node i on the path consider the number of siblings j with a smaller data item, i.e., $d_j < d_i$. To bound the expected number of such siblings j, we first consider on a fixed height h the number of tries G_h until the first node j' has drawn a height $h_{j'} > h$ (for each height h this results in the geometric sequence, Definition 4). Based on G_h, we consider the number of nodes that have drawn precisely the height $h_{j'} = h$ (for each height h, the geocoin-experiment Definition 5).

Note that this analysis turns out to be very simple since independence can be exploited in a restricted way and leads to a proper analysis with respect to small constants.

Definition 4. *We call a sequence $G = (G_1, \ldots, G_m)$ of m random experiments a* geometric sequence *if each G_h is chosen from a geometric distribution with $p_h^{geo} := \phi^h$. We denote its size$(G) := \sum_h G_h$ and say it* covers *all nodes if size$(G) \geq n$.*

For the analysis, we choose a fixed length of $m := \log_{1/\phi}(n)$ and modify G to $G' = (G_1, \ldots, G_{m-1}, n)$ such that G' covers all nodes with probability 1.

Based on a given geometric sequence, we define a sequence describing the number of messages sent by the nodes on a given height. We take the number of nodes G_j as a basis for a Bernoulli experiment where the success probability is the probability that a node sends a message on height h_j. This is $\Pr[h = h_j \mid h \leq h_j] = \frac{\phi^{h-1}(1-\phi)}{1-\phi^h}$.

Definition 5. *We denote a* geocoin-experiment *by a sequence $C = (C_1, \ldots, C_m)$ of random variables C_h which are drawn from the binomial distribution Binom$\left(n = G_h, p_h^{bin} = \frac{\phi^{h-1}(1-\phi)}{1-\phi^h}\right)$, i.e., C_h out of G_h successful coin tosses and each coin toss is successful with probability p_h^{bin}.*

We are now prepared to prove the second part of Lemma 1, i.e., a bound on the total communication for CFS and thus for INIT.

Theorem 2. *Let $h_{\max} \geq \log_{1/\phi}(n)$ hold. The* CFS *protocol uses an expected number of $\frac{1-\phi}{\phi}\log_{1/\phi}(n) + \frac{1}{\phi}$ messages in total.*

Proof. The number of messages sent is upper bounded by a geocoin-experiment C. Let $\mathscr{H} := \log_{1/\phi}(n)$. For $h < \mathscr{H}$ we use that the geometric distribution is memory-less and hence

$$\mathbb{E}[C_h] = (1 - p_h^{geo}) \cdot (p_h^{bin} + \mathbb{E}[C_h]) = (1 - \phi^h) \cdot \left(\frac{\phi^{h-1}(1-\phi)}{1-\phi^h} + \mathbb{E}[C_i]\right).$$

This can simply be rewritten as $\mathbb{E}[C_h] = \frac{1-\phi}{\phi}$.

For $h \geq \mathscr{H} = \log_{1/\phi}(n)$ we bound the number of messages by the total number of nodes with height at least $\mathscr{H}$. These can be described as the expectation of a Bernoulli experiment with n nodes and success probability $\phi^{\mathscr{H}-1}$ and hence we can bound $\mathbb{E}[C_{\geq \mathscr{H}}] \leq \phi^{\mathscr{H}-1} \cdot n = \frac{1}{\phi}$.

In total, we get

$$\sum_h \mathbb{E}[C_h] = \left(\sum_{h=1}^{\mathscr{H}-1} \mathbb{E}[C_i]\right) + \mathbb{E}[C_{\geq \mathscr{H}}] \leq \frac{1-\phi}{\phi}\log_{1/\phi}(n) + \frac{1}{\phi},$$

concluding the proof. □

We conclude the proof for the first lemma by this simple observation on the number of rounds. By the definition of the protocol it is easy to see that the server simply sends a broadcast message for each height h and receives a message by those nodes which fulfill a specific rule. Since the server can process each received message in the same round, $h_{\max}$ is obviously a strict upper bound for the number of rounds.

Observation 3. *The INIT operation uses at most $h_{\max} = \log n$ rounds.*

Algorithm 2. UPD(i, d) [Executed by node i]

```
1. Update $d_i^t$ by $d_i^{t+1} := d$.
2. Flip a coin with probability $p_{cnt} = \frac{c}{\log^{\kappa}(n)} \cdot \log n$.
3. if the coin flip was successful then
4.    send a message to the server; increase $cnt$.
5. if $cnt = c \cdot \log n$ holds then                [Executed by Server]
6.    restart the protocol, i.e., call INIT()
```

4 Update

To keep the data structure up to date we apply the following simple straightforward strategy: As long as there are fewer than $\log^c n$ updates processed since the last call of INIT, the precision of the approximated ranks can also only differ by an additional $\mathcal{O}(\log^c n)$ (for a predefined constant $c > 1$). We apply a simple standard counting technique to verify that the current number of processed UPD operations is $\mathcal{O}(\log^c n)$ in expectation. If more UPD operations are identified, the current data items in the data structure are discarded and the *Sketch*(t) is built from scratch.

Consider the protocol for the UPD operation as presented in Algorithm 2. It applies a randomized counting technique to identify that there are more than $\Theta(\log^{\kappa} n)$ updates since the last INIT operation. It is easy to verify by applying standard Chernoff bounds that the protocol identifies $cnt \leq 2\,c \log n$ with high probability. Thus, and applying a Chernoff bound again, it follows that the number of UPD operations that took place since the last INIT operation is upper bounded by $2 \log^{2c} n$ with high probability. With this, we can show the first part of Lemma 2.

Lemma 7. *After the last call of INIT, there are at most $\Theta(\log^{\kappa} n)$ UPD operations processed with high probability.*

The UPD operation sends a message with probability p_{cnt}, so it is easy to verify that the expected number of messages sent is upper bounded by p_{cnt}.

Now consider a sufficiently large instance (i.e., sufficiently many UPD operations). Assume that for a time step t at which INIT is called, t' denotes the next time step at which INIT is called to rebuild the data structure. Observe that $\mathcal{O}(\log n)$ messages where sent during $[t, t']$ by UPD and INIT operations in total. Since $\Omega(\log^{c/2} n)$ UPD operations where called with high probability, the amortized bound for one single UPD operation follows.

For the number of communication rounds, consider the same interval $[t, t']$ as described above. Since one execution of UPD uses a constant number of rounds (excluding the call of INIT) and the INIT operation is called a constant number of times, each UPD operation only uses an amortized constant number of rounds. These observations conclude the argumentation for the second part of Lemma 2:

Algorithm 3. WEAK SELECT(k)

```
1. Determine $\ell$ such that $k \in C_\ell$ holds.
2. output representative $r \in I_{\ell+1}$.
```

Lemma 8. *The UPD operation uses* $\mathcal{O}(1/\text{polylog}\, n)$ *messages in expectation and amortized* $\mathcal{O}(1)$ *number of rounds.*

5 Weak Select

For the sake of a complete presentation we shortly describe how the *Sketch*(t) is used to answer a weak approximate k-Select request.

The WEAK SELECT operation simply identifies the class ℓ in which the data item d with rank k is expected (see Algorithm 3). Then, the representative r in the class on level $\ell + 1$ is chosen.

Note that by the correctness of INIT and its analysis on the precision the correctness of the protocol follows. It is also easy to see that the protocol is executed by the server and thus does not need any further communication to the sensor nodes. Since no further argumentation is needed, we restate the following observation for completeness:

Observation 4. *Given a correct Sketch*(t)*, WEAK SELECT can be executed without any communication. It is correct with probability* $1 - 1/\text{polylog}\, n$.

6 Strong Approximate *k*-Select

In this section we present an algorithm which gives an (ε, δ)-approximation for the k-Select problem, i.e., a data item d is identified with a rank between $(1-\varepsilon)k$ and $(1 + \varepsilon)k$ with probability at least $1 - \delta$. In other words, we propose an algorithm and analyze its performance guarantee as claimed in Lemma 3.

Algorithm Description. We apply a standard sampling technique to select a data item as required. However, the data item given by the WEAK SELECT operation is too weak to directly be followed by a sampling technique (cf. Phase 3 in Algorithm 4). Thus, we add the following two phases:

(1) A data item d' is identified, such that a polylogarithmic error bound holds with high probability. It might be that a large number of sensor nodes (i.e., $\omega(k \cdot \text{polylog}\, n)$) 'survive' till the last phase and apply the costly sampling technique. With this step the event only occurs with probability at most $1/n$.

(2) The second phase applies $c \log \frac{1}{\delta'}$ calls of CFS to identify data items that have a rank between k and $42\,k$ with constant probability each. This number of calls is to amplify the (success) probability that the final data item d^* has a rank between k and $42\,k$ to at least $1 - \delta'$.

Algorithm 4. STRONG SELECT(ϕ, k, ε, δ)

```
 1. call WEAK SELECT($k$) and
    denote by $(d', h')$ the returned data item and its height.

    [Phase 1]
 2. Determine $\ell'$ such that $k \cdot \log^c n \in C_{\ell'}$ holds.
 3. repeat until a data item $d''$ is found (i.e., $d'' \neq$ nil)
 4.   Each node $i$ with $d_i \leq d'$ executes:
 5.     Call CFS($\phi$, $h$, $k \cdot \log^c n$) and let $(r, h_r)$ be the data item and height.
 6.     if $h_r \in h(C_{\ell'})$ then $d'' := r$.

    [Phase 2]
 7. for $j = 1, \ldots, c \log {}^1\!/\!{}_{\delta'}$ do in parallel
 8.   Each node $i$ with $d_i \leq d''$ executes:
 9.     call CFS($\phi$, $h_r$, $k$) on the active nodes and let $(d^*_j, h'_j)$ the output.
10. $d^* := Median(d^*_1, \ldots, d^*_{c \log 1/\delta'})$

    [Phase 3]
11. Each node $i$ with $d_i < d^*$ executes:
12.   Toss a coin with $p := \min\left(1, \frac{c}{k} \cdot \mathscr{S}_{\varepsilon,\delta}\right)$.
13.   On success send $d_i$ to the server.
14. The server sorts these values and outputs $d_{\tilde{k}}$, the $p \cdot k$-th smallest item.
```

Note that the internal probability δ' will be defined as $1/\text{polylog}\, n$ which is a result of the analysis. Important is that the calls of CFS do not change the information of *Sketch*(t) stored in the data structure. Here, they are only used 'read-only' and are not overwritten.

Analysis Outline. We split our analysis and consider each phase separately. First, we show that Phase 1 determines a data item d' with a rank which is by a polylogarithmic factor larger than k with high probability. This needs $\mathcal{O}(\log\log n)$ messages and $\mathcal{O}(\log\log n)$ rounds in expectation.

Afterwards, we consider Phase 2 which determines a data item d'' with a rank only a constant factor larger than k with probability at least $1 - \delta'$, where δ' can be chosen arbitrarily small.

Finally, Phase 3 applies a sampling technique to determine the final data item d which yields the property as required by Lemma 3.

We use the notation as given in the protocol and use d' to denote the data item given by the WEAK SELECT operation, d'' the data item determined by Phase 1, and d^* the data item given by Phase 2. We do not need any further analysis for the property of data item d' since we analyzed its precision (and the given success probability bounds) in the past sections.

Analysis of Phase 1. We consider Phase 1 of the STRONG SELECT operation and analyze the precision of the rank of item d'', the expected number of messages and the number of communication rounds.

Lemma 9. *For a given constant* c*, there exist constants* c_1, c_2*, such that Phase 1 as given in Algorithm 4 outputs a data item* d'' *with a rank between* $7k \cdot \log^{c_1}(n)$ *and* $7k \cdot \log^{c_2}(n)$ *with probability at least* $1 - n^{-c}$*.*

Proof. We use a simple argument to argue on the probability to obtain a data item within a multiplicative polylogarithmic precision bound:

Consider the event that the rank is strictly smaller than $7k \cdot \log^{c_1} n$. Thus, one node i of the $7k \log^{c_1} n - 1$ nodes has drawn a height $h_i \geq \log(7k \log^c n)$. We show (by applying Chernoff bounds) that this probability is upper bounded by $n^{-c'}$, where c' depends on c and c_1. For the remaining case (i.e., the rank is strictly larger than $7k \cdot \log^{c_2} n$) the same argument is applied.

Let X denote the rank of the data item d'' which is identified by STRONG SELECT. Now let X_1 be drawn from $Binom(n = 7k \log^{c_1} n, p = (1/2)^{\log(7k \log^c n)})$, and let $X_2 \sim Binom(n = 7k \log^{c_2} n, p = (1/2)^{\log(7k \log^c n)})$. Observe that it holds $\gamma_1 = \mathbb{E}[X_1] = \log^{c_1 - c} n$ and $\gamma_2 = \mathbb{E}[X_2] = \log^{c_2 - c} n$. Thus, $\Pr[X < 7k \log^{c_1} n] \leq \Pr[X_1 > (1 + (1/2) \log^{c - c_1} n) \cdot \log^{c_1 - c} n] \leq \exp(-\frac{1}{12}(\log^{c - c_1} n)) \leq n^{-\frac{c - c_1 - 1}{12}}$.

We obtain by the same argument similar results for the probability of the event that the rank is larger than the claimed bound. □

Lemma 10. *Phase 1 uses an amount of* $\mathcal{O}(\log \log n)$ *messages in expectation.*

Proof. We apply the law of total expectation and first consider the event that WEAK SELECT is successful. Afterwards, the number of messages for a failed call is considered.

First, consider the case that the WEAK SELECT operation is successfully within the precision bounds. Then, $\mathcal{O}(\log \log n)$ messages on expectation are used in this phase. On the other hand, consider the number of messages used if the number of nodes that take part in this phase is n. Then, the protocol needs $\mathcal{O}(\log n)$ messages. However, the probability that WEAK SELECT is not within these bounds is $1/\text{polylog}\, n$ which concludes the proof. □

To upper bound the time needed for Phase 1, simply determine the range of h and observe that this range is bounded by $\mathcal{O}(\log \log n)$. Since one data item is found with probability at least $1 - 1/\text{polylog}\, n$, in expectation after the second repetition a data item is found.

Observation 5. *Phase 1 of Algorithm 4 uses* $\mathcal{O}(\log \log n)$ *number of rounds.*

Analysis of Phase 2. Now consider one execution of the lines 7 to 10 as given in Algorithm 4 (and restated in Algorithm 5).

Lemma 11. *One execution of lines 8 and 9 of Phase 2 in Algorithm 4 outputs a data item* d *with* $rank(d) \in [k, 42k]$*, with probability at least* 0.6*.*

Algorithm 5. STRONG SELECT [Phase 2 restated]

```
7.  for $j = 1, \dots, c \log 1/\delta'$ do in parallel
8.      Each node $i$ with $d_i \leq d''$ executes:
9.          call CFS($\phi$, $h_r$, $k$) on the active nodes and let $(d_j^*, h_j')$ the output.
10. $d^* := Median(d_1^*, \dots, d_{c \log 1/\delta'}^*)$
```

Proof. The algorithm outputs the $(1/\phi)^\alpha$ smallest data item d_j^* the server gets as a response on height $h = h_{\min}$. To analyze its rank, simply consider the random number X of nodes i that observed smaller data items $d_i < d$. The claim follows by simple calculations: (i) $\Pr[X < k] \leq \frac{1}{5}$ and (ii) $\Pr[42k > X] \leq \frac{1}{5}$.

The event that X is (strictly) smaller than k holds if there are at least $(1/\phi)^\alpha$ out of k nodes with a random height at least $h_{\min}$. Let X_1 be drawn by a binomial distribution $Binom(n = k, p = \phi^{h_{\min}-1})$. It holds $\mathbb{E}[X_1] = k \cdot \phi^{h_{\min}-1} = \frac{1}{7} \cdot (\frac{1}{\phi})^\alpha$. Then, $\Pr[X < k] \leq \Pr[X_1 \geq (\frac{1}{\phi})^\alpha] = \Pr[X_1 \geq (1+6)\frac{1}{7\phi^\alpha}] \leq \exp(-\frac{1}{3}\frac{1}{7\phi^\alpha}6^2) \leq \frac{1}{5}$.

On the other hand, the event that X is (strictly) larger than $42k$ holds if there are fewer than $(1/\phi)^\alpha$ out of $42k$ nodes with a random height of at least $h_{\min}$. Let X_2 be drawn by a binomial distribution $Binom(n = 42k, p = \phi^{h_{\min}-1})$. It holds $\mathbb{E}[X_2] = (42k)\phi^{h_{\min}-1} = (42k)(7k)^{-1}\phi^{-\alpha} = \frac{6}{\phi^\alpha}$. Then, $\Pr[X > 42k] \leq \Pr[X_2 < \frac{1}{\phi^\alpha}] = \Pr[X_2 < (1 - (1 - \frac{1}{6}))\frac{6}{\phi^\alpha}] \leq \exp(-\frac{1}{2}(\frac{6}{\phi^\alpha}(1 - \frac{1}{6})^2) \leq \exp(-\frac{25}{12}) \leq \frac{1}{5}$. □

Note that we apply a standard boosting technique, i.e., we use $\mathcal{O}(\log \frac{1}{\delta'})$ independent instances, and consider the median of the outputs of all instances to be the overall output (cf. Algorithm 5). Thus, an output in the interval $[k, 42\,k]$ is determined with probability at least $1 - \delta'$.

Observation 6. *Phase 2 of Algorithm 4 outputs a data item d^* with $rank(d^*) \in [k, 42k]$ with probability at least $1 - \delta'$.*

Lemma 12. *Assume $\delta' \geq n^{-c}$ for a constant $c > 1$. The second phase of Algorithm 4 uses $\mathcal{O}(\log \frac{1}{\delta'} \cdot \log\log \frac{n}{k})$ messages in expectation.*

Proof. Consider one instance of Phase 2 and applying arguments from Theorem 2, the algorithm uses $\mathcal{O}(\log\log \frac{n}{k})$ messages in expectation for each iteration of Steps 8 and 9. This number of messages is multiplied by $\mathcal{O}(\log \frac{1}{\delta'})$, since we apply this number of executions in parallel.

It remains to show that the parallel execution does not need further messages to separate each execution from the others: In more detail, each instance of Steps 8 and 9 has to be executed with an additional identifier. Since $\delta' \leq n^{-c}$ holds, the identifier has a range of integer numbers between 1 and $\mathcal{O}(\log n)$ and thus needs additional $\mathcal{O}(\log\log)$ bits. Since a machine word has a size of $\mathcal{O}(\mathscr{B} + \log n)$ the identifier can be added to the message (or sent as a separate message such that the number of messages has a constant overhead). This concludes the proof. □

Algorithm 6. STRONG SELECT [Phase 3 restated]

```
11. Each node $i$ with $d_i < d^*$ executes:
12.     Toss a coin with $p := \min\left(1, \frac{c}{k} \cdot \mathscr{S}_{\varepsilon,\delta}\right)$.
13.     On success send $d_i$ to the server.
14. The server sorts these items and outputs $d_{\tilde{k}}$, the $p \cdot k$-th smallest item.
```

Since we run the $\mathcal{O}(\log \frac{1}{\delta'})$ instances in parallel, and the server is able to process all incoming messages within the same communication round, the number of communication rounds does not increase by the parallel executions.

Observation 7. *Phase 2 of Algorithm 4 uses $\mathcal{O}(\log\log n)$ rounds.*

6.1 Analysis of Phase 3

We are now prepared to propose the last phase of the algorithm which fulfills the required precision as stated in Lemma 3.

We consider the final phase of the algorithm, i.e., we apply a standard sampling technique (cf. Algorithm 6): The server broadcasts the value d^* which (as a result of the analysis of Phase 2) has a rank between k and $42\,k$ with probability at least $1 - 1/\text{polylog}\, n$. Each node i compares its data item d_i^t with d^* and only takes part in the sampling process if and only if $d_i^t \leq d^*$ holds. Then, with probability $p = \frac{c}{k}\frac{1}{\varepsilon^2}\log\frac{1}{\delta}$ node i sends its data item to the server. In turn, the server sorts each data item and outputs the $p \cdot k$-th smallest item, which has a rank of k in expectation.

For the sake of readability we introduce the notation $\mathscr{S}_{\varepsilon,\delta} := \frac{1}{\varepsilon^2}\log\frac{1}{\delta}$ and are now prepared to show Lemma 3:

Theorem 3. *Define $\delta' := 1/\text{polylog}\, n$. The STRONG SELECT operation (as presented in Algorithm 4) selects a data item $d_{\tilde{k}}$ with a rank in $[(1-\varepsilon)\,k, (1+\varepsilon)\,k]$ with probability at least $1-\delta$ using $\mathcal{O}(\mathscr{S}_{\varepsilon,\delta} + \log^2\log n)$ messages in expectation and $\mathcal{O}(\log_{1/\phi}\log n)$ communication rounds.*

Proof. From Lemma 12 we get that Phase 2 of the protocol uses an amount of at most $\mathcal{O}(\log\log\frac{n}{k}\log\frac{1}{\delta'})$ messages in expectation and runs for $\mathcal{O}(\log\log n)$ communication rounds. The remaining steps of Algorithm 4 need only one additional communication round and thus the stated bound on the communication rounds follows. We omit the proof for the correctness of the algorithm, i.e., with demanded probability the k-th smallest data item is approximated, since it is based on a simple argument using Chernoff bounds.

It remains to show the upper bound on the number of messages used. Formally, we apply the law of total expectation and consider the event that Phase 2 of Algorithm 4 determined a data item d^* with rank $k \leq rank(d^*) \leq 42k$ and the event $rank(d^*) > 42k$.

Observe that the sampling process in steps 2 and 3 yields $\mathcal{O}(\frac{rank(d^*)}{k}\mathscr{S}_{\varepsilon,\delta})$ messages in expectation. Consider the event that Phase 2 determined a data item d^* with rank $k \leq rank(d) \leq 42k$. Then, Phase 3 uses $\mathcal{O}(\mathscr{S}_{\varepsilon,\delta})$ messages in expectation. Now consider the event that Phase 2 determined a data item d^* with $d > 42\,k$. It uses $\mathcal{O}\left(\frac{\log^c n}{k}\mathscr{S}_{\varepsilon,\delta}\right)$ messages in expectation. Since the probability for this event is upper bounded by δ', the conditional expected number of messages is $\mathcal{O}\left(\frac{\log^c(n)}{k}\mathscr{S}_{\varepsilon,\delta} \cdot \delta'\right)$. Defining $\delta' := \log^{-c} n$ the bound follows as claimed. $\square$

6.2 One-Shot Approximate *k*-Select

For the sake of self-containment we propose a bound which considers all nodes to take part in the protocol.

Corollary 1. *Let c be a sufficiently large constant. Furthermore, let $N = n$, $\phi := \frac{1}{2}$, $h_{\max} := \log n$, and $\delta' := \frac{1}{\log^c(n)}$. The protocol uses an amount of at most $\mathcal{O}(\mathscr{S}_{\varepsilon,\delta} + \log n)$ messages in expectation and $\mathcal{O}(\log(\frac{n}{k}))$ expected rounds.*

This represents the case (with respect to the choice of ϕ) that a small number of messages and a large number of communication rounds are used. This observation is complemented by a lower bound of $\Omega(\log n)$ in [4].

7 Top-*k*

In this section we present an algorithm which identifies all k smallest data items currently observed by the sensor nodes, i.e., at a fixed time step t.

Note that by applying the MaximumProtocol (from [9]) k times and using the *Sketch*(t) from our data structure, the problem can be solved using $\mathcal{O}(k \cdot \log\log n)$ messages in expectation and $\mathcal{O}(k \cdot \log\log n)$ rounds. By applying the STRONG SELECT operation from the previous section (denote the output by d_K) and selecting all of the nodes i with a data item $d_i \leq d_K$, a bound of $\mathcal{O}(k + \log^2 \log n)$ expected messages and $\mathcal{O}(\log\log n)$ rounds in expectation follows. These bounds are subject to be improved to $\mathcal{O}(k + \log\log n)$ expected messages and $\mathcal{O}(k + \log\log n)$ expected rounds. Without our *Sketch*(t) the algorithm needs $k + \log n + 2$ expected messages and $\mathcal{O}(k + \log n)$ expected rounds, which might be of independent interest. We show a more general result which allows to trade-off between number of messages and number of rounds. This translates to $k + \frac{1-\phi}{\phi}\log_{1/\phi} n + \frac{1}{\phi}$ expected total communication and $\mathcal{O}(\phi \cdot k + \log_{1/\phi} n)$ expected rounds for an arbitrarily chosen $1/n \leq \phi \leq 1/2$.

Protocol Description. Revisiting the past section, where we added additional phases to improve precision, we take this idea and add as many phases in between as possible. In detail, we apply the idea of identifying the k largest values in a distributed (non-binary) search tree: The algorithm starts by drawing a random variable h_i from a geometric distribution, i.e., $\Pr[h_i = h] = \phi^{h-1}(1-\phi)$. Observe that a smaller choice of the failure probability ϕ results in smaller random heights

Algorithm 7. TOP-$k(\phi)$

```
Initialization()
  1. call WEAK SELECT and let $(d, h)$ denote the obtained data item and its height.
  2. Only nodes $i$ with $d_i \le d$ are considered.
  3. Each node $i$ draws a random variable $h_i$, i.i.d. from a geometric distribution with $p := 1 - \phi$
  4. Server defines $\ell := -\infty$, $u := \infty$, $S := \emptyset$
  5. call Top-$k$-Rec$(\ell, u, h)$
  6. Raise an error if $|S| < k$

Top-$k$-Rec$(\ell, u, h)$
  1. if $h = 0$ then
  2.    if $|S| = k$ then return $S$,
  3.    else end recursion
  4. Server probes sensor nodes $i$ with $\ell < d_i < u$ and $h_i \ge h$
     Let $r_1 < \ldots < r_j$ be the responses
  5. call Top-$k$-Rec$(\ell, r_1, h - 1)$
  6. $S \leftarrow S \cup r_1$
  7. for $i = 1$ to $j - 1$ do
  8.    call Top-$k$-Rec$(r_i, r_{i+1}, h - 1)$
  9.    $S \leftarrow S \cup r_{i+1}$
 10. call Top-$k$-Rec$(r_j, u, h - 1)$
```

h_i, but a larger expected number of 'siblings'. To perform an inorder treewalk the server identifies the siblings of a node with respect to the current path of the protocol by broadcasting values ℓ, u and h to identify all nodes i with values $\ell < d_i < u$ and a height of $h_i \ge h$. The protocol is shown in Algorithm 7.

Analysis. To prove that the TOP-k operation uses $\mathcal{O}(k + \log\log n)$ messages in expectation simply observe that the probability to send a message for a sensor node within the Top-k is 1. Consider the remaining nodes, i.e., consider the set V' of nodes that are not in the Top-k. To bound the number of messages, we simply upper bound the number of messages used to find the maximum within V'. Since WEAK SELECT gives a data item such that $k \cdot \log^{c_2} n$ nodes remain, and by the arguments in Theorem 1, it holds:

Lemma 13. *The TOP-k operation uses $\mathcal{O}(k + \log\log n)$ messages in expectation.*

We consider the number of rounds, which concludes the proof for Lemma 4.

Lemma 14. *The TOP-k operation uses at most $\mathcal{O}(k + \log\log n)$ exp. rounds.*

Proof. We structure the proof in two steps: First, we analyze the number of rounds used to determine the minimum (i.e., the data item with rank 1), and second, the number of communication rounds used to determine the Top-k.

Observe that the algorithm uses a linear amount of steps (linear in h), until it reaches $h_{\min} = 1$, after which the minimum is found. Afterwards, in each step the algorithm recursively probes for nodes successively smaller than the currently largest values, that are added to the output set S. Note that by the analysis in Theorem 2, the number of nodes that send a message in expectation in each round is $(1 - \phi)/\phi$ (for $h < \log_{1/\phi}(n)$). Thus, in each communication round there are $\Omega(\frac{1}{\phi})$ nodes in expectation that send a message, such that after an expected number of $\mathcal{O}(\phi \cdot k)$ rounds the Top-k protocol terminates. $\square$

Similar to the previous section, we can state a result for a one-shot computation of Top-k. This result might be of independent interest.

Corollary 2. *For $\phi := \frac{1}{2}$ and $h := \log n$, the Top-k protocol uses an amount of $k + \log n + 2$ messages in expectation and $\mathcal{O}(k + \log n)$ rounds.*

8 Future Research Perspectives

We see further applications of the *Sketch* in our data structure. Among others, one (direct) application is to output an (axis aligned) bounding box for the given data points. An interesting problem to consider is as follows: Each sensor node observes its position in the plane and our task is to output the (sensor nodes that form the) convex hull. The sensor nodes are mobile, i.e., they can move between two time steps by a bounded speed. Let n_h denote the number of nodes on the convex hull and observe that $\Omega(n_h)$ messages are needed to determine the output. With the algorithms in this paper the convex hull can be computed using $\mathcal{O}(n_h \cdot \log n)$ messages. We ask whether we may apply (some variant of) our *Sketch* such that $\mathcal{O}(n_h \cdot \log\log n)$ messages are sufficient to determine the points on the convex hull.

Revisiting the analysis of our data structure we observe that we reduce the communication especially if the adversary changes only a few data items at a time. Additionally, we analyze a worst-case adversary who changes data items with a small rank, i.e., with a polylogarithmic rank. It might be of interest to consider restrictions of the adversary to prove stronger bounds: The node which observes a new data item is chosen uniformly at random, or the new data item observed is 'close' to the old value.

References

1. Arackaparambil, C., Brody, J., Chakrabarti, A.: Functional monitoring without monotonicity. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 95–106. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02927-1_10
2. Babcock, B., Olston, C.: Distributed Top-K monitoring. In: ACM SIGMOD International Conference on Management of Data, pp. 28–39. ACM (2003)
3. Bemmann, P., et al.: Monitoring of domain-related problems in distributed data streams. In: Das, S., Tixeuil, S. (eds.) SIROCCO 2017. LNCS, vol. 10641, pp. 212–226. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72050-0_13
4. Biermeier, F., Feldkord, B., Malatyali, M., Meyer auf der Heide, F.: A communication-efficient distributed data structure for Top-k and k-select queries. In: Solis-Oba, R., Fleischer, R. (eds.) WAOA 2017. LNCS, vol. 10787, pp. 285–300. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89441-6_21
5. Cormode, G., Korn, F., Muthukrishnan, S., Srivastava, D.: Space-and time-efficient deterministic algorithms for biased quantiles over data streams. In: 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 263–272. ACM (2006)

6. Cormode, G., Korn, F., Muthukrishnan, S., Srivastava, D.: Effective computation of biased quantiles over data streams. In: 21st International Conference on Data Engineering, pp. 20–31. IEEE (2005)
7. Cormode, G., Muthukrishnan, S., Yi, K.: Algorithms for distributed functional monitoring. ACM Trans. Algorithms **7**(2), 21 (2011)
8. Cormode, G., Muthukrishnan, S., Yi, K.: Algorithms for distributed functional monitoring. In: 19th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1076–1085. Society for Industrial and Applied Mathematics (2008)
9. Mäcker, A., Malatyali, M., Meyer auf der Heide, F.: Online Top-k-position monitoring of distributed data streams. In: 29th International Parallel and Distributed Processing Symposium, pp. 357–364. IEEE (2015)
10. Mäcker, A., Malatyali, M., Meyer auf der Heide, F.: On competitive algorithms for approximations of Top-k-position monitoring of distributed streams. In: 30th International Parallel and Distributed Processing Symposium, pp. 700–709. IEEE (2016)
11. Madden, S., Franklin, M., Hellerstein, J., Hong, W.: The design of an acquisitional query processor for sensor networks. In: ACM SIGMOD International Conference on Management of Data, pp. 491–502. ACM (2003)
12. Marberg, J., Gafni, E.: An optimal shout-echo algorithm for selection in distributed sets. In: 23rd Allerton Conference on Communication, Control, and Computing (1985)
13. Muthukrishnan, S.: Data Streams: Algorithms and Applications. Now Publishers Inc. (2005)
14. Rotem, D., Santoro, N., Sidney, J.: Shout echo selection in distributed files. Networks **16**(1), 77–86 (1986)
15. Yi, K., Zhang, Q.: Optimal tracking of distributed heavy hitters and quantiles. Algorithmica **65**(1), 206–223 (2013)
16. Zengfeng, H., Yi, K., Zhang, Q.: Randomized algorithms for tracking distributed count, frequencies, and ranks. In: 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 295–306. ACM (2012)

What Is Known About Vertex Cover Kernelization?

Michael R. Fellows[1], Lars Jaffke[1(✉)], Aliz Izabella Király[1], Frances A. Rosamond[1], and Mathias Weller[2]

[1] Department of Informatics, University of Bergen, Bergen, Norway
{michael.fellows,lars.jaffke,aliz.kiraly,frances.rosamond}@uib.no
[2] CNRS, LIGM, Université Paris EST, Marne-la-Vallée, France
mathias.weller@u-pem.fr

Abstract. We are pleased to dedicate this survey on kernelization of the Vertex Cover problem, to Professor Juraj Hromkovič on the occasion of his 60th birthday. The Vertex Cover problem is often referred to as the *Drosophila* of parameterized complexity. It enjoys a long history. New and worthy perspectives will always be demonstrated first with concrete results here. This survey discusses several research directions in Vertex Cover kernelization. The *Barrier Degree* of Vertex Cover is discussed. We have reduction rules that kernelize vertices of small degree, including in this paper new results that reduce graphs almost to minimum degree five. Can this process go on forever? What is the minimum vertex-degree barrier for polynomial-time kernelization? Assuming the ETH Hypothesis, there is a minimum degree barrier. The idea of *automated kernelization* is discussed. We here report the first experimental results of an AI-guided branching algorithm for Vertex Cover whose logic seems amenable for application in finding reduction rules to kernelize small-degree vertices. The survey highlights a central open problem in parameterized complexity. Happy Birthday, Juraj!

1 Introduction and Preliminaries

A vertex cover of a graph is a subset of its vertices containing at least one endpoint of each of its edges. The Vertex Cover problem asks, given a graph G and an integer k, whether G contains a vertex cover of size at most k.

The study of the Vertex Cover problem lies at the roots of the theory of NP-completeness: It is one of Karp's 21 NP-complete problems [46] and plays a central role in the monograph of Garey and Johnson [34]. However, interest in the Vertex Cover problem reaches far beyond pure theory. One reason is that it naturally models *conflict resolution*,[1] a problem occurring in numerous scientific disciplines, with an international workshop devoted to it [2]. Other applications include classification methods (see, e.g., [35]), computational biology (e.g.,

M. R. Fellows, L. Jaffke, A. I. Király and F. A. Rosamond—Acknowledge support from the Bergen Research Foundation (BFS).

[1] In the textbook [16], the problem was entertainingly introduced as 'Bar Fight Prevention'.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 330–356, 2018.
https://doi.org/10.1007/978-3-319-98355-4_19

[13]), and various applications follow from the duality of VERTEX COVER with the CLIQUE problem (see, e.g., [1]). The latter finds numerous applications in fields such as computational biology and bioinformatics [9,47,48,64,71], computational chemistry [22,52,69] and electrical engineering [15,39].

In *parameterized/multivariate algorithmics* [16,20,60], the objects of study are computational problems whose instances are additionally equipped with a integer k, the *parameter*, typically expressing some structural measure of the instance of the problem. The goal is to design algorithms for hard problems whose runtime confines the combinatorial explosion to the parameter k rather than the size of the input. A parameterized problem is called *fixed-parameter tractable* if it can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$ where f is some computable function, k the parameter and n the input size. The second central notion in the field of parameterized algorithms is that of a *kernelization* [19,21,27], a polynomial-time algorithm (usually described as a set of *reduction rules*) that takes as input an instance (I, k) of a parameterized problem and outputs an equivalent instance (I', k'), where $|I'| + k' \leq g(k)$ for some computable function g.[2]

Kernelization (for the first time!) provided a theory of preprocessing with mathematically provable guarantees. On the other end, kernelization has immediate practical implications, as demonstrated by Karsten Weihe's problem [66,67] (see also [24,25]) concerning the train systems in Europe. By the means of two simple reduction rules, graphs (instances) on $10,000$ vertices are reduced to equivalent instances whose connected components are of size at most 50, making the reduced instance solvable exactly even by brute force in reasonable time, after the preprocessing, even though the general problem is NP-hard. Similar reduction rules have been successfully applied in the context of cancer research [4] and spread of virus [23].

The notions of fixed-parameter tractability and kernelization are tightly linked. It has been shown by Cai et al. that a parameterized problem is fixed-parameter tractable if and only if it has a (polynomial-time) kernelization algorithm [10]. Kernelization for the VERTEX COVER problem, which is often referred to as the *Drosophila* of parameterized complexity [20,31,37,60], enjoys a long history. In 1993, the first kernel on $\mathcal{O}(k^2)$ vertices was obtained, and is accredited to Buss [8], with more refined reduction rules given in [3]. Kernels with a linear number of vertices were obtained in various ways. Using classic graph theoretic results, Chor et al. gave a kernel on $3k$ vertices [14] (see also [26]), a kernel on $2k$ vertices was obtained via an LP-relaxation by Chen et al. [11] and another kernel on $2k$ vertices without the use of linear programming was obtained by Dehne et al. [17]. The next series of improvements gave kernels on $2k - c$ vertices [62] and the current champion which is due to Lampis has $2k - c \log k$ vertices [51], where in the latter two c is any fixed constant. Another kernel on $2k - \mathcal{O}(\log k)$

[2] As the focus of this text is on the VERTEX COVER problem, we refer to [36,54] for general surveys on the subject of kernelization and to [56] for a survey on the corresponding lower bound machinery.

vertices was observed in [58]. An experimental evaluation of several of the earlier kernels was carried out in [1].

There is no known subquadratic bound on the number of *edges* in any kernel for Vertex Cover, and the question whether such a kernel exists was a long standing open question in multivariate algorithmics. It was finally shown that up to logarithmic factors, Vertex Cover kernels with a quadratic number of edges are likely to be optimal: Dell and van Melkebeek, building on work of Bodlaender, Downey, Fellows and Hermelin [6,7], also Fortnow and Santhanam [33], showed that there is no kernel on $\mathcal{O}(n^{2-\varepsilon})$ bits, for any $\varepsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [18]. The latter would imply that the polynomial hierarchy collapses to its third level [70] which is widely considered to be implausible by complexity theorists.[3]

In another line of research, following the *parameter ecology* program [29], the existence of kernels for Vertex Cover w.r.t. parameters that take on smaller values than the vertex cover number was studied. Such parameterizations are typically referred to as *structural* parameterizations of Vertex Cover. The first such result is due to Jansen and Bodlaender who gave a kernel on $\mathcal{O}(\ell^3)$ vertices, where ℓ is the size of a feedback vertex set of the graph [45]. Further results include polynomial kernels where the parameter is the size of an odd cycle traversal or a König deletion set [50], the size of vertex deletion sets to maximum degree at most two [55], pseudoforest [32] and d-quasi forest [40]. Using the above mentioned lower bound machinery, it was shown that there is no kernel polynomial in the size of a vertex deletion set to chordal or perfect graphs unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [5,29].

As Vertex Cover is the primary intellectual "lab animal" in parameterized complexity, *new and worthy perspectives will always be demonstrated first with concrete results here.* We discuss several research directions in (Vertex Cover) kernelization. The first one is based on the observation that several reduction rules are known to kernelize vertices of small degree [8,30,63]; a natural question is whether this process can go on 'forever', i.e., whether we can find, for any fixed constant $d \in \mathbb{N}$, a set of reduction rules that kernelize in polynomial time to a reduced graph (the kernel) of minimum degree d. On the negative side, we observe that unless the Exponential-Time Hypothesis [42,43] fails, this is not the case even if the exponent in the polynomial-time kernelization is some arbitrary function of d. On the positive side, we give a clear account of reduction rules for Vertex Cover that were first observed by Fellows and Stege [30] that kernelize instances to minimum degree 'almost five' (see Theorem 5 for the exact statement) and discuss how this question is closely related to finding faster fpt-algorithms for Vertex Cover, a question that lies at the very heart of parameterized complexity research.

In the light of the ongoing machine-learning and artificial intelligence revolution, one might wonder whether AI could assist in the search for new reduction rules of parameterized problems as well. While this question seems far out, we

[3] We also refer to [44, pages 19f] and [68, Appendix A] for brief and convincing accounts of the implausibility of $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

report first experimental results of an AI-guided branching algorithm for Vertex Cover whose logic seems amenable for application in finding new reduction rules to kernelize to increasing minimum degree.

The rest of this paper is organized as follows. In the remainder of this section, we give preliminary definitions and introduce the necessary background. In Sect. 2 we review some classic Vertex Cover kernels. Section 3 is devoted to the topic of kernelizing small-degree vertices. We there give a description of reduction rules observed by Fellows and Stege [30] (see also [63]). In Sect. 4 we report results on an AI-guided branching algorithm whose ideas might lay the foundations of *automatically generated* reduction rules for Vertex Cover. We conclude with an open problem in Sect. 5.

Technical Preliminaries and Notation. For two integers a and b with $a < b$, we let $[a..b] := \{a, a+1, \dots, b\}$ and for a positive integer a, we let $[a] := [1..a]$.

Throughout the paper, each graph is finite, undirected and simple. Let G be a graph. We denote the vertex set of G by $V(G)$ and the edge set of G by $E(G) \subseteq \binom{V}{2}$. For a vertex $v \in V(G)$, we denote by $N(v)$ the *(open) neighborhood* of G, i.e., $N(v) := \{w \mid \{v, w\} \in E(G)\}$. The *degree of* v is the size of the neighborhood of v, i.e., $\deg(v) := |N(v)|$. We define the *closed neighborhood* of v as $N[v] := N(v) \cup \{v\}$. For a set of vertices $W \subseteq V(G)$, we let $N(W) := \bigcup_{w \in W} N(w)$ and $N[W] := N(W) \cup W$. For a set of vertices $\{v_1, \dots, v_r\}$, we use the shorthand $N(v_1, \dots, v_r) := N(\{v_1, \dots, v_r\})$.

For two graphs G and H, we denote by $H \subseteq G$ that H is a *subgraph* of G, i.e. that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For a vertex set $X \subseteq V(G)$, we denote by $G[X]$ the subgraph of G *induced* by X, i.e., $G[X] := (X, E(G) \cap \binom{X}{2})$. We let $G - X := G[V(G) \setminus X]$ and we use the shorthand $G - v$ for $G - \{v\}$. Furthermore, we define for an edge set $F \subseteq E(G)$, $G - F := (V(G), E(G) \setminus F)$ and for a set $\bar{F} \subseteq \binom{V(G)}{2}$, we let $G + \bar{F} := (V(G), E(G) \cup \bar{F})$.

A subgraph $P \subseteq G$ is called a *path* if all its vertices have degree at most two in P and there are precisely two distinct vertices in $V(P)$ that have degree one in P, called the *endpoints* of P. For $s, t \in V(G)$, a path is called (s,t)*-path* if it is a path with endpoints s and t.

We call two edges $e, f \in E(G)$ *adjacent* if they share an endpoint, i.e., if there exist vertices $v, w, x \in V(G)$ such that $e = \{v, w\}$ and $f = \{v, x\}$. A *matching* is a set of pairwise non-adjacent edges. A *co-matching* is a set of pairwise non-adjacent edges in the edge-complement of G, i.e., the graph whose vertex set is $V(G)$ and whose edge set is the complement of $E(G)$. We say that a matching (or co-matching) M *saturates* a set of vertices $W \subseteq V(G)$, if for all $v \in W$, there is a pair $\{v, w\} \in M$.

Given a set of vertices $X \subseteq V(G)$, we call the operation of adding to G a new vertex x with neighborhood $N(X)$ and deleting all vertices in X the *contraction* of X.

Exponential-Time Hypothesis (ETH). In 2001, Impagliazzo and Paturi made a conjecture about the complexity of 3-Sat, the problem of determining whether a given Boolean formula in conjunctive normal form with clauses of size at most 3 has a satisfying assignment. This conjecture is known as the *Exponential-Time*

Hypothesis (*ETH*) and has lead to a plethora of conditional lower bounds, see, e.g., the survey [53] or [16, Chap. 14]. Formally, ETH can be stated as follows.[4]

Conjecture 1 (ETH [42,43]). There is an $\varepsilon > 0$ such that 3-SAT on n variables cannot be solved in time $\mathcal{O}^*(2^{\varepsilon n})$.

2 Standard Methods

In this section, we review some classic results in VERTEX COVER kernelization. In particular, we discuss the Buss kernel [8] in Subsect. 2.1. Subsection 2.2 is devoted to the kernel based on the notion of a crown decomposition [14,26] (see Definition 1). A linear-programming-based kernel [11] is discussed in Subsect. 2.3.

We would like to remark that the technical parts of the expositions given in the remainder of this section are based on [16, Sections 2.2.1, 2.3 and 2.5] and we refer to this text for several details.

2.1 Buss Kernelization

The first kernel for VERTEX COVER appeared several years before the notion of kernelization was formally introduced and is attributed to Buss [8]. It relies on two observations. The first one is that by definition, there is no need to include an isolated vertex in a vertex cover, as it does not have any incident edges that need to be covered.

Reduction R.1. *If G has an isolated vertex v, then reduce (G, k) to $(G - v, k)$.*

The second observation is that, if G has a vertex v of degree more than k, then we have no choice but to include v in any size-k vertex cover of G: If we did not include v, we would have to include all of its at least $k + 1$ neighbors, exceeding the budget of k vertices we are given. Hence, G has a vertex cover of size k if and only if $G - v$ has a vertex cover of size $k - 1$, so we have observed that the following reduction rule is *safe*, meaning that the original instance is a YES-instance if and only if the reduced instance is a YES-instance.

Reduction R.2. *If G has a vertex v with $\deg(v) > k$, then reduce (G, k) to $(G - v, k - 1)$.*

Now, after exhaustively applying Reduction R.2, G has maximum degree at most k, so if G contains more than k^2 edges, then we are dealing with a NO-instance: It is not possible to cover more than k^2 edges with k vertices of degree at most k. On the other hand, if (G, k) is a YES-instance, then G has a vertex cover X of size at most k. After exhaustively applying Reduction R.1, G does not contain any isolated vertices so we can assume that every vertex of $V(G) \setminus X$ has a neighbor in X. Since the maximum degree of G is at most k, we can conclude that $|V(G) \setminus X| \leq k^2$, which implies that $|V(G)| \leq k^2 + k$. Hence, if G has more than $k^2 + k$ vertices, we can again conclude that we are dealing with a NO-instance. Since Reductions R.1 and R.2 clearly run in polynomial time, we have the following theorem.

[4] The $\mathcal{O}^*$-notation suppresses polynomial factors in n.

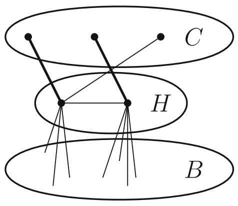


Fig. 1. Illustration of a crown decomposition (Definition 1). The fat edges in $G[C, H]$ show a matching saturating H.

Theorem 1 (Buss and Goldsmith [8]). VERTEX COVER *admits a kernel with at most $k^2 + k$ vertices and k^2 edges.*

2.2 Crown Reduction

The key insight above was that any vertex of degree at least $k + 1$ has to be contained in any size-k vertex cover of a graph. The kernel we present in this section follows a similar motivation. The goal is to identify a set of vertices that we can always *assume* to be contained in a size-k vertex cover of a graph. In other words, we want to find a set of vertices S, such that if G contains a vertex cover of size k then G contains a vertex cover of size k that contains S. The process of identifying such a set S is based on a structural decomposition of the input graph, called the *crown decomposition*. Formally, a crown decomposition is defined as follows and we illustrate it in Fig. 1.

Definition 1 (Crown Decomposition). *Let G be a graph. A* crown decomposition *of G is a partition (C, H, B) of $V(G)$, where C is called the* crown*, H the* head *and B the* body*, such that the following hold.*

(i) C is a non-empty independent set in G.
(ii) There are no edges between vertices in C and vertices in B.
(iii) $G[C, H]$ contains a matching that saturates H.

The motivation for using the above definition in VERTEX COVER kernelization is as follows. Suppose we are given a crown decomposition (C, H, B) of G and consider the bipartite graph $G[C, H]$. Clearly, any vertex cover of G has to cover the edges in $G[C \cup H]$. However, by (iii) we know that there is a matching in $G[C, H]$ saturating H, hence any vertex cover of $G[H, C]$ has size at least $|H|$. On the other hand, H is a vertex cover of $G[C, H]$ and since C is independent by (i), of $G[C \cup H]$. This allows us to conclude that G has a vertex cover of size k if and only if $G - (C \cup H)$ has a vertex cover of size $k - |H|$. Hence, the following reduction rule is safe.

Reduction R.3. *If G has a crown decomposition (C, H, B), then reduce (G, k) to $(G - (C \cup H), k - |H|)$.*

However, two questions remain. Namely whether we can find a crown decomposition of a graph in polynomial time and how to obtain the linear bound on the number of vertices in the resulting kernel. Both questions are answered by the following lemma whose proof is based on classic results in graph theory by König [49] and Hall [38], and polynomial-time algorithms for bipartite matching such as the classic algorithm due to Hopcroft and Karp [41].[5]

Lemma 1 (Lemma 2.14 in [16] based on [14]). *Let G be a graph on at least $3k+1$ vertices. There is a polynomial-time algorithm that either*

(1) finds a matching of size at least $k+1$ in G; or
(2) finds a crown decomposition of G.

Now, in Case 1 we can immediately conclude that (G, k) is a No-instance and in Case 2 we can apply Reduction R.3. By an exhaustive application of Lemma 1 in combination with Reduction R.3 (and Reduction R.1 to get rid of isolated vertices), we have the following theorem.

Theorem 2 (Chor et al. [14]). Vertex Cover *admits a kernel with at most $3k$ vertices.*

2.3 LP-Based Kernel

The Vertex Cover problem is one of many NP-hard problems that can be expressed as an integer linear program [61], a fact which is commonly exploited in the field of approximation algorithms [65]. In this section, we show how to use linear programming to obtain a kernel for Vertex Cover on at most $2k$ vertices. We first recall how to formulate Vertex Cover as an integer linear program.

For each vertex $v \in V(G)$, we introduce a variable $x_v \in \{0,1\}$ with the interpretation that $x_v = 1$ if and only if the vertex v is included in the vertex cover witnessed by a solution to the (integer) linear program. We can then formulate the constraints in a natural way, directly applying the definition of vertex covers: For each edge $uv \in E(G)$, the requirement that at least one of u and v has to be contained in the solution translates to the constraint $x_u + x_v \geq 1$. Since we are looking for a vertex cover of minimum size, the objective function minimizes the sum over all x_v's.

$$\begin{aligned} & \min \sum_{v \in V(G)} x_v \\ \text{subject to} \quad & x_u + x_v \geq 1 \quad \forall uv \in E(G) \qquad (1) \\ & x_v \in \{0,1\} \quad \forall v \in V(G) \qquad (2) \end{aligned}$$

To make the program feasible to compute, we relax the integrality constraints (2) to $x_v \in \mathbb{R}$, $x_v \geq 0$. (Note that we can drop the constraints $x_v \leq 1$ since the

[5] For a more fine-grained analysis one could apply the faster algorithm [57].

objective function is a minimization.) The resulting linear program is solvable in polynomial time, but may not always return a feasible solution for the original Vertex Cover instance. However, we are chasing a different goal here, a kernelization algorithm.

Given an optimal solution $(x_v)_{v \in V(G)}$ of the (relaxed) linear program, we define the sets $V_0 := \{v \in V(G) \mid x_v < \frac{1}{2}\}$, $V_1 := \{v \in V(G) \mid x_v > \frac{1}{2}\}$, and $V_{\frac{1}{2}} := \{v \in V(G) \mid x_v = \frac{1}{2}\}$. The key ingredient is the following theorem due to Nemhauser and Trotter [59].

Theorem 3 (Nemhauser and Trotter [59]). *There is a minimum vertex cover X of G such that $V_1 \subseteq X \subseteq V_1 \cup V_{\frac{1}{2}}$.*

We derive a reduction rule from Theorem 3. First, we note that in any Yes-instance of Vertex Cover, $\sum_{v \in V(G)} x_v \leq k$. Furthermore, let X be a vertex cover of G of size k with $V_1 \subseteq X$ and $X \cap V_0 = \emptyset$ (whose existence is guaranteed by Theorem 3), then $X \setminus V_1$ is a vertex cover of $G - (V_0 \cup V_1)$ of size $k - |V_1|$. Conversely, if $G - (V_0 \cup V_1)$ has a vertex cover X' of size k', we observe that by the constraints (1), for any edge $vw \in E(G)$ with $v \in V_0$, we have that $w \in V_1$. Hence, $X' \cup V_1$ is a vertex cover of G of size $k' + |V_1|$. We have argued that the following reduction rule is safe.

Reduction R.4. *Let $(x_v)_{v \in V(G)}$, V_0, $V_{\frac{1}{2}}$ and V_1 be as above. If $\sum_{v \in V(G)} x_v > k$, then conclude that we are dealing with a No-instance. Otherwise, reduce (G, k) to $(G - V_0 \cup V_1, k - |V_1|)$.*

The number of vertices in the reduced instance after applying Reduction R.4 is

$$|V(G) \setminus (V_0 \cup V_1)| = \left|V_{\frac{1}{2}}\right| = \sum\nolimits_{v \in V_{\frac{1}{2}}} 2x_v \leq 2 \cdot \sum\nolimits_{v \in V(G)} x_v \leq 2k,$$

so we have obtained the following kernel for Vertex Cover.[6]

Theorem 4 (Chen et al. [11]). Vertex Cover *admits a kernel with at most $2k$ vertices.*

3 Towards the Barrier – What Is the Maximum Minimum Vertex Degree of the Kernel that Can Be Achieved in Polynomial Time?

In the previous section, we have seen that by Reduction R.2 we can kernelize all vertices whose degree is larger than the target value k of the given vertex cover

[6] We would like to remark that while Linear Programming can be solved in polynomial time (and hence our reduction runs in polynomial time), the corresponding algorithms are often slow in practice. However, for the case of Vertex Cover there is good news: One can show that a solution of the above linear program can be found via a reduction to Bipartite Matching (see, e.g., [16, Sect. 2.5]) which has fast practical algorithms.

instance. Hence, after applying this rule exhaustively there will be no vertex of degree larger than k in the kernelized instance. But what about vertices of small degree? Vertices of degree zero, i.e., isolated vertices, can be removed from a Vertex Cover instance according to Reduction R.1. Furthermore, we will see below that there are fairly simple reduction rules that kernelize vertices of degree one and two (see Reductions R.5 and R.6). A natural question arises: Can this process go on 'forever', i.e., can we, for any fixed constant $d \in \mathbb{N}$, give a reduction rule that kernelizes all vertices of degree d from a given Vertex Cover instance?

The answer to this question is *probably not*—even if the degree of the polynomial in the runtime of the kernelization algorithm can depend on d: It is well-known (see, e.g., [16,20,28]) that unless ETH fails, there is some barrier constant $\zeta_{VC} > 0$ such that the fastest possible algorithm for Vertex Cover runs in time $(1 + \zeta_{VC})^k \cdot n^{\mathcal{O}(1)}$. If we could kernelize Vertex Cover in polynomial time to arbitrarily large minimum degree, one could devise a straightforward branching algorithm that runs in time $(1 + \zeta_{VC} - \varepsilon)^k \cdot n^{\mathcal{O}(1)}$, for some $0 < \varepsilon < \zeta_{VC}$, where ε can be arbitrarily close to the value of ζ_{VC}. We coin the corresponding integer $\delta_{VC} \in \mathbb{N}$ the *barrier degree* of Vertex Cover kernelization and now prove formally its existence (assuming ETH).

Proposition 1. *Unless ETH fails, there is some constant $\delta_{VC} \in \mathbb{N}$, such that Vertex Cover cannot be kernelized to instances of minimum degree δ_{VC}.*

Proof. Using standard arguments about branching algorithms (see, e.g., [16, Chap. 3]) one can show that there is an algorithm solving vertex cover in time $\lambda^k \cdot n^{\mathcal{O}(1)}$, where λ satisfies

$$\lambda \leq \lambda^d(\lambda - 1), \tag{3}$$

if the input graph always has a vertex of degree at least d to branch on. Now suppose that the statement of the proposition is false, then we can guarantee the existence of such a vertex for constant but arbitrarily large d (with only polynomial time overhead at each stage of the branching). Now let $\varepsilon > 0$ with $\varepsilon < \zeta_{VC}$. (Note that this implies that $\varepsilon < 1$ as $\zeta_{VC} < 0.2738$ [12].) We substitute λ with $(1 + \varepsilon)$ in (3) and obtain:

$$1+\varepsilon \leq (1+\varepsilon)^d \varepsilon \iff (1+\varepsilon)^d \geq \frac{1+\varepsilon}{\varepsilon} \iff d\log(1+\varepsilon) \geq \log\left(\frac{1+\varepsilon}{\varepsilon}\right)$$
$$\iff d \geq \frac{\log\left(\frac{1+\varepsilon}{\varepsilon}\right)}{\log(1+\varepsilon)} = \frac{\log(1+\varepsilon) - \log(\varepsilon)}{\log(1+\varepsilon)} = 1 - \frac{\log(\varepsilon)}{\log(1+\varepsilon)}.$$

This shows that for any such ε, there is a constant $d_\varepsilon \in \mathbb{N}$ such that, if we could kernelize Vertex Cover to minimum degree d_ε, then we could solve it in $(1 + \varepsilon)^k \cdot n^{\mathcal{O}(1)}$ time, where $\varepsilon < \zeta_{VC}$ by our choice. This contradicts ETH by, e.g., [28, Theorem 1]. □

The proof of Proposition 1 also provides some very natural motivation for the question of kernelizing Vertex Cover to larger and larger minimum degree;

Table 1. (Dependence on k of the) runtime of the resulting simple branching FPT-algorithm when using a kernelization algorithm to minimum degree d, for several values of d, versus the current fastest known FPT-algorithm for Vertex Cover [12].

Chen et al. [12]	$d = 5$	$d = 6$	$d = 7$	$d = 10$	$d = 25$	$d = 100$
1.2738^k	1.3247^k	1.2852^k	1.2555^k	1.1975^k	1.1005^k	1.0346^k

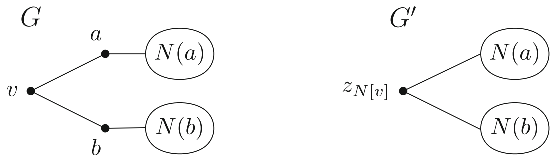


Fig. 2. Illustration of Reduction R.6. Note that, for our argument, it is immaterial whether the edge $\{a, b\}$ is present in G or not.

such kernels immediately provide new FPT-algorithms for the problem. In particular, kernelizing to minimum degree seven would already improve upon the current best known algorithm for Vertex Cover, yielding first progress in a very attractive research question in over a decade! We illustrate the runtime of such algorithms for several concrete values of d in Table 1.

In the remainder of this section, we present a set of reduction rules that were first observed by Fellows and Stege [30] to kernelize a vertex cover instance to minimum degree 'almost five', in the following sense: We show that a vertex can be kernelized if its degree is at most three or its degree is four and there are more than two edges between the vertices in its neighborhood.

We begin with a simple rule to deal with degree-one vertices and show that it is safe.

Reduction R.5. *If G has a pendant edge $\{u, v\}$ with $\deg(u) = 1$, then reduce (G, k) to $(G - \{u, v\}, k - 1)$.*

Proposition 2. *Reduction R.5 is safe, i.e., if G has a pendant edge $\{u, v\}$ with $\deg(u) = 1$, then G has a vertex cover of size k if and only if $G - \{u, v\}$ has a vertex cover of size $k - 1$.*

Proof. ($\Rightarrow$) Suppose G has a vertex cover X^* of size k. Since $\{u, v\}$ is an edge of G, at least one of u and v is contained in X^*. If $v \notin X^*$, then we let $X := X^* \setminus \{u\} \cup \{v\}$. Note that X is a vertex cover since v is the only neighbor of u. If $v \in X^*$, we simply let $X := X^*$. Since $v \in X$, $X \setminus \{v\}$ is a vertex cover of $G - \{u, v\}$ of size $k - 1$.

($\Leftarrow$) Let X' be a vertex cover of $G - \{u, v\}$ of size $k - 1$. We observe that any edge in $E(G) \setminus E(G - \{u, v\})$ is incident with v and conclude that $X' \cup \{v\}$ is a vertex cover of G of size k. □

The next reduction rule takes care of vertices of degree two and is illustrated in Fig. 2.

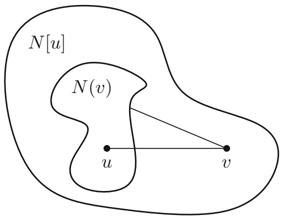


Fig. 3. Illustration of the situation of Reduction R.7.

Reduction R.6. *If G has a vertex v with $\deg(v) = 2$, then reduce (G, k) to $(G', k-1)$, where G' is the graph obtained from G by contracting $N[v]$ to a single vertex.*

Proposition 3. *Reduction R.6 is safe, i.e., G has a vertex cover of size k if and only if G' has a vertex cover of size $k-1$.*

Proof. Throughout the proof, we denote the neighborhood of v in G by $N(v) = \{a, b\}$ and the vertex in G' that was created due to the contraction of $N[v]$ by $z_{N[v]}$.

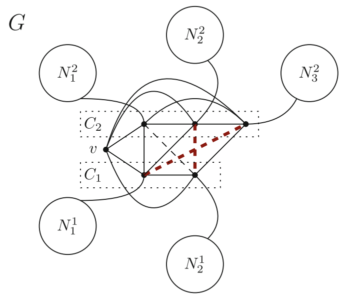
($\Rightarrow$) We observe that each edge in $E(G') \setminus E(G)$ has an endpoint in $\{z_{N[v]}\} \cup N(a, b)$. Let X be a vertex cover of G of size k. If $X \cap N(v) = \emptyset$, then $\{v\} \cup N(a, b) \subseteq X$ and we can conclude that $X \setminus \{v\}$ is a vertex cover of G'. If $N(v) \subseteq X$, then $X' := X \setminus N(v) \cup \{z_{N[v]}\}$ is a vertex cover of G'. (Note that in this case, X' has size at most $k-1$ as well.) If X contains precisely one vertex from $N(v)$, assume w.l.o.g. that $X \cap N(v) = \{a\}$, then $v \in X$ (otherwise the edge $\{v, b\}$ is not covered), so $X \setminus \{v, a\} \cup \{z_{N[v]}\}$ is a vertex cover of G' of size $k-1$.

($\Leftarrow$) Let X' be a vertex cover of G' of size $k-1$. We only have to distinguish the cases when $z_{N[v]} \in X'$ and when $z_{N[v]} \notin X'$. In the former case, $X' \setminus \{z_{N[v]}\} \cup \{a, b\}$ is a vertex cover of G, since each edge in $E(G) \setminus E(G')$ is incident with a vertex in $\{a, b\}$. In the latter case, $N(a, b) \subseteq X'$ since $z_{N[v]} \notin X'$, and we have that $X' \cup \{v\}$ is a vertex cover of G, as each edge in $E(G) \setminus E(G')$ has an endpoint in $\{v\} \cup N(a, b)$. In both cases, the size of the resulting vertex cover is k. □

Before we proceed with kernelizing vertices of degree larger than two, we require two auxiliary reduction rules. These will be crucially used to argue that we can exclude certain structures appearing in the subgraphs induced by the neighborhoods of small-degree vertices, and in the end will enable us to formulate reduction rules for vertices of degree three and several cases of degree-four vertices.

Reduction R.7. *If G has two adjacent vertices u and v such that $N(v) \subseteq N[u]$, then reduce (G, k) to $(G - u, k-1)$.*

For an illustration of the situation of Reduction R.7, see Fig. 3.



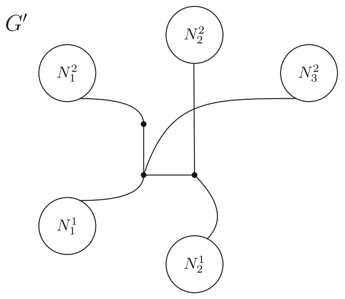


Fig. 4. Illustration of Reduction R.8. Non-edges in $G[C_1, C_2]$ are displayed with dotted lines and the bold dotted lines represent a co-matching in $G[C_1, C_2]$ saturating C_1.

Proposition 4. *Reduction R.7 is safe, i.e., if G has two adjacent vertices u and v, and $N(v) \subseteq N[u]$, then G contains a vertex cover of size k if and only if $G - u$ has a vertex cover of size $k - 1$.*

Proof. ($\Rightarrow$) Suppose G has a vertex cover X^* of size k. If $u \notin X^*$ then $N(u)$ must be in X^*, so by assumption, it contains $N[v] \setminus \{u\}$. But then, $X := X^* \setminus \{v\} \cup \{u\}$ is also a vertex cover of G of size k, so we can assume that $u \in X$. Then, $X \setminus \{u\}$ is a vertex cover of $G - u$ of size $k - 1$.

($\Leftarrow$) is immediate since for any vertex cover X' of $G - u$, $X' \cup \{u\}$ is a vertex cover of G. $\square$

The following reduction rule captures [30, Reductions R.4 and R.5] and is illustrated in Fig. 4. Note that due to its complexity, it will only be executed for vertices whose degree is bounded by some fixed constant α (independent of k).

Reduction R.8. *Suppose G has a vertex v such that the following hold. There is a partition (C_1, C_2) of $N(v)$ where*

(i) for $i \in [2]$, $G[C_i]$ is a clique.

Furthermore, there is a co-matching M of $G[C_1, C_2]$ such that

(ii) M saturates C_1 and
(iii) $G[V(M) \cap C_1, V(M) \cap C_2] + M$ is a complete bipartite graph.

Then, reduce (G, k) to $(G', k - |M|)$ where G' is the graph obtained from G by deleting v and contracting all pairs in M.

Proposition 5. *Reduction R.8 is safe, i.e., if its conditions are satisfied, then G contains a vertex cover of size k if and only if G' contains a vertex cover of size $k - |M|$.*

Proof. For $i \in [2]$, we let $M_i := V(M) \cap C_i$. For notational convenience, we will assume that the vertices that are kept in the contraction of the non-edges M are the vertices M_1 ($= C_1$, as M saturates C_1). Since by the assumption (i) of Reduction R.8, C_1 and C_2 induce cliques in G, we make

Observation 1. *Any vertex cover in G has to use at least $|C_i|-1$ vertices from C_i, where $i \in [2]$, and any vertex cover in G' has to use at least $|C_1|-1$ vertices from C_1.*

We now prove the proposition by a case analysis on the structure of the intersection of vertex covers of G and G' with $N_G(v)$ and $N_G(v) \setminus M_2$, respectively. Observation 1 will be used later to argue that we covered all possible cases.

Claim 1. *G contains a vertex cover X of size k such that $N_G(v) \subseteq X$ if and only if G' contains a vertex cover X' of size $k-|M|$ where $N_G(v) \setminus M_2 \subseteq X'$.*

Proof. ($\Rightarrow$) Let X be a vertex cover of G of size k such that $N_G(v) \subseteq X$. (Note that we can assume that $v \notin X$.) We have that $X' := X \setminus M_2$ is a vertex cover of $G^* := G - M_2$. By construction, any edge in $E(G') \setminus E(G^*)$ is incident with a vertex in $M_1 \subseteq X'$.

($\Leftarrow$) Let X' be a vertex cover of G' of size $k-|M|$ such that $N_G(v) \setminus M_2 \subseteq X'$. Then, $X := X' \cup M_2$ is a vertex cover of G, since every edge in $E(G) \setminus E(G')$ is either incident with a vertex in M_2 or a neighbor of v. For the latter case, we observe that $N_G(v) \subseteq X$. ⌟

Claim 2. *If G contains a vertex cover X of size k with $|C_i \setminus X| = 1$ for all $i \in [2]$, then G' contains a vertex cover X' of size at most $k-|M|$ with $|C_1 \setminus X'| \leq 1$.*

Proof. Let X be a vertex cover of G of size k and suppose that $C_1 \setminus X = \{v_1\}$ and $C_2 \setminus X = \{v_2\}$. We first observe that $v \in X$, since otherwise the edges $\{v, v_1\}$ and $\{v, v_2\}$ are not covered by X. By the same reasoning we have that $\{v_1, v_2\} \notin E(G)$.

Since M saturates C_1 by the assumption (ii) of Reduction R.8, we have that $v_1 \in M_1$. We prove the claim by a case analysis on the position of v_2.

Case 1 ($v_2 \notin M_2$): The assumption $v_2 \notin M_2$ implies that $M_2 \subseteq X$. We observe that $X^* := X \setminus (\{v\} \cup M_2)$ is a vertex cover of the graph $G^* := G - (\{v\} \cup M_2)$ of size $k - |M| - 1$. Observe that $V(G^*) = V(G')$ and let $X' := X^* \cup \{v_1\}$. By construction, any edge in $E(G') \setminus E(G^*)$ is incident with a vertex in $M_1 \subseteq X'$, hence X' is a vertex cover of G' and clearly, $|X'| = k - |M|$.

Case 2 ($v_2 \in M_2$): In this case, $X' := X \setminus (\{v\} \cup M_2)$ is a vertex cover of the graph $G^* := G - (\{v\} \cup M_2)$ of size $k - |M|$. Let $F' := E(G') \setminus E(G^*)$. Each edge $e' \in F'$ has an endpoint in M_1 by construction. If this endpoint is *not* v_1, then the edge e' is covered by X', since $M_1 \setminus \{v_1\} \subseteq X'$. We can assume that $e' = \{v_1, w\}$ for some $w \in N_{G'}(v_1)$. Since $\{v_1, v_2\} \notin E(G)$, $v_1, v_2 \in V(M)$ and by the assumption (iii) of Reduction R.8 ($G[M_1, M_2] + M$ is a complete bipartite graph), we can infer that $\{v_1, v_2\} \in M$. Hence, each neighbor of v_1 in G' is either a neighbor of v_1 or v_2 in G. Since X is a vertex cover of G with $v_1, v_2 \notin X$, we have that

$$X' = X \cap V(G') \supseteq (N_G(v_1) \cup N_G(v_2)) \cap V(G') = N_{G'}(v_1) \ni w,$$

so in this case, the edge $e' = \{v_1, w\}$ is covered by X' as well.

This concludes the proof of the claim. ⌟

Note that, if G contains a vertex cover X of size k that misses *precisely one* vertex w from $N_G(v)$, then we have that $v \in X$ and we can change that vertex cover to $X^* := X \setminus \{v\} \cup \{w\} \supseteq N_G(v)$ which is also of size k and apply Claim 1 to conclude that G' has a vertex cover of size (at most) $k - |M|$.

The next claim will complete the proof the proposition.

Claim 3. *If G' contains a vertex cover X' of size $k - |M|$ with $|C_1 \setminus X'| = 1$, then G contains a vertex cover X of size k with $|C_i \setminus X| = 1$, for all $i \in [2]$.*

Proof. Let X' be a vertex cover of G' of size $k - |M|$ with $C_1 \setminus X' = \{v_1\}$. Since M saturates C_1, we know that $v_1 \in M_1$. Let $v_2 \in M_2$ such that $\{v_1, v_2\} \in M$. We define

$$X := X' \cup \{v\} \cup (M_2 \setminus \{v_2\}) .$$

First note that X is of size k. Furthermore, any edge in $E(G) \setminus E(G')$ is either incident with v or with a vertex in M_2. The edges of the former type are clearly covered by X, since $v \in X$. Let $e \in E(G) \setminus E(G')$ be an edge incident with a vertex in M_2. If this vertex is not v_2, then it is in X, since $M_2 \setminus \{v_2\} \subseteq X$, so the edge e is covered. Otherwise, we have that $e = \{v_2, w\}$ for some $w \in N_G(v_2)$ and we observe the following. Any neighbor of v_2 in G is a neighbor of v_1 in G' by construction. (Recall that $\{v_1, v_2\} \in M$.) Since $v_1 \notin X'$ and X' is a vertex cover of G' (and combining with previous observations), we know that $X \supseteq X' \supseteq N_{G'}(v_1) \supseteq N_G(v_2) \ni w$, so the edge $e = \{v_2, w\}$ is covered by X as well. ⌟

By Observation 1, Claims 1, 2, and 3 exhaust all possible cases and we have proved Proposition 5. □

We are now ready to kernelize degree-three vertices.

Reduction R.9. *If none of the above reduction rules can be applied and G contains a vertex v of degree 3 with $N(v) = \{a, b, c\}$, then reduce (G, k) to (G', k) where G' is the graph on vertex set $V(G) \setminus \{v\}$ and edge set $(E(G) \cap \binom{V(G')}{2}) \cup F$, where*

$$F := \{\{a, b\}, \{b, c\}\} \cup \{\{a, x\} \mid x \in N_G(b)\} \cup \{\{b, y\} \mid y \in N_G(c)\} \cup \{\{c, z\} \mid z \in N_G(a)\} . \quad (4)$$

We illustrate the above reduction rule in Fig. 5.

Proposition 6. *Reduction R.9 is safe, i.e., if its conditions are satisfied, then G contains a vertex cover of size k if and only if G' contains a vertex cover of size k.*

Proof. We first show that we can assume that there are no edges between the vertices in $N(v)$.

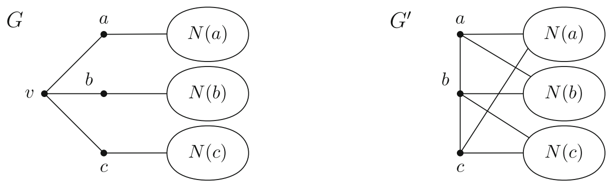


Fig. 5. Illustration of Reduction R.9.

Claim. *If none of the above reduction rules can be applied, then $N(v) = \{a, b, c\}$ is an independent set in G.*

Proof. If $G[N(v)]$ contains at least two edges, then these two edges have a common endpoint, say $x \in N(v)$. But then, $N(v) \subseteq N[x]$, so we could have applied Reduction R.7, a contradiction. If $G[N(v)]$ contains precisely one edge, assume w.l.o.g. that $\{a, b\} \in E(G)$, then we could have applied Reduction R.8 with $C_1 = \{c\}$, $C_2 = \{a, b\}$ and co-matching $M = \{\{a, c\}\}$, again a contradiction. ⌟

Due to the previous claim, we will assume that $N(v) = \{a, b, c\}$ is an independent set throughout the following.

Claim 4. *If G has a vertex cover of size k, then G' has a vertex cover of size at most k.*

Proof. We first observe that, for each edge $e' \in E(G')$, either $e' \in E(G - v)$ or $e' \in F$. Hence, any vertex cover X^* of $G - v$ is a vertex cover of G' if each edge in F has an endpoint in X^*, since by definition, X^* contains an endpoint of each edge in $E(G - v)$.

Let X be a vertex cover of G of size k. If $v \notin X$, then $N(v) = \{a, b, c\} \subseteq X$. By (4), each edge in F has at least one endpoint in $\{a, b, c\}$ and hence in X, so we can conclude that X is a vertex cover of G' of size k.

Suppose $v \in X$ and note for the remainder of the proof that $X \setminus \{v\}$ is a vertex cover of $G - v$ of size $k - 1$. We argue that we can assume that at most one vertex from $N(v)$ is contained in X: For the case that $N(v) \subseteq X$, we can apply the same argument as above to conclude that $X \setminus \{v\}$ is a vertex cover of G' of size $k - 1$. If X contains precisely two vertices from $N(v) = \{a, b, c\}$, assume w.l.o.g. that $\{a, b\} \subseteq X$, then $X' := X \setminus \{v\} \cup \{c\}$ is a vertex cover of G' of size k, since again, X' contains $N(v)$.

We assume that X contains at most one vertex from $N(v)$. If X contains no vertex of $N(v)$, then X must contain all of $N_G(a, b, c)$. Hence the only edges in F that are not covered by X – see (4) – are incident with the vertex b. Together with the fact that $X \setminus \{v\}$ is a vertex cover of $G - v$, we can conclude that $X \setminus \{v\} \cup \{b\}$ is a vertex cover of G'.

From now on, we assume that precisely one vertex of $N(v)$ is contained in the vertex cover X of G. If $a \in X$, then $b, c \notin X$ and hence $N_G(b, c) \subseteq X$. Again, $X \setminus \{v\}$ is a vertex cover of $G - v$ and we observe that any edge in $e' \in F$ that does not have an endpoint in $X \setminus \{v\}$ is incident with the vertex c. By (4), either

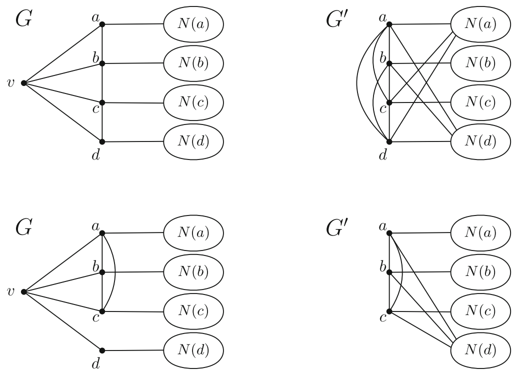


Fig. 6. Illustration of Reduction R.10, with Reduction R.10.1 displayed at the top and Reduction R.10.2 displayed at the bottom.

$e' = \{b, c\}$ or $e' = \{c, z\}$ for some $z \in N(a)$. We can conclude that $X \setminus \{v\} \cup \{c\}$ is a vertex cover of G'. The remaining cases can be argued for similarly: If $b \in X$, then $X \setminus \{v\} \cup \{a\}$ is a vertex cover of G' and if $c \in X$, then $X \setminus \{v\} \cup \{b\}$ is a vertex cover of G'. ⌟

Claim 5. *If G' has a vertex cover of size k then G has a vertex cover of size k.*

Proof. Throughout the following, let X' be a vertex cover of G' of size k. Since $\{a, b, c\}$ is *not* an independent set in G', we know that X' has to contain at least one vertex of $\{a, b, c\}$. If $\{a, b, c\} = N(v) \subseteq X'$, then X' contains an endpoint of each edge in $E(G) \setminus E(G') = \{\{v, x\} \mid x \in \{a, b, c\}\}$, so we can conclude that X' is a vertex cover of G.

We now consider the cases when X' contains precisely two vertices from $\{a, b, c\}$. If $\{a, b\} \subseteq X'$ and hence $c \notin X'$, then X' contains $N_G(a)$ as well, to cover the edges between the vertex c and vertices in $N(a)$. It follows that $X' \setminus \{a\}$ is a vertex cover of $G - v$. Since each edge in $E(G) \setminus E(G - v)$ is incident with v, we can conclude that $X' \setminus \{a\} \cup \{v\}$ is a vertex cover of G. By similar arguments we have that if $X' \cap \{a, b, c\} = \{b, c\}$, then $X' \setminus \{b\} \cup \{v\}$ is a vertex cover of G and if $X' \cap \{a, b, c\} = \{a, c\}$, then $X' \setminus \{c\} \cup \{v\}$ is a vertex cover of G.

It remains to argue the case when X' contains precisely one vertex from $\{a, b, c\}$. Note that the only possible such case is when this vertex is b. If X' contained only the vertex a (resp., c), then the edge $\{b, c\}$ (resp., $\{a, b\}$) would remain uncovered by X'. Suppose $X' \cap \{a, b, c\} = \{b\}$, so $a, c \notin X'$, implying that $N(a, b, c) \subseteq X'$. Hence, $X' \setminus \{b\}$ is a vertex cover of $G - v$ and $X' \setminus \{b\} \cup \{v\}$ is a vertex cover of G. ⌟

In the light of Claims 4 and 5, the proposition is proved. □

The next reduction rule kernelizes all vertices that have degree four and whose neighborhood induces a subgraph with more than two edges.

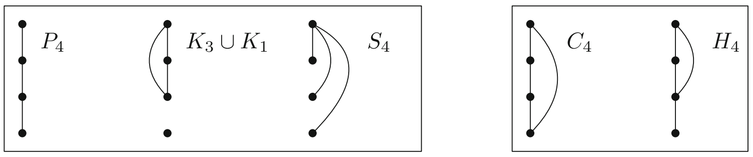


Fig. 7. In the left box, all pairwise non-isomorphic graphs on four vertices and three edges and in the right box, all pairwise non-isomorphic graphs on four vertices and four edges are shown.

Reduction R.10. *If G contains a vertex v with degree four such that $G[N(v)]$ has at least three edges and none of the above rules apply, then reduce (G, k) to (G', k') according to one of the following cases (up to renaming the vertices in $N(v) = \{a, b, c, d\}$).*

Reduction R.10.1. *$G[N(v)]$ is an (a, d)-path. In this case, let $k' := k$ and G' be the graph on vertex set $V(G) \setminus \{v\}$ and edge set $(E(G) \cap \binom{V(G')}{2}) \cup F$, where*

$$F := \binom{N(v)}{2} \cup \{\{x, y\} \mid x \in \{a, b\}, y \in N(d)\} \qquad (5)$$
$$\cup \{\{x, y\} \mid x \in \{c, d\}, y \in N(a)\}\,.$$

Reduction R.10.2. *$G[\{a, b, c\}]$ is a clique and d is isolated in $G[N(v)]$. In this case, $k' := k - 1$ and G' is the graph on vertex set $V(G) \setminus \{v, d\}$ and edge set $(E(G) \cap \binom{V(G')}{2}) \cup F$, where*

$$F := \{\{x, y\} \mid x \in \{a, b, c\} \text{ and } y \in N(d)\}$$

We illustrate Reductions R.10.1 and R.10.2 in Fig. 6.

Proposition 7. *Reduction R.10 is safe, i.e., if its conditions are satisfied, then either Reduction R.10.1 or Reduction R.10.2 applies and G has a vertex cover of size k if and only if G' has a vertex cover of size k' (with G' and k' depending on the case that applies).*

Proof. Throughout the proof, let $v \in V(G)$ denote the vertex satisfying the conditions of Reduction R.10. We first show that it is sufficient to consider the cases described in Reductions R.10.1 and R.10.2.

Claim. *If G and v are as above and none of the above reduction rules can be applied, then one of the cases Reductions R.10.1 and R.10.2 applies.*

Proof. If $G[N(v)]$ contains at least five edges, then there has to be a vertex $x \in \{a, b, c, d\}$ which is incident with three of these edges. Hence, $N(v) \subseteq N[x]$ and we could have applied Reduction R.7, a contradiction. Suppose there are four edges in $G[N(v)]$. There are only two non-isomorphic graphs on four vertices and four edges, see the right hand side of Fig. 7. If $G[N(v)]$ induces a C_4, assume w.l.o.g. that its vertices appear in the order $a - b - c - d - a$, then we can partition $N(v)$ into cliques $C_2 = \{a, b\}$ and $C_1 = \{c, d\}$ and obtain a co-matching $M = \{\{a, c\}, \{b, d\}\}$ which satisfies the conditions of Reduction R.8, since M saturates C_1 and $G[C_1, C_2] + M$ is a complete bipartite graph: we have that $\{b, c\}, \{a, d\} \in E(G)$. If $G[N(v)]$ induces an H_4, assume w.l.o.g. that the 3-cycle is $a - b - c - a$ and d is the pendant vertex adjacent to c, then we can partition $N(v)$ into cliques $C_2 = \{a, b, c\}$ and $C_1 = \{d\}$ and have a co-matching $M = \{a, d\}$ saturating C_1. Hence, we could have applied Reduction R.8, a contradiction.

We can assume that $G[N(v)]$ contains precisely three edges. There are three pairwise non-isomorphic graphs on four vertices and three edges, shown in the left-hand side of Fig. 7. If $G[N(v)]$ induces a star (S_4) with center a, then we have that $N(v) \subseteq N[a]$, so we could have applied Reduction R.7, a contradiction. If $G[N(v)]$ induces a P_4, we can apply Reduction R.10.1, if $G[N(v)]$ induces a $K_3 \cup K_1$, then we can apply Reduction R.10.2. ⌟

We now prove that Reduction R.10.1 is safe.

Claim 6. *If the conditions of Reduction R.10.1 hold, then G contains a vertex cover of size k if and only if G' contains a vertex cover of size k.*

Proof. ($\Rightarrow$) Suppose G has a vertex cover X of size k. If $N(v) \subseteq X$, then every edge in $E(G') \setminus E(G)$ has at least one endpoint in X by construction, so X is a vertex cover of G' as well. If X contains precisely three vertices of $N(v)$, suppose w.l.o.g. that $X \cap N(v) = \{a, b, c\}$, then v has to be contained in X as well, otherwise the edge $\{v, d\}$ remains uncovered. Hence, $X \setminus \{v\} \cup \{d\} \supseteq N(v)$ is a vertex cover of G' of size k.

From now on suppose that X contains precisely two vertices from $N(v)$ and note that in all of the following cases, $v \in X$. The only vertex covers of $G[N(v)]$ of size two are $\{b, c\}$, $\{a, c\}$ and $\{b, d\}$. First observe that any triple of vertices from $N_G(v)$ covers the edges $\binom{N(v)}{2}$, so we do not have to consider them explicitly in the following discussion.

Suppose $X \cap N(v) = \{b, c\}$. Then, $N(a, d) \subseteq X$, so all edges in $E(G') \setminus E(G) \setminus \binom{N(v)}{2}$ are covered by X. We can conclude that $X \setminus \{v\} \cup \{a\}$ (also $X \setminus \{v\} \cup \{d\}$) is a vertex cover of G'. If $X \cap N(v) = \{a, c\}$, then $N(b, d) \subseteq X$ and all edges in $E(G') \setminus E(G) \setminus \binom{N(v)}{2}$ that are not covered by X are between d and $N(a)$, so $X \setminus \{v\} \cup \{d\}$ is a vertex cover of G'. Similarly, if $X \cap N(v) = \{b, d\}$, then $X \setminus \{v\} \cup \{a\}$ is a vertex cover of G'.

Since $G[N(v)]$ does not have a vertex cover of size at most 1, this concludes the proof of the first direction.

($\Leftarrow$) Since $N_G(v)$ is a clique in G', any vertex cover of G' contains at least three vertices from $N_G(v)$. Let X' be a vertex cover of G' of size k. If $N_G(v) \subseteq X'$, then X' is also a vertex cover of G since each edge in $E(G) \setminus E(G')$ has an endpoint in $N_G(v)$. In the remainder, we can assume that X' contains precisely three vertices from $N_G(v)$. Suppose $X' \cap N_G(v) = \{a, b, c\}$. Since $d \notin X'$, we have that $N(a, d) \subseteq X'$, hence all edges between a and $N(a)$ are covered by $N(a) \subseteq X'$. Together with the observation that $\{b, c\}$ is a vertex cover of $G[N(v)]$, we can conclude that $X' \setminus \{a\} \cup \{v\}$ is a vertex cover of G. If $X' \cap N_G(v) = \{a, b, d\}$, then $N(a, c) \subseteq X'$ since $c \notin X'$. By the same reasoning as before, we can observe that $X' \setminus \{a\} \cup \{v\}$ is a vertex cover of G. Similarly, if $X' \cap N_G(v) = \{b, c, d\}$ or if $X' \cap N_G(v) = \{a, c, d\}$ then we can argue that $X' \setminus \{d\} \cup \{v\}$ is a vertex cover of G. ⌟

We now prove the safeness of Reduction R.10.2.

Claim 7. *If the conditions of Reduction R.10.2 hold, then G contains a vertex cover of size k if and only if G' contains a vertex cover of size $k - 1$.*

Proof. ($\Rightarrow$) Let X be a vertex cover of G of size k. Observe that all edges in $E(G') \setminus E(G)$ have an endpoint in $\{a, b, c\}$ or in $N(d)$. If $N_G(v) \subseteq X$, then X clearly contains a vertex cover of G' of size at most $k - 1$. Suppose X does not contain all of $N_G(v)$, implying that $v \in X$. Since $\{a, b, c\}$ is a clique in G, X contains at least two vertices from $\{a, b, c\}$, assume w.l.o.g. that $X \cap \{a, b, c\} = \{a, b\}$. If $d \in X$, then $X \setminus \{v, d\} \cup \{c\}$ is a vertex cover of G' and if $d \notin X$, then $N(d) \subseteq X$, so X already covers all edges between $\{a, b, c\}$ and $N(d)$, implying that $X \setminus \{v\}$ is a vertex cover of G'.

($\Leftarrow$) Let X' be a vertex cover of G' of size $k - 1$. Since $G'[\{a, b, c\}]$ is a clique, X' contains at least two vertices from $\{a, b, c\}$. If $\{a, b, c\} \subseteq X'$, then $X' \cup \{d\}$ is a vertex cover of G. If X' contains precisely two vertices from $\{a, b, c\}$, then $N_G(d) \subseteq X'$, so all edges between d and $N(d)$ (in G) are already covered by X'. We can conclude that in this case, $X' \cup \{v\}$ is a vertex cover of G. ⌟

This concludes the proof that Reduction R.10 is safe. □

It is easy to see that Reductions R.5, R.6, R.7, R.9 and R.10 can be executed in polynomial time. We observe (naively) that Reduction R.8 can be executed in time $\mathcal{O}\left(n \cdot 2^{\alpha} \cdot \alpha! \cdot \alpha^{\mathcal{O}(1)}\right)$, where α denotes the degree of the vertex v. Since for our arguments it is sufficient to apply Reduction R.8 for $\alpha \leq 4$, we can assume that all its executions run in polynomial time as well. An exhaustive application of the reduction rules presented in this section therefore yields the following theorem.

Theorem 5 (cf. Fellows and Stege [30]). *There is a polynomial-time algorithm that given an instance (G, k) of* Vertex Cover *outputs an equivalent instance (G', k'), where $k' \leq k$,*

(I) the minimum degree of G' is at least four and
(II) for all vertices $v \in V(G')$ with $\deg_{G'}(v) = 4$, $G'[N(v)]$ contains at most two edges.

4 Automated Vertex Cover Kernelization

Many reduction rules for various problems are instances of the same class that can be described as "find a subgraph H with boundary X and replace it with a graph H^*". For example, Reduction R.6 can be formulated as "find a P_3 (a, v, b)

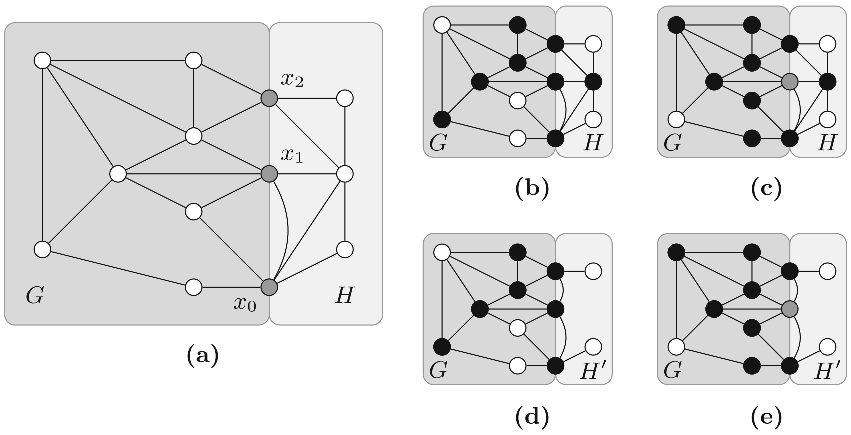


Fig. 8. Illustration of notation. **(a)** shows a graph $G \oplus H$ resulting from gluing G (darker background) and H (lighter background). **(b)** shows a vertex cover of $G \oplus H$ compatible with $\{2\}$, minimizing the intersection with $V(H)$ and, thus, implying $P_H^G(\{2\}) = 4$. **(c)** shows a vertex cover S of $G \oplus H$ compatible with $\{0, 1, 2\}$, minimizing the intersection with $V(H)$ and, thus, implying $P_H^G(\{0, 1, 2\}) = 3$. Note that S is not necessarily minimum or even minimal for $G \oplus H$. **(d) and (e)** show the same as (b) and (c) but with the graph H' resulting from the application of Reduction R.7 to a vertex in H, proving $P_{H'}^G(X) = P_H^G(X) - 1$ for $X \in \{\{2\}, \{0, 1, 2\}\}$.

with boundary $\{a, b\}$ (a kind of "local surgery") and replace it with $z_{N[v]}$". Note however, that boundary connections might change during the replacement. Proposed reduction rules of this type can be checked in roughly $2^{|X|}$ times the time it takes to compute a minimum vertex cover in H and H^*.

Notation. See Fig. 8 for an illustration. Let G be a graph in which t non-isolated vertices $x_1, \ldots, x_t$ are bijectively labeled with the integers in $[t]$. Then, we call G *t-boundaried.* If $\{x_1, \ldots, x_t\}$ is an independent set in G, we call G *strongly t-boundaried.* For a t-boundaried graph H and a strongly t-boundaried graph G, we let $G \oplus H$ denote the result of *gluing* G and H, that is, identifying the vertices with the same label in the disjoint union of G and H. We call a set $S \subseteq V(G)$ *compatible* with a set $X \subseteq [t]$ in G if $N_G(x_i) \subseteq S \iff i \in X$ for all $i \in [t]$. Let $P_H^G \colon 2^{[t]} \to \mathbb{N}$ be such that, for all $X \subseteq [t]$, $P_H^G(X)$ is the smallest number of vertices of H contained in any vertex cover S of $G \oplus H$ that is compatible with X in G (and ∞ if no such S exists). Then, we call P_H^G the *profile* of H in G. It turns out that the profile is indeed independent of G, so we drop the superscript.

Lemma 2. *Let G, G' be strongly t-boundaried, let H be t-boundaried and let $X \subseteq [t]$. Then, $P_H^G(X) = P_H^{G'}(X)$ for all $X \subseteq [t]$.*

Proof. Let $X \subseteq [t]$ and let S and S' be vertex covers of $G \oplus H$ and $G' \oplus H$ that are compatible with X in G and G', respectively, such that $|S \cap V(H)|$ and $|S' \cap V(H)|$

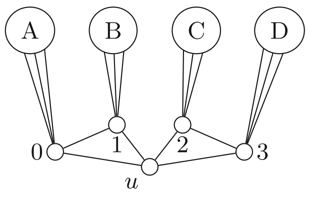
are minimum among all such vertex covers. To prove the claim, we show that $S^* := (S \setminus V(H)) \cup (S' \cap V(H))$ is a vertex cover of $G \oplus H$. By symmetry, the same follows for S and S' inversed, which then implies the lemma. Towards a contradiction, assume that $G \oplus H$ contains an edge uv such that $u, v \notin S^*$. Then, exactly one of u and v is in H as, otherwise, S is not a vertex cover of $G \oplus H$. Without loss of generality, let $u \in V(H)$ and $v \notin V(H)$. Thus, u is a boundary vertex x_i and $v \notin S$. Since S and S' are compatible with X in G and G', respectively, we know that $N_G(x_i) \subseteq S \iff i \in X \iff N_{G'}(x_i) \subseteq S'$ and, as $v \in N_G(u) \setminus S$, we have $N_{G'}(u) \nsubseteq S'$. However, since S' is a vertex cover of $G' \oplus H$, we have $u \in S'$, which contradicts $u \notin S^*$ since $u \in V(H)$. □

We observe that two t-boundaried graphs H and H' with the same profile can be swapped for one another in any graph G without changing the size of an optimal vertex cover, that is, $G \oplus H$ and $G \oplus H'$ have the same vertex cover number. More generally, for any $c \in \mathbb{N}$, we say that H and H' are *c-equivalent* if $\forall_{X \subseteq [t]}\ P_H(X) = P_{H'}(X) + c$. In this way, for any fixed size t, the profile gives rise to an equivalence relation on the set of t-boundaried graphs. This relation allows automated discovery of reduction rules that remove vertices with undesirable properties from the input graph. The idea is, for each induced subgraph H having an undesirable property Π, to replace H by some c-equivalent H' that does not suffer from Π, while reducing k by c.

Lemma 3. *Let G be strongly t-boundaried, let H and H' be t-boundaried and c-equivalent for some $c \in \mathbb{N}$, and let $k \in \mathbb{N}$. Then, $G \oplus H$ has a vertex cover of size at most k if and only if $G \oplus H'$ has a vertex cover of size at most $k - c$.*

Proof. As "⇒" is completely analogous to "⇐", we only prove the latter. To this end, let S be a smallest vertex cover of $G \oplus H$ that, among all such vertex covers, minimizes $|S \cap V(H)|$. Let $X := \{i \mid N_G(x_i) \subseteq S\}$ and note that S is compatible with X in G and $|S \cap V(H)| = P_H(X)$. Let S' be a smallest vertex cover of $G \oplus H'$ that is compatible with X in G and, among all such vertex covers, minimizes $|S' \cap V(H')|$. As H and H' are c-equivalent, we know that $|S \cap V(H)| = |S' \cap V(H')| + c$. We show that $S^* := (S \setminus V(H)) \cup (S' \cap V(H))$ is a vertex cover of $G \oplus H'$ (clearly, $|S^*| \leq |S \setminus V(H)| + |S' \cap V(H')| = |S \setminus V(H)| + |S \cap V(H)| + c = |S| + c$). Towards a contradiction, assume that there is an edge uv of $G \oplus H'$ with $u, v \notin S^*$. If $u, v \notin V(H')$, then S is not a vertex cover of $G \oplus H$ and, if $u, v \in V(H')$, then S' is not a vertex cover of $G \oplus H'$. Thus, without loss of generality, $u \in V(H')$ and $v \notin V(H')$, implying that u is a boundary vertex x_i. Since $u \notin S^*$, we know that $u \notin S'$ and, since S' is a vertex cover of $G \oplus H'$, we have $N_G(u) \subseteq S'$. Since S' is compatible with X, we have $i \in X$ and, since S is compatible with X, we have $N_G(u) \subseteq S$, implying $v \in S$ which contradicts $v \notin S^*$ since $v \notin V(H)$. □

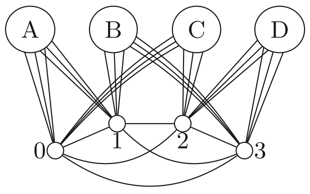
Given a t-boundaried graph H and a property Π, we can enumerate all t-boundaried graphs H' that are c-equivalent to H for some c and that do not suffer from Π.



(a)

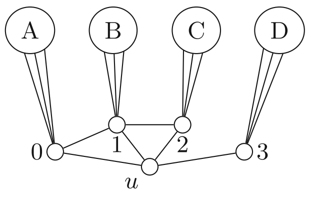
		A		$\overline{A}$	
		B	$\overline{B}$	B	$\overline{B}$
C	D	3	3	3	4
	$\overline{D}$	3	3	3	4
$\overline{C}$	D	3	3	3	4
	$\overline{D}$	4	4	4	4

(b)



(c)

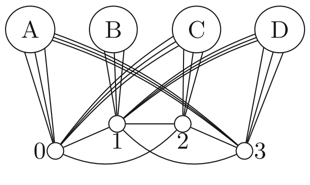
Fig. 9. Illustration of Reduction R.11. **(a)** shows the degree-four vertex u with two edges in its neighborhood. A–D represent the sets of neighbors of 0–3, respectively, in the rest of the graph (A–D may mutually intersect or be empty). **(b)** shows the profile of (a) where the entry 3 at position $(A, \overline{B}, C, D)$ means that three vertices are needed to cover all edges of (a), assuming all vertices in A, C, and D are already in the cover. Indeed, $\{u, 1, 2\}$ is a size-3 cover in this case. **(c)** shows a subgraph that is 0-equivalent to (a), that is, (b) is also the profile of (c).



(a)

		A		$\overline{A}$	
		B	$\overline{B}$	B	$\overline{B}$
C	D	2	2	3	3
	$\overline{D}$	3	3	4	4
$\overline{C}$	D	3	3	3	4
	$\overline{D}$	4	4	4	4

(b)



(c)

Fig. 10. Illustration of Reduction R.12. **(a)** shows the degree-four vertex u with two edges in its neighborhood. A–D represent the sets of neighbors of 0–3, respectively, in the rest of the graph. **(b)** shows the profile of (a) where the entry 2 at position $(A, \overline{B}, C, D)$ means that two vertices are needed to cover all edges of (a), assuming all vertices in A, C, and D are already in the cover. Indeed, $\{u, 1\}$ is a size-2 cover in this case. **(c)** shows a subgraph that is 0-equivalent to (a), that is, (b) is also the profile of (c).

Two Examples. A proof-of-concept implementation[7] was used to attack the remaining cases of degree-four vertices (see Sect. 3). For a given t-boundaried graph H or profile P_H and a given number n, the implementation enumerates all strongly t-boundaried, n-vertex graphs H' and outputs H' if $P_H(X) = P_{H'}(X)$ for all $X \subseteq [t]$. Feeding the graphs displayed in Figs. 9a and 10a, the implementation yielded, in 5s and 6s, respectively, reduction rules that remove degree-four vertices whose neighborhood contains exactly two edges.

Reduction R.11. *Let G contain the 4-boundaried graph H depicted in Fig. 9a as an induced subgraph. Then, replace H by the 4-boundaried graph H' depicted in Fig. 9c.*

[7] https://github.com/igel-kun/VC_min_deg.

Reduction R.12. *Let G contain the 4-boundaried graph H depicted in Fig. 10a as an induced subgraph. Then, replace H by the 4-boundaried graph H' depicted in Fig. 10c.*

By Lemma 3, correctness of Reductions R.11 and R.12 can be verified by convincing oneself that the profiles in Figs. 9 and 10 are indeed the profiles of the graphs of Figs. 9c and 10c, respectively (implying that the subgraphs are 0-equivalent).

Indeed, Reductions R.11 and R.12 cover all cases of degree-four vertices with two edges in the neighborhood, leaving only the cases of one and no edges in the neighborhood in order to be able to reduce to graphs of minimum degree five.

5 Conclusion and Open Problems

In Sect. 3, we have discussed the *barrier degree* constant δ_{VC} for Vertex Cover kernelization: We observed that, for some $\delta_{VC} \in \mathbb{N}$, Vertex Cover cannot be kernelized to instances of minimum degree δ_{VC} unless ETH fails. In terms of FPT algorithms, the equivalent concept is that of the existence of the *barrier constant* $\zeta_{VC} > 0$ which is such that there is no algorithm for Vertex Cover running in time $(1 + \zeta_{VC})^k \cdot n^{\mathcal{O}(1)}$ modulo ETH (e.g., [16,20,28]). So far it is only known that $\zeta_{VC} < 0.2738$ [12] and that $\delta_{VC} > 3$ (Sect. 3, see also [30]). However, observe that the question of determining the concrete value of δ_{VC} is much more tangible than the one of finding the value of ζ_{VC}: Suppose one can show that a reduction rule that kernelizes degree-d vertices violates ETH, for some $d \in \mathbb{N}$. Then one might be able to adapt the gadgets used in that proof to show an ETH-violation via a reduction rule for degree $d+1, d+2, \ldots$ vertices as well. We pose: What is the exact value of δ_{VC}?

The main theme of this paper has been to gather (from hitherto unpublished sources), carefully verify, and advance research on the question: to what minimum degree d can the (*formidable naturally parameterized*) Vertex Cover problem be kernelized to kernels of minimum degree d, even if the exponent of the polynomial running time bound grows wildly in d? ETH enforces a limit.

References

1. Abu-Khzam, F.N., Collins, R.L., Fellows, M.R., Langston, M.A., Suters, W.H., Symons, C.T.: Kernelization algorithms for the vertex cover problem: theory and experiments. In: Arge, L., Italiano, G.F., Sedgewick, R. (eds.) Proceedings 6th Workshop on Algorithm Engineering and Experiments and 1st Workshop on Analytic Algorithms and Combinatorics (ALENEX/ANALC), pp. 62–69. SIAM (2004)
2. Aydoğan, R., Baarslag, T., Gerding, E., Jonker, C.M., Julian, V., Sanchez-Anguix, V. (eds.): COREDEMA 2016. LNCS (LNAI), vol. 10238. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57285-7
3. Balasubramanian, R., Fellows, M.R., Raman, V.: An improved fixed-parameter algorithm for vertex cover. Inf. Process. Lett. **65**(3), 163–168 (1998)
4. Berretta, R., Moscato, P.: Cancer biomarker discovery: the entropic hallmark. PLoS ONE **5**(8), e12262 (2010)

5. Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Kernelization lower bounds by cross-composition. SIAM J. Discrete Math. **28**(1), 277–305 (2014)
6. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels (extended abstract). In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5125, pp. 563–574. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70575-8_46
7. Bodlender, H., Downey, R., Fellows, M., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. **75**, 423–434 (2009)
8. Buss, J.F., Goldsmith, J.: Nondeterminism within P. SIAM J. Comput. **22**(3), 560–572 (1993)
9. Butenko, S., Wilhelm, W.E.: Clique-detection models in computational biochemistry and genomics. Eur. J. Oper. Res. **173**(1), 1–17 (2006)
10. Cai, L., Chen, J., Downey, R.G., Fellows, M.R.: Advice classes of parameterized tractability. Ann. Pure Appl. Log. **84**(1), 119–138 (1997)
11. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: further observations and further improvements. J. Algorithms **41**(2), 280–301 (2001)
12. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theor. Comput. Sci. **411**(40–42), 3736–3756 (2010). Previously appeared in MFCS 2006 as 'Improved parameterized upper bounds for vertex cover'
13. Cheng, T.M.K., Yu-En, L., Vendruscolo, M., Blundell, T.L., et al.: Prediction by graph theoretic measures of structural effects in proteins arising from non-synonymous single nucleotide polymorphisms. PLoS Comput. Biol. **4**(7), e1000135 (2008)
14. Chor, B., Fellows, M., Juedes, D.: Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 257–269. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30559-0_22
15. Cong, J., Smith, M.L.: A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design. In: Proceedings 30th International Design Automation Conference, pp. 755–760. ACM (1993)
16. Cygan, M., et al.: Parameterized Algorithms, 1st edn. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21275-3
17. Dehne, F., Fellows, M., Rosamond, F., Shaw, P.: Greedy localization, iterative compression, and modeled crown reductions: new FPT techniques, an improved algorithm for SET SPLITTING, and a novel $2k$ kernelization for VERTEX COVER. In: Downey, R., Fellows, M., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 271–280. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28639-4_24
18. Dell, H., van Melkebeek, D.: Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. J. ACM (JACM) **61**(4), 23 (2014). Previously appeared in STOC 2010
19. Downey, R.G., Fellows, M.R.: Parameterized computational feasibility. In: Clote, P., Remmel, J.B. (eds.) Feasible Mathematics II. PCS, pp. 219–244. Springer, Boston (1995). https://doi.org/10.1007/978-1-4612-2566-9_7
20. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. TCS. Springer, London (2013). https://doi.org/10.1007/978-1-4471-5559-1
21. Downey, R.G., Fellows, M.R., Stege, U.: Parameterized complexity: a framework for systematically confronting computational intractability. In: Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future, vol. 49, pp. 49–99 (1999)

22. Downs, G.M., Willett, P.: Similarity searching in databases of chemical structures. Rev. Comput. Chem. **7**, 1–66 (1996)
23. Enright, J., Meeks, K.: Deleting edges to restrict the size of an epidemic: a new application for treewidth. Algorithmica **80**, 1–33 (2017). Previously appeared in COCOA 2015
24. Fellows, M.R.: Parameterized complexity: new developments and research frontiers. In: Downey, R.G., Hirschfeldt, D.R. (eds.) Aspects of Complexity: Minicourses in Algorithmics, Complexity and Computational Algebra. De Gruyter Series in Logic and Its Applications, vol. 4, pp. 51–72. De Gruyter, Kaikoura (2000)
25. Fellows, M.R.: Parameterized complexity: the main ideas and some research frontiers. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 291–307. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45678-3_26
26. Fellows, M.R.: Blow-ups, win/win's, and crown rules: some new directions in *FPT*. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 1–12. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39890-5_1
27. Fellows, M.R.: The lost continent of polynomial time: preprocessing and kernelization. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 276–277. Springer, Heidelberg (2006). https://doi.org/10.1007/11847250_25
28. Fellows, M.R.: Some open problems in parameterized complexity related to the work of Jianer Chen. Tsinghua Sci. Technol. **19**(4), 325–328 (2014)
29. Fellows, M.R., Jansen, B.M.P., Rosamond, F.A.: Towards fully multivariate algorithmics: parameter ecology and the deconstruction of computational complexity. Eur. J. Comb. **34**(3), 541–566 (2013)
30. Fellows, M.R., Stege, U.: An improved fixed-parameter tractable algorithm for vertex cover (1999)
31. Fomin, F.V., Gaspers, S., Golovach, P.A., Kratsch, D., Saurabh, S.: Parameterized algorithm for eternal vertex cover. Inf. Process. Lett. **110**(16), 702–706 (2010)
32. Fomin, F.V., Strømme, T.J.F.: Vertex cover structural parameterization revisited. In: Heggernes, P. (ed.) WG 2016. LNCS, vol. 9941, pp. 171–182. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53536-3_15
33. Fortnow, L., Santhanam, R.: Infeasibility of instance compression and succinct PCPs for NP. In: Proceedings Fortieth Annual ACM Symposium on Theory of Computing (STOC), pp. 133–142. ACM (2008)
34. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
35. Gottlieb, L.-A., Kontorovich, A., Krauthgamer, R.: Efficient classification for metric data. IEEE Trans. Inf. Theory **60**(9), 5750–5759 (2014)
36. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News **38**(1), 31–45 (2007)
37. Guo, J., Niedermeier, R., Wernicke, S.: Parameterized complexity of vertex cover variants. Theory Comput. Syst. **41**(3), 501–520 (2007)
38. Hall, P.: On representatives of subsets. J. Lond. Math. Soc. **10**(1), 26–30 (1935)
39. Hamzaoglu, I., Patel, J.H.: Test set compaction algorithms for combinational circuits. In: Proceedings IEEE/ACM International Conference on Computer-Aided Design, pp. 283–289. ACM (1998)
40. Hols, E.-M.C., Kratsch, S.: Smaller parameters for vertex cover kernelization. arXiv preprint arXiv:1711.04604 (2017)
41. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. **2**(4), 225–231 (1973)
42. Impagliazzo, R., Paturi, R.: On the complexity of k-sat. J. Comput. Syst. Sci. **62**(2), 367–375 (2001)

43. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**(4), 512–530 (2001)
44. Jansen, B.M.P.: The power of data reduction. Kernels for fundamental graph problems. Ph.D. thesis, Utrecht University, The Netherlands (2013)
45. Jansen, B.M.P., Bodlaender, H.L.: Vertex cover kernelization revisited. Theory Comput. Syst. **53**(2), 263–299 (2013)
46. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds.) Complexity of Computer Computations. IRSS, pp. 85–103. Springer, Boston (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
47. Karp, R.M.: Heuristic algorithms in computational molecular biology. J. Comput. Syst. Sci. **77**(1), 122–128 (2011)
48. Koch, I., Lengauer, T., Wanke, E.: An algorithm for finding maximal common subtopologies in a set of protein structures. J. Comput. Biol. **3**(2), 289–306 (1996)
49. König, D.: Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. Math. Ann. **77**(4), 453–465 (1916)
50. Kratsch, S., Wahlström, M.: Representative sets and irrelevant vertices: new tools for kernelization. In: Proceedings 53rd Annual Symposium on Foundations of Computer Science (FOCS), pp. 450–459. IEEE (2012)
51. Lampis, M.: A kernel of order $2k - c\log k$ for vertex cover. Inf. Process. Lett. **111**(23), 1089–1091 (2011)
52. Leach, A.R., Gillet, V.J.: An Introduction to Chemoinformatics. Springer, Dordrecht (2007)
53. Lokshtanov, D., Marx, D., Saurabh, S.: Lower bounds based on the exponential time hypothesis. Bull. EATCS **3**(105), 41–71 (2013)
54. Lokshtanov, D., Misra, N., Saurabh, S.: Kernelization – preprocessing with a guarantee. In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday. LNCS, vol. 7370, pp. 129–161. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30891-8_10
55. Majumdar, D., Raman, V., Saurabh, S.: Kernels for structural parameterizations of vertex cover-case of small degree modulators. In: Proceedings 10th International Symposium on Parameterized and Exact Computation (IPEC). Leibniz International Proceedings in Informatics (LIPIcs), vol. 43, pp. 331–342. Schloss Dagstuhl Publishing (2015)
56. Misra, N., Raman, V., Saurabh, S.: Lower bounds on kernelization. Discrete Optim. **8**(1), 110–128 (2011)
57. Mucha, M., Sankowski, P.: Maximum matchings via Gaussian elimination. In: Proceedings 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 248–255. IEEE (2004)
58. Narayanaswamy, N.S., Raman, V., Ramanujan, M.S., Saurabh, S.: LP can be a cure for parameterized problems. In: Dürr, C., Wilke, T. (eds.) Proceedings 29th Symposium on Theoretical Aspects of Computer Science (STACS). Leibniz International Proceedings in Informatics (LIPIcs), vol. 14, pp. 338–349. Schloss Dagstuhl Publishing (2012)
59. Nemhauser, G.L., Trotter, L.E.: Properties of vertex packing and independence system polyhedra. Math. Program. **6**(1), 48–61 (1974)
60. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and its Applications, vol. 31. Oxford University Press, Oxford (2002)

61. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, New York (1998)
62. Soleimanfallah, A., Yeo, A.: A kernel of order $2k - c$ for vertex cover. Discrete Math. **311**(10), 892–895 (2011)
63. Stege, U.: Resolving conflicts in problems from computational biology. Ph.D. thesis, ETH Zuerich (2000)
64. Strickland, D.M., Barnes, E., Sokol, J.S.: Optimal protein structure alignment using maximum cliques. Oper. Res. **53**(3), 389–402 (2005)
65. Vazirani, V.V.: Approximation Algorithms, 1st edn. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-662-04565-7
66. Weihe, K.: Covering trains by stations or the power of data reduction. In: Proceedings 1st Conference on Algorithms and Experiments (ALEX 1998), Trento, Italy, pp. 1–8 (1998)
67. Weihe, K.: On the differences between "practical" and "applied". In: Näher, S., Wagner, D. (eds.) WAE 2000. LNCS, vol. 1982, pp. 1–10. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44691-5_1
68. Weller, M.: Aspects of preprocessing applied to combinatorial graph problems. Ph.D. thesis, TU Berlin (2013)
69. Willett, P., Barnard, J.M., Downs, G.M.: Chemical similarity searching. J. Chem. Inf. Comput. Sci. **38**(6), 983–996 (1998)
70. Yap, C.K.: Some consequences of non-uniform conditions on uniform classes. Theoret. Comput. Sci. **26**(3), 287–300 (1983)
71. Yeger-Lotem, E., et al.: Network motifs in integrated cellular networks of transcription-regulation and protein-protein interaction. Proc. Nat. Acad. Sci. U.S.A. **101**(16), 5934–5939 (2004)

A Survey on the Complexity of Flood-Filling Games

Michael R. Fellows[1], Frances A. Rosamond[1], Maise Dantas da Silva[2], and Uéverton S. Souza[2](✉)

[1] University of Bergen, Bergen, Norway
{michael.fellows,frances.rosamond}@uib.no
[2] Fluminense Federal University, Niterói, Brazil
maisedantas@id.uff.br, ueverton@ic.uff.br

Abstract. This survey is offered in honour of the special occasion of the birthday celebration of science and education pioneer Professor Juraj Hromkovič. In this survey, we review recent results on one-player flood-filling games on graphs, Flood-It and Free-Flood-It, in which the player aims to make the board monochromatic with a minimum number of *flooding moves*. As for many colored graph problems, flood-filling games have relevant interpretations in bioinformatics. The original versions of Flood-It and Free-Flood-It are played on $n \times m$ grids, but several studies were devoted to analyzing the complexity of these games when the "board" (the graph) belongs to other graph classes. A complete mapping of the complexity of flood-filling games on trees is presented, charting the consequences of single and aggregate parameterizations. The Flood-It problem on trees and the Restricted Shortest Common Supersequence (RSCS) problem are analogous. Flood-It remains NP-hard when played on 3-colored trees. A general framework for reducibility from Flood-It to Free-Flood-It is revisited. The complexity behavior of these games when played on various kinds of graphs is surveyed, such as Cartesian products of cycles and paths, circular grids, split graphs, co-comparability graphs, and AT-free graphs. We review a recent investigation of the parameterized complexity of Flood-It when the size of a minimum vertex cover is the structural parameter. Some educational aspects of the game are also reviewed. Happy Birthday, Juraj!

Keywords: Combinatorial games · Flood-filling games · Flood-It Free-Flood-It · Graph algorithms · Parameterized complexity

1 Introduction

Flood-It is a one-player combinatorial game (also known as the Mad Virus Game), originally played on a colored board consisting of an $n \times m$ grid, where each tile/cell of the board has an initial color from a fixed color set. In the classic game, two tiles are *neighboring* tiles if they lie in the same row (resp. column) and in consecutive columns (resp. rows). A sequence C of tiles is a *path* when every

© Springer Nature Switzerland AG 2018
H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 357–376, 2018.
https://doi.org/10.1007/978-3-319-98355-4_20

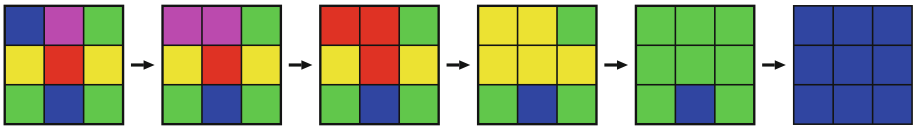


Fig. 1. An optimal sequence of moves to flood a 3×3 grid. (Color figure online)

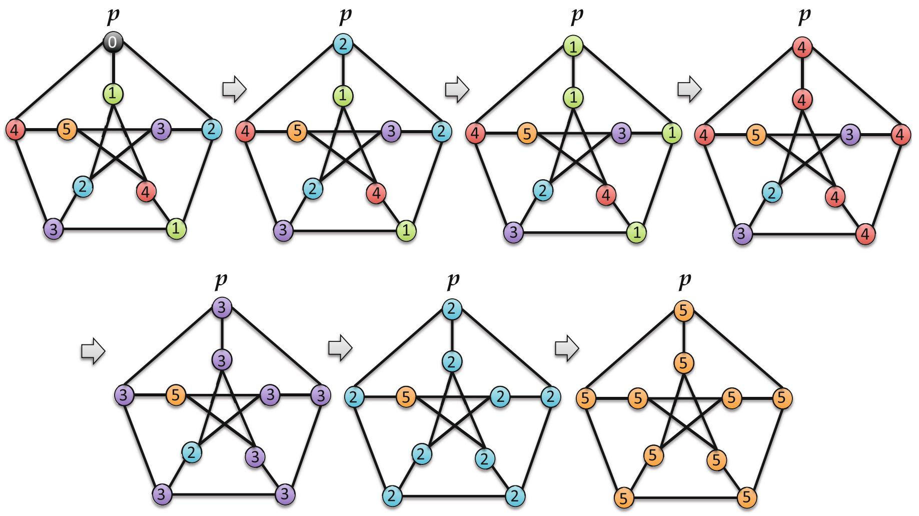


Fig. 2. An optimal sequence of moves to flood a 6-colored Petersen graph. (Color figure online)

pair of consecutive tiles in C is formed by neighboring tiles. A *monochromatic path* is a path in which all the tiles have the same color. Two tiles a and b are *m-connected* when there is a monochromatic path between them. In Flood-It, a move consists of assigning a new color c_i to the top left tile p (the *pivot*) and also to all the tiles m-connected to p immediately before the move. The objective of the game is to make the board monochromatic ("flood the board") with the minimum number of moves. Figure 1 shows a sequence of moves to flood a 3×3 grid colored with five colors.

A variation of Flood-It is Free-Flood-It, where the player can freely choose which tile will be the pivot of each move. In addition, these games can easily be generalized to be played on any graph with an initial coloring. Figure 2 shows a sequence of moves to flood a graph using a fixed pivot vertex p.

Many complexity issues regarding Flood-It and Free-Flood-It have recently been investigated. In [1], Arthur, Clifford, Jalsenius, Montanaro, and Sach show that Flood-It and Free-Flood-It are NP-hard on $n \times n$ grids colored with at least three colors. Meeks and Scott [28] prove that Free-Flood-It is solvable in polynomial time on $1 \times n$ grids, and also that Flood-It and Free-Flood-It remain NP-hard on $3 \times n$ grids colored with at least four colors.

Up to the authors' knowledge, the complexity of Flood-It on $3 \times n$ grids colored with three colors remains an open question. Clifford, Jalsenius, Montanaro, and Sach present in [7] a polynomial-time algorithm for Flood-It on $2 \times n$ grids. In [29], Meeks and Scott show that Free-Flood-It remains NP-hard on $2 \times n$ grids.

Fleischer and Woeginger [35] prove that Flood-It is NP-hard on trees. Analysing the complexity of Flood-It on non-grid graphs, Fleischer and Woeginger [14,35] also prove that Flood-It (also called the Honey-Bee-Solitaire problem) remains NP-hard even when restricted to split graphs, but that it is polynomial-time solvable on co-comparability graphs. When Flood-It is played on paths ($1 \times n$ grids) the problem is trivially solvable if the pivot has degree one. However, allowing the pivot be any vertex of the path, the problem is equivalent to the Shortest Common Supersequence problem (SCS) for two sequences [13,40], a very well-studied problem that does not have a known linear-time algorithm.

By similar arguments, it can be seen that Flood-It on cycles can be solved in polynomial time. In [28], Meeks and Scott show that Free-Flood-It on paths can be solved in $O(n^6)$ time. Thus, by the approach of removing the last vertex that will be flooded, we obtain an algorithm to solve Free-Flood-It on cycles in $O(n^7)$ time. In [25], it was shown that Free-Flood-It can be solved in polynomial time on 2-colored graphs.

In [30], it is shown the elegant fact that the minimum number of moves required to flood *any* given graph G is equal to the minimum, taken over all spanning trees T of G, of the number of moves required to flood T. Meeks and Vu present some extremal properties of flood-filling games in [31]. Fukui et al. [15] show that if the number of colors is not bounded, Free-Flood-It is NP-complete even on caterpillars and proper interval graphs, but when the number of colors is a fixed constant, the game can be solved in XP time on interval graphs. Hon et al. [19] present a polynomial-time algorithm for Flood-It on AT-free graphs. In [41], Souza, Protti and Dantas da Silva describe polynomial-time algorithms to play Flood-It on C_n^2 (the second power of a cycle on n vertices), $2 \times n$ circular grids ($2 \times n$ grids where the first and last tiles in a same row are neighboring tiles). Souza et al. also show that Free-Flood-It is NP-hard on C_n^2 and $2 \times n$ circular grids. More recently, in [8,12] there was presented a parameterized complexity analysis of Flood-It, parameterizing with respect to the vertex cover number, and the neighborhood diversity of the input.

Flood-filling Games in Bioinformatics. Since the 90's, an increasing number of papers on biological applications have been dealt with as combinatorial problems. Vertex-colored graph problems have several applications in bioinformatics [9]. The Colored Interval Sandwich Problem has applications in DNA physical mapping [11,16] and in perfect phylogeny [27]; vertex-recoloring problems appear in protein-protein interaction networks and phylogenetic analysis [6,33]; the Graph Motif Problem [9] was introduced in the context of metabolic network analysis [24]; the Intervalizing Colored Graphs Problem [5] models DNA physical mapping [11]; and the Triangulating Colored Graph Problem [5] is polynomially equivalent to the Perfect Phylogeny Problem [17].

Flood-Filling games on colored graphs are also related to many problems in bioinformatics. As shown in [13,40], Flood-It played on trees is analogous to a restricted case of the Shortest Common Supersequence Problem [18]. Consequently, these games inherit from the Shortest Common Supersequence Problem many applications in bioinformatics, such as: microarray production [36], DNA sequence assembly [3], and a close relationship to multiple sequence alignment [38]. In addition, some disease spreading models, described in [2], work in a similar way to flood-filling games.

Additional Definitions and Notation. Neighboring tiles naturally correspond to neighboring vertices of a graph G representing the board; therefore, from now on, we use the term *vertex* instead of *tile*. A subgraph H of G is *adjacent* to a vertex $v \in V(G)$ if v has a neighbor in $V(H)$. A *flood move*, or just *move*, is a pair $m = (p, c)$ where p is the *pivot* of m (the vertex chosen to have its color changed by m), and c is the new color assigned to p; in this case, we also say that color c is *played in move m*. In Flood-It all moves have the same pivot. A subgraph H is said to be *flooded* when H becomes monochromatic. A vertex v is *flooded by a move m* if the color of v is played in m and v becomes m-connected to new vertices after playing m. We say that a move m *floods a vertex v by a vertex w* if v and w are neighbors and move m changes the color of w to flood v. A (free-)flooding is a sequence of moves in (Free-)Flood-It which floods G (the entire board). An optimal (free-)flooding is a flooding with a minimum number of moves. A move $m = (p, c)$ *is played on subgraph H* if $p \in V(H)$. An *island* is a vertex v colored with a color c such that no neighbor of v is colored with c. Let G_n be a graph with n vertices, the k-th power of G_n, denoted by G_n^k, is the graph formed by G_n plus edges between vertices at a distance at most k. Thus, P_n^k and C_n^k is the k-th power of a path P_n and a cycle C_n, respectively. A *circular grid* is an $n \times m$ grid with the additional property that the first and the last tiles in a same row are neighboring tiles. Finally, we denote by $\Pi \propto^f \Pi'$ a reduction from a problem Π to a problem Π' via a computable function f.

The formal definitions of the most studied flood-filling games are as follows.

Flood-It (decision version)
Instance: A colored graph G with a pivot vertex p, an integer λ.
Question: Is there a sequence of at most λ flood moves which makes the graph monochromatic, using p as the pivot in all moves?

Free-Flood-It (decision version)
Instance: A colored graph G, an integer λ.
Question: Is there a sequence of at most λ flood moves which makes the graph monochromatic?

Definition 1. *Let Π be a flood-filling game and let $S = \{s_1, \ldots, s_n\}$ be a subset of the aspects of Π (see below). $[S_1]$-$\Pi(S_2)$ is the family of parameterized problems where the aspects in $S_1 \subseteq S$ are fixed constants and the aspects in $S_2 \subseteq S$ are aggregate parameters.*

As an example, $[d]$-Flood-It(c) is the family of parameterized problems where d (maximum distance of the pivot) is a fixed constant and c (number of colors) is the parameter. We consider the following aspects of the problem:

- c = number of colors
- λ = number of moves
- d = maximum distance of the pivot
- o = maximum orbit
- t = number of leaves
- r = number of bad moves, $r = (\lambda - c)$

Given a vertex-colored graph G, the *orbit* of a color b in G, o_b, is the number of occurrences of b in G. We say the the maximum orbit of a vertex-colored graph G is the maximum orbit of a color used in G. A *good move* for a color c_a is a move that floods all non-flooded vertices with color c_a. A move that is not good is a *bad move*. As in Free-Flood-It there is no fixed pivot, for such a game the parameter d stands for the diameter of the graph.

2 Flood-Filling Games on Grids

In this section we summarize the main results on the complexity of flood-filling games on grids.

Lemma 1 (Clifford et al. [7]). *For $c \geq 4$, $[c]$-Flood-It and $[c]$-Free-Flood-It are NP-hard on grids. Further, for an unbounded number of colours c, there is no polynomial-time constant-factor approximation algorithm, unless $P = NP$.*

Lemma 2 (Clifford et al. [7]). *$[c = 3]$-Flood-It and $[c = 3]$-Free-Flood-It on grids are NP-hard.*

Theorem 1 (Clifford et al. [7]). *Flood-It is polynomial-time solvable on $2 \times n$ grids.*

Theorem 2 (Meeks and Scott [28]). *$[c = 4]$-Flood-It remains NP-hard when restricted to $3 \times n$ grids.*

Theorem 3 (Meeks and Scott [28]). *$[c = 4]$-Free-Flood-It remains NP-hard when restricted to $3 \times n$ grids.*

Theorem 4 (Meeks and Scott [28]). *Free-Flood-It can be solved in polynomial time on $1 \times n$ grids.*

Theorem 5 (Meeks and Scott [28], Lagoutte et al. [25]). *$[c = 2]$-Free-Flood-It can be solved in polynomial time on general graphs.*

Theorem 6 (Meeks and Scott [29]). *Free-Flood-It remains NP-hard when restricted to $2 \times n$ grids.*

Theorem 7 (Meeks and Scott [29]). *Free-Flood-It(c) can be solved in $O(2^c \times n^{10})$ time on $2 \times n$ grids.*

In [7], Clifford et al. also provided a discussion about approximating the number of moves, and grids where each tile is coloured uniformly at random. Up to the authors' knowledge, the complexity of Flood-It and Free-Flood-It on $k \times n$ $(k \geq 3)$ grids colored with three colors remains as an open question.

Tables 1 and 2 summarize the results of this section.

Table 1. Complexity of Flood-It

#colors\ grid size	$1 \times n$	$2 \times n$	$3 \times n$	$k \times n$	$n \times n$
2	P	P	P	P	P
3	P	P	open	open	NP-hard [7]
4	P	P	NP-hard [28]	NP-hard	NP-hard
unbounded	P	P [7]	NP-hard	NP-hard	NP-hard

Table 2. Complexity of Free-Flood-It

#colors\ grid size	$1 \times n$	$2 \times n$	$3 \times n$	$k \times n$	$n \times n$
2	P	P	P	P	P [25,28]
3	P	P	open	open	NP-hard [7]
4	P	P [29]	NP-hard [28]	NP-hard	NP-hard
unbounded	P [28]	NP-hard [29]	NP-hard	NP-hard	NP-hard

3 Flood-Filling Games on Trees

Now, we revisit a multivariate investigation of the complexity of Flood-It and Free-Flood-It when played on trees presented in [13,40], where the authors analyze the complexity consequences of parameterizing flood-filling problems in various ways.

3.1 Flood-It on Trees

We start this section by remarking that Flood-It played on a tree is equivalent to Flood-It played on a rooted tree whose root is the pivot.

Theorem 8 (Fellows et al. [13], Souza et al. [40]). *$[d]$-Flood-It on trees remains NP-hard when $d = 2$.*

The proof of Theorem 8 uses a reduction from the Vertex Cover Problem. The authors show that there is a vertex cover of size k in a graph G if and only if there is a flooding with $n + k$ moves in the associated tree T. Given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, they construct a tree T as follows:

– create a pivot root s with color c_s;
– for each edge $e_i = uv$ of G, add to T a subset of vertices $E_i = \{u'_i, v'_i, u''_i, v''_i\}$ such that u'_i, v'_i are children of s, v''_i is a child of u'_i, and u''_i is a child of v'_i;
– define a distinct color c_u for each $u \in V(G)$, and color all vertices of the form u'_i, u''_i (for all i) with the color c_u.

Figure 3 shows a graph G and its associated tree T.

Theorem 8 shows that the problem remains NP-hard even for a very restricted class of trees. Notice that Flood-It and Free-Flood-It are trivially solvable when T is a star.

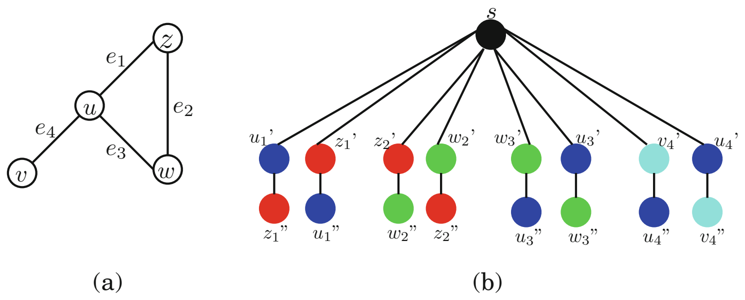


Fig. 3. (a) A graph G; (b) tree T obtained from G.

Corollary 1. *$[o, d]$-Flood-It on trees is NP-hard even when $o = 4$ and $d = 2$.*

Corollary 1 follows by restricting the reduction presented in Theorem 8 to cubic graphs. In addition, in [13,40] the authors also show the following result.

Theorem 9. *$[d]$-Flood-It(c) is in FPT and admits a polynomial kernelization.*

Analogous Problems

Next we present a very interesting observation provided in [13].

Definition 2. *Two optimization problems Π and Π' are said to be* analogous *if there exist linear-time reductions f, g such that:*

1. *$\Pi \propto^f \Pi'$ and $\Pi' \propto^g \Pi$;*
2. *every feasible solution s for an instance I of Π implies a feasible solution s' for $f(I)$ such that size$(s) =$ size(s');*
3. *every feasible solution s' for an instance I' of Π' implies a feasible solution s for $g(I')$ such that size$(s') =$ size(s).*

Next we have an equivalent definition for decision problems. Denote by $Y(\Pi)$ the set of all instances I of Π yielding a yes-answer for the question "$I \in Y(\Pi)$?".

Definition 3. *Two decision problems Π and Π' in NP are said to be* analogous *if there exist linear-time reductions f, g such that:*

1. *$\Pi \propto^f \Pi'$ and $\Pi' \propto^g \Pi$;*
2. *every easy checkable certificate $\mathcal{C}$ for the yes-answer of the question "$I \in Y(\Pi)$?" implies an easy checkable certificate $\mathcal{C}'$ for the yes-answer of the question "$f(I) \in Y(\Pi')$?" such that size$(\mathcal{C})$ = size$(\mathcal{C}')$;*
3. *every easy checkable certificate $\mathcal{C}'$ for the yes-answer of the question "$I' \in Y(\Pi')$?" implies an easy checkable certificate $\mathcal{C}$ for the yes-answer of the question "$g(I') \in Y(\Pi)$?" such that size$(\mathcal{C}')$ = size$(\mathcal{C})$.*

Definition 4. *Let Π and Π' be analogous decision problems. The parameterized problems $\Pi(k_1, \ldots, k_t)$ and $\Pi'(k'_1, \ldots, k'_t)$ are said to be* p-analogous *if there exist FPT reductions f, g and a one-to-one correspondence $k_i \leftrightarrow k'_i$ such that:*

1. *$\Pi(k_1, \ldots, k_t) \propto^f \Pi'(k'_1, \ldots, k'_t)$ and $\Pi'(k'_1, \ldots, k'_t) \propto^g \Pi(k_1, \ldots, k_t)$;*
2. *every easy checkable certificate $\mathcal{C}$ for the yes-answer of the question "$I \in Y(\Pi(k_1, \ldots, k_t))$?" implies an easy checkable certificate $\mathcal{C}'$ for the yes-answer of the question "$f(I) \in Y(\Pi'(k'_1, \ldots, k'_t))$?" such that $k'_i = \varphi'_i(k_i)$ for some function φ'_i $(1 \le i \le t)$;*
3. *every easy checkable certificate $\mathcal{C}'$ for the yes-answer of the question "$I' \in Y(\Pi'(k'_1, \ldots, k'_t))$?" implies an easy checkable certificate $\mathcal{C}$ for the yes-answer of the question "$g(I') \in Y(\Pi(k_1, \ldots, k_t))$?" such that $k_i = \varphi_i(k'_i)$ for some function φ_i $(1 \le i \le t)$.*

Two straightforward consequences of the above definitions are: (a) if Π and Π' are analogous problems then Π is in P (is NP-hard) if and only if Π' is in P (is NP-hard); (b) if $\Pi(k_1, \ldots, k_\ell)$ and $\Pi'(k'_1, \ldots, k'_\ell)$ are p-analogous problems then $\Pi(k_1, \ldots, k_\ell)$ is in FPT (admits a polynomial kernel/is W[1]-hard) if and only if $\Pi'(k'_1, \ldots, k'_\ell)$ is in FPT (admits a polynomial kernel/is W[1]-hard).

Fleischer and Woeginger used a reduction from the Fixed Alphabet Shortest Common Supersequence Problem [35] to prove that Flood-It on trees is NP-hard even when the number of colors is fixed. In [13], the authors show that Flood-It on trees and Restricted Shortest Common Supersequence(RSCS) are analogous problems. RSCS is a variant of SCS - Shortest Common Supersequence [10].

Shortest Common Supersequence (SCS)
(decision version)
Instance: A set of strings $S = s_1, \ldots, s_\ell$ over an alphabet Σ, an integer Λ.
Question: Does there exist a string $s \in \Sigma$ of length at most Λ that is a supersequence of each string in S?

Restricted Shortest Common Supersequence (RSCS)
(decision version)
Instance: A set of ρ-strings $R = r_1, \ldots, r_\ell$ over an alphabet Σ, an integer Λ. (A ρ-string is a string with no identical consecutive symbols.)
Question: Does there exist a string $r \in \Sigma$ of length at most Λ that is a supersequence of each ρ-string in R?

Let SCS($|\Sigma_1|, \ell_1$) stand for the SCS problem parameterized by $|\Sigma_1|$ and ℓ_1 (ℓ_1 is the number of strings). The notation RSCS($|\Sigma_2|, \ell_2$) is used similarly.

Theorem 10 (Fellows et al. [13]). *SCS($|\Sigma_1|, \ell_1$) is FPT-reducible to RSCS ($|\Sigma_2|, \ell_2$).*

Theorem 11 (Fellows et al. [13]).

(a) Flood-It on trees and RSCS are analogous problems.
(b) Flood-It(c, t, λ) on trees is p-analogous to RSCS($|\Sigma|, \ell, \Lambda$).

By Theorem 11, results valid for RSCS can be inherited by Flood-It on trees:

Corollary 2 (Fellows et al. [13]). *[t]-Flood-It on trees is solvable in polynomial time.*

Corollary 3 (Fellows et al. [13]). *Flood-It(t, c) on trees is W[1]-hard.*

Phylogenetic Colored Trees

Flood-It played on trees can be applied to scheduling. Each color corresponds to an operation in the sequential process of manufacturing an object. In the input tree T, paths from the pivot to the leaves correspond to the manufacturing sequences for a number of different objects that share the same production line. A flooding to T then corresponds to a schedule of operations for the production line that allows all of the different objects to be manufactured. It may reasonably be the case that each object to be manufactured requires any given operation to be applied at most once.

Theorem 12 (Fellows et al. [13]). *[r]-Flood-It on general graphs can be solved in polynomial time.*

Definition 5. *A colored rooted tree is a pc-tree (phylogenetic colored tree) if no color occurs more than once in any path from the root to a leaf.*

Corollary 4 (Fellows et al. [13]). *Flood-It on trees remains NP-hard even when restricted to pc-trees with pivot root.*

Corollary 5 (Fellows et al. [13]). *Flood-It(t) on pc-trees with pivot root is W[1]-hard.*

Definition 6. *A pc-tree T is a cpc-tree (complete pc-tree) if each color occurs exactly once in any path from the root to a leaf.*

Cpc-trees are a special subclass of pc-trees. Many hard cases of Flood-It on pc-trees are easy to solve when restricted to cpc-trees; for example, while Flood-It on pc-trees remains NP-hard when d is the parameter, Flood-It on cpc-trees is trivially solved in FPT time.

As in biological applications the phylogenetic sequences are often complete, the complexity of flood-filling games for complete pc-trees is an interesting issue.

Theorem 13 (Fellows et al. [13]). *Flood-It on trees remains NP-hard even when restricted to cpc-trees with pivot root.*

In Corollary 1, it was shown that Flood-It remains NP-hard even when restricted to pc-trees with maximum orbit 4. Theorem 14 shows that Flood-It on cpc-trees can be solved in polynomial time when considering the maximum orbit as the parameter.

Theorem 14 (Fellows et al. [13]). $[o]$*-Flood-It on cpc-trees can be solved in polynomial time.*

Flood-It on 3-Colored Trees

Flood-It on 2-colored graphs is trivially solvable. Fleischer and Woeginger [35] proved that Flood-It remains NP-hard when restricted to 4-colored trees. Raiha and Ukkonen [37] proved that Shortest Common Supersequence over a binary alphabet is NP-complete, and Middendorf [32] proved that Shortest Common Supersequence over a binary alphabet remains NP-complete even if the given strings have the same length and each string contains exactly two ones.

In Middendorf's proof the instances of Shortest Common Supersequence do not have two consecutive ones; hence, without loss of generality, we can assume that the last character of each input string is '0' (since after each '1' there is a '0'). Using this fact, in [13] was proved the following theorem.

Theorem 15 (Fellows et al. [13]). $[c]$*-Flood-It on trees remains NP-hard when* $c = 3$*.*

3.2 Free-Flood-It on Trees

Theorem 16 (Fellows et al. [13]). $[r]$*-Free-Flood-It on general graphs can be solved in polynomial time.*

The proof of Theorem 16 is similar to the proof presented in Theorem 12 in [13].

Now we present a general framework for reducibility from Flood-It to Free-Flood-It provided in [13].

Definition 7. *Let G be a graph, $v \in V(G)$, and ℓ a positive integer. The graph $\psi(G, v, \ell)$ is constructed as follows: (i) create ℓ disjoint copies $G_1, \ldots, G_\ell$ of G; (ii) contract the copies $v_1, v_2, \ldots, v_\ell$ of v into a single vertex v^*.*

Definition 8. *Let $\mathscr{F}$ be a class of graphs. Then:*

$$\psi(\mathscr{F}) = \{G \mid G = \psi(G', v, \ell) \text{ for some triple } (G' \in \mathscr{F}, v \in V(G'), \ell > 0)\ \}.$$

Definition 9. *A class $\mathscr{F}$ of graphs is closed under operator ψ if $\psi(\mathscr{F}) \subseteq \mathscr{F}$.*

Examples of classes closed under ψ are chordal graphs and bipartite graphs.

Theorem 17 (Fellows et al. [13]). *Flood-It played on $\mathscr{F}$ is reducible in polynomial time to Free-Flood-It played on $\psi(\mathscr{F})$.*

Corollary 6 (Fellows et al. [13]). *Let $\mathscr{F}$ be a class of graphs closed under ψ. Then Flood-It played on $\mathscr{F}$ is reducible in polynomial time to Free-Flood-It played on $\mathscr{F}$.*

NP-hardness results valid for Flood-It can be inherited by Free-Flood-It:

Corollary 7 (Fellows et al. [13]). *$[d]$-Free-Flood-It on trees is NP-hard even when $d = 4$.*

Corollary 8 (Fellows et al. [13]). *Free-Flood-It on cpc-trees is NP-hard.*

Corollary 9 (Fellows et al. [13]). *$[c]$-Free-Flood-It on trees is NP-hard even when $c = 3$.*

The next theorem implies that Flood-It on pc-trees and Free-Flood-It on pc-trees are analogous, and parameterized versions of these problems are p-analogous.

Theorem 18. *In Free-Flood-It on pc-trees, there always exists an optimal free-flooding which is a flooding with pivot root.*

Corollary 10 (Fellows et al. [13]). *Free-Flood-It(t) on pc-trees is W[1]-hard.*

Corollary 11 (Fellows et al. [13]). *Free-Flood-It(t, r) on pc-trees with pivot root is in FPT.*

Table 3 summarizes the results presented in [13].

4 Flood-Filling Games on Other Classes of Graphs

In this section we consider the complexity of flood-filling games played on other classes of boards, such as split graphs, co-comparability graphs, powers of paths, powers of cycles and circular grids.

In [35], Flood-It was denoted by Honey-Bee-Solitaire.

Theorem 19 (Fleischer and Woeginger [35]). *Flood-It can be solved in polynomial time on co-comparability graphs.*

Theorem 20 (Fleischer and Woeginger [35]). *Flood-It on split graphs is NP-hard.*

Recently, Hon et al. [19] claim that Flood-It on AT-free graphs can be solved in polynomial time.

Theorem 21 (Hon et al. [19]). *Flood-It can be solved in polynomial time on AT-free graphs graphs.*

Table 3. Multivariate analysis of flood-filling games on trees.

Problem	Instance	Fixed constant	Parameters	Complexity
Flood-It	cpc-trees		–	NP-hard
	trees	$c \geq 3$	–	NP-hard
	pc-trees		c	FPT
	graphs		λ	FPT
	cpc-trees		d	FPT
	cpc-trees	o	–	Polynomial
	pc-trees		k	W[1]-hard
	graphs	r	–	Polynomial
	cpc-trees		r	FPT
	trees		c, d	FPT
	graphs		c, o	FPT
	trees		c, k	W[1]-hard
	trees		c, r	FPT
	pc-trees	$d \geq 2$, $o \geq 4$	–	NP-hard
	trees		d, k	FPT
	trees		o, k	W[1]-hard
	trees		k, r	FPT
Free-Flood-It	cpc-trees		–	NP-hard
	trees	$c \geq 3$	–	NP-hard
	trees	$d \geq 4$	–	NP-hard
	pc-trees		k	W[1]-hard
	cpc-trees		r	FPT
	trees		o, k	W[1]-hard
	pc-trees		k, r	FPT

The next results focus on the powers of some classes of graphs, and circular grids.

Corollary 12 (Souza et al. [41]). *Flood-It is solvable in polynomial time on $2 \times n$ circular grids.*

Lemma 3 (Souza et al. [41]). *Flood-It on C_n^2 is a particular case of Flood-It on circular grids.*

Figure 4 illustrates a C_n^2 as a particular instance of Flood-It on circular grids.

Corollary 13 (Souza et al. [41]). *Flood-It can be solved in polynomial time on C_n^2.*

Corollary 14 (Souza et al. [41]). *Flood-It can be solved in polynomial time on P_n^2.*

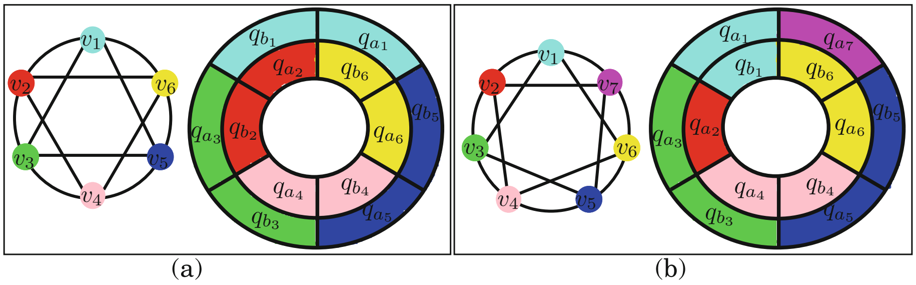


Fig. 4. (a) $2 \times n$ circular grid for even n; (b) $2 \times n$ circular grid for odd n.

In [31], Meeks and Vu present some upper bounds to the maximum number of moves that might be required to flood a arbitrary graph, which they show to be tight for particular classes of graphs. They also determine this maximum number of moves exactly when the underlying graph is a path, cycle, or a blow-up of a path or cycle (Table 4).

Table 4. Complexity of flood-filling games on particular graph classes

	Flood-It	Free-Flood-It
Caterpillars	P [35]	NP-hard [15]
P_n^2	P [41]	NP-hard [41]
Proper interval	P [35]	NP-hard [15]
Co-comparability	P [35]	NP-hard [15]
AT-free	P [19]	NP-hard [15]
C_n^2	P [41]	NP-hard [41]
$2 \times n$ circular grid	P [41]	NP-hard [41]
Split	NP-hard [35]	NP-hard [15]

Free-Flood-It

Theorem 22 (Fukui et al. [15]). *Free-Flood-It is NP-hard even on proper interval graphs, or even on caterpillars. These results still hold even if the maximum degree of the graphs is bounded by 3.*

Theorem 23 (Fukui et al. [15]). *Free-Flood-It on an interval graph can be solved in $O(4^c c^2 n^3)$ time.*

Notice that Theorem 23 shows that the problem is fixed-parameter tractable on interval graphs considering the number of colors as the parameter.

Flood-It on paths can be easily solved in $O(n^2)$ time by a dynamic programming [40], and as show in [41], the problem remains polynomially solvable when played on $2 \times n$ circular grids, C_n^2 and P_n^2. Although Free-Flood-It can be solved in polynomial time when played on paths and cycles, in [41], they show that Free-Flood-It is NP-hard when played on C_n^2, P_n^2 or circular grids.

Theorem 24 (Souza et al. [41]). *Free-Flood-It remains NP-hard on C_n^2.*

Corollary 15 (Souza et al. [41]). *Free-Flood-It remains NP-hard on P_n^2.*

Corollary 16 (Souza et al. [41]). *Free-Flood-It remains NP-hard on $2 \times n$ circular grids.*

5 The Size of a Minimum Vertex Cover as Parameter

From the parameterized complexity point of view, we consider the complexity of the Flood-It game played on graphs with bounded minimum vertex cover. We revisit the results presented in [8,12].

In the literature, there exist some results considering bounded values for different parameters of Flood-It. For instance, Flood-It is NP-hard on $n \times n$ grids colored with at least three *colors* [1], and it is also NP-hard on trees with *diameter* at most four [13,40]. In [13,40], the authors show some parameterized complexity results on Flood-It on trees, such as: Flood-It is W[1]-hard on trees when the number of leaves and the number of colors are parameters. On the other hand, it is easy to verify in $O^*(\lambda^\lambda)$ time whether Flood-It has a solution of size at most λ, and to obtain a kernel of size $O(c^d)$ where c and d are the parameters, the number of colors and the diameter of the graph, respectively. At this point, we review the parameterized complexity of Flood-It game considering the minimum vertex cover of the board (graph) as the parameter.

In [8,12], they present a parameterized complexity analysis of Flood-It with respect to the vertex cover number and with respect to the neighborhood diversity.

Theorem 25 (Fellows et al. [8,12]). *Flood-It is in FPT when parameterized by the vertex cover number ($|vc|$).*

Definition 10. *A graph $G(V,E)$ has neighborhood diversity $nd(G) = t$ if one can partition V into t sets $V_1, \dots, V_t$ such that, for all $v \in V$ and every $i \in \{1, \dots, t\}$, either v is adjacent to every vertex in V_i or it is adjacent to none of them. Note that each part V_i of G is either a clique or an independent set.*

The parameter neighborhood diversity is a natural generalization of the vertex cover number. In 2012, Lampis [26] showed that for every graph G we have $nd(G) \leq 2^{|vc|} + |vc|$, where $|vc|$ is the vertex cover number of G. The optimal neighborhood diversity decomposition of a graph G can be computed in $\mathcal{O}(n^3)$ time [26].

Corollary 17 (Fellows et al. [8,12]). *Flood-It is fixed-parameter tractable when parameterized by the neighborhood diversity.*

Theorem 26 (Fellows et al. [8,12]). *Flood-It admits polynomial kernelization when parameterized by the neighborhood diversity* (*nd*) *and the number of colors* (c)*.*

Theorem 27 (Fellows et al. [8,12]). *Flood-It parameterized by the vertex cover number does not admit a polynomial kernel, unless coNP $\subseteq$ NP/poly, even restricted to bipartite or chordal graphs.*

Corollary 18 (Fellows et al. [8,12]). *Flood-It does not admit a polynomial kernel, unless coNP $\subseteq$ NP/poly, even when the vertex cover number and the maximum number of bad moves to be played are considered as an aggregate parameter.*

Based on the Exponential Time Hypothesis (ETH) and the Strong Exponential Time Hypothesis (SETH), in [12] they obtained the following bounds for Flood-It.

Theorem 28 (Fellows et al. [12]). *Flood-It cannot be solved in*

1. $2^{o(|vc|+i_c)}n^{\mathcal{O}(1)}$ *time, unless ETH fails, and*
2. $(2-\varepsilon)^{i_c}n^{\mathcal{O}(1)}$ *time, unless SETH fails,*

even when the input graph is either bipartite or chordal, and i_c denotes the minimum number of colors of a maximum independent set of G.

Theorem 28 provides a strong bound for bipartite and chordal graphs. The next result gives us bounds for very restricted subclasses of bipartite and chordal graphs.

Theorem 29 (Fellows et al. [12]). *Unless ETH fails, Flood-It cannot be solved in $2^{o(|vc|+i_c)}n^{\mathcal{O}(1)}$ time, even when the input graph G is a tree with height two or a split graph formed by a clique and a set of pendant vertices.*

In [12] the author also provided an exact algorithm for Flood-It.

Theorem 30 (Fellows et al. [12]). *Flood-It can be solved in $\mathcal{O}(2^{\mathcal{O}(|vc|\, log(i_c\,|vc|))}$ $n^{\mathcal{O}(1)})$ time, where $|vc|$ is the vertex cover number and i_c is the minimum number of colors of a maximum independent set of G.*

In [30] they show that the minimum number of moves required to flood any given graph G is equal to the minimum, taken over all spanning trees T of G, of the number of moves required to flood T. This result can be applied to give polynomial-time algorithms for flood-filling problems.

Corollary 19 (Meeks and Scott [30]). *Free-Flood-It is solvable in polynomial time on subdivisions of any fixed graph H.*

6 Final Considerations

We revisited recent results (see [1,7,8,12–15,19,25,28–31,35,40,41]) on flood-filling games. Many complexity issues on Flood-It and Free-Flood-It have recently been investigated, and these games have presented interesting behavior when played on non-grid graphs. We also briefly presented a multivariate investigation of the complexity of Flood-It and Free-Flood-It when played on trees provided in [13,39,40]. During our analysis we remember that Flood-It on trees is analogous to Restricted Shortest Common Supersequence, and Flood-It remains NP-hard on 3-colored trees. We also revisited a general framework for reducibility from Flood-It to Free-Flood-It. From the parameterized complexity point of view, we revisited recent results which show that: Flood-It game played on graphs with bounded minimum vertex cover is fixed-parameter tractable, and admits polynomial kernelization when besides the minimum vertex cover, the number of colors is also part of the parameter.

6.1 Open Problems

We present some open problems on flood-filling games:

- What are the complexities of $[c = 3]$-Flood-It and $[c = 3]$-Free-Flood-It on $k \times n$ grids, in the case that $k \geq 3$ is a fixed integer?
- Does Flood-It(r) remain fixed-parameter tractable on general pc-trees?

Motivated by the general framework for reducibility from Flood-It to Free-Flood-It presented in [13], the following question arises:

- Is there any graph class for which Free-Flood-It can be solved in polynomial time, but Flood-It is NP-hard?

In addition, it is interesting to identify new graph classes for which Flood-It can be non-trivially solvable in polynomial time. It is also interesting to identify other single parameters for which Flood-It is fixed-parameter tractable.

We could think of games as a way of defining graph parameters. The paper [34], gives a game-defined look at graph structure. From this perspective, they show that it follows that branch-width is polynomially computable for planar graphs. Maybe a parameter such as "flood-filling number" will be similarly a useful structural parameter.

6.2 Flood-Filling Games in Teaching Computational Thinking

From the beginning of time, humans have generated knowledge of the world around them and have developed procedures (or algorithms) in order to reach their goals [21]. This makes computer science crucial to the development of society, and modular thinking as a fundamental tool for human creative work [20].

In [22], an efficient and didactic method of teaching computational thinking (or algorithmic thinking) by introducing the concepts gradually is described. This is done by using the spiral curriculum, which goes back to the research of Jerome Bruner in the 1960s. The Center for Computer Science Education of ETH Zurich follows this principle when designing teaching materials. The increase in knowledge occurs gradually, along with the development of intuition and the ability to abstract. Thus, students can be introduced to computational thinking from primary school, learning to develop their programs in a modular and structured way.

The elaboration of the spiral curriculum has many levels of depth [21], making it an excellent tool for developing algorithmic thinking. The algorithm concept is very abstract and can be introduced gradually. Initially, one can work with specific inputs of a problem and intuitive strategies. Then, one can work with more robust strategies, analyzing the universe of viable solutions when possible. At a further level, one can analyze the quality of the solutions, and then look for good solutions, when the analysis of all solutions is not possible anymore.

One of the well-studied methods for teaching computational thinking is the use of digital games. We can find in the literature both studies of the didactic use of games already available and commonly used for recreation [4] and the development of games and platforms specifically aimed at computer education [22,23].

The flood-filling game is available on a variety of websites and applications and it can be configured for board dimensions and number of colors. Using the spiral approach, one can initially propose matches to the students with a small board and a small number of colors so that they find viable solutions and even list all of them. At this stage, they may observe that certain flooding moves do not change the number of tiles flooded, while other choices increase the flood. They may notice that among the colors that increase the flood, some increase it more than others. When comparing all the solutions found, they learn to measure the quality of each solution, noting that some have fewer moves than others. The goal of the game is then understood and the optimal solutions are those that have the least number of moves. Looking at these solutions, some concepts of simple and robust strategies, such as greedy, can be introduced. As the size of the board and the number of colors is increased, finding all the solutions is no longer a feasible task. Students are then prompted to search for heuristics that find a good solution. At this step, one can observe and discuss some algorithm development strategies, which can guarantee optimal solutions for some problems but not for others, such as flood-filling, and it is possible to explain the concept of NP-difficulty. The flood-filling game is useful for demonstrating and teaching many basic concepts of computation, from trivial definitions about graphs such as "neighborhood" and "the distance between two vertices", to more complex concepts. Clearly, a study on which concepts should be inserted at each age and school subject is needed [20].

References

1. Arthur, D., Clifford, R., Jalsenius, M., Montanaro, A., Sach, B.: The complexity of flood filling games. In: Boldi, P., Gargano, L. (eds.) FUN 2010. LNCS, vol. 6099, pp. 307–318. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13122-6_30
2. Aschwanden, C.: Spatial simulation model for infectious viral disease with focus on sars and the common flu. In: 37th Annual Hawaii International Conference on System Sciences, HICSS (2004)
3. Barone, P., Bonizzoni, P., Vedova, G.D., Mauri, G.: An approximation algorithm for the shortest common supersequence problem: an experimental analysis. In: ACM Symposium on Applied, Computing, pp. 56–60 (2001)
4. Becker, K.: Teaching with games: the minesweeper and asteroids experience. J. Comput. Sci. Coll. **17**, 23–33 (2001)
5. Bodlaender, H.L., Fellows, M.R., Hallett, M.T., Wareham, T., Warnow, T.: The hardness of perfect phylogeny, feasible register assignment and other problems on thin colored graphs. Theor. Comput. Sci. **244**, 167–188 (2000)
6. Chor, B., Fellows, M., Ragan, M.A., Razgon, I., Rosamond, F., Snir, S.: Connected coloring completion for general graphs: algorithms and complexity. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 75–85. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73545-8_10
7. Clifford, R., Jalsenius, M., Montanaro, A., Sach, B.: The complexity of flood-filling games. Theory Comput. Syst. **50**(1), 72–92 (2012)
8. dos Santos Souza, U., Rosamond, F., Fellows, M.R., Protti, F., da Silva, M.D.: The flood-it game parameterized by the vertex cover number. In: LAGOS 2015 - VIII Latin-American Algorithms, Graphs and Optimization Symposium, Electronic Notes in Discrete Mathematics, vol. 50, pp. 35–40 (2015)
9. Fellows, M.R., Fertin, G., Hermelin, D., Vialette, S.: Sharp tractability borderlines for finding connected motifs in vertex-colored graphs. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 340–351. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73420-8_31
10. Fellows, M.R., Hallett, M.T., Stege, U.: Analogs and duals of the mast problem for sequences and trees. J. Algorithms **49**(1), 192–216 (2003)
11. Fellows, M.R., Hallett, M.T., Wareham, H.T.: DNA physical mapping: three ways difficult. In: Lengauer, T. (ed.) ESA 1993. LNCS, vol. 726, pp. 157–168. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57273-2_52
12. Fellows, M., Protti, F., Rosamond, F., da Silva, M.D., Souza, U.S.: Algorithms, kernels and lower bounds for the flood-it game parameterized by the vertex cover number. Discrete Appl. Math. **245**, 94–100 (2017)
13. Fellows, M.R., dos Santos Souza, U., Protti, F., da Silva, M.D.: Tractability and hardness of flood-filling games on trees. Theor. Comput. Sci. **576**, 102–116 (2015)
14. Fleischer, R., Woeginger, G.J.: An algorithmic analysis of the honey-bee game. In: Boldi, P., Gargano, L. (eds.) FUN 2010. LNCS, vol. 6099, pp. 178–189. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13122-6_19
15. Fukui, H., Otachi, Y., Uehara, R., Uno, T., Uno, Y.: On complexity of flooding games on graphs with interval representations. In: Akiyama, J., Kano, M., Sakai, T. (eds.) TJJCCGG 2012. LNCS, vol. 8296, pp. 73–84. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45281-9_7
16. Golumbic, M., Kaplan, H., Shamir, R.: On the complexity of dna physical mapping. Adv. Appl. Math. **15**, 251–261 (1994)

17. Gusfield, D.: Efficient algorithms for inferring evolutionary tree. Networks **21**, 19–28 (1981)
18. Hallett, M.T.: An integrated complexity analysis of problems from computational biology. Ph.D. thesis, University of Victoria (1996)
19. Hon, W.-K., Kloks, T., Liu, F.-H., Liu, H.-H., Wang, H.-L.: Flood-it on at-free graphs. arXiv preprint arXiv:1511.01806 (2015)
20. Hromkovič, J.: Homo informaticus: why computer science fundamentals are an unavoidable part of human culture and how to teach them. Bull. EATCS **115**, 112–122 (2015)
21. Hromkovič, J., Lacher, R.: The Computer science way of thinking in human history and consequences for the design of computer science curricula. In: Dagiene, V., Hellas, A. (eds.) ISSEP 2017. LNCS, vol. 10696, pp. 3–11. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71483-7_1
22. Hromkovic, J., Kohn, T., Komm, D., Serafini, G.: Algorithmic thinking from the start. In: The Education Column, Bulletin of the EATCS, p. 121 (2017)
23. Hromkovič, J., Serafini, G., Staub, J.: XLogoOnline: a single-page, browser-based programming environment for schools aiming at reducing cognitive load on pupils. In: Dagiene, V., Hellas, A. (eds.) ISSEP 2017. LNCS, vol. 10696, pp. 219–231. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71483-7_18
24. Lacroix, V., Fernandes, C.G., Sagot, M.F.: Motif search in graphs: application to metabolic networks. IEEE/ACM Trans. Comput. Biol. Bioinf. **3**(4), 360–368 (2006)
25. Lagoutte, A., Noual, M., Thierry, E.: Flooding games on graphs. Discrete Appl. Math. **164**, 532–538 (2014)
26. Lampis, M.: Algorithmic meta-theorems for restrictions of treewidth. Algorithmica **64**(1), 19–37 (2012)
27. McMorris, F.R., Warnow, T.J., Wimer, T.: Triangulating vertex-colored graphs. SIAM J. Discrete Math. **7**(2), 296–306 (1994)
28. Meeks, K., Scott, A.: The complexity of flood-filling games on graphs. Discrete Appl. Math. **160**, 959–969 (2012)
29. Meeks, K., Scott, A.: The complexity of free-flood-it on $2 \times n$ boards. Theor. Comput. Sci. **500**, 25–43 (2013)
30. Meeks, K., Scott, A.: Spanning trees and the complexity of flood-filling games. Theory Comput. Syst. **54**(4), 731–753 (2014)
31. Meeks, K., Vu, D.K.: Extremal properties of flood-filling games. arXiv preprint arXiv:1504.00596 (2015)
32. Middendorf, M.: More on the complexity of common superstring and supersequence problems. Theor. Comput. Sci. **125**, 205–228 (1994)
33. Moran, S., Snir, S.: Convex recolorings of strings and trees: definitions, hardness results and algorithms. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 218–232. Springer, Heidelberg (2005). https://doi.org/10.1007/11534273_20
34. Seymour, P.D., Thomas, R.: Call routing and the ratcatcher. Combinatorica **14**, 217–241 (1994)
35. Woeginger, G.J., Fleischer, R.: An algorithmic analysis of the honey-bee game. Theor. Comput. Sci. **452**, 75–87 (2012)
36. Rahmann, S.: The shortest common supersequence problem in a microarray production setting. Bioinformatics **19**(Suppl. 2), ii156–ii161 (2003)
37. Raiha, K.-J., Ukkonen, E.: The shortest common supersequence problem over binary alphabet is NP-complete. Theor. Comput. Sci. **16**, 187–198 (1981)
38. Sim, J., Park, K.: The consensus string problem for a metric is NP-complete. J. Discrete Algorithms **1**(1), 111–117 (2003)

39. Souza, U.S., Protti, F., Dantas da Silva, M.: Inundação em grafos. In: Proceedings of the 16th Congreso Latino Iberoamericano de Investigación Operativa & 44th Simpósio Brasileiro de Pesquisa Operacional, CLAIO/SBPO 2012 (2012)
40. dos Santos Souza, U., Protti, F., da Silva, M.D.: Parameterized complexity of flood-filling games on trees. In: Du, D.-Z., Zhang, G. (eds.) COCOON 2013. LNCS, vol. 7936, pp. 531–542. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38768-5_47
41. dos Santos Souza, U., Protti, F., Silva, M.: An algorithmic analysis of flood-it and free-flood-it on graph powers. Discrete Math. Theor. Comput. Sci. **16**(3), 279 (2014)

Infinity and Finite Arithmetic

Walter Gander(✉)

Department of Computer Science, ETH Zurich, Zürich, Switzerland
gander@inf.ethz.ch

Abstract. Juraj Hromkovič discusses in his book *Algorithmic Adventures* [4] the notion of infinity. The title of Chap. 3 is *Infinity Is Not Equal to Infinity, or Why Infinity Is Infinitely Important in Computer Science.* Infinity is indeed important in mathematics and computer science. However, a computer is a finite machine. It can represent only a finite set of numbers and an algorithm cannot run forever. Nevertheless we are able to deal with infinity and compute limits on the computer as is shown in this article.

1 Infinity Is Man-Made

The popular use of infinity describes something that never ends. An ocean may be said to be infinitely large. The sky is so vast that one might regard it as infinite. The number of stars we see in the sky seems to be infinite. And the time passes by, it never stops, it runs infinitely forever. Christians end the Lord's Prayer mentioning eternity, never stopping time:

For thine is the kingdom, the power, and the glory, for ever and ever.

However, we know today that all what we experience is *finite*. One liter of water contains 3.343×10^{25} molecules H_2O or 1.003×10^{26} hydrogen and oxygen atoms (according to Wolfram Alpha[1]). Every human is composed of a finite number of molecules, though it may be difficult to actually count them. In principle one could also count the finite number of atoms which form our planet.

The following quotation is attributed to Albert Einstein (though there is no proof for this):

"Two things are infinite: the universe and human stupidity; and I'm not sure about the universe."

Today scientists believe what Einstein refers to. Wolfram Alpha estimates that the universe contains 10^{80} atoms.

Thus we have to conclude:

Infinity does not exists in nature – it is man-made.

[1] https://www.wolframalpha.com/.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 377–392, 2018.
https://doi.org/10.1007/978-3-319-98355-4_21

2 Infinity in Mathematics

The Encyclopaedia Britannica[2] defines

> **Infinity**, the concept of something that is unlimited, endless, without bound. The common symbol for infinity, ∞, was invented by the English mathematician John Wallis in 1657. Three main types of infinity may be distinguished: the mathematical, the physical, and the metaphysical. Mathematical infinities occur, for instance, as the number of points on a continuous line or as the size of the endless sequence of counting numbers: $1, 2, 3, \ldots$. Spatial and temporal concepts of infinity occur in physics when one asks if there are infinitely many stars or if the universe will last forever. In a metaphysical discussion of God or the Absolute, there are questions of whether an ultimate entity must be infinite and whether lesser things could be infinite as well.

The notion of infinity in mathematics is not only very useful but also necessary. Calculus would not exist without the concept of limit computations. And this has consequences in physics. Without defining and computing limits we would e.g. not be able to define velocity and acceleration.

The model for a set with countable infinite elements are the natural numbers $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$. Juraj Hromkovič discusses the well known fact in [4] that the set of all rational numbers

$$\mathbb{Q} = \left\{ \frac{p}{q} \mid p, q \in \mathbb{N}, q \neq 0 \right\}$$

has the same cardinality as $\mathbb{N}$, thus $|\mathbb{N}| = |\mathbb{Q}|$. On the other hand there are more real numbers even in the interval $[0, 1]$, thus $|\mathbb{N}| < |\mathbb{R}|$. Juraj Hromkovič points in [4] also to the known fact that the real numbers are uncountable and that *there are at least two infinite sets of different sizes.*

We shall not discuss the difference in size of these infinite sets of numbers, rather we will concentrate in the following on computing limits.

3 Infinite Series

Infinite series $\sum_{k=1}^{\infty} a_k$ occur frequently in mathematics and one is interested if the partial sums

$$s_n = \sum_{k=1}^{n} a_k, \quad \lim_{n \to \infty} s_n = ?$$

converge to a limit or not.

It is well known that the *harmonic sum*

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots \tag{1}$$

[2] https://www.britannica.com/topic/infinity-mathematics#toc252429.

diverges. We can make this plausible by the following argument. We start with the geometric series and integrate:

$$\frac{1}{1-t} = 1 + t + t^2 + t^3 + \cdots, \quad |t| < 1$$

$$\int_0^z \frac{dt}{1-t} = -\log(1-z) = z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4} + \cdots, \quad z < 1.$$

If we let $z \to 1$ then the right hand side tends to the harmonic series, but the left hand side $-\log(1-z) \to \infty$ thus suggests that the harmonic series diverges.

By dividing the last equation by z we obtain

$$\frac{-\log(1-z)}{z} = 1 + \frac{z}{2} + \frac{z^2}{3} + \frac{z^3}{4} + \cdots$$

and by integrating we get the dilogarithm function

$$Li2(z) = \int_0^z \frac{-\log(1-u)}{u}\, du = z + \frac{z^2}{4} + \frac{z^3}{9} + \frac{z^4}{16} + \cdots.$$

For $z = 1$ we get the well known series of the reciprocal squares

$$\int_0^1 \frac{-\log(1-z)}{z}\, dz = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots = \frac{\pi^2}{6}.$$

This series is also the value of the ζ-function

$$\zeta(z) = 1 + \frac{1}{2^z} + \frac{1}{3^z} + \frac{1}{4^z} + \cdots$$

for $z = 2$. By dividing the dilogarithm function by z and integrating we get

$$\int_0^1 \frac{Li2(z)}{z}\, dz = 1 + \frac{1}{2^3} + \frac{1}{3^3} + \frac{1}{4^3} + \cdots = \zeta(3).$$

We can compute the numerical value for $\zeta(3)$ but a nice result as for $\zeta(2)$ is still not known.

4 Infinity and Numerical Computations

4.1 Straightforward Computation Fails

Consider again the harmonic series. The terms converge to zero. Summing up the series using floating point arithmetic such a series converges! The following program sums up the terms of the series until the next term is so small that it does not changes the partial sum `s` anymore.

```
s=1; term=1; k=1;
while s+term ~= s
  k=k+1; term=1/k; s=s+term;
end
```

If we would let this program run on a laptop, we would need to wait a long time till it terminates. In fact it is known (see [5], p. 556) that

$$s_n = \sum_{k=1}^{n} \frac{1}{k} = \log(n) + \gamma + \frac{1}{2n} + O\left(\frac{1}{n^2}\right)$$

where $\gamma = 0.57721566490153286\ldots$ is the *Euler-Mascheroni Constant*.

For $n = 10^{15}$ we get $s_n \approx \log(n) + \gamma = 35.116$ and numerically in IEEE arithmetic we have $s_n + 1/n = s_n$. So the harmonic series converges on the computer to $s \approx 35$.

My laptop needs 4 s to compute the partial sum s_n for $n = 10^9$. To go to $n = 10^{15}$ the computing time would be $T = 4 \cdot 10^6$ s or about 46 days! Furthermore the result would be affected by rounding errors.

Stammbach makes in [7] similar estimates and concludes:

> Das Beispiel ist instruktiv: Bereits im sehr einfachen Fall der harmonischen Reihe ist der Computer nicht in der Lage, den Resultaten der Mathematik wirklich gerecht zu werden. Und er wird es nie können, denn selbst der Einsatz eines Computers, der eine Million Mal schneller ist, bringt in dieser Richtung wenig.

If one uses the straightforward approach, he is indeed right with his critical opinion towards computers. However, it is well-known that the usual textbook formulas often cannot be used on the computer without a careful analysis as they may be unstable and/or the results may suffer from rounding errors. Already Forsythe [2] noticed that even the popular formula for the solutions of a quadratic equation has to be modified for the use in finite arithmetic. It is the task of numerical analysts to develop robust algorithms which work well on computers.

Let's consider also the series of the inverse squares:

$$\zeta(2) = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots.$$

Here the straightforward summation works in reasonable time, since the terms converge rapidly to zero. We can sum up until the terms become negligible compared with the partial sum:

```
s=1; term=1; n=1;
while s+term ~= s
  n=n+1; term=1/n^2; s=s+term;
end
```

The program terminates on my laptop in 0.4 s with $n = 94'906'266$ and $s = 1.644934057834575$. It is well known than numerically it is preferable to sum backwards starting with the smaller terms first. Doing so, with

```
n=94906266; s=0;
for k=n:-1:1
  s=s+1/k^2;
end
```

we get another value $s = 1.644934056311514$. Do the rounding error affect both results so much? Let's compare the results using Maple with more precision:

```
Digits:=30:
s:=0:
for k from 1 to 94906266 do
  s:=s+1.0/k^2:
od:
s                    1.64493405631151440597651536455
```

The result is the same as with the backwards summation. But we are far away from the limit value:

$$\frac{\pi^2}{6} - s = 1.0536 \times 10^{-8}$$

Thus also this series cannot be summed up in a straightforward fashion. On the computer it converges too early to a wrong limit.

Numerical analysts have developed summation methods for accelerating convergence. In the following we shall discuss some of theses techniques: *Aitken acceleration*, *extrapolation* and the ε-algorithm.

4.2 Aitkens Δ^2-Acceleration

Let us first consider Aitkens Δ^2-Acceleration [3]. Let $\{x_k\}$ be a sequence which converges linearly to a limit s. This means that for the error $x_k - s$ we have

$$\lim_{k\to\infty} \frac{x_{k+1} - s}{x_k - s} = \rho \neq 0, \quad |\rho| < 1.$$

Thus asymptotically

$$x_n - s \sim \rho\,(x_{n-1} - s) \sim \rho^n\,(x_0 - s)$$

or when solved for x_n we get a model of the asymptotic behavior of the sequence:

$$x_n \sim s + C\rho^n, \quad C = x_0 - s.$$

If we replace "$\sim$" by "$=$" and write the last equation for $n-2, n-1$ and n, we obtain a system of three equations which we can solve for ρ, C and s. Using Maple

```
solve({x[n-2]=s+C*rho^(n-2),x[n-2+1]=s+C*rho^(n-1),
       x[n]=s+C*rho^n},{rho,C,s}):
assign(%): s:=simplify(s);
```

we get the solution

$$s = \frac{x_n x_{n-2} - x_{n-1}^2}{x_n - 2\,x_{n-1} + x_{n-2}}.$$

Forming the new sequence $x'_n = s$ and rearranging, we get

$$x'_n = x_{n-2} - \frac{(x_{n-1} - x_{n-2})^2}{x_n - 2\,x_{n-1} + x_{n-2}} = x_{n-2} - \frac{(\Delta x_{n-2})^2}{\Delta^2 x_{n-2}}$$

which is called *Aitken's Δ^2-Acceleration.* The hope is that the new sequence $\{x'_n\}$ converges faster to the limit.

Assuming that $\{x'_n\}$ converges also linearly we can iterate this transformation and end up with a triangular scheme.

$$\begin{array}{llll}
x_k & x'_k & x''_k & \cdots \\
\hline
x_1 & & & \\
x_2 & & & \\
x_3 & x'_1 & & \\
x_4 & x'_2 & & \\
x_5 & x'_3 & x''_1 & \\
\vdots & \vdots & \vdots & \ddots
\end{array}$$

Let's apply this acceleration to compute the Euler-Mascheroni Constant γ. First we generate a sequence of $K+1$ partial sums

$$x_k = \sum_{j=1}^{2^k} \frac{1}{j} - \log(2^k), \quad k = 0, 1, \ldots, K.$$

```
function x=HarmonicPartial(K);
% HARMONICPARTIAL computes K+1 partial sums of the harmonic series
% and subtracts the logaritm.
y=[1:2^K]; y=1./y;
s=1; x=s;
for k=1:K
   s=s+sum(y(2^(k-1)+1:2^k)); x=[x s-log(2^k)];
end
x=x';
```

Then we program the function for the iterated Aitken-Scheme:

```
function T=AitkenAcc(x)
% AITKENACC tries to accelerate the convergent sequence x
%           with repeated Aitken-Delta-Square transformations
n=length(x); m=floor((n+1)/2);
T=zeros(n,m);
T(:,1)=x;
for j=2:m
  for k=2*j-1:n
    Delta2=T(k,j-1)-2*T(k-1,j-1)+T(k-2,j-1);
    if Delta2==0, break, end
    T(k,j)=T(k-2,j-1)-(T(k-1,j-1)-T(k-2,j-1))^2/Delta2;
  end
end
```

Now we can call the main program

```
K=8
x=HarmonicPartial(K);
A=AitkenAcc(x)
```

We get

```
1.000000000000000
0.806852819440055
0.697038972213442 0.552329999700925
0.638415601177307 0.571280073489448
0.608140270989212 0.575806621446670 0.577227192023427
0.592759292636793 0.576875788763135 0.577206420206234
0.585007820346097 0.577132432059184 0.577213495239782 0.577211697690028
0.581116828669555 0.577195084788389 0.577215319609806 0.577215953496545
0.579167518337717 0.577210549043978 0.577215616874797 0.577215674740153
                                                                        0.577215691876342
```

The Aitken extrapolation gives 7 correct decimal digits of γ using only the first 256 terms of the series.

The convergence of the sequence $\{x_k\}$ is linear with the factor $\rho \approx 0.5$ as we can see by computing the quotients $(x_{k+1} - \gamma)/(x_k - \gamma)$

```
(x(2:K)-0.57721566490153)./(x(1:K-1)-0.57721566490153)
```

```
   0.543154359030451
   0.521794077934416
   0.510751519455785
   0.505304547186624
   0.502629772912587
   0.501308676280856
   0.500652713588389
```

4.3 Extrapolation

We follow here the theory given in [3]. Extrapolation is used to *compute limits.* Let h be a discretization parameter and $T(h)$ an approximation of an unknown quantity a_0 with the following property:

$$\lim_{h \to 0} T(h) = a_0. \tag{2}$$

The usual assumption is that $T(0)$ is *difficult to compute* – maybe numerically unstable or requiring infinitely many operations. If we compute some function values $T(h_i)$ for $h_i > 0$, $i = 0, 1, \ldots, n$ and construct the interpolation polynomial $P_n(x)$ then $P_n(0)$ will be an approximation for a_0.

If a limit $a_0 = \lim_{m\to\infty} s_m$ is to be computed then, using the transformation $h = 1/m$ and $T(h) = s_m$, the problem is reduced to $\lim_{h=0} T(h) = a_0$.

To compute the sequence $\{P_n(0)\}$ for $n = 0, 1, 2, \ldots$ it is best to use Aitken-Neville interpolation (see [3]). The hope is that the diagonal of the Aitken-Neville scheme will converge to a_0. This is indeed the case if there exists an asymptotic expansion for $T(h)$ of the form

$$T(h) = a_0 + a_1 h + \cdots + a_k h^k + R_k(h) \quad \text{with} \quad |R_k(h)| < C_k h^{k+1}, \tag{3}$$

and if the sequence $\{h_i\}$ is chosen such that

$$h_{i+1} < c h_i \quad \text{with some} \quad 0 < c < 1,$$

i.e. if the sequence $\{h_i\}$ converges sufficiently rapidly to zero. In this case, the diagonals of the Aitken-Neville scheme converge faster to a_0 than the columns, see [1].

Since we extrapolate for $x = 0$ the recurrence for computing the Aitken-Neville scheme simplifies to

$$T_{ij} = \frac{h_i T_{i-1,j-1} - h_{i-j} T_{i,j-1}}{h_i - h_{i-j}}. \tag{4}$$

Furthermore if we choose the special sequence

$$h_i = h_0 2^{-i}, \tag{5}$$

then the recurrence becomes

$$T_{ij} = \frac{2^j T_{i,j-1} - T_{i-1,j-1}}{2^j - 1}. \tag{6}$$

Note that this scheme can also be interpreted as an algorithm for *elimination of lower order error terms* by taking the appropriate linear combinations. This process is called *Richardson Extrapolation* and is the same as Aitken–Neville extrapolation.

Consider the expansion

$$\begin{aligned} T(h) &= a_0 + a_1 h^2 + a_2 h^4 + a_3 h^6 + \cdots, \\ T\left(\tfrac{h}{2}\right) &= a_0 + a_1 \left(\tfrac{h}{2}\right)^2 + a_2 \left(\tfrac{h}{2}\right)^4 + a_3 \left(\tfrac{h}{2}\right)^6 + \cdots, \\ T\left(\tfrac{h}{4}\right) &= a_0 + a_1 \left(\tfrac{h}{4}\right)^2 + a_2 \left(\tfrac{h}{4}\right)^4 + a_3 \left(\tfrac{h}{4}\right)^6 + \cdots. \end{aligned} \tag{7}$$

Forming the quantities

$$T_{11} = \frac{4T\left(\frac{h}{2}\right) - T(h)}{3} \text{ and } T_{21} = \frac{4T\left(\frac{h}{4}\right) - T\left(\frac{h}{2}\right)}{3},$$

we obtain

$$\begin{aligned} T_{11} &= a_0 - \tfrac{1}{4} a_2 h^4 - \tfrac{5}{16} a_3 h^6 + \cdots, \\ T_{21} &= a_0 - \tfrac{1}{64} a_2 h^4 - \tfrac{5}{1024} a_3 h^6 + \cdots. \end{aligned}$$

Thus we have eliminated the term with h^2. Continuing with the linear combination

$$T_{22} = \frac{16 T_{21} - T_{11}}{15} = a_0 + \frac{1}{64} a_3 h^6 + \cdots$$

we eliminate the next term with h^4.

Often in the asymptotic expansion (3) the odd powers of h are missing, and

$$T(h) = a_0 + a_2h^2 + a_4h^4 + \cdots \tag{8}$$

holds. In this case it is advantageous to extrapolate with a polynomial in the variable $x = h^2$. This way we obtain faster approximations of (8) of higher order. Instead of (4) we then use

$$T_{ij} = \frac{h_i^2 T_{i-1,j-1} - h_{i-j}^2 T_{i,j-1}}{h_i^2 - h_{i-j}^2}. \tag{9}$$

Moreover, if we use the sequence (5) for h_i, we obtain the recurrence

$$T_{ij} = \frac{4^j T_{i,j-1} - T_{i-1,j-1}}{4^j - 1}, \tag{10}$$

which is used in the *Romberg Algorithm* for computing integrals.

For the special choice of the sequence h_i according to (5) we obtain the following extrapolation algorithm:

```
function A=ANS(x,factor);
% ANS  Aitken-Neville-Scheme for x,  factor is 2 or 4
K=length(x);
A(1,1)=x(1);
for i=2:K
  A(i,1)=x(i); vhj=1;
  for j=2:i
    vhj=vhj*factor;
    A(i,j)=(vhj*A(i,j-1)-A(i-1,j-1))/(vhj-1);
  end;
end
```

Let's turn now to the series with the inverse squares. We have mentioned before that this series is a special case (for $z = 2$) of the ζ-function

$$\zeta(z) = \sum_{k=1}^{\infty} \frac{1}{k^z}.$$

To compute $\zeta(2)$ we apply the Aitken-Neville scheme to extrapolate the limit of partial sums:

$$s_m = \sum_{k=1}^{m} \frac{1}{k^2}, \quad \zeta(2) = \lim_{n\to\infty} s_m.$$

So far we did not investigate the asymptotic behavior of s_m. Assuming that all powers of $1/m$ are present, we extrapolate with (6).

```
% compute partial sums s_m=\sum_{k=1}^(2^m), k=1,..,K
K=8;
y=[1:2^K]; y=1./y.^2;
for j=0:K-1
  s=sum(y(1:2^j)); x(j+1)=s;
end
x=x';
A=ANS(x,2)
```

We get the following results (we truncated the numbers to save space):

```
1.00000
1.25000 1.50000
1.42361 1.59722 1.629629
1.52742 1.63123 1.642569 1.644418529
1.58434 1.64127 1.644617 1.644909465 1.6449421945
1.61416 1.64398 1.644893 1.644933169 1.6449347501 1.64493451001
1.62943 1.64469 1.644928 1.644934037 1.6449340953 1.64493407423
1.63715 1.64487 1.644933 1.644934065 1.6449340678 1.64493406692

                                                  1.644934067322
                                                  1.644934066809 1.644934066805390
```

We have used $2^7 = 128$ terms of the series and obtained as extrapolated value for the limit $A_{8,8} = 1.644934066805390$. The error is $\pi^2 - A_{8,8} = 4.28 \cdot 10^{-11}$ so the extrapolation works well.

Asymptotic Expansion of ζ-Function. Consider the partial sum

$$s_{m-1} = \sum_{k=1}^{m-1} \frac{1}{k^z} = \zeta(z) - \sum_{k=m}^{\infty} \frac{1}{k^z}.$$

Applying the *Euler-MacLaurin Summation Formula* (for a derivation see [3]) to the tail we get the asymptotic expansion

$$\sum_{k=m}^{\infty} \frac{1}{k^z} \sim \frac{1}{z-1}\frac{1}{m^{z-1}} + \frac{1}{2}\frac{1}{m^z} + \frac{1}{z-1}\sum_{j=1} \binom{1-z}{2j} \frac{B_{2j}}{m^{z-1+2j}}.$$

The B_k are the *Bernoulli numbers*:

$$B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}, B_4 = -\frac{1}{30}, B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, \ldots$$

and $B_3 = B_5 = B_7 = \ldots = 0$. In general the series on the right hand side does not converge. Thus we get

$$\sum_{k=1}^{m-1} \frac{1}{k^z} + \frac{1}{2}\frac{1}{m^z} \sim \zeta(z) - \frac{1}{z-1}\sum_{j=0} \binom{1-z}{2j} \frac{B_{2j}}{m^{z-1+2j}}.$$

For $z = 3$ we obtain an asymptotic expansion with only even exponents

$$\sum_{k=1}^{m-1} \frac{1}{k^3} + \frac{1}{2}\frac{1}{m^3} \sim \zeta(3) - \frac{1}{2m^2} - \frac{1}{4m^4} + \frac{1}{12m^6} - \frac{1}{12m^8} \pm \cdots \qquad (11)$$

And for $z = 2$ we obtain

$$\sum_{k=1}^{m-1} \frac{1}{k^2} + \frac{1}{2}\frac{1}{m^2} \sim \zeta(2) - \frac{B_0}{m} - \frac{B_2}{m^3} - \frac{B_4}{m^5} - \cdots \tag{12}$$

which is an expansion with only odd exponents.

Knowing these asymptotic expansions, there is no need to accelerate convergence by extrapolation. For instance we can choose $m = 1000$ and use (11) to compute

$$\sum_{k=1}^{m-1} \frac{1}{k^3} + \frac{1}{2}\frac{1}{m^3} + \frac{1}{2m^2} + \frac{1}{4m^4} - \frac{1}{12m^6} = 1.202056903159593$$

and obtain $\zeta(3)$ to machine precision.

If, however, knowing only that the expansion has even exponents, we can extrapolate

```
K=8; m=1;
for j=1:K
  s=0;
  for k=1:m-1
    s=s+1/k^3;
  end
  x(j)=s+1/2/m^3;
  m=2*m;
end
A=ANS(x',4)
```

and get $A_{8,8} = 1.202056903159594$, and thus $\zeta(3)$ also to machine precision.

Could we also take advantage if the asymptotic development as e.g. in (12) contains only odd exponents? We need to modify the extrapolation scheme following the idea of Richardson by eliminating lower order error terms. If

$$T(h) = a_0 + a_1 h + a_2 h^3 + a_3 h^5 + a_4 h^7 + \cdots$$

we form the extrapolation scheme

$$\begin{array}{llll}
T_{11} = T(h) & & & \\
T_{12} = T(h/2) & T_{22} = 2T_{12} - T_{11} & & \\
T_{31} = T(h/4) & T_{32} = 2T_{32} - T_{12} & T_{33} = \dfrac{2^3 T_{32} - T_{22}}{2^3 - 1} & \\
\vdots & \vdots & \vdots & \ddots
\end{array}$$

Then T_{k2} has eliminated the term with h and T_{k3} has also eliminated the term with h^3. In general, for $h_i = h_0 2^{-i}$, we extrapolate the limit $\lim_{k\to\infty} x_k$ by initializing $T_{i1} = x_i, i = 1, 2, 3, \ldots$ and

$$T_{ij} = \frac{2^{2j-3} T_{ij-1} - T_{i-1j-1}}{2^{2j-3} - 1}, \quad i = 2, 3, \ldots,\ j = 2, 3, \ldots, i$$

This scheme is computed by the following function:

```
function A=ANSodd(x);
% ANSodd extrapolation for x with only odd exponents
K=length(x);
A(1,1)=x(1);
for i=2:K
  A(i,1)=x(i); vhj=2;
  for j=2:i
    A(i,j)=(vhj*A(i,j-1)-A(i-1,j-1))/(vhj-1);
    vhj=vhj*4;
  end;
end
```

We now extrapolate again the partial sum for the inverse squares:

```
K=8; m=1;
for j=1:8
  s=0;
  for k=1:m-1
    s=s+1/k^2;
  end
  x(j)=s+1/2/m^2;
  m=2*m;
end
A=ANSodd(x)
```

This time we converge to machine precision (we omitted again digits to save space):

```
0.500
1.125 1.750000
1.392 1.659722 1.64682539
1.519 1.646857 1.64502024 1.64496201552
1.582 1.645177 1.64493716 1.64493448049 1.64493426368
1.613 1.644964 1.64493416 1.64493407101 1.64493406778 1.644934067405063
1.629 1.644937 1.64493407 1.64493406688 1.64493406685 1.644934066849075
1.637 1.644934 1.64493406 1.64493406684 1.64493406684 1.644934066848226

                                                   1.644934066848803
                                                   1.644934066848226 1.644934066848226
```

4.4 The ε-Algorithm

In this section we again follow the theory given in [3]. Aitken's Δ^2-Acceleration uses as model for the asymptotic behavior of the error

$$x_n - s \sim C\rho^n.$$

By replacing "$\sim$" with "$=$" and by using three consecutive iterations we obtained in Subsect. 4.2 a nonlinear system for ρ, C and s. Solving for s we obtain a new sequence $\{x'\}$. A generalization of this was proposed by Shanks [6]. Consider the asymptotic error model

$$x_n - s \sim \sum_{i=1}^{k} a_i \rho_i^n, \quad \text{for } k > 1.$$

Replacing again "$\sim$" with "$=$" and using $2k+1$ consecutive iterations we get a system of nonlinear equations

$$x_{n+j} = s_{n,k} + \sum_{i=1}^{k} a_i \rho_i^{n+j}, \quad j = 0, 1, \ldots, 2k.$$

Assuming we can solve this system, we obtain a new sequence $x_n' = s_{n,k}$. This is called a *Shanks Transformation.*

Solving this nonlinear system is not easy and becomes quickly unwieldy. In order to find a different characterization for the Shanks Transformation, let $P_k(x) = c_0 + c_1 x + \cdots + c_k x^k$ be the polynomial with zeros $\rho_1, \ldots, \rho_k$, normalized such that $\sum c_i = 1$, and consider the equations

$$\begin{aligned} c_0(x_n - s_{n,k}) &= c_0 \sum_{i=1}^{k} a_i \rho_i^n \\ c_1(x_{n+1} - s_{n,k}) &= c_1 \sum_{i=1}^{k} a_i \rho_i^{n+1} \\ \vdots \quad &= \quad \vdots \\ c_k(x_{n+k} - s_{n,k}) &= c_k \sum_{i=1}^{k} a_i \rho_i^{n+k}. \end{aligned}$$

Adding all these equations, we obtain the sum

$$\sum_{j=0}^{k} c_j (x_{n+j} - s_{n,k}) = \sum_{i=1}^{k} a_i \rho_i^n \underbrace{\sum_{j=0}^{k} c_j \rho_i^j}_{P_k(\rho_i)=0},$$

and since $\sum c_i = 1$, the extrapolated value becomes

$$s_{n,k} = \sum_{j=0}^{k} c_j x_{n+j}. \tag{13}$$

Thus $s_{n,k}$ is a linear combination of successive iterates, a weighted average. If we knew the coefficients c_j of the polynomial, we could directly compute $s_{n,k}$.

Wynn established in 1956, see [8], the remarkable result that the quantities $s_{n,k}$ can be computed recursively. This procedure is called the *ε-algorithm.* Let

$\varepsilon_{-1}^{(n)} = 0$ and $\varepsilon_0^{(n)} = x_n$ for $n = 0, 1, 2, \ldots$. From these values, the following table using the recurrence relation

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + \frac{1}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}} \tag{14}$$

is constructed:

$$\begin{array}{lllll}
\varepsilon_{-1}^{(0)} & & & & \\
 & \varepsilon_0^{(0)} & & & \\
\varepsilon_{-1}^{(1)} & & \varepsilon_1^{(0)} & & \\
 & \varepsilon_0^{(1)} & & \varepsilon_2^{(0)} & \\
\varepsilon_{-1}^{(2)} & & \varepsilon_1^{(1)} & & \varepsilon_3^{(0)} \\
 & \varepsilon_0^{(2)} & & \varepsilon_2^{(1)} & & \cdots \\
\varepsilon_{-1}^{(3)} & & \varepsilon_1^{(2)} & & \cdots \\
 & \varepsilon_0^{(3)} & & \cdots & \\
\varepsilon_{-1}^{(4)} & & \cdots & &
\end{array} \tag{15}$$

Wynn showed that $\varepsilon_{2k}^{(n)} = s_{n,k}$ and $\varepsilon_{2k+1}^{(n)} = \frac{1}{S_k(\Delta x_n)}$, where $S_k(\Delta x_n)$ denotes the Shanks transformation of the sequence of the differences $\Delta x_n = x_{n+1} - x_n$. Thus every second column in the ε-table is in principle of interest. For the MATLAB implementation, we write the ε-table in the lower triangular part of the matrix E, and since the indices in MATLAB start at 1, we shift appropriately:

$$\begin{array}{llll}
0 = \varepsilon_{-1}^{(0)} = E_{11}, & & & \\
0 = \varepsilon_{-1}^{(1)} = E_{21} & x_1 = \varepsilon_0^{(0)} = E_{22}, & & \\
0 = \varepsilon_{-1}^{(2)} = E_{31} & x_2 = \varepsilon_0^{(1)} = E_{32} & \varepsilon_1^{(0)} = E_{33}, & \\
0 = \varepsilon_{-1}^{(3)} = E_{41} & x_3 = \varepsilon_0^{(2)} = E_{42} & \varepsilon_1^{(1)} = E_{43} & \varepsilon_2^{(0)} = E_{44}.
\end{array} \tag{16}$$

We obtain the algorithm

```
function [s,Er]=EpsilonAlgorithm(x);
% EPSILONALGORITHM computes the eps-scheme E for sequence x.
%   Output is the reduced scheme Er (only even columns) and
%   diagonal element s.
n=length(x);
E=zeros(n+1,n+1);
for i=1:n
  E(i+1,2)=x(i);
end
for i=3:n+1
  for j=3:i
    D=E(i,j-1)-E(i-1,j-1);
    if D==0, s=E(i,j-1); return, end
    E(i,j)=E(i-1,j-2)+1/D;
  end
```

```
end
Er=E(2:end,2:2:end); s=E(end,end);
```

The performance of the ε-algorithm is shown by accelerating the partial sums of the series

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}\pm\cdots=\ln 2.$$

We first compute the partial sums and then apply the ε-algorithm

```
format short e
k=4;
v=1;
for j=1:2*k+1
  y(j)=v/j; v=-v;
end
x=cumsum(y);
[s,Er]=EpsilonAlgorithm(x);
Er
log(2)-s
```

We obtain the result

```
Er =
   1.0000e+00            0            0            0            0
   5.0000e-01            0            0            0            0
   8.3333e-01   7.0000e-01            0            0            0
   5.8333e-01   6.9048e-01            0            0            0
   7.8333e-01   6.9444e-01   6.9333e-01            0            0
   6.1667e-01   6.9242e-01   6.9309e-01            0            0
   7.5952e-01   6.9359e-01   6.9317e-01   6.9315e-01            0
   6.3452e-01   6.9286e-01   6.9314e-01   6.9315e-01            0
   7.4563e-01   6.9335e-01   6.9315e-01   6.9315e-01   6.9315e-01
>> log(2)-s
ans =  -1.5179e-07
```

It is quite remarkable that we can obtain a result with about 7 decimal digits of accuracy by extrapolation using only partial sums of the first 9 terms, especially since the last partial sum still has no correct digit!

Acknowledgment. The author wishes to thank the referees for their very careful reading the paper.

References

1. Stoer, J., Bulirsch, R.: Introduction to Numerical Analysis. Springer, New York (1991)
2. Forsythe, G.E.: How Do You Solve a Quadratic Equation? Stanford Technical report No. CS40, 16 June 1966
3. Gander, W., Gander, M.J., Kwok, F.: Scientific Computing, an Introduction Using Maple and Matlab. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04325-8
4. Hromkovič, J.: Algorithmic Adventures. Springer, Berlin (2009). https://doi.org/10.1007/978-3-540-85986-4
5. Knopp, K.: Theorie und Anwendungen der unendlichen Reihen. Springer, Heidelberg (1947)
6. Shanks, D.: Non-linear transformation of divergent and slowly convergent sequences. J. Math. Phys. **34**, 1–42 (1955)
7. Stammbach, U.: Die harmonische Reihe: Historisches und Mathematisches. El. Math. **54**, 93–106 (1999)
8. Wynn, P.: On a device for computing the $e_m(S_n)$-transformation. MTAC **10**, 91–96 (1956)

A Modern View on Stability of Approximation

Ralf Klasing[1] and Tobias Mömke[2](✉)

[1] CNRS, LaBRI, University of Bordeaux, Talence, France
ralf.klasing@labri.fr

[2] University of Bremen, Bremen, Germany
moemke@uni-bremen.de

Abstract. In order to attack hard optimization problems that do not admit any polynomial-time approximation scheme (PTAS) or α-approximation algorithm for a reasonable constant α (or even with a worse approximability), Hromkovič et al. [12,31] introduced the notion of *Stability of Approximation*. The main idea of the stability concept is to try to split the set of all input instances into (potentially infinitely many) classes with respect to the achievable approximation ratio. The crucial point in applying this concept is to find parameters that well capture the difficulty of problem instances of the particular hard problem. The concept of stability of approximation turns out to be ubiquitous in the research of approximation algorithms and is applicable beyond approximation. In the literature, however, the relation to stability of approximation has stayed implicit. The purpose of this survey is to take a broader view and to explicitly analyze algorithmic problems and their algorithms with respect to stability of approximation.

"I would like to express my belief that, in order to make essential progress in algorithmics, one has to move from measuring the problem hardness in a worst-case manner to classifying the hardness of the instances of the investigated algorithmic problems."

Juraj Hromkovič [33]

1 Introduction

The study of algorithms investigates for which algorithmic problems a solution with specific properties can be obtained such that the computation does not exceed given resource constraints. The properties of the solutions are determined by the structure of the solution such as for instance "the solution is a tree" or "the solution is a sorted list of numbers". Some common resource measures are time

Research partially funded by Deutsche Forschungsgemeinschaft grant MO2889/1-1, and by "The Investments for the Future" Programme IdEx Bordeaux - CPU (ANR-10-IDEX-03-02).

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 393–408, 2018.
https://doi.org/10.1007/978-3-319-98355-4_22

and space (e.g., the algorithm should run in polynomial time and/or space), but there are further reasonable measures such as the amount of randomness, advice, security, non-determinism, etc.

In this survey, we are not able to cover the broad and deep field of algorithmics as a whole. Instead, we focus our attention on optimization problems. We therefore consider problems where the solutions are associated with a cost or profit, and we want to find a feasible solution (complying with the specified properties) such that the cost is minimized or the profit is maximized.

For simplicity, let us for now concentrate on minimization problems. Let us consider the traveling salesman problem (TSP). We are given a complete graph G with a cost function $\mathsf{cost}\colon E(G) \to \mathbb{R}^+$. The feasible solutions are exactly the Hamiltonian cycles of G, i.e., all sets of edges such that each vertex has degree 2 and the induced subgraph of these edges is connected. The goal is to find a feasible solution (Hamiltonian cycle) of minimum cost.

We would like to find a solution in polynomial time. Since the (general) TSP is known to be NP-hard, unless $\mathsf{P} = \mathsf{NP}$ we cannot guarantee to find an optimal solution. We therefore naturally obtain the notion of approximation algorithms: we do not restrict our view to optimal solutions. Instead we allow a set of feasible solutions that satisfy the following quality constraints: an algorithm A is an α-approximation algorithm if all solutions computed by A are at most a factor α worse than the optimum.

It is well-known that without restricting the cost function, the TSP cannot be approximated in polynomial time.[1] The situation is different if the cost function cost is a metric (in particular, cost satisfies the triangle inequality, i.e., for all distinct vertices $u, v, w \in V(G)$, $\mathsf{cost}(u, w) \leq \mathsf{cost}(u, v) + \mathsf{cost}(v, w)$). In metric graphs, the TSP can be approximated in polynomial time with an approximation ratio $\alpha = 1.5$ using Christofides' algorithm [21]. If we further restrict the costs such that $\mathsf{cost}(u, v) = 1$ for all $u, v \in V(G)$, the problem boils down to computing an *arbitrary* Hamiltonian cycle, which can be done in polynomial time (since G is complete).

The approximability of the TSP reveals a concept that we can see in a broader context: the hardness of a problem (here measured in the achievable approximation ratio) depends on the set of admissible instances. The more we restrict the class of instances, the easier it is to compute a solution to the problem. The dependency of admissible instances and approximation was first formalized by Hromkovič et al. [12,31], where the authors introduced the notion of *Stability of Approximation.*

[1] Since the encoding of numbers contributes to the size of the instance, the approximation ratio of the TSP is bounded by some function. For example, if the instance is an n-vertex graph with edge costs in the order of 2^{2^n}, the encoding of the instance consists of more than 2^n bits which allows for an exponential running time in n. Since these considerations are merely an artefact of the machine model, we consider optimization problems with a super-polynomial lower bound on the approximation ratio as inapproximable.

The idea behind this concept is to find a parameter (characteristic) of the input instances that captures the hardness of particular inputs. An approximation algorithm is called *stable* with respect to this parameter if its approximation ratio grows with this parameter but not with the size of the input instances. Note that the idea of the concept of approximation stability is similar to that of stability of numerical algorithms. Instead of observing the size of the change of the output value according to a small change of the input value, one looks for the size of the change of the approximation ratio according to a small change in the specification of the set of consistent input instances.

The concept of stability of approximation turns out to be ubiquitous in research on approximation algorithms and is applicable beyond approximation. In the literature, however, the relation to stability of approximation has stayed implicit. The purpose of this survey is to take a broader view and to explicitly analyze algorithmic problems and their algorithms with respect to stability of approximation.

The survey is organized as follows. In Sect. 2, we formally introduce the concept of stability of approximation. In Sect. 3, we study the concept in the context of graph problems where the edge costs form a metric. In Sect. 4, we investigate stability of approximation in the context of scheduling problems. In Sect. 5, we consider the problems of vertex coloring and maximum independent set in bounded-degree graphs. In Sect. 6, we study stability of approximation in the context of parameterized algorithms. In Sect. 7, we consider graph problems where the vertices are points in some space and the distances are measured according to the properties of the space. In particular, we consider Euclidean space and doubling spaces. In Sect. 8, we show that the concept of stability of approximation naturally translates to concepts such as online algorithms, by studying online scheduling and online matching. In Sect. 9, we draw several conclusions from our investigation of stability of approximation.

2 Stability of Approximation

We now give the formal definition of the stability of approximation algorithms [12,14,31,32].

In general, an optimization problem U is determined by a set of instances L_I, a set of feasible solutions $\mathcal{M}$ with subsets $\mathcal{M}(x)$ for each $x \in L_I$, a cost function $\mathsf{cost}\colon \mathcal{M} \to \mathbb{R}^+$ and a goal (minimization or maximization).[2] Additionally, we distinguish between instances L_I that we want to allow, and all instances L that are feasible (and thus $L_I \subseteq L$). For instance, when considering the metric TSP, L_I contains all (encodings of) edge-weighted graphs where the cost function is a metric. The definition of the TSP, however, does not depend on the cost restriction. Instead, we could use an arbitrary cost function. Then L is the set of all (encodings of) edge-weighted graphs $G = (V, E)$ with all cost functions $\mathsf{cost}\colon E \to \mathbb{R}^+$.

[2] For simplicity we assume instances and solutions to be encoded over the binary alphabet.

Let U be an optimization problem. For every $x \in L_I$, we define $Output_U(x) = \{y \in \mathcal{M}(x) \mid \mathsf{cost}(y) = \mathsf{goal}\{\mathsf{cost}(z) \mid z \in \mathcal{M}(x)\}\}$ as the set of optimal solutions for x and U, and $Opt_U(x) = \mathsf{cost}(y)$ for some $y \in Output_U(x)$. We say that an algorithm A is a *consistent algorithm for* U if, for every input $x \in L_I$, A computes an output $A(x) \in \mathcal{M}(x)$. We say that A *solves* U if, for every $x \in L_I$, A computes an output $A(x)$ from $Output_U(x)$. The time complexity of A is defined as the function $Time_A(n) = \max\{Time_A(x) \mid x \in L_I \cap \Sigma_I^n\}$ from $\mathbb{N}$ to $\mathbb{N}$, where $Time_A(x)$ is the length of the computation of A on x, and Σ_I is an alphabet called the *input alphabet* of U.

Let U be an optimization problem, and let A be a consistent algorithm for U. For every $x \in L_I$, the *approximation ratio* $R_A(x)$ *of* A *on* x is defined as $R_A(x) = \max\left\{\frac{\mathsf{cost}(A(x))}{Opt_U(x)}, \frac{Opt_U(x)}{\mathsf{cost}(A(x))}\right\}$. For any $n \in \mathbb{N}$, we define the *approximation ratio of* A as $R_A(n) = \max\{R_A(x) \mid x \in L_I \cap \Sigma_I^n\}$. For any positive real $\delta > 1$, we say that A is a δ*-approximation algorithm for* U if $R_A(x) \leq \delta$ for every $x \in L_I$. For every function $f\colon \mathbb{N} \to \mathbb{R}^{>1}$, we say that A is an $f(n)$*-approximation algorithm for* U if $R_A(n) \leq f(n)$ for every $n \in \mathbb{N}$.

Stability of approximation is concerned with the approximation behavior of instances in $L \setminus L_I$. To this end, we consider a distance function $h\colon L \to \mathbb{R}^{\geq 0}$ that expresses how much we deviate from L_I. We require h to be polynomial time computable and that $h(x) = 0$ for all $x \in L_I$.

For each $r \in \mathbb{R}^{\geq 0}$, we consider the ball $B_{r,h}(L_I) := \{x \in L \mid h(x) \leq r\}$. Let A be an algorithm that computes a feasible solution for each $x \in L$ and an α-approximation for each $x \in L_I$ and $\alpha \geq 1$. Then A is called *p-stable according to* h if for each $0 \leq r \leq p$ there is a constant α'_r such that A is an α'_r-approximation algorithm for each $x \in B_{h,r}(L_I)$. We call A *stable according to* h if it is p-stable according to h for all $p \in \mathbb{R}^{\geq 0}$.

If in the definition we replace the constant α'_r by a function $f_r\colon \mathbb{N} \to \mathbb{R}^{\geq 0}$ that maps the length of the encoding of the instance to a number, we call A f_r*-quasistable according to* h.

3 Relaxing the Metric

The original context in which stability of approximation was studied was in the context of graph problems where the edge costs form a metric [12,14,30,31].

A complete weighted graph $G = (V, E, \mathsf{cost})$ is called Δ_β-metric, for some $\beta \geq 1/2$, if the cost function $\mathsf{cost}(\cdot,\cdot)$ satisfies $\mathsf{cost}(v, v) = 0$, $\mathsf{cost}(u, v) = \mathsf{cost}(v, u)$, and the β-triangle inequality, i.e., $\mathsf{cost}(u, v) \leq \beta \cdot (\mathsf{cost}(u, x) + \mathsf{cost}(x, v))$ for all vertices $u, v, x \in V$. If $\beta > 1$ then we speak about the *relaxed triangle inequality*, and if $\beta < 1$ we speak about the *sharpened triangle inequality*.

The concept of *stability of approximation* has been successfully applied to several fundamental hard optimization problems. A nice example is the Traveling Salesman Problem (TSP), which does not admit any polynomial-time approximation algorithm with an approximation ratio bounded by a polynomial in the size of the input instance, but is $\frac{3}{2}$-approximable for metric input instances. Here, one can characterize the input instances by their "distance" to metric instances. This can be expressed by the β-triangle inequality for any $\beta \geq 1$.

In a sequence of papers [1,2,7,10–13,39], it was shown that

1. there are stable approximation algorithms for the TSP whose approximation ratio[3] grows with β, and
2. for every $\beta > \frac{1}{2}$ one can prove explicit lower bounds on the polynomial-time approximability growing with β.

Hence, one can partition the set of all input instances of the Traveling Salesman Problem into infinitely many subclasses according to the degree of violation of the triangle inequality, and for each subclass one can guarantee upper and lower bounds on the approximation ratio (that are linear in β).

Similar studies demonstrated that the β-triangle inequality can serve as a measure of hardness of the input instances for other optimization problems as well. In particular, for the problem of constructing 2-connected spanning subgraphs of a given complete edge-weighted graph, it was proved in [8] (with an explicit lower bound on the approximability) that this minimization problem is APX-hard even for graphs satisfying a sharpened triangle inequality for any $\beta > \frac{1}{2}$, i.e., even if the edge costs are from an interval $[1, 1+\varepsilon]$ for an arbitrarily small $\varepsilon > 0$. On the other hand, an upper bound of $\frac{2}{3} + \frac{1}{3} \cdot \frac{\beta}{1-\beta}$ on the approximability is achieved in [8] for all inputs satisfying the sharpened triangle inequality. For the problem of finding, for a given positive integer $k \geq 2$ and an edge-weighted graph G, a minimum k-edge- or k-vertex-connected spanning subgraph, a linear-time approximation algorithm was given in [9] with approximation ratio $\frac{\beta}{1-\beta}$ for any $\frac{1}{2} \leq \beta < 1$ (which does not depend on k).

Certain problems, such as the star p-hub center problem (Δ_β-SpHCP) [19] and the p-hub center problem (Δ_β-pHCP) [20], exhibit a *phase transition phenomenon* when parameterized by the β-triangle inequality. More precisely, considering the class $\mathcal{C}_\beta$ of Δ_β-metric graphs, there exists β_0 such that these problems are solvable in polynomial time when restricted to $\mathcal{C}_\beta$ for any $\beta \leq \beta_0$, whereas for $\beta > \beta_0$ they are approximable in the class $\mathcal{C}_\beta$ with upper and lower bounds on the approximation ratio depending on β.

More precisely, Chen et al. [19] showed that for an arbitrary $\varepsilon > 0$, to approximate Δ_β-SpHCP with a ratio $g(\beta) - \varepsilon$ is NP-hard, and $r(\beta)$-approximation algorithms were given for the same problem, where $g(\beta)$ and $r(\beta)$ are functions of β. If $\beta \leq \frac{3-\sqrt{3}}{2}$, one has $r(\beta) = g(\beta) = 1$, i.e., Δ_β-SpHCP is polynomial-time solvable. If $\frac{3-\sqrt{3}}{2} < \beta \leq \frac{2}{3}$, one has $r(\beta) = g(\beta) = \frac{1+2\beta-2\beta^2}{4(1-\beta)}$. For $\frac{2}{3} \leq \beta \leq 1$, $r(\beta) = \min\{\frac{1+2\beta-2\beta^2}{4(1-\beta)}, 1 + \frac{4\beta^2}{5\beta+1}\}$. Moreover, for $\beta \geq 1$, one has $r(\beta) = \min\{\beta + \frac{4\beta^2-2\beta}{2+\beta}, 2\beta+1\}$ and $g(\beta) = \beta + \frac{1}{2}$. For $\beta \geq 2$, the approximability of the problem (i.e., upper and lower bound) is linear in β.

Chen et al. [20] proved that for an arbitrary $\varepsilon > 0$, to approximate Δ_β-pHCP with a ratio $\tilde{g}(\beta) - \varepsilon$ is NP-hard, and $\tilde{r}(\beta)$-approximation algorithms

[3] The currently best approximation ratio is $\min\{4\beta, 3\beta/4 + 3\beta^2/4\}$ [7,39].

were given for the same problem, where $\tilde{g}(\beta)$ and $\tilde{r}(\beta)$ are functions of β. If $\beta \leq \frac{3-\sqrt{3}}{2}$, one has $\tilde{r}(\beta) = \tilde{g}(\beta) = 1$, i.e., Δ_β-pHCP is polynomial-time solvable. If $\frac{3-\sqrt{3}}{2} < \beta \leq \frac{5+\sqrt{5}}{10}$, one has $\tilde{r}(\beta) = \tilde{g}(\beta) = \frac{3\beta-2\beta^2}{3(1-\beta)}$. For $\frac{5+\sqrt{5}}{10} \leq \beta \leq 1$, $\tilde{r}(\beta) = \min\{\beta+\beta^2, \frac{4\beta^2+5\beta+1}{5\beta+1}\}$ and $\tilde{g}(\beta) = \frac{4\beta^2+3\beta-1}{5\beta-1}$. Moreover, for $\beta \geq 1$, one has $\tilde{r}(\beta) = 2\beta$ and $\tilde{g}(\beta) = \beta \cdot \frac{4\beta-1}{3\beta-1}$. For $\beta \geq 1$, the approximability of the problem (i.e., upper and lower bound) is linear in β.

4 Processing Time

We now turn our attention to a basic and well-studied scheduling problem. We want to minimize the flow time on multiple machines. Formally, we are given m identical machines and a set J of n jobs. Each job j has a processing time $1 \leq p_j \leq P$ and release time r_j. Jobs can be preempted, i.e., we can interrupt the processing of a running job and continue later without any penalties. The entire job has to be processed on the same machine and the task of the algorithm is to decide which job is scheduled on which machine.

For a given algorithm, C_j denotes the completion time of a job j, and the flow time of j is defined as $F_j = C_j - r_j$, i.e., the time that has passed between the release time and the completion time. Now, the goal is to minimize the total flow time $\sum_j F_j$.

Leonardi and Raz [37,38] showed that there is an $O(\log(\min\{\frac{n}{m}, P\}))$-approximation algorithm for this problem. At each point in time during the execution of the algorithm, it chooses those tasks that have the shortest remaining processing time (SRPT).

The parameter P leads to a natural distance function in the context of stability of approximation. We define L as the set of all instances, without restricting the maximum processing time of a job. Then L_I induces the subproblem with unit processing times, i.e., all processing times are exactly 1. For an instance x we define $h(x) = P(x) - 1$, where $P(x)$ is the maximum processing time over all jobs in x. Clearly, h is computable in polynomial time and $h(x) = 0$ for $x \in L_I$.

Since the approximation ratio increases monotonously with the distance from L_I, also the conditions for $B_{r,h}$ are satisfied. We conclude that the SRPT algorithm is stable according to h.

Garg and Kumar [26] considered a generalization of the problem, where each job can only be assigned to a specific subset of machines. Formally, for each job j there is a subset $S(j)$ of machines on which j may be scheduled. They showed that there is an $O(\log P)$-approximation algorithm for this problem. Additionally, they showed a lower bound of $\Omega\left(\frac{\log P}{\log \log P}\right)$, which means that the ratio is almost tight. From our analysis above, we directly obtain that also this algorithm is stable with respect to h.

5 Degree-Bounded Graphs

We now consider degree-bounded graphs. For a graph $G = (V, E)$, we write Δ for the maximum degree over all vertices in V.

5.1 Coloring

The maximum degree plays an important role when we want to color the vertices of G. We would like to find a minimum size set of colors such that each vertex is assigned a color and each edge is incident to two vertices with different colors. The chromatic number of G is the minimum number of colors sufficient to color all vertices of G. It is NP-hard to approximate the chromatic number of an n-vertex graph with an approximation ratio of $n^{1-\varepsilon}$ [44], for an arbitrary $\varepsilon > 0$.

There is a basic and well-known result on graph coloring: one can color each graph with $\Delta+1$ colors. We simply color the graph greedily. We iteratively choose uncolored vertices and color them with the first color c from $\{1, 2, \ldots, \Delta+1\}$ such that no neighbor is colored c. The color c exists because of the degree bound: there are at most Δ colors with which neighbors can be colored. In particular, the algorithm directly gives a $(\Delta+1)$-approximation algorithm. We can improve the approximation ratio by observing that one can check in polynomial time if a graph is bipartite (cf. [23]), which is equivalent to being 2-colorable. If the graph is bipartite, we can efficiently determine a 2-coloring. We can therefore either compute an optimal solution or we can assume that the optimum is at least 3. The modified algorithm has an approximation ratio of $(\Delta+1)/3$.

The degree of G determines a useful distance function. We define L as the set of all instances, without restricting the degree. Then L_I induces the subproblem with $\Delta = 3$, i.e., the first number such that graph coloring is NP-hard. For an instance x we define $h(x) = \max\{0, \Delta - 3\}$. The value of h is easy to compute and $h(x) = 0$ for $x \in L_I$.

Since the approximation ratio increases monotonously with Δ, we conclude that the coloring algorithm is stable according to h.

5.2 Independent Set

We continue with the maximum independent set problem in bounded-degree graphs. We would like to find a maximum-size set of vertices S such that they are pairwise non-adjacent.

A well-known result of Halldórsson and Radhakrishnan [29] states that there is a $(\Delta+2)/3$-approximation algorithm for this problem. The approximation ratio follows from a sophisticated analysis which includes the use of a generalization of Turán's bound.

As for coloring, finding a maximum independent set is easy for degree-2-bounded graphs (we cannot do better than to take every other vertex on each path or cycle). Therefore, the same sets of problem instances and the same distance function as for coloring also apply for maximum independent set. The algorithm is therefore stable according to h.

6 Parameterization

Stability of approximation also appears in the context of parameterized algorithms. As an example, let us again consider graph coloring.

While graph coloring is NP-hard to approximate in general (see Subsect. 5.1), for restricted graph classes the situation is much better. For instance, coloring a perfect graph is polynomially solvable [27].

Here, we focus on a class of graphs determined by their pathwidth. The pathwidth expresses the similarity of a graph to a simple path. In fact, the graphs of pathwidth 1 are exactly $\{K_3, K^*_{1,3}\}$-free graphs (or, equivalently, disjoint unions of caterpillar trees), where $K^*_{1,3}$ is the graph obtained from $K_{1,3}$ by subdividing each edge exactly once [24]. Since here we only use this parameter indirectly, we refer the reader to the book of Downey and Fellows [25] for a formal definition.

A graph has pathwidth zero if and only if it is a single vertex. For pathwidths $k \geq 2$, we have the following result.

Theorem 1 (Kierstead and Trotter [34]). *For graphs of pathwidth k there is a $(3k-2)$-approximation algorithm for graph coloring.*

Therefore, the pathwidth determines a natural distance function. We set L_I to be the set of all graphs of pathwidth zero. Then the distance function h is simply specified by $h(x) = k$ if the given instance x is a graph of pathwidth k.

We claim that the algorithm of Kierstead and Trotter is k-stable for each constant k. Since the dependence of the approximation ratio on k is monotonous, there is a constant α_r for each $r < k$ such that for all instances in $B_{r,h}(L_I)$ the algorithm is an α_r-approximation. It is left to show that h is polynomially computable. This is the point where we have to use that k is a constant: computing the pathwidth is a hard problem. For constant k, however, there is a polynomial time algorithm that computes the pathwidth (in the language of parameterized algorithms, computing the pathwidth is *fixed-parameter tractable*) [15,17].

For non-constant k, the problem of computing the pathwidth is NP-hard to approximate [16]. We therefore conclude that even though the algorithm of Kierstead and Trotter is k-stable for each constant k, it is neither stable nor quasi-stable.

7 Dimensionality

Graph problems usually have a natural version of the problem where the vertices are points in some space and the distances are measured according to the properties of the space. For instance, we can consider the TSP where the vertices are points in the Euclidean plane. Then a vertex u is a point (u_1, u_2) and the distance between two vertices u and v is the Euclidean distance $((u_1-v_1)^2+(u_2-v_2)^2)^{1/2}$.

One of the important factors influencing the hardness of a problem is the dimension of the considered space. Generally speaking, a higher dimension increases the freedom to choose instances and therefore leads to harder problems.

7.1 Euclidean Space

We now consider the Euclidean space more thoroughly. Many of the well-studied problems that are APX-hard in general metric spaces allow for a PTAS in the

Euclidean plane $\mathbb{R}^2$. This is the case for the traveling salesman problem [3,4], the Steiner tree problem [3,4], the Steiner forest problem [18], the k-median problem [5,35], and many related problems.

The hardness of these problems, however, crucially depends on the dimension of the Euclidean space. For $\log n$ dimensions, Trevisan showed that the TSP is APX-hard [43]. This forces us to focus on spaces $\mathbb{R}^d$ for small values of d (either constant, or in some cases in the order of $\log\log n$).

The running time of the $(1+\varepsilon)$-approximation algorithms also depends on ε to be a fixed constant. Since $(1+\varepsilon)$ is a factor, the absolute error increases with the input size n, and it is therefore desirable to have an ε that decreases with increasing n. We are interested in how small ε can be (depending on n) such that the algorithm still runs in polynomial time.

To this end, we again take the TSP as a showcase problem. Arora [3] showed how to obtain a randomized $(1 + 1/c)$-approximation in time $O(n(\log n)^{(O(\sqrt{d})\cdot c)^{d-1}})$. By choosing $\beta \in O(\sqrt{d})^{d-1}$, we obtain the running time $O(n(\log n)^{\beta\cdot c^{d-1}})$, and we want the time to be polynomial, i.e.,

$$\begin{aligned} n(\log n)^{\beta\cdot c^{d-1}} &= n^{O(1)} && \text{which implies}\\ \beta\cdot c^{d-1} &= \log_{\log n} n^{O(1)} = \frac{O(\log n)}{\log\log n} && \text{and thus}\\ c &= \Big(\frac{O(\log n)}{O(\sqrt{d})^{d-1}\log\log n}\Big)^{1/(d-1)}. \end{aligned}$$

We can formulate this in terms of stability of approximation by choosing L_I to be all 1-dimensional instances, L all d-dimensional instances for $d \geq 1$ and specifying the distance function h as follows. Let $d(x)$ be the number of dimensions used in instance x. Then $h(x) = d(x) - 1$. We conclude that the Euclidean TSP is $f(r)$-quasistable according to h with

$$f(r) = 1 + 1/\Big(\frac{O(\log n)}{O(\sqrt{\lfloor r+1\rfloor})^{\lfloor r+1\rfloor-1}\log\log n}\Big)^{1/(\lfloor r+1\rfloor-1)}.$$

7.2 Doubling Spaces

In the previous section, the only reason to restrict our view to the Euclidean space was that Arora's algorithm uses its properties. The dimensionality behaves orthogonally to the restrictions imposed by the Euclidean space. To this end, we now focus on a more general setting. Given an arbitrary metric space $\mathbb{M}$ with a distance function dist, let B_ℓ denote a ball in this space with radius ℓ according to dist.[4] Then the doubling dimension of $\mathbb{M}$ is the minimal number d such that for every ℓ, every ball B_ℓ can be covered by 2^d balls $B_{\ell/2}$ of radius $\ell/2$. Such a metric space is called a *doubling space* of dimension d. As an example, the doubling dimension of a 2-dimensional space with ℓ_1-norm is 2, since each

[4] Note that here we do not talk about the relaxation of a class of instances, but about the standard notation of balls in a metric space.

square can be covered by $2^2 = 4$ squares of half the radius. Doubling spaces of dimension $O(d)$ generalize the Euclidean space with d dimensions, and are significantly more general (cf. [6,22,28,42] and references therein).

As an example of a stability result according to the dimension of a doubling space we use the Maximum Scatter TSP (MSTSP). In the MSTSP, we are given a TSP instance. Now, the goal is to find a tour such that the *smallest* cost over all edges in the tour is *maximized*. In other words, the objective is to only use long jumps. The idea is that the order of points visited is scattered as much as possible within the considered graph. A solution to the MSTSP is required, for instance, when drilling holes and the work piece is heated up in the process. Then one should first process parts of the work piece far away from the last hole in order to prevent overheating.

Kozma and Mömke [36] showed that there is a PTAS for the MSTSP in doubling spaces of dimension d at most $O(\log \log n)$, where n is the number of vertices. More precisely, the result is a $(1+\varepsilon)$-approximation algorithm running in time $\tilde{O}\big(n^3 + 2^{O(K \log K)}\big)$, where $K \le \big(\frac{13}{\varepsilon}\big)^d$. We want to determine ε such that

$$\tilde{O}\big(n^3 + 2^{O(K \log K)}\big) = \log^c n \cdot \big(n^3 + 2^{O(K \log K)}\big) = n^{O(1)}. \tag{1}$$

The factor $\log^c n$ and the term n^3 can be hidden in the exponent $O(1)$ of the right-hand side of (1). We obtain $2^{O(K \log K)} = n^{O(1)}$. We further simplify the equation and obtain

$$K \log K = (13/\varepsilon)^d \log(13/\varepsilon)^d = O(\log n).$$

For sufficiently small ε then $1/\varepsilon^{d-1/2} < \log n$ and therefore

$$\varepsilon > (\log n)^{-1/(d-1/2)}.$$

The remaining steps are analogous to our analysis of the Euclidean space. It is sensible to define L_I to be the one-dimensional instances and to use the distance function $h(x) = d(x) - 1$. Then the algorithm is $f(r)$-quasistable according to h with

$$f(r) = 1 + (\log n)^{-1/(\lfloor r+1 \rfloor - 1/2)}.$$

8 Online Algorithms

The concept of stability of approximation is not restricted to approximation algorithms. It naturally translates to concepts such as online algorithms.

The source of hardness in online algorithms comes from incomplete knowledge rather than from time restrictions: we (i.e., the algorithm) have to take decisions before knowing the entire instance.

Formally, an online problem U is defined as follows. An instance of U is given as a sequence of requests $r_1, r_2, \ldots$ An online algorithm A for U has to provide a sequence of answers $a_1, a_2, \ldots$ The answer a_i depends on the requests $r_1, r_2, \ldots, r_i$ and the answers $a_1, a_2, \ldots, a_{i-1}$.

In order to focus on the essence of the problem, the algorithm does not have a time restriction. It is sufficient to provide a computable function that determines the answers.

There are several models of online computation. One of the most used and most reasonable models is to assume an oblivious adversary. This model provides a measure of worst-case complexity because we assume an adversary that knows our algorithm and tries to provide a worst possible request sequence for it. The adversary, however, is oblivious which means that he has to fix the request upfront, without knowing the algorithm's answers. (This does not change anything for deterministic online algorithms, but it provides a huge advantage for randomized online algorithms.)

Online problems are typically optimization problems and as in approximation algorithms, we would like to have a guaranteed quality of the computed solution. Unlike in approximation algorithms, however, the need to compromise quality in online algorithms is unconditional: we do not depend on conjectures such as $\mathsf{P} \neq \mathsf{NP}$. The analog to the approximation ratio in online algorithms is called *competitive ratio.*

An algorithm is *c-competitive*, if there is a constant $\alpha > 0$ such that for each sequence of requests with cost opt of an optimal offline solution, the cost alg of the solution computed by the algorithm is at most $\alpha + c \cdot \mathsf{opt}$.

We define stability for online algorithms analogous to Sect. 2, but we replace the approximation ratio by the competitive ratio. We note that unless we require the online algorithm to run in polynomial time, the requirement of the distance function h to be polynomial time computable is not essential for online algorithms. We therefore only require h to be a computable function.

8.1 Online Scheduling

We continue with the online view on the problems that we analyzed in Sect. 4. Again, we want to minimize the flow-time on multiple machines. We have formally defined the offline version of this problem in Sect. 4. The difference to the offline version is that now, the jobs arrive in an online fashion. Let $j_1, j_2, \ldots, j_n$ be the jobs of a problem instance. We then order the jobs by increasing release time. The i-th request reveals the i-th job j_i. Before a new job j_{i+1} (or the end of the input) is revealed, the algorithm has to fix the machine on which j_i will be processed.

Leonardi and Raz [37,38] showed a tight bound $\Theta(\log P)$ on the competitive ratio of this problem. Recall that P is the maximal processing time of a single job. Note that the upper bound matches the offline upper bound. For online algorithms, however, the lower bound has changed.

Again, P leads to a natural distance function in the context of stability of approximation. Recall that the set L contains all instances, without restricting the maximum processing time of a job and that L_I induces the subproblem with unit processing times. For an instance x we define $h(x) = P(x) - 1$, where $P(x)$ is the maximum processing time over all jobs in x.

Since the competitive ratio increases monotonously with the distance from L_I, also the conditions for $B_{r,h}$ are satisfied. We conclude that the SRPT algorithm is stable according to h.

Unlike in the case of offline algorithms, however, this result does not translate to the version of the problem where each job can only be assigned to a specific subset of machines. Garg and Kumar [26] showed that in this case, the competitive ratio is unbounded, independent of P. Therefore, for this version of the problem there is no algorithm that is stable according to h.

8.2 Online Matching

The k-server problem is one of the central problems studied in online computation. We are given a metric space $\mathbb{M}$ and a set S of k servers placed on various locations in $\mathbb{M}$. The initial positions of the servers are known. The online requests are points in $\mathbb{M}$ and have to be answered with a server from S, which then has to move to the requested point. The cost of moving a server equals the distance in $\mathbb{M}$ and the goal is to minimize the overall distance traveled by servers.

The k-server problem formalizes natural problems such as for instance real time delivery: the servers are delivery cars and the metric space is determined by clients within a street network. In such a network, it is natural to consider a capacitated version of the k-server problem. We assign a capacity to the servers: each delivery car can serve at most ℓ different clients.

If $\ell = 1$, the capacitated k-server problem boils down to a bipartite matching between the servers and the clients: each server is matched to at most one client and each client is served by exactly one server. The resulting problem is known as online bipartite matching.

Raghvendra [41] presented an algorithm for online bipartite matching which in the following we call RM-algorithm (for "robust matching," following the notation of the authors). Nayyar and Raghvendra [40] afterwards obtained a better analysis based on the parameter $\mu_{\mathbb{M}}(S)$ which is defined as the maximum ratio of the traveling salesman tour and the diameter over all subsets of S (or, more precisely, the graphs induced by S). To use the parameter, we rely on our relaxation of stability for online algorithms in which we defined the distance function to be only computable and did not require polynomial time. The meaning of the parameter $\mu_{\mathbb{M}}(S)$ becomes clear when we consider concrete classes of metric spaces $\mathbb{M}$. If $\mathbb{M}$ is the space determined by the shortest path metric of a line, $\mu_{\mathbb{M}}(S) = 2$. If $\mathbb{M}$ is a doubling space of dimension d, $\mu_{\mathbb{M}}(S) = O(n^{1-1/d})$.

The result of Nayyar and Raghvendra is that the RM-algorithm has a competitive ratio of $O(\mu_{\mathbb{M}}(S) \log^2 n)$, while every algorithm has a competitive ratio of $\Omega(\mu_{\mathbb{M}}(S))$. To analyze the stability of the RM-algorithm, we now fix the needed parameters.

We set L to the set of all feasible instances independent of the value of $\mu_{\mathbb{M}}(S)$. One can check that for an arbitrary metric space on three points the value of $\mu_{\mathbb{M}}(S)$ is at least 2, matching the value for a shortest path metric of a line. Since the latter class of instances is non-trivial, it is natural to set L_I to be all instances with $\mu_{\mathbb{M}}(S) \leq 2$. Let S_x be the set of servers located in the metric space $\mathbb{M}_x$ for an instance $x \in L$. We then define $h(x) = \max\{0, \mu_{\mathbb{M}_x}(S_x) - 2\}$, which satisfies all required conditions. Since the upper bound on the competitive ratio is non-constant, we do not have an analysis that states that the algorithm is stable according to h. However, we can claim that it is f-quasistable according to h for the function f with $f(n) = O(\mu_{\mathbb{M}}(S) \log^2 n)$.

9 Conclusion

We have seen that stability of approximation is a natural concept that appears in many circumstances. We can draw several conclusions from our investigation.

(i) Stability of approximation can be seen as a parameterized version of approximation algorithms. If the approximation ratio monotonously increases with a given parameter, this parameter is a natural base for a stability distance function. Conversely, for stability of approximation results, we could also state the algorithms as algorithms for general problem instances parameterized on the distance function.

(ii) Stability of approximation has similarities with the study of time complexity. The relation becomes clear when considering polynomial time approximation schemes. In this survey, we determined the impact of stability on the error ε. An alternative way to handle a PTAS would be to fix a specific running time and to determine the error ε depending on the stability distance function. We can see the situation as a pareto curve with the stability distance on one axis and the running time on another axis.

(iii) The concept of stability is not restricted to approximation algorithms. For example, it directly translates to stability of competitiveness in online algorithms. This insight motivates to further investigate the scope of stability in further contexts.

To summarize, we observe that classifying the hardness of the instances of the investigated algorithmic problems is a successful approach that is important in modern algorithmic research.

Acknowledgment. We thank the anonymous referees for their careful reading and valuable comments, which helped to improve the presentation of the paper.

References

1. Andreae, T.: On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality. Networks **38**(2), 59–67 (2001)
2. Andreae, T., Bandelt, H.-J.: Performance guarantees for approximation algorithms depending on parametrized triangle inequalities. SIAM J. Discret. Math. **8**(1), 1–16 (1995)
3. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM **45**(5), 753–782 (1998)
4. Arora, S.: Approximation schemes for NP-hard geometric optimization problems: a survey. Math. Progr. **97**(1–2), 43–69 (2003)

5. Arora, S., Raghavan, P., Rao, S.: Approximation schemes for Euclidean k-medians and related problems. In: Proceedings of 30th Annual ACM Symposium on the Theory of Computing (STOC 1998), pp. 106–113 (1998)
6. Bartal, Y., Gottlieb, L.-A., Krauthgamer, R.: The traveling salesman problem: low-dimensionality implies a polynomial time approximation scheme. SIAM J. Comput. **45**(4), 1563–1581 (2016)
7. Bender, M.A., Chekuri, C.: Performance guarantees for the TSP with a parameterized triangle inequality. Inf. Process. Lett. **73**(1–2), 17–21 (2000)
8. Böckenhauer, H.-J., Bongartz, D., Hromkovič, J., Klasing, R., Proietti, G., Seibert, S., Unger, W.: On the hardness of constructing minimal 2-connected spanning subgraphs in complete graphs with sharpened triangle inequality. Theor. Comput. Sci. **326**(1–3), 137–153 (2004)
9. Böckenhauer, H.-J., et al.: On k-connectivity problems with sharpened triangle inequality. J. Discret. Algorithms **6**(4), 605–617 (2008)
10. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: Approximation algorithms for the TSP with sharpened triangle inequality. Inf. Process. Lett. **75**(3), 133–138 (2000)
11. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: An improved lower bound on the approximability of metric TSP and approximation algorithms for the TSP with sharpened triangle inequality. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 382–394. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46541-3_32
12. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem. Theor. Comput. Sci. **285**(1), 3–24 (2002)
13. Böckenhauer, H.-J., Seibert, S.: Improved lower bounds on the approximability of the traveling salesman problem. RAIRO - Theor. Inf. Appl. **34**(3), 213–255 (2000)
14. Böckenhauer, H.-J., Seibert, S., Hromkovič, J.: Stability of approximation. In: Handbook of Approximation Algorithms and Metaheuristics. Chapman and Hall/CRC (2007)
15. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**(6), 1305–1317 (1996)
16. Bodlaender, H.L., Gilbert, J.R., Hafsteinsson, H., Kloks, T.: Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. J. Algorithms **18**(2), 238–255 (1995)
17. Bodlaender, H.L., Kloks, T.: Efficient and constructive algorithms for the pathwidth and treewidth of graphs. J. Algorithms **21**(2), 358–402 (1996)
18. Borradaile, G., Klein, P.N., Mathieu, C.: A polynomial-time approximation scheme for Euclidean Steiner forest. ACM Trans. Algorithms **11**(3), 19:1–19:20 (2015)
19. Chen, L.-H., et al.: Approximability and inapproximability of the star p-hub center problem with parameterized triangle inequality. J. Comput. Syst. Sci. **92**, 92–112 (2018)
20. Chen, L.-H., Hsieh, S.-Y., Hung, L.-J., Klasing, R.: The approximability of the p-hub center problem with parameterized triangle inequality. In: Cao, Y., Chen, J. (eds.) COCOON 2017. LNCS, vol. 10392, pp. 112–123. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62389-4_10
21. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report 388, Graduate School of Industrial Administration, Carnegie-Mellon University (1976)
22. Clarkson, K.L.: Nearest neighbor queries in metric spaces. Discret. Comput. Geom. **22**(1), 63–93 (1999)

23. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press, Cambridge (2009)
24. Ding, G., Dziobiak, S.: On 3-connected graphs of path-width at most three. SIAM J. Discret. Math. **27**(3), 1514–1526 (2013)
25. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science. Springer, London (2013). https://doi.org/10.1007/978-1-4471-5559-1
26. Garg, N., Kumar, A.: Minimizing average flow-time: upper and lower bounds. In: Proceedings of 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), pp. 603–613. IEEE Computer Society (2007)
27. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Algorithms and Combinatorics, vol. 2. Springer, Heidelberg (1988). https://doi.org/10.1007/978-3-642-78240-4
28. Gupta, A. Krauthgamer, R., Lee, J.R.: Bounded geometries, fractals, and low-distortion embeddings. In: Proceedings of 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003), pp. 534–543. IEEE Computer Society (2003)
29. Halldórsson, M.M., Radhakrishnan, J.: Greed is good: approximating independent sets in sparse and bounded-degree graphs. Algorithmica **18**(1), 145–163 (1997)
30. Hromkoviĉ, J.: Stability of approximation algorithms and the knapsack problem. In: Karhumäki, J., Maurer, H., Paun, G., Rozenberg, G. (eds.) Jewels are Forever, pp. 238–249. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-642-60207-8_21
31. Hromkovič, J.: Algorithmics for Hard Problems - Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-662-05269-3
32. Hromkovič, J.: Stability of approximation in discrete optimization. In: Levy, J.-J., Mayr, E.W., Mitchell, J.C. (eds.) TCS 2004. IIFIP, vol. 155, pp. 3–18. Springer, Boston, MA (2004). https://doi.org/10.1007/1-4020-8141-3_2
33. Hromkovič, J.: Algorithmics – is there hope for a unified theory? In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 181–194. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13182-0_17
34. Kierstead, H.A., Trotter, W.T.: An extremal problem in recursive combinatorics. Congr. Numer. **33**, 143–153 (1981)
35. Kolliopoulos, S.G., Rao, S.: A nearly linear-time approximation scheme for the Euclidean k-median problem. SIAM J. Comput. **37**(3), 757–782 (2007)
36. Kozma, L., Mömke, T.: Maximum scatter TSP in doubling metrics. In: Proceedings of 28th ACM-SIAM Symposium on Discrete Algorithms (SODA 2017), pp. 143–153 (2017)
37. Leonardi, S., Raz, D.: Approximating total flow time on parallel machines. In: Proceedings of 29th Annual ACM Symposium on the Theory of Computing (STOC 1997), pp. 110–119. ACM (1997)
38. Leonardi, S., Raz, D.: Approximating total flow time on parallel machines. J. Comput. Syst. Sci. **73**(6), 875–891 (2007)
39. Mömke, T.: An improved approximation algorithm for the traveling salesman problem with relaxed triangle inequality. Inf. Process. Lett. **115**(11), 866–871 (2015)
40. Nayyar, K., Raghvendra, S.: An input sensitive online algorithm for the metric bipartite matching problem. In: Proceedings of 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2017), pp. 505–515. IEEE Computer Society (2017)

41. Raghvendra, S.: A robust and optimal online algorithm for minimum metric bipartite matching. In: APPROX-RANDOM. LIPIcs, vol. 60, pp. 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
42. Talwar, K.: Bypassing the embedding: algorithms for low dimensional metrics. In: Proceedings of 36th Annual ACM Symposium on Theory of Computing (STOC 2004), pp. 281–290 (2004)
43. Trevisan, L.: When Hamming meets Euclid: the approximability of geometric TSP and Steiner tree. SIAM J. Comput. **30**(2), 475–485 (2000)
44. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory Comput. **3**(1), 103–128 (2007)

$\mathcal{NP}$-Hardness of Equilibria in Case of Risk-Averse Players

Marios Mavronicolas[1] and Burkhard Monien[2](✉)

[1] Department of Computer Science, University of Cyprus, 1678 Nicosia, Cyprus
mavronic@ucy.ac.cy

[2] Faculty of Electrical Engineering, Computer Science and Mathematics, University of Paderborn, 33102 Paderborn, Germany
bm@upb.de

Abstract. We study the complexity of deciding the existence of *mixed equilibria* for minimization games where players use *valuations* other than expectation to evaluate their *costs*. We consider *risk-averse* players seeking to minimize the sum $\mathsf{V} = \mathsf{E} + \mathsf{R}$ of *expectation* E and a *risk valuation* R of their costs; R is non-negative and vanishes exactly when the cost incurred to a player is constant over all choices of strategies by the other players. In a V-*equilibrium,* no player can unilaterally reduce her cost.

The results presented in this paper show that the question whether a 2-player game with such risk-modeling valuations V has a V-equilibrium is strongly $\mathcal{NP}$-hard under very mild assumptions. We only have to ask that the valuation functions are strictly quasiconcave or that they fulfill the Weak-Equilibrium-for-Expectation property and that additionally some 2-strategy game has no V-equilibrium. These conditions are fulfilled for the functions from Markowitz's Mean-Variance approach like $\mathsf{E} + \mathsf{Var}$, $\mathsf{E} + \mathsf{SD}$ and the Sharpe Ration, and also for Conditional Value-at-Risk, recently very popular for modeling volatile economic circumstances.

1 Introduction

1.1 Motivation

In a *game,* each *player* is using a *mixed strategy,* a probability distribution over her *strategies*; her *cost* depends on the choices of all players and is evaluated by a *valuation*, a function from probability distributions to reals. The most prominent valuation in *Non-Cooperative Game Theory* is *expectation*, where each player minimizes her expected cost. A drawback of expectation is that it may not accommodate *risk* and its impact on strategic decision (cf. [3]). Indeed, *risk-averse* players [1] are willing to accept a larger amount of payment rather than to gamble and take the risk of a larger cost. According to [15], "a risk-averse

Partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-the-Fly-Computing" (SFB 901), and by funds for the promotion of research at University of Cyprus.

player is willing to pay something for certainty". In many experiments (see [19]) it has been demonstrated that expectation is not well-suited to model human behaviour under risk. This was observed by Daniel Bernoulli already in 1739. A short description of corresponding results is given below in Subsect. 1.2.

So, valuations other than expectation have been sought (cf. [1,10,22,35,37]). *Concave* valuations, such as *variance* and *standard deviation,* are well-suited to model risk-averse minimizing players. Already in 1906, Fisher [14] proposed attaching standard derivation to expectation as an additive measure of risk. In his seminal paper [22], Markowitz introduced the *Mean-Variance* approach to portfolio maximization, advocating the minimization of variance constrained on some lower bound on the expected return. Popular valuations are *(i)* $\mathsf{E} - \gamma \cdot \mathsf{Var}$, where E and Var stand for *expectation* and *variance,* respectively, and $\gamma > 0$ describes the risk tolerance (see [10]), and *(ii)* the *Sharpe Ratio* $\mathsf{SR} = \mathsf{E}/\mathsf{SD}$ [35], where SD stands for *standard deviation.* The *Mean-Variance* paradigm [22] created Modern Portfolio Theory [10] as a new field and initiated a tremendous amount of research — see the surveys [20,36] for an overview. However, in the *Mean-Variance* paradigm [22], only expectation and variance were used for evaluating the return; this choice is justified only if the return is normally distributed [17].

Therefore this approach is not-well suited to model the losses that might be incurred in finance or in the insurance industry. Real financial data are characterized by fat tails. For these applications Conditional Value at Risk became the most prominent measure during the last years. CVaR is recognized as a suitable model of risk in volatile economic circumstance [20,31,32,34]; it is widely used for modeling and optimizing hedge and credit risks. There were recently two quite notable developments in the insurance domain and the financial domain, namely the issue of the *Solvency II Directive* in the insurance domain and of *Basel III Rules* in the financial domain, which enforce a tremendous increase to the role and applicability of CVaR.

A significant advantage of expectation is that it guarantees the existence of a *Nash equilibrium* [26,27], where each player is playing a *best-response* mixed strategy and could not unilaterally reduce her expected cost. Existence of equilibria (for minimization games) extends to *convex* valuations [7,11], but may fail for non-convex and even for *concave* ones. *Crawford's game* [6, Sect. 4] was the *first* counterexample game with no equilibrium for a certain valuation; for more counterexamples, see [9,23].

1.2 Human Behaviour Under Risk

In Experimental Economics, experimental methods are applied to study economic questions, e.g., in order to test the validity of economic theories.

The paper of Kahnemann and Tversky [19] presents a critique of expected utility theory as a descriptive model of decision-making under risk.

- Possibility of winning, certainty effect
  - A: 80% chance of winning \$4,000
    B: \$3,000 with certainty
    20% choose A, 80% choose B
  - A: 50% chance of winning a three-week tour of England, France, and Italy
    B: One-week tour of England with certainty
    22% choose A, 78% choose B
- Possibility of losing
  - A: 80% chance of losing \$4,000
    B: Losing \$3,000 with certainty
    92% choose A, 8% choose B

Already in 1739, in his paper "Exposition of a New Theory on the Measure of Risk" Daniel Bernoulli [3] studies an example where expectation is not suited to model human behaviour.

A person owns a lottery ticket:

- It yields with equal probability either nothing or 20,000 ducats.

A poor person may want to sell this ticket for 9,000 ducats. While a rich man may want to buy it for this price.

1.3 Framework

We embarked on a research direction making different-from-classical assumptions. We model the valuation of each player as the sum $\mathsf{V} = \mathsf{E} + \mathsf{R}$, where E and R are the *expectation* and *risk valuation,* respectively. R has the *Risk-Positivity* property: the value of R is non-negative – it is 0, yielding no risk, exactly when the cost incurred to a player is constant over all choices of strategies by the other players. We focus on the associated decision problem, denoted as $\exists\mathsf{V}$-EQUILIBRIUM, asking, given a game G, whether G has a V-*equilibrium,* where no player could unilaterally reduce her cost (as evaluated by V).

We shall focus on strictly quasiconcave valuations. A key property of a strictly quasiconcave valuation, called *Optimal-Value*, we prove and exploit is that it maintains the same optimal value over all convex combinations of strategies supported in a given best-response mixed strategy (Theorem 1).

The *Weak-Equilibrium-for-Expectation* property [23, Sect. 2.6] requires that all strategies supported in a player's best-response mixed strategy induce the same conditional expectation, taken over all random choices of the other players, for her cost. Thus the player is holding a *unique* expectation for her cost no matter which of her supported strategies she ends up choosing. This property holds vacuously for Nash equilibria [26, 27] and it holds for several of the valuation functions we consider, but not for all. It is an open problem to classify the subclass of strictly quasiconcave valuations with this property.

1.4 Contribution and Related Work

Fiat and Papadimitriou [13] were the first in Computer Science to consider algorithmic problems about equilibria for games where players are not expectation-optimizers. Specifically, they introduced the equilibrium computation problem in games where risk-averse players use valuations more general than expectation, and addressed the complexity of deciding the existence of such equilibria. Subsequently, Mavronicolas and Monien [23] focused on the concave valuation *expectation plus variance,* and established structural and complexity results [23, Sect. 3].

Mavronicolas and Monien [24,25] developed a framework for games with players minimizing an $(\mathsf{E}+\mathsf{R})$-valuation V. Our framework enabled proving the strong $\mathcal{NP}$-hardness of $\exists$V-EQUILIBRIUM in the simple case of games with two players, and for many significant choices of $(\mathsf{E}+\mathsf{R})$-valuations V. These were the *first* complexity results for deciding the existence of equilibria in the context of risk-modeling valuations.

The results from these 3 papers are presented in this article in a unified approach. The results show that the question whether a 2-player game with such risk-modeling valuations V has a V-equilibrium is strongly $\mathcal{NP}$-hard under very mild assumptions. We only have to ask that the valuation functions are strictly quasiconcave or that they fulfill the Weak-Equilibrium-for-Expectation property and that additionally some 2-strategy game has no V-equilibrium. These conditions are fulfilled for the functions from Markowitz's Mean-Variance approach like $\mathsf{E}+\mathsf{Var}$, $\mathsf{E}+\mathsf{SD}$ and the Sharpe Ration, and also for Conditional Value-at-Risk, recently very popular for modeling volatile economic circumstances.

Complexity results for the related optimization problems do also exist: Maximizing the expectation of portfolio-return of an investment for a given threshold on variance and for a given threshold on Var_α, respectively, were shown to be $\mathcal{NP}$-hard in [21] and in [2], respectively.

A different aspect is studied by Brautbar et al. [5]. They introduce a class of function valuations coming from Markowitz's Mean-Variance paradigm [22] and show that for valuations from this class non-pure equilibria are very rare.

In [5] it is also shown, following a proof idea described already by Crawford [6], that for concave valuations a correlated equilibrium always exists.

Finally we want to mention that [24] also contains the result that the problem $\exists$V-EQUILIBRIUM is strongly $\mathcal{NP}$-hard also for 2-strategy games provided that $\mathsf{V}=\mathsf{E}+\mathsf{Var}$ or $\mathsf{V}=\mathsf{E}+\mathsf{SD}$.

2 (Strict) Convexity and (Strict) Concavity

Definition 1. *A function* $\mathsf{f}\colon \mathsf{D}\to\mathbb{R}$ *on a convex set* $\mathsf{D}\subset\mathbb{R}^n$ *is* convex *(resp.,* strictly convex*) if for every pair of points* $x, y\in\mathsf{T}$*, for all* $\lambda\in[0,1]$ *(resp.,* $\lambda\in(0,1)$*),* $\mathsf{f}(\lambda\,x+(1-\lambda)\,y)\leq\lambda\,\mathsf{f}(x)+(1-\lambda)\,\mathsf{f}(y)$ *(resp.,* $\mathsf{f}(\lambda\,x+(1-\lambda)\,y)<\lambda\,\mathsf{f}(x)+(1-\lambda)\,\mathsf{f}(y)$*). A function* $\mathsf{f}\colon\mathsf{D}\to\mathbb{R}$ *on a convex set* D *is* concave *(resp.,* strictly concave*) if* $-\mathsf{f}$ *is convex (resp., strictly convex).*

We give a characterization of convexity (resp., concavity) for differentiable functions.

Lemma 1. *Assume that the function* $\mathsf{f}\colon \mathsf{D} \rightarrow \mathbb{R}$ *on the convex set* $\mathsf{D} \subset \mathbb{R}^n$ *is differentiable. Then,* f *is convex (resp., concave) if and only if for all pairs* $x, y \in \mathsf{D}$*,* $\mathsf{f}(y) \geq \mathsf{f}(x) + \nabla \mathsf{f}(x)^{\mathrm{T}}(y-x)$ *(resp.,* $\mathsf{f}(y) \leq \mathsf{f}(x) + \nabla \mathsf{f}(x)^{\mathrm{T}}(y-x)$*).*

2.1 Quasiconvexity and Quasiconcavity

Quasiconvexity (resp., quasiconcavity) is a generalization of convexity (resp., concavity).

Definition 2. *A function* $\mathsf{f}\colon \mathsf{D} \rightarrow \mathbb{R}$ *on a convex set* $\mathsf{D} \subset \mathbb{R}^n$ *is* quasiconvex *(resp.,* quasiconcave*) if for every pair of points* $x, y \in \mathsf{D}$*, for all* $\lambda \in (0,1)$*,* $\mathsf{f}(\lambda x + (1-\lambda)y) \leq \max\{\mathsf{f}(x), \mathsf{f}(y)\}$ *(resp.,* $\mathsf{f}(\lambda x + (1-\lambda)y) \geq \min\{\mathsf{f}(x), \mathsf{f}(y)\}$*).*

An introduction into quasiconvexity/quasiconcavity and many examples can be found in [4].

As a nice historical survey in [16] shows, the notion of quasiconcavity emerged in Economics and Mathematics from the works of three authors, namely von Neumann [28], DeFinetti [8] and Fenchel [12], and started the research field of *generalized convexity*. Quasiconcave functions make a very rich class with many applications in Mathematical Economics; for example, in Microeconomics, the convexity of customers' preference relations is a very desirable feature, implying the quasiconcavity of their utilities [33].

2.2 Strict Quasiconvexity and Strict Quasiconcavity

Definition 3. *A quasiconvex (resp., quasiconcave) function* $\mathsf{f}\colon \mathsf{D} \rightarrow \mathbb{R}$ *on a convex set* $\mathsf{D} \subset \mathbb{R}^n$ *is* strictly quasiconvex *(resp.,* strictly quasiconcave*)* [30] *if for every pair of points* $x, y \in \mathsf{D}$*,* $\mathsf{f}(x) \neq \mathsf{f}(y)$ *implies that for all* $\lambda \in (0,1)$*,* $\mathsf{f}(\lambda x + (1-\lambda)y) < \max\{\mathsf{f}(x), \mathsf{f}(y)\}$ *(resp.,* $\mathsf{f}(\lambda x + (1-\lambda)y) > \min\{\mathsf{f}(x), \mathsf{f}(y)\}$*).*

A strictly quasiconcave valuation incurs to a player mixing two particular mixed strategies a cost larger than the minimum of the individual costs incurred by the two mixed strategies when the two costs are not equal. A significant property of strictly quasiconcave valuations, making them interesting to consider in the context of equilibrium computation, is that they do not exclude the existence of mixed equilibria, whereas strictly concave valuations do.

The notion of strictness for strictly quasiconcave functions is weaker than for strictly concave functions: for the former, strictness allows that $\mathsf{f}(\lambda x+(1-\lambda)y) = \min\{\mathsf{f}(x), \mathsf{f}(y)\}$, with $\lambda \in (0,1)$, in the case $\mathsf{f}(x) = \mathsf{f}(y)$, whereas the equality is excluded for the latter. A notable property of a strictly quasiconcave function is a property of concave functions: a local maximum is also a global maximum [30, Theorem 2].

We prove:

Proposition 1. *Consider a pair of a convex function* $\mathsf{g}\colon \mathsf{D} \rightarrow \mathbb{R}$ *and a concave function* $\mathsf{h}\colon \mathsf{D} \rightarrow \mathbb{R}$*. Then, the function* $\mathsf{f} := \mathsf{g}/\mathsf{h}$ *is strictly quasiconvex.*

3 Definitions and Background

3.1 Games

For an integer $n \geq 2$, an *n-players game* G, or *game,* consists of *(i)* n finite sets $\{S_i\}_{i \in [n]}$ of *strategies,* and *(ii)* n *cost functions* $\{\mu_i\}_{i \in [n]}$, each mapping $\mathcal{S} = \times_{i \in [n]} S_i$ to the reals. So, $\mu_i(\mathbf{s})$ is the *cost* of player $i \in [n]$ on the *profile* $\mathbf{s} = \langle s_1, \ldots, s_n \rangle$ of strategies, one per player.

A *mixed strategy* for player $i \in [n]$ is a probability distribution p_i on S_i; the *support* of player i in p_i is the set $\sigma(p_i) = \{\ell \in S_i \mid p_i(\ell) > 0\}$. Denote as $\Delta_i = \Delta(S_i)$ the set of mixed strategies for player i. Player i is *pure* if for each strategy $s_i \in S_i$, $p_i(s_i) \in \{0, 1\}$; else she is *non-pure.* Denote as p_i^ℓ the *pure strategy* of player i choosing the strategy ℓ with probability 1.

A *mixed profile* is a tuple $\mathbf{p} = \langle p_1, \ldots, p_n \rangle$ of n mixed strategies, one per player; denote as $\Delta = \Delta(S) = \times_{i \in [n]} \Delta_i$ the set of mixed profiles. The mixed profile $\mathbf{p}$ induces probabilities $\mathbf{p}(\mathbf{s})$ for each profile $\mathbf{s} \in \mathcal{S}$ with $\mathbf{p}(\mathbf{s}) = \prod_{i' \in [n]} p_{i'}(s_{i'})$. For a player $i \in [n]$, the *partial profile* $\mathbf{s}_{-i}$ (resp., *partial mixed profile* $\mathbf{p}_{-i}$) results by eliminating the strategy s_i (resp., the mixed strategy p_i) from $\mathbf{s}$ (resp., $\mathbf{p}$). The partial mixed profile $\mathbf{p}_{-i}$ induces probabilities $\mathbf{p}(\mathbf{s}_{-i})$ for each partial profile $\mathbf{s}_{-i} \in \mathcal{S}_{-i} := \times_{i' \in [n] \setminus \{i\}} S_{i'}$ with $\mathbf{p}(\mathbf{s}_{-i}) = \prod_{i' \in [n] \setminus \{i\}} p_{i'}(s_{i'})$.

3.2 $(\mathsf{E} + \mathsf{R})$-Valuations

For a player $i \in [n]$, a *valuation function* V_i or *valuation* for short, is a real-valued function on $\Delta(\mathcal{S})$, yielding a value $\mathsf{V}_i(\mathbf{p})$ to each mixed profile $\mathbf{p}$, so that in the special case where $\mathbf{p}$ is a profile $\mathbf{s}$, $\mathsf{V}_i(\mathbf{s}) = \mu_i(\mathbf{s})$. A *valuation* $\mathsf{V} = \langle \mathsf{V}_1, \ldots, \mathsf{V}_n \rangle$ is a tuple of valuations, one per player; G^{V} denotes G together with V.

An $(\mathsf{E}+\mathsf{R})$-*valuation* [24, Definition 2.1] is a valuation of the form $\mathsf{V} = \mathsf{E}+\mathsf{R}$, where E is the *expectation valuation* with $\mathsf{E}_i(\mathbf{p}) = \sum_{\mathbf{s} \in S} \mathbf{p}(\mathbf{s})\, \mu_i(\mathbf{s})$ for $i \in [n]$, and R is the *risk valuation,* a continuous valuation with the *Risk-Positivity* property: For each player $i \in [n]$ and mixed profile $\mathbf{p}$, (C.1) $\mathsf{R}_i(\mathbf{p}) \geq 0$ and (C.2) $\mathsf{R}_i(\mathbf{p}) = 0$ if and only if for each profile $\mathbf{s} \in \mathcal{S}$ with $\mathbf{p}(\mathbf{s}) > 0$, $\mu_i(\mathbf{s})$ remains constant over all choices of strategies by the other players. In such a case, $\mathsf{V}_i(\mathbf{p}) = \mathsf{E}_i(\mathbf{p}) = \mu_i(\mathbf{s})$ for any profile $\mathbf{s} \in \mathcal{S}$ with $\mathbf{p}(\mathbf{s}) > 0$.

We shall deal with cases where for a player $i \in [n]$ and a mixed profile $\mathbf{p}$, $\{\mu_i(\mathbf{s}) \mid \mathbf{p}(\mathbf{s}) > 0\} = \{a, b\}$ with $a < b$, so that $\mathsf{R}_i(\mathbf{p})$ depends on the three parameters a, b and q, where $q := \sum_{\mathbf{s} \in \mathcal{S} \mid \mu_i(\mathbf{s}) = b} \mathbf{p}(\mathbf{s})$. Then, denote

$$\widehat{\mathsf{V}}_i(a, b, q) := a + q(b - a) + \widehat{\mathsf{R}}_i(a, b, q).$$

3.3 V-Equilibrium

Fix a player $i \in [n]$. The pure strategy p_i^ℓ is a V_i-*best pure response* to a partial mixed profile $\mathbf{p}_{-i}$ if

$$\mathsf{V}_i\left(p_i^\ell, \mathbf{p}_{-i}\right) = \min\left\{\mathsf{V}_i\left(p_i^{\ell'}, \mathbf{p}_{-i}\right) \mid \ell' \in S_i\right\}.$$

So, the pure strategy p_i^ℓ minimizes the valuation $\mathsf{V}_i\left(., \mathbf{p}_{-i}\right)$ of player i over her pure strategies. The mixed strategy p_i is a V_i-*best response* to $\mathbf{p}_{-i}$ if

$$\mathsf{V}_i\left(p_i, \mathbf{p}_{-i}\right) = \min\left\{\mathsf{V}_i\left(p_i', \mathbf{p}_{-i}\right) \mid p_i' \in \Delta(S_i)\right\}.$$

So, the mixed strategy p_i minimizes the valuation $\mathsf{V}_i\left(., \mathbf{p}_{-i}\right)$ of player i over her mixed strategies. The mixed profile $\mathbf{p}$ is a V-*equilibrium* if for each player i, the mixed strategy p_i is a V_i-best response to $\mathbf{p}_{-i}$. So, no player could unilaterally deviate to another mixed strategy to reduce her cost. Denote as $\exists$V-EQUILIBRIUM the algorithmic problem of deciding, given a game G, the existence of a V-equilibrium for G^{V}.

3.4 The Weak-Equilibrium-for-Expectation Property

The mixed profile $\mathbf{p}$ has the *Weak Equilibrium* property [23, Sect. 2.6] for player $i \in [n]$ in the game G^{V} if for each pair of strategies $\ell, \ell' \in \sigma(p_i)$, $\mathsf{V}_i\left(p_i^\ell, \mathbf{p}_{-i}\right) = \mathsf{V}_i\left(p_i^{\ell'}, \mathbf{p}_{-i}\right)$. The mixed profile $\mathbf{p}$ has the *Weak Equilibrium* property [23, Sect. 2.6] in G^{V} if it has the *Weak Equilibrium* property for each player $i \in [n]$ in G^{V}. The valuation V has the *Weak-Equilibrium-for-Expectation* property if the following condition holds for every game G. For each player $i \in [n]$, if p_i is a V_i-best-response to $\mathbf{p}_{-i}$, then $\mathbf{p}$ has the *Weak Equilibrium* property for player i in the game G^{E}: for each pair of strategies $\ell, \ell' \in \sigma(p_i)$, $\mathsf{E}_i\left(p_i^\ell, \mathbf{p}_{-i}\right) = \mathsf{E}_i\left(p_i^{\ell'}, \mathbf{p}_{-i}\right)$.

3.5 The Optimal-Value Property

We show in [25, Theorem 9]:

Theorem 1 (The Optimal-Value Property). *Fix a game G, a player $i \in [n]$ and a partial mixed profile $\mathbf{p}_{-i}$. Assume that (A.1) the valuation $\mathsf{V}_i\left(p_i, \mathbf{p}_{-i}\right)$ is strictly quasiconcave in p_i, and (A.2) $\widehat{p}_i$ is a V_i-best response to $\mathbf{p}_{-i}$. Then, for any mixed strategy q_i with $\sigma\left(q_i\right) \subseteq \sigma\left(\widehat{p}_i\right)$, $\mathsf{V}_i\left(q_i, \mathbf{p}_{-i}\right) = \mathsf{V}_i\left(\widehat{p}_i, \mathbf{p}_{-i}\right)$.*

4 Valuations

4.1 Definition

We introduce E-strict concavity from [24, Definition 2.2]:

Definition 4 (E-Strict Convexity, E-Strict Concavity). *Fix a player i. The $(\mathsf{E} + \mathsf{R})$-valuation V_i is E-strictly* convex *(resp., E-strictly* concave*) if for every game G, the following conditions hold for a fixed partial mixed profile $\mathbf{p}_{-i}$:*

(1) V_i is convex (resp., concave) in the mixed strategy p_i.

(2) For a pair of mixed strategies $p_i', p_i'' \in \Delta(S_i)$, if $\mathsf{E}_i\left(p_i', \mathbf{p}_{-i}\right) \neq \mathsf{E}_i\left(p_i'', \mathbf{p}_{-i}\right)$, then for any $\lambda \in (0, 1)$,

$$\mathsf{V}_i\left(\lambda p_i' + (1-\lambda)p_i'', \mathbf{p}_{-i}\right) < \max\{\mathsf{V}_i\left(p_i', \mathbf{p}_{-i}\right), \mathsf{V}_i\left(p_i'', \mathbf{p}_{-i}\right)\}$$
$$(\textit{resp.}, \mathsf{V}_i\left(\lambda p_i' + (1-\lambda)p_i'', \mathbf{p}_{-i}\right) > \min\{\mathsf{V}_i\left(p_i', \mathbf{p}_{-i}\right), \mathsf{V}_i\left(p_i'', \mathbf{p}_{-i}\right)\}\ ).$$

4.2 The Weak-Equilibrium-for-Expectation Property

We show in [24, Proposition 3.2]:

Theorem 2. *Take a player $i \in [n]$ where V_i is* E-strictly concave*. Then,* V *has the* Weak-Equilibrium-for-Expectation *property for player i.*

4.3 Valuation Functions

We will consider the following valuation functions, where $\mathsf{Var}, \mathsf{SD}$ and ESR denote variance, standard derivation and the Extended Sharpe Ratio, respectively. CVar denotes Conditional Value at Risk and will be defined and studied in Sect. 5. ESR is an adjustment of the Sharpe Ratio to minimization games.

(1) $\mathsf{V} = \mathsf{E} + \gamma \cdot \mathsf{Var}, \gamma > 0$
(2) $\mathsf{V} = \mathsf{E} + \gamma \cdot \mathsf{SD}, \gamma > 0$
(3) $\mathsf{V}^{\nu} = \nu^{-1}(\sum_{s \in S} p(s) \cdot \nu(\mu_i(s))$
(4) $\mathsf{ESR} = \frac{M \cdot \mathsf{E}}{M - \mathsf{Var}}$, M constant
(5) CVar

Here $\mathsf{E}(p) = \sum_{s \in S} p(s)\mu_i(s), \mathsf{Var}(p) = \sum_{s \in S} p(s)(\mu_i(s) - \mathsf{E}_i(p))^2$, and $\mathsf{SD}(p) = \sqrt{\mathsf{Var}(p)}$. The function ν is increasing and a strictly convex function.

The valuation V_i^{ν} is popular in Acturial Risk Theory (see [18]). If $\nu(x) = x^2$, then $V^{\nu} = \sqrt{\sum_{s \in S} p(s)\mu_i^2(s)}$.

Proposition 2. *The valuation functions* $\mathsf{Var}, \mathsf{SD}, \mathsf{ESR} - E$ *and* $\mathsf{V}^{\nu} - \mathsf{E}$ *fulfill the Risk Positivity Property.*

Proposition 3. *The valuation functions* $\mathsf{E} + \gamma \cdot \mathsf{Var}, \mathsf{E} + \gamma \cdot \mathsf{SD}, \mathsf{V}^{\nu}$ *are concave. The valuation function* ESR *is strictly quasiconcave.*

Theorem 3 (Mavronicolas and Monien [24, Lemma 2.3, Corollary 3.3]**).**

(i) The valuation functions $\mathsf{E} + \gamma \cdot \mathsf{Var}$ *and* $\mathsf{E} + \gamma \cdot \mathsf{SD}$ *are* E*-strictly concave.*
(ii) The valuation functions $\mathsf{E} + \gamma \cdot \mathsf{Var}, \mathsf{E} + \gamma \cdot \mathsf{SD}$ *and* ESR *fulfill the Weak-Equilibrium-for-Expectation property.*

Consider a concave valuation V^{ν}, for an increasing and strictly convex function ν. Since ν^{-1} is also increasing, a mixed profile p is a V^{ν}-equilibrium for a game G if and only if p is an E-equilibrium for the game G^{ν} constructed from G by setting for each player $i \in [n]$ and profile $\mathsf{s} \in S$, $\mu_i^{\nu}(\mathsf{s}) := \nu(\mu_i(\mathsf{s}))$. Since every game has an E-equilibrium, this implies that there is a V^{ν}-equilibrium for G, and the associated search problem for a V^{ν}-equilibrium is total; it is in $\mathcal{PPAD}$ [29] for 2-player games.

The situation is different for the other valuation functions.

For $0 < \delta < 1$, the *Crawford game* $\mathsf{G}_C(\delta)$ is the 2-player game with bimatrix

$$\begin{pmatrix} \langle 1+\delta, 1+\delta \rangle & \langle 1, 1+2\delta \rangle \\ \langle 1, 1+2\delta \rangle & \langle 1+2\delta, 1 \rangle \end{pmatrix}$$

from [24]; it is a generalization of a game from [6, Sect. 4]. $\mathsf{G}_C(\delta)$ has no pure equilibrium [24, Lemma 5.11]. We use the Weak-Equilibrium-for-Expectation property to prove:

Theorem 4 (Mavronicolas and Monien [24, Lemma 5.11] and [25, Theorem 12]**).** *For δ with $0 < \delta < 1$ and $\mathsf{V} = \mathsf{E} + \gamma \cdot \mathsf{Var}$ or $\mathsf{V} = \mathsf{E} + \gamma \cdot \mathsf{SD}$ or $\mathsf{V} = \mathsf{ESR}$ the game $\mathsf{G}_C(\delta)$ has no V-equilibrium.*

4.4 Mean-Variance Preference Functions

Brautbar et al. [5, Sect. 3.1] study a class of valuations, coming from the Mean-Variance paradigm of Markowitz [22] and termed as Mean-Variance Preference Functions and show that for valuations from this class non-pure equilibria are very rare. We rephrase their definition [5, Definition 1] to fit into the adopted setting of minimization games:

Definition 5 (see Brautbar et al. [5]**).** *Fix a player $i \in [n]$. A Mean-Variance Preference Function is a valuation $\mathsf{V}_i(p_i, \mathbf{p}_{-i}) := \mathsf{G}_i(\mathsf{E}_i(p_i, \mathbf{p}_{-i}), \mathsf{Var}_i(p_i, \mathbf{p}_{-i}))$ which satisfies:*

(1) $\mathsf{V}_i(p_i, \mathbf{p_{-i}})$ *is concave in* p_i.
(2) G_i *is non-decreasing in its first argument* $(\mathsf{E}_i(p_i, \mathbf{p_{-i}}))$.
(3) Fix a partial mixed profile $\mathbf{p_{-i}}$ *and a nonempty convex subset* $\Delta \subseteq \Delta_i = \Delta(\mathsf{S}_i)$ *such that* $\mathsf{V}_i(p_i, \mathbf{p_{-i}})$ *is constant on* Δ. *Then, both* $\mathsf{E}_i(p_i, \mathbf{p_{-i}})$ *and* $\mathsf{Var}_i(p_i, \mathbf{p_{-i}})$ *are constant on* Δ.

So, a Mean-Variance Preference Function simulataneously generalizes and restricts the $(\mathsf{E} + \mathsf{R})$-valuations; it generalizes "+" to G but restricts R to Var. Note that condition (3) in the above definition may be seen as a generalization of the Weak-Equilibrium-for-Expectation property conditioned on the assumption that $\mathsf{V}_i(p_i, \mathbf{p_{-i}})$ is constant on a nonempty convex set $\Delta \subseteq \Delta_i$. It is proved in [5, Claim 1] that $\mathsf{E} + \mathsf{Var}$ is a Mean-Variance Preference Function. Since $\mathsf{E} + \mathsf{Var}$ is E-strictly concave (Theorem 3(i)) and has the Optimal-Value property (Theorem 1), the established condition (3) for $\mathsf{E} + \mathsf{Var}$ is a special case of our general result that every E-strictly concave valuation has the Weak-Equilibrium-for-Expectation property (Theorem 2).

Theorem 5 (Boyd and Vandenberghe [4, Theorem 1]**).** *Consider a two-player game with Mean-Variance Preference Functions V_1 and V_2 where, for each player $i \in \{1, 2\}$ and strategy profile $s \in S$, the cost $\mu_i(s) \in \mathbb{R}$ is drawn i.i.d. from an absolutely continuous distribution (with respect to Lebesque). Then with probability 1 the game does not have a non-pure V-equilibrium.*

5 Conditional Value-at-Risk (CVaR_α)

Our presentation draws from [31,32].

Fix throughout a confidence level $\alpha \in [0,1)$. Some of the tools to study the properties of CVaR_α come from *Value-at-Risk,* denoted as VaR_α. They are both examples of a *valuation*, a function from probability distributions to reals. We consider a random variable P, representing a distribution of loss; the cumulative distribution function of P may be continuous or discontinuous. A *discrete* random variable P has a discrete distribution function, making a special case of a discontinuous cumulative distribution function: it takes on values $0 \leq a_1 < a_2 < \ldots < a_\ell$ with probabilities $p_j := \mathbb{P}(\mathsf{P} = a_j)$, $1 \leq j \leq \ell$; the values $a_1, a_2, \ldots, a_\ell$ are called *scenaria* in the risk literature. In this case, we shall identify P with the probability vector $\mathbf{p}$. Note that in defining $\mathsf{VaR}_\alpha(\mathbf{p})$ and $\mathsf{CVaR}_\alpha(\mathbf{p})$, the values $a_1, \ldots, a_\ell$ have to be fixed. Denote the cumulative distribution function of $\mathbf{p}$ as $\Pi(\mathbf{p}, z) = \sum_{k \in [\ell] \mid a_k \leq z} p_k$.

5.1 Value-at-Risk (VaR_α)

The *Value-at-Risk* of P at *confidence level* $\alpha \in [0,1)$, denoted as $\mathsf{VaR}_\alpha(\mathsf{P})$, is defined as $\mathsf{VaR}_\alpha(\mathsf{P}) = \inf\{\zeta \mid \mathbb{P}[\mathsf{P} \leq \zeta] \geq \alpha\}$; this definition applies to both continuous and discontinuous distributions. So, $\mathbb{P}[\mathsf{P} \leq \mathsf{VaR}_\alpha(\mathsf{P})] \geq \alpha$, with equality holding when the cumulative distribution function is continuous at $\mathsf{VaR}_\alpha(\mathsf{P})$ — put in other words, when there is no *probability atom* at $\mathsf{VaR}_\alpha(\mathsf{P})$. When P is a discrete random variable, $\mathsf{VaR}_\alpha(\mathbf{p}) = \min\left\{a_k \mid \sum_{j=1}^{k} p_j \geq \alpha\right\}$. Note that VaR_α is a discontinuous function of α. In this case, denote as $\kappa_\alpha = \kappa_\alpha(\mathbf{p})$ the unique index such that $\sum_{k \in [\kappa_\alpha]} p_k \geq \alpha > \sum_{k \in [\kappa_\alpha - 1]} p_k$. So, $a_{\kappa_\alpha} = \mathsf{VaR}_\alpha(\mathbf{p})$. In the full version of the paper, we give an example establishing that VaR_α is not an $(\mathsf{E} + \mathsf{R})$-valuation. We shall later use an interesting property of VaR_α (cf. [4, Exercise 3.24 (f)]), for which we provide a new proof.

Lemma 2. *With $\alpha \in (0,1)$, $\mathsf{VaR}_\alpha(\mathbf{p})$ is quasiconcave and quasiconvex in the probabilities, but neither strictly quasiconcave nor strictly quasiconvex.*

5.2 Conditional Value-at-Risk (CVaR_α)

CVaR_α accounts for losses no smaller than VaR_α. For random variables with a continuous cumulative distribution function, the *Conditional Value-at-Risk* of P, denoted as $\mathsf{CVaR}_\alpha(\mathsf{P})$, is the conditional expectation of P subject to $\mathsf{P} \geq \mathsf{VaR}_\alpha(\mathsf{P})$ [34, Definition 2]. So, $\mathsf{CVaR}_\alpha(\mathsf{P}) = \mathsf{E}(\mathsf{P} \mid \mathsf{P} \geq \mathsf{VaR}_\alpha(\mathsf{P}))$. Note that $\alpha = 0$ is a degenerate case with $\mathsf{CVaR}_0(\mathsf{P}) = \mathsf{E}(\mathsf{P})$. The general definition of CVaR_α for random variables with a possibly discontinuous cumulative distribution function is obtained as follows (cf. [32, Proposition 6]). Denote as $\mathsf{CVaR}_\alpha^+(\mathsf{P})$ the conditional expectation of P subject to $\mathsf{P} > \mathsf{VaR}_\alpha(\mathsf{P})$; $\mathsf{CVaR}_\alpha^+(\mathsf{P})$ is also called the *Conditional Tail Expectation* or *Upper* CVaR [34]. Set $\lambda_\alpha(\mathsf{P}) := \dfrac{\mathbb{P}[\mathsf{P} \leq \mathsf{VaR}_\alpha(\mathsf{P})] - \alpha}{1 - \alpha}$. Note that $\lambda_\alpha(\mathsf{P})$ provides for the possibility that there is a probability atom (that is, the cumulative distribution function is discontinuous) at $\mathsf{VaR}_\alpha(\mathsf{P})$. The numerator of $\lambda_\alpha(\mathsf{P})$ is the "excess" probability mass at $\mathsf{VaR}_\alpha(\mathsf{P})$, i.e., the probability mass beyond α. The denominator $1 - \alpha$ is the support size of $\mathbb{P}_\alpha$, the *α-tail distribution of* P, denoted as $\mathbb{P}_\alpha$ and defined as follows: $\mathbb{P}_\alpha[\mathsf{P} \leq \zeta] = 0$ if $\zeta < \mathsf{VaR}_\alpha(\mathsf{P})$, and $\mathbb{P}_\alpha[\mathsf{P} \leq \zeta] = \dfrac{\mathbb{P}[\mathsf{P} \leq \zeta] - \alpha}{1-\alpha}$ if $\zeta \geq \mathsf{VaR}_\alpha(\mathsf{P})$. Thus, $\lambda_\alpha(\mathsf{P}) = \mathbb{P}_\alpha[\mathsf{P} \leq \mathsf{VaR}_\alpha(\mathsf{P})]$. Hence, we obtain the *weighted average formula* for $\mathsf{CVaR}_\alpha(\mathsf{P})$ [32, Proposition 6]:

$$\mathsf{CVaR}_\alpha(\mathsf{P}) := \lambda_\alpha(\mathsf{P}) \cdot \mathsf{VaR}_\alpha(\mathsf{P}) + (1 - \lambda_\alpha(\mathsf{P})) \cdot \mathsf{CVaR}^+_\alpha(\mathsf{P}).$$

So, CVaR_α is the weighted average of VaR_α and the expectation of losses strictly exceeding VaR_α. Note that $\mathsf{CVaR}_\alpha(\mathsf{P}) = \mathsf{VaR}_\alpha(\mathsf{P})$ when $\lambda_\alpha(\mathsf{P}) = 1$; note also that for continuous cumulative distributions, $\lambda_\alpha(\mathsf{P}) = 0$ and $\mathsf{CVaR}_\alpha(\mathsf{P}) = \mathsf{CVaR}^+_\alpha(\mathsf{P})$, the expected loss strictly exceeding VaR_α. For a discrete random variable P, identified with the probability vector $\mathbf{p}$, the weighted average formula reduces to the *discrete weighted average formula* (cf. [32, Proposition 8]):

$$\mathsf{CVaR}_\alpha(\mathbf{p}) = \frac{1}{1-\alpha}\left(\left(\sum_{k \in [\kappa_\alpha]} p_k - \alpha\right) a_{\kappa_\alpha} + \sum_{k \geq \kappa_\alpha + 1} p_k a_k\right).$$

Recall the *minimization function* $\mathsf{F}_\alpha(\mathbf{p}, z) := z + \frac{1}{1-\alpha} \sum_{k \in [n] \mid a_k > z} (a_k - z) p_k$, for a probability vector $\mathbf{p}$ and a number $z \in \mathbb{R}$, from [31, Sect. 2]. Note that F_α is linear in its first argument. The continuity of F_α in z follows from its convexity (in z) [32, Theorem 10]. We provide a direct and simple proof, bypassing convexity, for the discrete setting.

Lemma 3. *The function $\mathsf{F}_\alpha(\mathbf{p}, z)$ is continuous in z.*

We now consider some monotonicity properties of $\mathsf{F}_\alpha(\mathbf{p}, z)$ in z, drawing on the continuity of F_α (Lemma 3); they offer refinements of properties from [32, Proposition 5], which will be needed later. In order to take into account the possibility that some probabilities may be 0, denote, for an index $j \in [\ell]$, $\mathsf{next}(\mathbf{p}, j) := \min\{k > j \mid p_k \neq 0\}$; $\mathsf{next}(\mathbf{p}, j) := \ell + 1$ when $p_k = 0$ for all $j < k \leq \ell$.

Lemma 4. *The following implications hold:*

(1) If $\Pi(\mathbf{p}, \mathsf{VaR}_\alpha(\mathbf{p})) > \alpha$, then:
(1/a) $\mathsf{F}_\alpha(\mathbf{p}, z)$ increases strictly monotone in z for $z \geq \mathsf{VaR}_\alpha(\mathbf{p})$.
(1/b) $\mathsf{F}_\alpha(\mathbf{p}, z)$ decreases strictly monotone in z for $z \leq \mathsf{VaR}_\alpha(\mathbf{p})$.
(2) If $\Pi(\mathbf{p}, \mathsf{VaR}_\alpha(\mathbf{p})) = \alpha$, then:
(2/a) $\mathsf{F}_\alpha(\mathbf{p}, z)$ increases strictly monotone for $z \geq a_{\mathsf{next}(\mathbf{p}, \kappa_\alpha(\mathbf{p}))}$.
(2/b) $\mathsf{F}_\alpha(\mathbf{p}, z)$ is constant for $\mathsf{VaR}_\alpha(\mathbf{p}) \leq z \leq a_{\mathsf{next}(\mathbf{p}, \kappa_\alpha(\mathbf{p}))}$.
(2/c) $\mathsf{F}_\alpha(\mathbf{p}, z)$ decreases strictly monotone for $z \leq \mathsf{VaR}_\alpha(\mathbf{p})$.

By Lemma 3, Lemma 4 immediately implies a result from [32, Theorem 10]:

Corollary 1. $\mathsf{CVaR}_\alpha(\mathbf{p}) = \min_z \mathsf{F}_\alpha(\mathbf{p}, z) = \mathsf{F}_\alpha(\mathbf{p}, \mathsf{VaR}_\alpha(\mathbf{p}))$.

We continue to look at two significant properties:

Proposition 4. *With $\alpha \in (0,1)$, CVaR_α is an $(\mathsf{E}+\mathsf{R})$-valuation.*

Proposition 5. *With $\alpha \in (0,1)$, CVaR_α is concave in the probabilities.*

We continue with a characterization of the cases where CVaR_α is constant over all convex combinations of probability vectors $\mathbf{p}$ and $\mathbf{q}$; interestingly, one of the two cases requires that $\mathsf{VaR}_\alpha(\mathbf{p}) = \mathsf{VaR}_\alpha(\mathbf{q})$. We use the quasiconcavity and quasiconvexity of VaR_α in the probabilities (Lemma 2) to prove:

Theorem 6 (Mavronicolas and Monien [25, Theorem 3]**).** *Fix a confidence level $\alpha \in (0,1)$ and consider probability vectors $\mathbf{p}$ and $\mathbf{q}$ with $\mathsf{VaR}_\alpha(\mathbf{p}) \leq \mathsf{VaR}_\alpha(\mathbf{q})$ and $\mathsf{CVaR}_\alpha(\mathbf{p}) = \mathsf{CVaR}_\alpha(\mathbf{q})$. Then, $\mathsf{CVaR}_\alpha(\lambda \cdot \mathbf{p} + (1-\lambda) \cdot \mathbf{q})$ is constant for $\lambda \in [0,1]$ if and only if:*

(1) $\mathsf{VaR}_\alpha(\mathbf{p}) = \mathsf{VaR}_\alpha(\mathbf{q})$, *or:*
(2) $\alpha = \Pi(\mathbf{p}, \mathsf{VaR}_\alpha(\mathbf{p}))$ *and* $\kappa_\alpha(\mathbf{q}) \leq \mathsf{next}(\mathbf{p}, \kappa_\alpha(\mathbf{p}))$.

Because of Proposition 5 and Theorem 1 the Optimal-Value property holds and is used to prove:

Theorem 7 (Weak-Equilibrium-for-VaR_α Property [25, Theorem 4]**).** *Fix a confidence level $\alpha \in (0,1)$. Take a CVaR_α-best response p_i of player $i \in [n]$ to the partial mixed profile $\mathbf{p}_{-i}$, where $p^1, p^2 \in \sigma(p_i)$ with $\mathsf{VaR}_\alpha(\langle p^1, \mathbf{p}_{-i}\rangle) \leq \mathsf{VaR}_\alpha(\langle p^2, \mathbf{p}_{-i}\rangle)$. Then:*

(1) $\mathsf{VaR}_\alpha(\langle p^1, \mathbf{p}_{-i}\rangle) = \mathsf{VaR}_\alpha(\langle p^2, \mathbf{p}_{-i}\rangle)$, *or:*
(2) $\alpha = \Pi\left(\langle p^1, \mathbf{p}_{-i}\rangle, \mathsf{VaR}_\alpha(\langle p^1, \mathbf{p}_{-i}\rangle)\right)$
and $\kappa_\alpha(\langle p^2, \mathbf{p}_{-i}\rangle) \in \left[\kappa_\alpha\left(\langle p^1, \mathbf{p}_{-i}\rangle\right), \mathsf{next}\left(\mathbf{p}, \kappa_\alpha\left(\langle p^1, \mathbf{p}_{-i}\rangle\right)\right)\right]$.

Using this theorem we show in [25, Theorem 6]:

Theorem 8. *With $\alpha \in \left(\frac{1}{3}, 1\right)$ and $\delta > 0$, $\mathsf{G}_C(\delta)$ has no CVaR_α-equilibrium.*

6 Complexity Results

We state the *Master's Theorem for $\mathcal{NP}$-hardness* from [24,25]:

Theorem 9 (Mavronicolas and Monien [24, Theorem 5.1] **and** [25, Theorem 7]**).** *Fix an $(\mathsf{E}+\mathsf{R})$-valuation V such that:*

(1) V *has the* Weak-Equilibrium-for-Expectation *property or* V *has the Optimal-Value property.*
(2) There is a rational number δ with $0 < \delta \leq \frac{1}{4}$ such that:
  (2/a) $\widehat{\mathsf{R}}(1, 1+\rho, q) < \frac{1}{2}$ *for each probability $q \in [0,1]$ with $0 < \rho \leq 2\delta$.*
  (2/b) $\widehat{\mathsf{V}}(1, 1+\rho, r) < \widehat{\mathsf{V}}(1, 2, q)$ *for all $0 \leq r \leq q \leq 1$ with $0 < \rho \leq 2\delta$.*

(2/c) The Crawford game $\mathsf{G}_C(\delta)$ *with bimatrix*

$$\begin{pmatrix} \langle 1+\delta, 1+\delta\rangle & \langle 1, 1+2\delta\rangle \\ \langle 1, 1+2\delta\rangle & \langle 1+2\delta, 1\rangle \end{pmatrix}$$

has no V*-equilibrium.*

Then, $\exists\mathsf{V}$*-EQUILIBRIUM is strongly* $\mathcal{NP}$*-hard for 2-player games.*

Property $(2/\alpha)$ holds if $\widehat{\mathsf{R}}(1, 1+2\delta), q)$ is continuous in δ and q. And, for the functions $\mathsf{E} + \gamma \cdot \mathsf{Var}, \mathsf{E} \cdot \gamma \cdot \mathsf{SD}, \mathsf{ESR}$ and CVaR_α property $(2/b)$ can be shown easily. Therefore Theorems 4 and 8 imply:

Theorem 10 (Mavronicolas and Monien [24,25]). *Consider the valuation functions* $\mathsf{V} = \mathsf{E}+\gamma\cdot\mathsf{Var}, \mathsf{V} = \mathsf{E}+\gamma\cdot\mathsf{SD}, \mathsf{V} = \mathsf{ESR}$ *or* $\mathsf{V} = \mathsf{CVaR}_\alpha$ *with* $1/3 < \alpha < 1$, *then* $\exists\mathsf{V}$*-Equilibrium is strongly* $\mathcal{NP}$*-hard for 2-player games.*

References

1. Arrow, K.J.: The Theory of Risk-Aversion, reprinted in Aspects of the Theory of Risk Bearing. Y. J. Assatio, Helsinki (1965)
2. Benati, S., Rizzi, R.: A mixed integer linear programming formulation of the optimal mean/value-at-risk portfolio problem. Eur. J. Oper. Res. **176**, 423–434 (2007)
3. Bernoulli, D.: Exposition of a new theory on the measurement of risk, commentaries of the imperial academy of science of Saint Petersburg (in Latin), 1738. Engl. Transl. Econ. **22**(1), 23–36 (1954)
4. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
5. Brautbar, M., Kearns, M., Syed, U.: Private and third-party randomization in risk-sensitive equilibrium concepts. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence, pp. 723–728, July 2010
6. Crawford, V.P.: Equilibrium without independence. J. Econ. Theory **50**(1), 127–154 (1990)
7. Debreu, G.: A social equilibrium existence theorem. Proc. Nat. Acad. Sci. U.S.A. **38**, 886–893 (1952)
8. DeFinetti, B.: Sulle stratificazioni convesse. Annali di Matematica **30**, 173–183 (1949)
9. Denel, E., Safra, Z., Segel, U.: Existence and dynamic consistency of nash equilibrium with non-expected utility preferences. J. Econ. Theory **55**(2), 229–246 (1991)
10. Elton, E.J., Gruber, M.J., Brown, S.J., Goetzmann, W.N.: Modern Portfolio Theory and Investment Analysis. Wiley, Hoboken (2007)
11. Fan, K.: Fixed point and minimax theorems in locally convex topological linear spaces. Proc. Nat. Acad. Sci. **38**(52), 121–126 (1952)
12. Fenchel, W.: Convex Cones, Sets and Functions. Lecture Notes, Department of Mathematics, Princeton University (1953)
13. Fiat, A., Papadimitriou, C.: When the players are not expectation maximizers. In: Kontogiannis, S., Koutsoupias, E., Spirakis, P.G. (eds.) SAGT 2010. LNCS, vol. 6386, pp. 1–14. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16170-4_1

14. Fisher, I.: The Nature of Capital and Income. The Macmillan Company (1906)
15. Friedmann, M., Savage, L.J.: The utility analysis of choices involving risk. J. Polit. Econ. **56**(4), 279–304 (1948)
16. Guerraggio, A., Molho, E.: The origins of quasi-concavity: a development between mathematics and economics. Historia Mathematica **31**, 62–75 (2004)
17. Hanoch, G., Levy, H.: The efficiency analysis of choices involving risk. Rev. Econ. Stud. **36**(3), 335–346 (1969)
18. Kaas, R., Goodvaerts, M., Dhaene, J., Denuit, M.: Modern Actuarial Risk Theory Using R, 2nd edn. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70998-5
19. Kahnemann, D., Tversky, A.: Prospect theory: an analysis of decision under risk. Econometrica **47**(2), 263–291 (1979)
20. Krokhmal, P., Zabarankin, M., Uryasev, S.: Modeling and optimization of risk. Surv. Oper. Res. Manag. Sci. **16**, 49–66 (2011)
21. Mannor, S., Tsitsiklis, J.N.: Mean-variance optimization in Markov decision processes. Eur. J. Oper. Res. **231**(3), 645–653 (2013)
22. Markowitz, H.: Portfolio selection. J. Finance **7**, 77–91 (1952)
23. Mavronicolas, M., Monien, B.: Minimizing expectation plus variance. Theory Comput. Syst. **57**(3), 617–654 (2015)
24. Mavronicolas, M., Monien, B.: The complexity of equilibria for risk-modeling valuations. Theor. Comput. Sci. **634**, 67–96 (2016)
25. Mavronicolas, M., Monien, B.: Conditional value-at-risk: structure and complexity of equilibria. In: Bilò, V., Flammini, M. (eds.) SAGT 2017. LNCS, vol. 10504, pp. 131–143. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66700-3_11
26. Nash, J.F.: Equilibrium points in n-person games. Proc. Nat. Acad. Sci. U.S.A. **36**(1), 48–49 (1950)
27. Nash, J.F.: Non-cooperative games. Ann. Math. **54**(2), 286–295 (1951)
28. von Neumann, J.: Zur Theorie der Gesellschaftsspiele. Math. Ann. **100**, 295–326 (1928)
29. Papadimitriou, C.H.: On the complexity of the parity argument and other inefficient proofs of existence. J. Comput. Syst. Sci. **48**(3), 498–532 (1994)
30. Ponstein, J.: Seven kinds of convexity. SIAM Rev. **9**(1), 115–119 (1967)
31. Rockafellar, R.T., Uryasev, S.: Optimization of conditional value-at-risk. J. Risk **2**, 21–41 (2000)
32. Rockafellar, R.T., Uryasev, S.: Conditional value-at-risk for general loss distributions. J. Bank. Finance **26**, 1443–1471 (2002)
33. Rubinstein, A.: Lecture Notes in Microeconomic Theory. Princeton University Press, Princeton (2006)
34. Sarykalin, S., Serraino, G., Uryasev, S.: Value-at-risk vs. conditional value-at-risk in risk management and optimization, Chapter 13. In: Tutorials in Operations Research (2008)
35. Sharpe, W.F.: A simplified model for portfolio analysis. Manag. Sci. **9**(2), 277–293 (1963)
36. Steinbach, M.C.: Markowitz revisited: mean-variance models in financial portfolios. SIAM Rev. **43**(1), 31–85 (2001)
37. Stoyanov, S.V., Rachev, S.T., Fabozzi, F.: Optimal financial portfolios. Appl. Math. Finance **14**(5), 401–436 (2007)

Firefly-Inspired Algorithm for Job Shop Scheduling

Joss Miller-Todd(✉), Kathleen Steinhöfel(✉), and Patrick Veenstra(✉)

Department of Informatics, King's College London,
30 Aldwych, Bush House, London WC2B 4BG, UK
{joss.miller-todd,kathleen.steinhofel,patrick.veenstra}@kcl.ac.uk

Abstract. Current research strongly suggests that hybrid local search algorithms are more effective than heuristics based on homogenous methods. Accordingly, this paper presents a new hybrid method of Simulated Annealing and Firefly Algorithm [SAFA] for the Job Shop Scheduling Problem (JSSP) with the objective of minimising the makespan. We provide an experimental analysis of its performance based on a set of established benchmarks.

Simulated Annealing [SA] continues to be a viable approach in identifying optimal and near optimal makespans for the JSSP. Similarly, recent deployments of the Firefly Algorithm [FA] have delineated its effectiveness for solving combinatorial optimisation problems efficiently. Therefore, the hybrid algorithm in question aims to combine the acclamatory strengths of SA and FA while overcoming their respective deficiencies.

Keywords: Job shop scheduling · Firefly · Simulated Annealing

1 Introduction

Scheduling deals with the allocation of resources to operations over given time periods, with the aim of optimising one or more objectives [16]. The Job Shop Scheduling Problem (JSSP) is one of many formulations in scheduling, and has attracted many researchers investigating properties of NP-hard problems and methods to tackle them. It is an important optimisation problem and, for the authors in particular, the problem that sparked, based on [20], the collaboration with Juraj Hromkovič and his research group, see [9,10,14].

There are many variants (relaxations and restrictions) of JSSP. In this paper, we consider the standard version defined as follows. There are n jobs in the set $J = \{j_1, j_2, \ldots, j_n\}$ and m machines $M = \{m_1, m_2, \ldots, m_m\}$. Each job $j \in J$ consists of m operations; one operation for each machine. Let $\pi_j(i)$ be the operation for job j performed on machine i. The operations π_j must be performed in a specific order. Let $\mathbf{O} \in [1, m]^{m \times n}$ contain the order of the operations for each of the jobs. The column vector $\mathbf{o}_j$ contains the order of operations for the job j. To be explicit, $o_{ij} \in [1, m]$ is the position of $\pi_j(i)$ in the sequence of operations for j. So if $o_{ij} = 4$, then operation $\pi_j(i)$ must be the 4-th operation

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 423–433, 2018.
https://doi.org/10.1007/978-3-319-98355-4_24

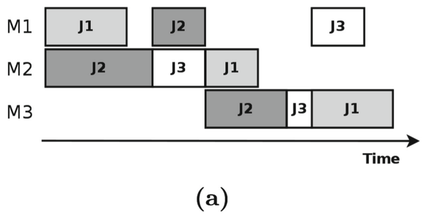


Fig. 1. Two possible schedules for an example instance of the job shop scheduling problem with three jobs on three machines, with times and orders defined in (1). The problem is to find the schedule that leads to the shortest makespan, that is, the shortest possible time needed to complete all operations.

to run for job j. Finally, the elements p_{ij} of the matrix $\mathbf{P} \in \mathbb{R}^{m \times n}$ denote the time taken for operation $\pi_j(i)$, $j \in J$, $i \in M$. Each machine can only process a single operation at a time, and once an operation has started, no preemption is permitted.

A schedule is a matrix $\mathbf{C} \in \mathbb{R}^{m \times n}$ containing the completion times of the operations. Element c_{ij} is the completion time of operation $\pi_j(i)$. Let the makespan L of $\mathbf{C}$ be $\max_{i,j} \mathbf{C}_{ij}$. The job shop scheduling problem is to find a valid schedule such that the makespan L (the total time taken for all jobs to complete) is minimised.

As an example, take a problem instance with three machines and three jobs, with the time matrix $\mathbf{P}$ and order matrix $\mathbf{O}$ as follows:

$$\mathbf{P} = \begin{array}{c} \begin{array}{ccc} J_1 & J_2 & J_3 \end{array} \\ \begin{pmatrix} 3 & 2 & 2 \\ 2 & 4 & 2 \\ 3 & 3 & 1 \end{pmatrix} \end{array} \begin{array}{c} \\ M_1 \\ M_2 \\ M_3 \end{array} \qquad \mathbf{O} = \begin{array}{c} \begin{array}{ccc} J_1 & J_2 & J_3 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 1 \\ 3 & 3 & 2 \end{pmatrix} \end{array} \begin{array}{c} \\ M_1 \\ M_2 \\ M_3 \end{array} \qquad (1)$$

Figure 1 shows two examples of valid schedules for this example instance of JSSP.

In this paper, we propose an algorithm for creating schedules that minimise the makespan of JSSP instances by combining the Simulated Annealing algorithm with a Firefly-inspired methodology. The rationale behind this combination is to escape from local minima and avoid becoming constrained in the solution space.

The JSSP has been a focal point for many researchers over the last few decades, primarily due to the growing need for efficiency accompanied by the rapid increase in the speed of computing [2]. This, together with the more recent developments in the availability of cloud resources allowing for large scale distributed and parallel computation, has opened up many additional avenues in terms of algorithmic techniques for working with optimisation problems.

The JSSP has a core motivation in production planning and manufacturing systems. Reduced costs and efficient production lines are a direct result of fast

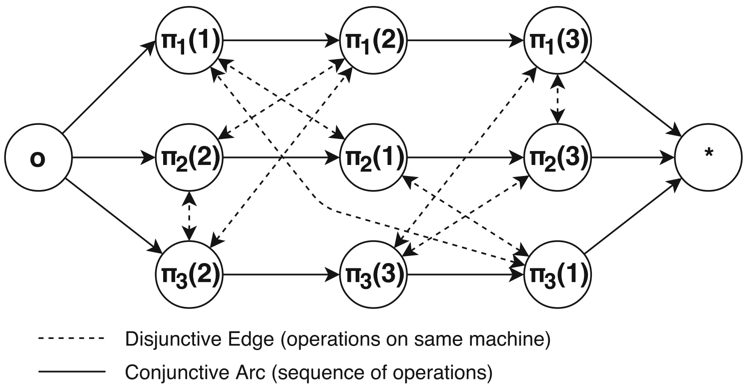

o
π1(1)
π1(2)
π1(3)
π2(2)
π2(1)
π2(3)
π3(2)
π3(3)
π3(1)
*
Disjunctive Edge (operations on same machine)
Conjunctive Arc (sequence of operations)

Fig. 2. Disjunctive graph exemplified from the previous section (Fig. 1). Recall that $\pi_j(i)$ refers to the operation for the j-th job on the i-th machine. Solid arcs indicate the order of operations for each job. Dashed edges indicate operations on the same machine.

and accurate scheduling solvers. Advancements in job shop scheduling also allow for prioritisation of operations and improved predicted completion times [1].

This paper builds on the research of Steinhöfel et al. on SA [20] and her work with Albrecht et al. on FA [13] by evaluating their symbiosis for the job shop scheduling problem. Moreover, the presented method is motivated by the property of a backbone structure, see [22], and the Firefly idea which enables the search of areas close to the estimated backbones of optimal schedules.

2 Background

The publication of a seminal paper by Johnson (1954) is largely credited as being the starting line for scheduling research [11]. The article outlines two and three-stage production schedules with the objective of minimising the total elapsed time. Early methods of solving JSSP instances include priority rules, branch and bound, integer programming, Monte-Carlo methods, stochastic analysis, algorithms with learning, and enumerative algorithms [15,17].

An important graphical representation of the JSSP, is the disjunctive graph proposed by Roy and Sussman [17].

Figure 2 shows a disjunctive graph for the example defined in the previous section. This graphical formulation provides an outline of all feasible schedules, enabling better interpretation than the usual Gantt chart representations. Modification of the schedule structure is achieved by activating disjunctive edges to represent an ordering of tasks on a machine path. Figure 3 outlines a concrete schedule equivalent to instance (a) in Fig. 1.

The source **o** and dummy vertex * act as start and end points respectively, allowing for full navigation of the directed graph; the longest path from **o** to * is the makespan. The makespan can be calculated by weighting all vertices by the

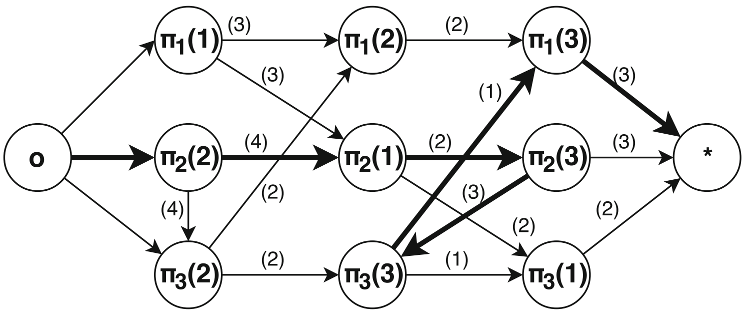


Fig. 3. Disjunctive graph of the schedule instance (a) from the previous section (Fig. 1). Recall that $\pi_j(i)$ refers to the operation for the j-th job on the i-th machine. The vertex weights are shown on their outgoing edges, and the thick arcs show a longest path with respect to vertex weights.

respective **P** values of each operation $\pi_j(i)$ and summing the total processing time of all operations on the longest path.

Note that, in [4], the graph derived from a disjunctive graph by considering a concrete schedule has exactly one arc for every disjunctive edge, let us call these *disjunctive arcs*. But any disjunctive arc shortcutting a directed path of disjunctive arcs does not contribute to a longest path (in terms of vertex weights) since the shortcut misses at least one vertex. Thus, we can omit these shortcuts from the graph.

Potts and Strusevich (2009) demonstrate a simple lower bound of JSSP instances by removing all disjunctive edges in such a graph and finding the maximum weight path in the resulting sub-graph [18]. Various more complex lower bounds are evaluated by dividing a JSSP instance into sub-problems by machine, calculating the lower bound for the single instance and selecting the optimal result.

2.1 Local Search

Local Search (LS) methods involve starting with some initial solution, and then iteratively moving to solutions that are similar, but have better objective function values. This iterative process repeats until a local optimum is found. A survey of LS methods for JSSP can be found in Vaessens et al. [24].

Let F be the set of feasible solutions for some problem instance. The cost of any $x \in F$ is defined by $c\colon F \to \mathbb{R}$. The neighbourhood function $N\colon F \to \{0,1\}^F$ determines which solutions in F are similar to x. That is, for $x \in F$, the solutions in the neighbourhood $N(x) \subseteq F$ are said to be neighbours to x. The execution of a local-search algorithm defines a walk in F such that each solution visited is a neighbour of the previously visited one. A solution $x \in F$ is called a local minimum with respect to a neighbourhood function N if $c(x) \leq c(y)$ for all $y \in N(x)$.

A major drawback of LS methods is that of becoming trapped in a local optimum. One attempt at addressing this problem is Simulated Annealing (SA). Annealing in metallurgy is the method of heating and then cooling metals to maximise the size of crystals. SA simulates this process for optimisation problems. As in LS methods, there is a neighbourhood function and a cost function. In SA, rather than always moving to a solution with a better objective value, there is some probability of moving to worse solutions. This probability is determined by the temperature, which reduces with time. A higher temperature means the algorithm is more likely to move to a worse solution. This allows the algorithm to leave local minima and explore more of the solution space.

In 1992, Laarhoven et al. first formulated the SA approach for the JSSP [25], along with further results by Blazewicz et al. [4], Steinhöfel et al. [20] and Satake et al. [19], respectively. SA has proven to be an effective technique for efficiently finding approximate results, overcoming to some extent the aforementioned limitations of LS.

2.2 Evolutionary Algorithms and Hybrids

Various evolutionary algorithms have been used as heuristics for solving JSSP instances, including Genetic Algorithms (GA) [7], Ant Colony Optimisation (ACO) [5] and Particle Swarm Optimisation (PSO) [6]. Instead of improving a single candidate solution, these meta-heuristics improve a population of solutions.

The method developed in this paper follows a line of research on hybrid approaches that combine SA with nature-inspired algorithms. Our method combines SA with the Firefly Algorithm (FA), which is one of the more recent nature inspired optimisation algorithms, published in 2008 by Xin-She Yang [6]. Fireflies are characterised by their luminescence, using their emittance of light to attract other fireflies - essentially their main form of communication.

The intensity of light emitted is directly proportional to the volume and rate at which other flies converge toward it. The attractiveness of the flies is linked to their objective function and monotonic decay of the attractiveness with distance, allowing individual flies to be attracted to any other firefly in the population, assuming they are close enough. The algorithm developed in this paper simplifies this approach somewhat, and resembles more accurately the algorithm developed by Steinhöfel et al. on protein structure prediction in lattice models [13]. This implementation considers a single target, setting the optimal instance as the beacon of light for the rest of the population. In this scenario, the so called fireflies all attempt to move in the same direction.

The rationale supporting the use of FA for the JSSP emanates from an understanding of the solution space, wherein schedule instances close to the optimal are more likely to have similar edge orientations. In addition, updates of the beacon during execution allow for the population to constantly change direction, increasing the likelihood of exploring optimal zones.

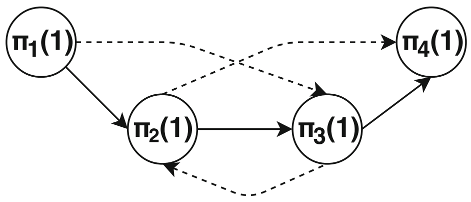


Fig. 4. Transition depicting edge flip on a machine path.

3 Our Algorithm

The method presented in this paper is a hybrid of Simulated Annealing (SA) and the Firefly Algorithm (FA).

The neighbourhood function determines for each solution x a subset of F. A solution belongs to that subset only if it can be reached from x with a single modification step, i.e., $N\colon F \rightarrow \{0,1\}^F$. That is, $N(x) \subseteq F$ is the set of feasible schedules similar to $x \in F$.

We define neighbours based on edge switching. Edges on a machine path can be switched such that a machine path $\pi_1(1)$-$\pi_2(1)$-$\pi_3(1)$-$\pi_4(1)$ can transition to $\pi_1(1)$-$\pi_3(1)$-$\pi_2(1)$-$\pi_4(1)$. A visual representation of this is provided in Fig. 4.

Given the disjunctive graph of a JSSP instance, switching a machine edge on the longest path will always result in a feasible (acyclic) schedule [20]. The choice of which edge on the longest path to flip can be determined by its rate of occurrence across all longest paths; known as its edge popularity. By flipping the most popular edge on a longest path, the likelihood of the neighbour having divergent longest paths, and hence more disparate makespan values, is greater.

However, computing all longest paths for every schedule is time consuming. We employ a different method, as follows. We switch a random machine edge and check if the resulting schedule is feasible. This can be determined by considering a sub-graph around the flipped edge in the full disjunctive graph. This allows us to efficiently check if a schedule is feasible. If it is feasible, it is considered a neighbour. If not, it is not.

An empirical analysis on real datasets showed that on average approximately 80% of edge flips resulted in a feasible schedule.

The full algorithm is shown in Algorithm 1. First, a set X of r initial schedules is generated. For each $x \in X$, a greedy local search is performed until each x is a local optimum. That is, after the greedy search, for every $x \in X$, we have that $\forall j \in N(x), L_x \leq L_j$. Let Θ be the current best solution. Simulated annealing is executed on Θ. And parallel to this, the firefly algorithm is performed using the $|X| - 1$ schedules in the set $X \setminus \Theta$.

Algorithm 1: SAFA Algorithm

```
input  : $t_0^{(\mathrm{sa})} \leftarrow$ starting temperature for SA, $c^{(\mathrm{sa})} \leftarrow$ cooling rate for SA,
         $t_0^{(\mathrm{fa})} \leftarrow$ starting temperature for FA, $c^{(\mathrm{fa})} \leftarrow$ cooling rate for FA,
         $r \leftarrow$ number of initial solutions
output: Best schedule found
1  $X \leftarrow$ set of $r$ initial schedules
2  for $x \in X$ do
3      $x \leftarrow$ result of greedy local search on $x$ until local optimum found
4  end
5  $\Theta \leftarrow \mathrm{argmin}_x L_x$
6  Begin SA thread with $\Theta$
7  Begin Fireflies thread with $X \setminus \Theta$
8  Return $\Theta$ when both threads finished

   Fireflies Thread

9  $temp^{(fa)} \leftarrow t_0^{(\mathrm{fa})}$
10 while $temp^{(fa)} > 1$ do
11     for $x \in X \setminus \Theta$ do
12         if $temp\ /t_0 > U(0,1)$ then
13             $x \leftarrow x$ with random machine edge flipped
14         else
15             $x \leftarrow x$ moved towards $\Theta$ with edge flip
16     end
17     $temp^{(fa)} \leftarrow temp^{(fa)} \cdot (1 - c)$
18     if any $L_x < L_\Theta$, replace $\Theta$ with that $x$
19 end

   SA Thread

20 $temp^{(sa)} \leftarrow t_0^{(\mathrm{sa})}$
21 $y \leftarrow \Theta$
22 while $temp^{(sa)} > 1$ do
23     if $\Theta$ was replaced by firefly thread, set $y \leftarrow \Theta$
24     $y' \leftarrow$ random neighbour in $N(y)$
25     if $L_{y'} < L_y$ then
26         $y = y'$
27     else if $temp\ /t_0 > U(0,1)$ then
28         $y \leftarrow y'$
29     $temp^{(sa)} \leftarrow temp^{(sa)} \cdot (1 - c)$
30 end
```

In a given iteration of simulated annealing, if a randomly selected neighbour solution is better than the current solution, we move to it. If not, then we move to it with some transition probability. This probability is proportional to a temperature that reduces (cools) from iteration to iteration. Early on in the SA procedure, the temperature is high and the probability of moving to worse solutions is higher compared to later in the SA procedure when the temperature is lower.

In the firefly algorithm, we also have a temperature that reduces over time. The temperature in this case, is relative to the luminescence of the firefly. We may consider the reduction in temperature equivalent to the intensification of the luminescence of Θ. With some probability proportional to the luminescence, a firefly moves towards Θ with an edge flip. Otherwise it moves to a random neighbour.

If at any point, a schedule with a makespan smaller than Θ is found, it replaces Θ, and the current schedule being considered in the SA thread is replaced by that new Θ.

Starting temperatures and cooling rates are chosen such that the number of iterations increases with input size. The starting population of the fireflies is altered such that the runtime is acceptable for the resources available.

4 Experimental Results

Table 1 shows the results of the SAFA algorithm on a selection of benchmark JSSP instances. The ID's[1] are constructed based on their origin, with benchmarks from Fisher & Thompson (ft) [8], Lawrence (la) [12], Storer, Wu and Vaccari (swv) [21], Taillard (ta) [23], Yamada and Nakano (yn) [26] and Applegate and Cook (orb) [3].

The SAFA algorithm was executed 30 times for each benchmark problem. Therefore, the SAFA AF (average found) is the average makespan found out of 30 runs. The makespan is calculated by reversing Dijkstras algorithm and finding the longest path from the source **o** to the dummy vertex *, as seen in Fig. 2.

A multitude of benchmarks have been considered for the purpose of evaluating the accuracy of the hybrid algorithm. The benchmarks have been selected based on their recognisability, size and comparability. Accordingly, results are compared to the best-found makespans, including a dissimilarity percentage between the best-found solutions of the SAFA hybrid and the known Upper Bound (UB). All computations were performed on a cloud server hosting a standard Intel dual-core processor and 7.5 gb of memory.

The SA parameter settings for the individual benchmarks are dependent on the size and complexity of the instance. In reference to Algorithm 1, the starting temperatures $t_0^{(\mathrm{sa})}$ and $t_0^{(\mathrm{fa})}$ are initiated with approximately the same value across all benchmarks. In contrast, the cooling rate parameters $c^{(\mathrm{sa})}$ and $c^{(\mathrm{fa})}$ decrease as the proportions of the benchmark increase. In other words, a lower

[1] http://jobshop.jjvh.nl.

Table 1. Results on benchmark problems. LB = lower bound. BF = best found with existing methods. SAFA BF and AF = Best found and average found respectively with SAFA method presented in this paper. % Diff = Percentage difference between the best found makespan of SAFA method compared to the best found. Note: ** = BF > LB

ID	Size	LB	BF	SAFA.BF	SAFA.AF	% Diff
ft06	6×6	55	55	55	55.00	0
ft10	10×10	930	930	937	948.63	0.7
ft20	20×5	1165	1165	1178	1181.25	1.1
la01	10×5	666	666	666	666.00	0
la10	15×5	958	958	958	958.00	0
la11	20×5	1222	1222	1222	1222.00	0
la23	15×10	1032	1032	1032	1032.00	0
la34	30×10	1721	1721	1721	1721.00	0
la35	30×10	1888	1888	1888	1888.00	0
la36	15×15	1268	1268	1306	1329.70	2.9
la37	15×15	1397	1397	1458	1485.33	4.3
swv11	50×10	2983	2983	3809	3868.10	27.6
swv13	50×10	3104	3104	3926	4013.30	26.4
swv17	50×10	2794	2794	2794	2794.00	0
swv18	50×10	2852	2852	2852	2852.00	0
ta69	100×20	3071	3071	3332	3352.33	8.4
ta71	100×20	5464	5464	6087	6230.33	11.4
ta76	100×20	5342	5342	5854	5854.00	9.5
yn02**	20×20	861	904	995	1053.80	10.0
yn03**	20×20	827	892	971	1018.00	8.8
orb01	10×10	1059	1059	1085	1089.80	2.4
orb02	10×10	888	888	895	897.30	0.7
orb03	10×10	1005	1005	1015	1037.28	0.9
orb04	10×10	1005	1005	1012	1014.50	0.6
orb05	10×10	887	887	894	895.50	0.7
orb06	10×10	1010	1010	1028	1037.75	1.7
orb07	10×10	397	397	407	407.00	2.5
orb08	10×10	899	899	928	937.50	3.2
orb09	10×10	934	934	948	953.75	1.4
orb10	10×10	944	944	957	957.00	1.3

cooling rate parameter is used for larger benchmark instances. This property allows SA to execute within a middle-ground of efficiency and efficaciousness, while enabling FA with enough iterations for fireflies to achieve their objective function - fully transitioning toward the state of the optimal schedule. This characteristic is essential for the firefly population given there is a greater likelihood of Θ being replaced when the underlying structures, and therein longest paths, become alikened to one another.

5 Conclusion

We have designed a hybrid SAFA algorithm for solving the JSSP. The algorithm has been implemented and executed on a number of different benchmarks. The experimental results are encouraging, wherein the best-found solution was reached for a number of instances and near-optimal solutions found for a wide range of others. The algorithm will be complemented by a more fine-grained analysis in the future. FA improves on the underlying SA algorithm at multiple junctures throughout its execution, with results reflecting conclusive remarks similar to other applications of FA for optimisation problems. Given the results of this paper, future work will be directed at utilising information and estimations of the backbone structure even more. In general, the combination of SA and FA works well and could further be tailored to reach best-found solutions for the larger JSSP instances as well as also being applied to solving other combinatorial optimisation problems.

References

1. Abas, M., Abbas, A., Khan, W. A.: Scheduling job shop - a case study. In: IOP Conference Series: Materials Science and Engineering, vol. 146, p. 12052. IOP Publishing (2016)
2. Adams, J., Balas, E., Zawack, D.: The shifting bottleneck procedure for job shop scheduling. Manag. Sci. **34**(3), 391–401 (1988)
3. Applegate, D., Cook, W.: A computational study of the job-shop scheduling problem. ORSA J. Comput. **3**(2), 149–156 (1991)
4. Błażewicz, J., Domschke, W., Pesch, E.: The job shop scheduling problem: conventional and new solution techniques. Eur. J. Oper. Res. **93**(1), 1–33 (1996)
5. Dorigo, M., Maniezzo, V., Colorni, A.: Positive feedback as a search strategy. Dipartimento di Elettronica, Politecnico di Milano, Italy, Technical report 91-016 (1991)
6. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: 1995 Proceedings of the Sixth International Symposium on Micro Machine and Human Science, MHS 1995, pp. 39–43. IEEE (1995)
7. Falkenauer, E., Bouffouix, S.: A genetic algorithm for job shop. In: 1991 IEEE International Conference on Robotics and Automation, Proceedings, pp. 824–829. IEEE (1991)
8. Fisher, H.: Probabilistic learning combinations of local job-shop scheduling rules. Industrial scheduling, pp. 225–251 (1963)
9. Hromkovič, J., Mömke, T., Steinhöfel, K., Widmayer, P.: Job shop scheduling with unit length tasks: bounds and algorithms. Algorithmic Oper. Res. **2**(1), 1–14 (2007)

10. Hromkovič, J., Steinhöfel, K., Widmayer, P.: Job shop scheduling with unit length tasks: bounds and algorithms. ICTCS 2001. LNCS, vol. 2202, pp. 90–106. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45446-2_6
11. Johnson, S.M.: Optimal two-and three-stage production schedules with setup times included. Naval Res. Logist. (NRL) **1**(1), 61–68 (1954)
12. Lawrence, S.: Resouce constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement). Carnegie-Mellon University, Graduate School of Industrial Administration (1984)
13. Maher, B., Albrecht, A.A., Loomes, M., Yang, X.S., Steinhöfel, K.: A firefly-inspired method for protein structure prediction in lattice models. Biomolecules **4**(1), 56–75 (2014)
14. Mömke, T.: On the power of randomization for job shop scheduling with k-units length tasks. RAIRO Theor. Inform. Appl. **43**(2), 189–207 (2009)
15. Muth, J.F., Thompson, G.L.: Industrial scheduling. Prentice-Hall, Upper Saddle River (1963)
16. Pinedo, M. L.: Scheduling: Theory, and Systems (2008)
17. Potts, C.N., Strusevich, V.A.: Fifty years of scheduling: a survey of milestones. J. Oper. Res. Soc. **60**(1), S41–S68 (2009)
18. Roy, B., Sussmann, B.: Les problemes d'ordonnancement avec contraintes disjonctives. Technical report 9 (1964)
19. Satake, T., Morikawa, K., Takahashi, K., Nakamura, N.: Simulated annealing approach for minimizing the makespan of the general job-shop. Int. J. Prod. Econ. **60**, 515–522 (1999)
20. Steinhöfel, K., Albrecht, A., Wong, C.K.: Two simulated annealing-based heuristics for the job shop scheduling problem. Eur. J. Oper. Res. **118**(3), 524–548 (1999)
21. Storer, R.H., Wu, S.D., Vaccari, R.: New search spaces for sequencing instances with application to job shop scheduling. Manag. Sci. **38**, 1495–1509 (1992)
22. Streeter, M.J., Smith, S.F.: How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. J. Artif. Intell. Res. (JAIR) **26**, 247–287 (2006)
23. Taillard, E.: Benchmarks for basic scheduling problems. Eur. J. Oper. Res. **64**(2), 278–285 (1993)
24. Vaessens, R.J.M., Aarts, E.H.L., Lenstra, J.K.: Job shop scheduling by local search. INFORMS J. Comput. **8**(3), 302–317 (1996)
25. Van Laarhoven, P.J.M., Aarts, E.H.L., Lenstra, J.K.: Job shop scheduling by simulated annealing. Oper. Res. **40**(1), 113–125 (1992)
26. Yamada, T., Nakano, R.: A genetic algorithm applicable to large-scale job-shop instances. In: Manner, Manderick (eds.) Parallel Instance Solving from Nature 2 (1992)

Rendezvous of Asynchronous Mobile Robots with Lights

Takashi Okumura[1], Koichi Wada[2(✉)], and Yoshiaki Katayama[3]

[1] Graduate School of Science and Engineering, Hosei University, Tokyo 184-8584, Japan
takashi.okumura.4e@stu.hosei.ac.jp
[2] Faculty of Science and Engineering, Hosei University, Tokyo 184-8485, Japan
wada@hosei.ac.jp
[3] Graduate School of Engineering, Nagoya Institute of Technology, Nagoya 466-8555, Japan
katayama@nitech.ac.jp

Abstract. We study a *Rendezvous* problem for 2 autonomous mobile robots in asynchronous settings with persistent memory called *light*. It is well known that Rendezvous is impossible when robots have no lights in basic common models, even if the system is semi-synchronous. On the other hand, Rendezvous is possible if robots have lights with a constant number of colors in several types of lights [9,21]. In asynchronous settings, Rendezvous can be solved by robots with 3 colors of lights in non-rigid movement and with 2 colors of lights in rigid movement, respectively [21], if the robots can use not only their own light but also the other robot's light (*full-light*), where non-rigid movement means robots may be stopped before reaching the computed destination but can move a minimum distance $\delta > 0$, and rigid movement means robots can reach the computed destination. In semi-synchronous settings, Rendezvous can be solved with 2 colors of full-lights in non-rigid movement.

In this paper, we show that in asynchronous settings, Rendezvous can be solved with 2 colors of full-lights in non-rigid movement if robots know the value of the minimum distance δ. We also show that Rendezvous can be solved with 2 colors of full-lights in general non-rigid movement if we consider some reasonable restricted class of asynchronous settings.

1 Introduction

Background and Motivation

The computational issues of autonomous mobile robots have been research object in distributed computing fields. In particular, a large amount of work has been dedicated to the research of theoretical models of autonomous mobile robots [1–3,6,12,15,18,19]. In the basic common setting, a robot is modeled as a point in a two-dimensional plane and its capability is quite weak. We usually assume that robots are *oblivious* (no memory to record past history), *anonymous* and *uniform* (robots have no IDs and run identical algorithms) [8]. Robots operate in

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 434–448, 2018.
https://doi.org/10.1007/978-3-319-98355-4_25

Look-Compute-Move (LCM) cycles in this model. In the Look operation, robots obtain a snapshot of the environment and they execute the same algorithm with the snapshot as an input to the Compute operation, and move towards the computed destination in the Move operation. Repeating these cycles, all robots perform a given task. It is difficult for these very weak robot systems to complete the task. Revealing the weakest capability of robots sufficient to attain a given task is one of the most interesting challenges in the theoretical research of autonomous mobile robots.

The problem considered in this paper is *Gathering*, which is one of the most fundamental tasks of autonomous mobile robots. Gathering is the process of n mobile robots, initially located on arbitrary positions, to meet within finite time at a location, not known a priori. When there are two robots in this setting, this task is called *Rendezvous*. In this paper, we focus on Rendezvous in asynchronous settings and we reveal the weakest additional assumptions for this problem.

Since Gathering and Rendezvous are simple but essential problems, they have been intensively studied and a number of possibility and/or impossibility results have been shown under different assumptions [1–3,5–7,10,13–18]. The solvability of Gathering and Rendezvous depends on the activation schedule and the synchronization level. Usually three basic types of schedulers are identified, the fully synchronous (FSYNC), the semi-synchronous (SSYNC) and the asynchronous (ASYNC). In the FSYNC model, there are common rounds and in each round all robots are activated simultaneously and Compute and Move are done instantaneously. The SSYNC is the same as FSYNC except that in each round only a subset of the robots is activated. In the ASYNC scheduler, there is no restriction on the notion of time, Compute and Move in each cycle can take an unpredictable amount of time, and the time interval between successive activations is also unpredictable (but these times must be finite). Gathering and Rendezvous are trivially solvable in FSYNC by using an algorithm that moves to the center of gravity. However, Rendezvous cannot be solved even in SSYNC without any additional assumptions [8].

In [4], persistent memory called *light* has been introduced to reveal the relationship between ASYNC and SSYNC and they show that asynchronous robots with lights equipped with a constant number of colors are strictly more powerful than semi-synchronous robots without lights. In order to solve Rendezvous without any other additional assumptions, robots with lights have been introduced [4,9,21].

Table 1 shows results on solving Rendezvous by robots with lights with each scheduler and movement restriction. In the table, *full-light* means that robots can see not only the lights of other robots but also their own light. In the movement restriction, Rigid means that robots can reach the computed destination. In Non-Rigid, robots may be stopped before reaching the computed destination but move a minimum distance $\delta > 0$. Non-Rigid($+\delta$) means the setting is Non-Rigid and the robots know the value δ. Gathering of robots with lights is discussed in [20].

Table 1. Rendezvous algorithms by robots with full-lights.

Scheduler	Movement	Full-light [21]	No-light [8,17]
FSYNC	Non-rigid	–	◯
SSYNC	Non-rigid	2	×
	Rigid	–	
	Non-rigid(+δ)	–	
ASYNC	Non-rigid	3	×
	Rigid	2	
	Non-rigid(+δ)	?	

◯ and × mean solvable and unsolvable, respectively,
– indicates that this part has been solved under a weaker condition, and
? means this part is not solved.

Our Contribution

In this paper, we consider whether we can solve Rendezvous in ASYNC with the optimal number of colors of light. In SSYNC, Rendezvous cannot be solved with one color but can be solved with 2 colors in Non-Rigid and full-light. On the other hand, Rendezvous in ASYNC can be solved with 3 colors in Non-Rigid and full-light, and with 2 colors in Rigid.

In this paper, we show that Rendezvous in ASYNC can be solved with 2 colors in Non-Rigid(+δ) and full-light. We give a basic Rendezvous algorithm with 2 colors of full-lights (A and B)[1] and it can solve Rendezvous in ASYNC and Rigid, and its variant can also solve Rendezvous in ASYNC and Non-Rigid(+δ). These two algorithms can behave correctly if the initial color of each robot is A. However if the initial color of each robot is B, the algorithm cannot solve Rendezvous in ASYNC and Rigid. It is still open whether Rendezvous can be solved with 2 colors in ASYNC and Non-Rigid. However, we introduce some restricted class of ASYNC called *LC-atomic* ASYNC, and we show that our basic algorithm can solve Rendezvous in this scheduler and Non-Rigid with arbitrary initial color. Here, LC-atomic ASYNC means that we consider the interval from the beginning of each Look operation to the end of the corresponding Compute operation as an atomic one, that is, no robot can observe between the beginning of each Look operation and the end of each Compute operation in every cycle. This is a reasonable sufficient condition so that Rendezvous can be solved with the optimal number of colors of light in ASYNC and No-Rigid.

2 Model and Preliminaries

We consider a set of n anonymous mobile robots $\mathcal{R} = \{r_1, \ldots, r_n\}$ located in $\mathbb{R}^2$. Each robot r_i has a persistent state $\ell(r_i)$ called *light* which may be taken from a finite set of colors L.

[1] This is essentially the same as that in [21].

We denote by $\ell(r_i, t)$ the color of light the robot r_i has at time t and by $p(r_i, t) \in \mathbb{R}^2$ the position occupied by r_i at time t represented in some global coordinate system. Given two points $p, q \in \mathbb{R}^2$, $dis(p, q)$ denotes the distance between p and q.

Each robot r_i has its own coordinate system where r_i is located at its origin at any time. These coordinate systems do not necessarily agree with those of other robots. This means that there is no common unit of distance and no common knowledge of directions of its coordinates and clockwise orientation (*chirality*).

At any point in time, a robot can be active or inactive. When a robot r_i is activated, it executes *Look*-*Compute*-*Move* operations:

– **Look:** The robot r_i activates its sensors to obtain a snapshot which consists of pairs of a light and a position for every robot with respect to its own coordinate system. Since the result of this operation is a snapshot of the positions of all robots, the robot does not notice the movement, even if the robot sees other moving robots. We assume robots can observe all other robots (unlimited visibility).
– **Compute:** The robot r_i executes its algorithm using the snapshot and its own color of light (if it can be utilized) and returns a destination point des_i with respect to its coordinate system and a light $\ell_i \in L$ to which its own color is set.
– **Move:** The robot r_i moves to the computed destination des_i. A robot r is said to *collide* with robot s at time t if $p(r, t) = p(s, t)$ and r is performing *Move* at time t, and r's collision is *accidental* if its destination is not $p(r, t)$. Since robots are seen as points, we assume that accidental collisions are immaterial. A moving robot, upon causing an accidental collision, proceeds in its movement without changes, in a "hit-and-run" fashion [8]. The robot may be stopped by an adversary before reaching the computed destination.[2] If stopped before reaching its destination, a robot moves at least a minimum distance $\delta > 0$. Note that without this assumption an adversary could make it impossible for any robot to ever reach its destination, following a classical Zenonian argument [8]. If the distance to the destination is at most δ, the robot can reach it. In this case, the movement is called *Non-Rigid*. Otherwise, it is called *Rigid*. If the movement is Non-Rigid and the robots know the value of δ, it is called *Non-Rigid(+δ)*.

A scheduler decides which subset of robots is activated for every configuration. The schedulers we consider are asynchronous or semi-synchronous and it is assumed that for every robot r and time t, there exists a time $t'(\geq t)$ at which r becomes active. Usually this scheduler is said to be *fair* [8].

– **ASYNC:** The asynchronous (ASYNC) scheduler activates the robots independently, and the duration of each Compute, Move and the time between successive activities is finite and unpredictable. As a result, a robot can be seen while moving and the snapshot and its actual configuration are not the same, and so its computation may be done with the old configuration.

[2] E.g., due to limits to its motion energy.

- **SSYNC:** The semi-synchronous (SSYNC) scheduler activates a subset of all robots synchronously and their Look-Compute-Move cycles are performed at the same time. We can assume that activated robots at the same time obtain the same snapshot, and their Compute and Move are executed instantaneously. In SSYNC, we can assume that each activation defines a discrete time called *round* and Look-Compute-Move is performed instantaneously in one round. A subset of activated robots in each round is determined by an adversary and robots do not know about this subset.

As a special case of SSYNC, if all robots are activated in each round, the scheduler is called fully-synchronous (FSYNC).

In this paper, we consider ASYNC and we assume the following.

In a Look operation, a snapshot of a time t_L is taken and we say that *the Look operation is performed at time* t_L. Each Compute operation of r_i is assumed to be done at an instant time t_C and its color of light $\ell_i(t)$ and its destination des_i are assigned to the computed values at the time t_C. In a *Move* operation, when its movement begins at t_B and ends at t_E, we say that its movement is performed during $[t_B, t_E]$, its beginning and ending of the movement are denoted by $Move_{BEGIN}$ and $Move_{END}$, and its $Move_{BEGIN}$ and $Move_{END}$ occur at t_B and t_E, respectively. In the following, *Compute*, $Move_{BEGIN}$ and $Move_{END}$ are abbreviated by *Comp*, $Move_B$ and $Move_E$, respectively. When some cycle has no movement (robots change only colors of lights, or their destinations are the current positions), we can assume that the Move operation in this cycle is omitted, since we can consider the Move operation can be performed just before the next Look operation.

Also we consider the following restricted classes of ASYNC.

Let a robot execute a cycle. If any other robot cannot execute any *Look* operation between the *Look* operation and the following *Comp* operation in the cycle, its ASYNC is said to be *LC-atomic*. Thus we can assume that in LC-atomic ASYNC, *Look* and *Comp* operations in every cycle are performed at the same time. If any other robot cannot execute any *Look* operation between the $Move_B$ and the following $Move_E$, its ASYNC is said to be *Move-atomic*. In this case *Move* operations in all cycles can be considered to be performed instantaneously and at time t_M. In Move-atomic ASYNC, when a robot r observes another robot r' performing a *Move* operation at time t_M, r observes the snapshot after the moving of r'.

In our settings, robots have persistent lights and can change their colors at an instant time in each Compute operation. We consider the following robot models according to visibility of lights.

- *full-light*, the robot can recognize not only colors of lights of other robots but also its own color of light.
- *external-light*, the robot can recognize only colors of lights of other robots but cannot see its own color of light. Note that the robot can change its own color.
- *internal-light*, the robot can recognize only its own color of light but cannot see colors of lights of other robots.

Table 2. Rendezvous algorithms by robots with three kinds of lights.

Scheduler	Movement	Full-light [21]	External-light [9]	Internal-light [9]	No-light [8,17]
FSYNC	Non-rigid	–	–	–	◯
SSYNC	Non-rigid	2	3	?	×
	Rigid	–	?	6	
	Non-rigid($+\delta$)	–	?	3	
ASYNC	Non-rigid	3	?	?	×
	Rigid	2	12	?	
	Non-rigid($+\delta$)	?	3	?	

◯ and × mean solvable and unsolvable, respectively,
– indicates that this part has been solved under a weaker condition, and
? means that this part is not solved.

An n-*Gathering* problem is defined as follows. Given $n(\geq 2)$ robots initially placed at arbitrary positions in $\mathbb{R}^2$, they congregate at a single location which is not predefined in finite time. In the following, we consider the case that $n = 2$ and the 2-Gathering problem is called *Rendezvous*.

3 Previous Results for Rendezvous

Rendezvous is not solvable in the basic model without lights.

Theorem 1 (Flocchini et al. [8]). *Rendezvous is deterministically unsolvable in SSYNC even if chirality is assumed.*

If robots have a constant number of colors in their lights, Rendezvous can be solved with three kinds of lights, as shown in Table 2. Since robots are oblivious and anonymous, the colors can be used to break symmetry of robots and configurations [9,21]. Since full-lights are stronger assumptions than external-lights and internal-lights, the number of colors of full-lights used in algorithms is fewer than that of external-lights and internal-lights. Also previous algorithms and our algorithms with full-lights are in a class of algorithms called $\mathcal{L}$-algorithms [21], which means that each robot may only compute a destination point on the line connecting two points robots are located at by using only the colors of lights of current robots. The algorithms of this class are of interest because they operate also when the coordinate system of a robot is not self-consistent (i.e., it can unpredictably rotate, change its scale or undergo a reflection) [9]. On the other hand, there does not exist any $\mathcal{L}$-algorithm with external-lights and internal-lights in ASYNC and SSYNC, respectively [9].

It is still an open problem whether Rendezvous can be solved in ASYNC with 2 colors in Non-Rigid. In the following, we will show that Rendezvous is solved in ASYNC and full-light with 2 colors, if we assume (1) Rigid movement, (2) Non-Rigid movement and knowledge of the minimum distance δ robots move, (3) LC-atomic. In these cases, we can construct optimal Rendezvous algorithms with respect to the number of colors in ASYNC.

Algorithm 1. Rendezvous(scheduler, movement, initial-color)

```
Parameters: scheduler, movement-restriction, initial-color
Assumptions: full-light, two colors ($A$ and $B$)
1   case me.light of
2   $A$:
3       if $other.light = A$ then
4           $me.light \leftarrow B$
5           $me.des \leftarrow$ the midpoint of $me.position$ and $other.position$
6       else $me.des \leftarrow other.position$
7   $B$:
8       if $other.light = A$ then
9           $me.des \leftarrow me.position$        // stay
10      else $me.light \leftarrow A$
11  endcase
```

4 Asynchronous Rendezvous for Robots with Lights

4.1 Basic Rendezvous Algorithm

In this section, two robots are denoted by r and s. Let t_0 be the starting time of the algorithm.

Given a robot *robot*, an operation *op* ($\in \{Look, Comp, Move_B, Move_E\}$), and a time t, $t^+(robot, op)$ denotes the time *robot* performs the first *op* after t (inclusive) if there exists such an operation, and $t^-(robot, op)$ denotes the time *robot* performs the first *op* before t (inclusive) if there exists such an operation. If t is the time the algorithm terminates, $t^+(robot, op)$ is not defined for any *op*. When *robot* does not perform *op* before t and $t^-(robot, op)$ does not exist, $t^-(robot, op)$ is defined to be t_0.

A time t_c is called a *cycle start time* if the next performed operations of both r and s after t are Look operations, or otherwise, the robots performing the operations neither change their colors of lights nor move. In the latter case, we can consider that these operations can be performed before t_c, and the subsequent *Look* operation can be performed as the first operation after t_c.

Algorithm 1 is used as a basic Rendezvous algorithm which has three parameters, namely scheduler, movement restriction and an initial color of light, and it assumes full-light and uses two colors A and B.

We will show that Rendezvous(ASYNC, Rigid, A) and Rendezvous(LC-atomic ASYNC, Non-Rigid, any[3]) solve Rendezvous, and that some variant of Rendezvous(ASYNC, Non-Rigid($+\delta$), A) also solves Rendezvous.

Lemma 1. *Assume that time t_c is a cycle start time and $\ell(r, t_c) = \ell(s, t_c) = B$ in Rendezvous(ASYNC, Non-Rigid, any). If $dis(p(r, t_c)), p(s, t_c)) = 0$, then two robots r and s do not move after t_c.*

[3] Either A or B will do.

Lemma 2. *Let robot r perform a Look operation at time t in Rendezvous(ASYNC, Non-Rigid, any). If $t^-(s, Comp) \leq t$ and $\ell(r,t) \neq \ell(s,t)$, then there exists a time $t^*(> t)$ such that r and s succeed in rendezvous at time t^* by Rendezvous(ASYNC, Non-Rigid, any).*

Proof. If $\ell(r,t) = B$ and $\ell(s,t) = A$, then r does not change the color and stays at the current position. If s performs a *Look* operation at $t^+(s, Look)$, s does not change the color and s's destination is $p(r,t)$. Since both r and s do not change the colors after the time $t^+(s, Look)$, r stays at $p(r,t)$ and the destination of s is $p(r,t)$. Thus r and s succeed in rendezvous at some time $t^* \geq t^+(s, Move_E)$ even in Non-Rigid.

If $\ell(r,t) = A$ and $\ell(s,t) = B$, then r does not change the color and computes the destination as $p(s,t)$. When s finishes the *Move* operation at $t' = t^+(s, Move_E)$, s is located at $p(s,t')$. If $t' \leq t$, since r's destination is $p(s,t)$ and $p(s,t)$ is not changed (even if s performs a *Look* operation after t' and before t), r and s succeed in rendezvous at some time $t^* \geq t^+(r, Move_E)$.

Otherwise ($t < t'$), if r performs *Look* operations before t', these destinations are different since s is moving, but the color is not changed and $\ell(r,t') = A$. Since s stays at $p(s,t')$ after t', r and s succeed in rendezvous at some time $t^* \geq t^+(r, Move_E)$.

In both cases r and s do not move after t^* by the algorithm. □

4.2 ASYNC and Rigid Movement

If Rigid movement is assumed, asynchronous Rendezvous can be done with 2 colors.

Theorem 2. *Rendezvous(ASYNC, Rigid, A) solves Rendezvous.*

Proof. Let t_0 be the starting time of the algorithm and let r and s be two robots whose colors of lights are A. Without loss of generality, r is assumed to perform the *Look* operation first at time t_1, that is, $t_1 = t_0^+(r, Look)$. Let $t_2 = t_0^+(s, Look)$. There are two cases: (I) $t_1 \leq t_2 < t_0^+(r, Comp)$ and (II) $t_0^+(r, Comp) \leq t_2$.

Case (I): Since $\ell(s,t_2) = A$ and $\ell(r,t_2) = A$, s moves to the midpoint of $p(r,t_0)$ and $p(s,t_0)$ at time $t_3 = t_0^+(s, Move_E)$. Robot s changes its color of light from A to B at time $t_0^+(s, Comp)$ and $\ell(s,t_3) = B$. There are two cases (I-1) $t'_3 = t_0^+(r, Move_E) < t_3$ and (I-2) $t_3 \leq t'_3 = t_0^+(r, Move_E)$.

Case (I-1): Robot r reaches the destination at time t'_3 but s does not reach the destination ($t'_3 = t_0^+(r, Move_E) < t_3$). If r does not perform any operations during $[t'_3, t_3]$, t_3 becomes a cycle start time and then r and s rendezvous at time t_3 and the two robots do not move after t_3 by Lemma 1.

Otherwise, r performs several operations during $[t'_3, t_3]$. Suppose r performs at least one *Look* operation and one *Comp* operation during $[t'_3, t_3]$ and t_C denotes the time r performs the *Comp* operation ($t_C =$

$t_3'^{+}(r, Comp)$). Note that r only changes its color of light and does not move in this cycle. Then its color of light is changed to A at t_C, and $\ell(r, t_C) = A$ and $\ell(s, t_C) = B$. Thus, the next *Look* operation of r or s after t_C satisfies the conditions of Lemma 2, and r and s succeed in rendezvous. The remaining case is that r performs only a *Look* operation during $[t_3', t_3]$. Let t_L be the time r performs the *Look* operation. Since r observes $\ell(s, t_L) = B$ and r and s are located at the same point at t_3, this case is the same as the first case.

Case (I-2): Interchanging the roles of r and s, this case can be reduced to Case (I-1).

Case (II): Since $(t_0)^+(r, Comp) \leq t_2$ and $\ell(r, t_2) \neq \ell(s, t_2)$, r and s succeed in rendezvous by Lemma 2. □

Note that this algorithm does not terminate and we cannot change the algorithm so that the fixed one can terminate. It is an open problem whether there exists an algorithm which solves Rendezvous and terminates with two colors in ASYNC. Also there exists an execution such that Rendezvous(Async, Rigid, any) does not work in general. In fact, if the initial colors of lights for both robots are B, this algorithm cannot solve Rendezvous. Figure 1 shows a counterexample where Rendezvous(Async, Rigid, B) does not work. Since the colors of lights at t_5 are B, this execution repeats forever and achieves only convergence, that is, the robots move arbitrarily close to each other, but might not rendezvous within finite time. This counterexample also shows that Rendezvous (Move-atomic ASYNC, Rigid, B) does not work. However, if we assume LC-atomic ASYNC, we can show that Rendezvous(LC-atomic ASYNC, Rigid, B) solves Rendezvous.

Lemma 3. *Rendezvous(LC-atomic ASYNC, Rigid, B) solves Rendezvous.*

Proof. In LC-atomic ASYNC, any *Look* operation and the following *Comp* operation are performed at the same time and this operation is denoted by LC. Let t_c be a cycle start time and let r perform an LC operation first and let $t_1 = t_c^+(r, LC)$. There are two cases: (I) $t_1 = t_c^+(s, LC)$ and (II) $t_1 < t_c^+(s, LC) = t_2$.

Case (I): In this case, since t_1 becomes a cycle start time and $\ell(r, t_1) = \ell(s, t_1) = A$, the lemma holds by Theorem 2.

Case (II): In this case, since $\ell(r, t_1) = A$ and $\ell(s, t_2) = B$, the lemma holds by Lemma 2. □

In an execution of Rendezvous(Async, Non-Rigid, A) starting from $\ell(r, t_0) = \ell(s, t_0) = A$, if r and s perform one cycle simultaneously, their colors can be changed into B without attaining Rendezvous, that is, there is a cycle start time $t_c (\geq t_0)$ such that $\ell(r, t_c) = \ell(s, t_c) = B$. Thus it cannot solve Rendezvous even if both initial colors of lights are A. In Subsect. 4.3, we will show that if ASYNC is restricted to LC-atomic, Rendezvous can be solved in Non-Rigid with two colors from any initial colors of lights.

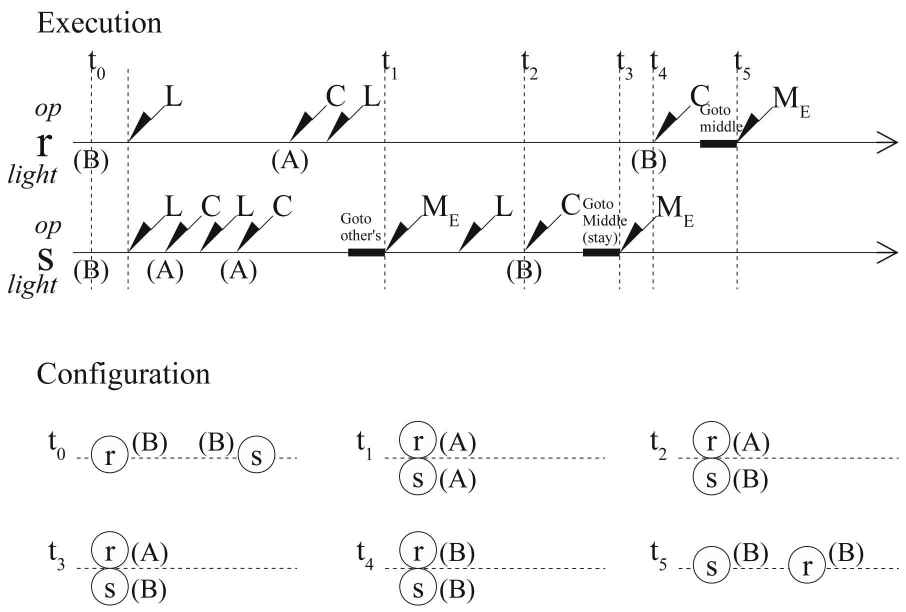


Fig. 1. Rendezvous(ASYNC, Rigid, B) cannot solve Rendezvous in general.

4.3 LC-Atomic ASYNC and Non-Rigid Movement

Let t_c be a cycle start time of the algorithm. There are three cases according to the colors of lights of two robots r and s, (I) $\ell(r,t_c) \neq \ell(s,t_c)$, (II) $\ell(r,t_c) = \ell(s,t_c) = A$, and (III) $\ell(r,t_c) = \ell(s,t_c) = B$.

Lemma 4. *If $\ell(r,t_c) \neq \ell(s,t_c)$ and the algorithm starts at t_c, then there exists a time $t^*(\geq t)$ such that r and s succeed in rendezvous at time t^* by Rendezvous (LC-atomic ASYNC, Non-Rigid, any).*

Proof. This is obvious from Lemma 2. □

Lemma 5. *If $\ell(r,t_c) = \ell(s,t_c) = B$ and the algorithm starts at t_c, then there exists a time $t^*(\geq t)$ such that r and s succeed in rendezvous at time t^* by Rendezvous(LC-atomic ASYNC, Non-Rigid, any) or t^* is a cycle start time and $\ell(r,t^*) = \ell(s,t^*) = A$.*

Proof. Let r perform the LC operation first and let $t_1 = t_c^+(r, LC)$. There are two cases: (I) $t_1 = t_c^+(s, LC)$ and (II) $t_1 < t_c^+(s, LC) = t_2$.

(I) In this case, $\ell(r,t_1) = \ell(s,t_1) = B$ and there are two cases: (I-1) $t_1^+(r, LC) \neq t_1^+(s, LC)$ and (I-2) $t_1^+(r, LC) = t_1^+(s, LC)$.

(I-1) This case can be proven by Lemma 2.

(I-2) Letting $t^* = t_1^+(r, LC) = t_1^+(s, LC)$, $\ell(r,t^*) = \ell(s,t^*) = A$ and t_* becomes a cycle start time. Also at the time t_* rendezvous is succeeded or $dis(p(r,t^*), p(s,t^*)) \leq dis(p(r,t_c), p(s,t_c)) - 2\delta$.

(II) Since $\ell(r,t_1) = A$ and $\ell(s,t_2) = B$, this case is proven by Lemma 2. □

Algorithm 2. RendezvousWithDelta (ASYNC, Non-Rigid ($+\delta$), A)

```
Assumptions: full-light, two colors ($A$ and $B$)
case $dis(me.position, other.position) (= DIST)$ of
$DIST > 2\delta$:
    if $me.light = other.light = B$ then
        $me.des \leftarrow$ the point moving by $\delta/2$ from $me.position$ to $other.position$
    else $me.light \leftarrow B$
$2\delta \geq DIST \geq \delta$:
    if $me.light = other.light = A$ then
        $me.light \leftarrow B$
        $me.des \leftarrow$ the midpoint of $me.position$ and $other.position$
    else $me.light \leftarrow A$
$\delta > DIST$:        // Rendezvous(ASYNC, Rigid, A)
    case me.light of
    $A$:
        if $other.light = A$ then
            $me.light \leftarrow B$
            $me.des \leftarrow$ the midpoint of $me.position$ and $other.position$
        else $me.des \leftarrow other.position$
    $B$:
        if $other.light = A$ then $me.des \leftarrow me.position$        // stay
        else $me.light \leftarrow A$
    endcase
endcase
```

Lemma 6. *If $\ell(r, t_c) = \ell(s, t_c) = A$ and the algorithm starts at t_c, then there exists a time $t^*(\geq t_c)$ such that r and s succeed in rendezvous at time t^* by Rendezvous(LC-atomic ASYNC, Non-Rigid, any) or t^* is a cycle start time, $\ell(r, t^*) = \ell(s, t^*) = A$ and $dis(p(r, t^*), p(s, t^*)) \leq dis(p(r, t_c), p(s, t_c)) - 2\delta$.*

Proof. Let $t_1 = t_c^+(r, LC)$ and $t_c^+(s, LC)$. If $t_1 = t_2$, then letting $t^* = t_1$, t^* is a cycle start time and $\ell(r, t^*) = \ell(s, t^*) = A$. Otherwise, Lemma 2 proves this case. □

Lemmata 4 to 6 imply the next theorem.

Theorem 3. *Rendezvous(LC-atomic ASYNC, Non-Rigid, any) solves Rendezvous.*

4.4 ASYNC and Non-Rigid Movement($+\delta$)

Although it is still open whether asynchronous Rendezvous can be solved in Non-rigid with two colors of lights, if we assume Non-Rigid($+\delta$), we can solve Rendezvous modifying Rendezvous(ASYNC, Non-Rigid($+\delta$), A) and using the minimum moving value δ in it.

Let $dist_0 = dis(p(r, t_0), p(s, t_0))$ and let RendezvousWithDelta (Algorithm 2) begin with $\ell(r, t_0) = \ell(s, t_0) = A$. If $dist_0 > 2\delta$, both robots do not move until both colors of lights become B (**lines** 3-5) and there exists a cycle start time $t_1(> t_0)$ such that $\ell(r, t_1) = \ell(s, t_1) = B$. After $\ell(r, t_1) = \ell(s, t_1) = B$, the distance between r and s is reduced by $\delta/2$ without changing the colors of lights (**line** 4) and the distance falls in $[2\delta, \delta]$ and both colors of lights become A at a cycle starting time t_2. After $\ell(r, t_2) = \ell(s, t_2) = A$, we can use Rendezvous(ASYNC, Rigid, A) since $2\delta \geq dis(p(r, t_2), p(s, t_2)) \geq \delta$. Therefore, rendezvous is succeeded. Note that in Algorithm 2, the initial pair of colors of r and s is $(\ell(r, t_0), \ell(s, t_0)) = (A, A)$ and it is changed into $(\ell(r, t_1), \ell(s, t_1)) = (B, B)$ without changing the distance of r and s. And it is changed into $(\ell(r, t_2), \ell(s, t_2)) = (A, A)$ when the distance becomes between δ and 2δ. These *mode* changes are necessary and our algorithm does not work correctly if these mode changes are not incorporated in the algorithm. If the algorithm starts with $(\ell(r, t_0), \ell(s, t_0)) = (A, A)$ and does not use the mode change, when the distance becomes between δ and 2δ at some time t', the configuration becomes $(\ell(r, t'), \ell(s, t')) = (B, B)$ and therefore Rendezvous fails from the configuration.

Lemma 7. *If $dist_0 > 2\delta$, in any execution of RedezvousWithDelta(ASYNC, Non-Rigid(+δ), A),*

(1) there exists a cycle start time $t_1(> t_0)$ such that $\ell(r, t_1) = \ell(s, t_1) = B$ and $dis(p(r, t_1), p(s, t_1)) = dist_0$, and
(2) there exists a cycle start time $t_2(> t_1)$ such that $\ell(r, t_2) = \ell(s, t_2) = A$ and $2\delta \geq dis(p(r, t_2), p(s, t_2)) \geq \delta$.

Proof. (1) Without loss of generality, r performs the *Look* operation first and let t_{rL} be such a time. The color of r is changed from A to B at a time t_{rC}. Since $\ell(s, t_0) = A$, s performs a *Look* operation at a time $t_{sL}(\geq t_{rL})$ and changes its color from A to B at a time t_{sC}. Then, let $t_1 = \max(t_{rC}, t_{sC})$. If a *Comp* operation is performed immediately after t_1, the robot does not change its color of light, since the robot performs the preceding *Look* operation before t_1. Thus, t_1 becomes a cycle start time.

(2) Since t_1 is a cycle start time, we can assume that the algorithm starts at t_1 with $\ell(r, t_1) = \ell(s, t_1) = B$. The distance $dist_0$ is reduced by $\delta/2$ every cycle of each robot after t_1. Since $dist_0 > 2\delta$, $dist_0$ can be denoted by $x(\delta/2) + \varepsilon$, where $x \geq 4$ and $0 \leq \varepsilon < \delta/2$.

Let t be the time of the $(x-2)$-th *Look* operation among *Look* operations r and s performed after t_1, and without loss of generality, let r be the robot performing the $(x-2)$-th *Look* operation. Note that among $(x-3)$ *Move* operations between t_1 and t at least $\max(0, x-4)$ *Move* operations have been completed and at most one *Move* operation has not been completed yet.

Let $t' = t^-(s, Look)$.[4] We have two situations (Fig. 2). One is that (I) $(x-3)$ *Move* operations are completed until t. This case satisfies $2\delta \geq dis(p(r, t), p(s, t)) \geq \delta$. The other is that (II) the $(x-3)$-th *Move* operation s performs has not been completed at t.[5] The latter case is divided into two cases

[4] If s performed no *Look* operations after t_1, $t' = t_1$.
[5] This case includes $t < t'^+(s, Move_B)$.

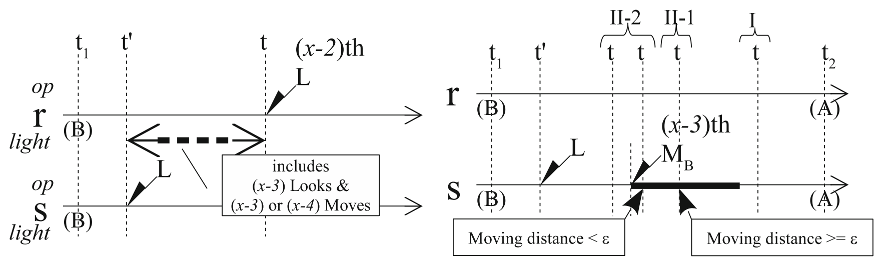


Fig. 2. Situations in the proof of Lemma 7

(II-1) $2\delta \geq dis(p(r,t), p(s,t)) \geq \delta$ and (II-2) $dis(p(r,t), p(s)) > 2\delta$, according to the time r performs the *Look* operation.

Case (I) and (II-1): Since $2\delta \geq dis(p(r,t), p(s,t)) \geq \delta$, r changes its color of light to A at $t^+(r, Comp)$. When s performs a *Look* operation at $t_{sL} = t^+(s, Look)$, s observes $2\delta \geq dis(p(r, t_{sL}), p(s, t_{sL})) \geq \delta$ and $\ell(s, t_{sL}) = B$, and changes its color of light to A at $t_{sC} = t^+(s, Comp)$. Letting $t_2 = \max(t_{rC}, t_{sC})$, t_2 becomes a cycle start time as follows.
When $t_2 = T_{rC}$, s changes its color of light to A at $t_{sC}(\leq t_{rC})$. Even if s performs a *Look* operation at t_L after t_{sC} before $t_2 = t_{rC}$, s does not change its color at $t_L^+(s, Comp)$ since $\ell(r, t_L) = B$. The case that $_2 = T_{sC}$ can be proven similarly.

Case (II-2): Since $dis(p(r,t), p(s,t)) > 2\delta$, r reduces the distance by $\delta/2$. Then, r performs the *Move* operation and subsequently performs the next *Look* operation at $t_{rL} = t^+(r, Look)$. After that it changes its color of light to A at $t_{rC} = t^+(r, Comp)$, since $\delta \leq dis(p(r, t_{rL}), p(s, t_{rL})) \leq 2\delta$. The next *Look* operation of s is performed after $t'^+(s, Move_E)$, and s changes its color of light to A at $t_{sC} = t^+(s, Comp)$. Robot s changes its color of light to A at t_{sC}. Letting $t_2 = \max(t_{rC}, t_{sC})$, we can prove that t_2 becomes a cycle start time similar to the former case. $\square$

The following two lemmata can be proven similarly to the proof of Theorem 2.

Lemma 8. *If $2\delta \geq dist_0 \geq \delta$, then RedezvousWithDelta(ASYNC, Non-Rigid $(+\delta)$, A) solves Rendezvous.*

Lemma 9. *If $dist_0 > \delta$, then RedezvousWithDelta(ASYNC, Non-Rigid $(+\delta)$, A) solves Rendezvous.*

Lemmata 7 to 9 imply the following theorem.

Theorem 4. *RedezvousWithDelta(ASYNC, Non-Rigid$(+\delta)$, A) solves Rendezvous.*

5 Concluding Remarks

We have shown that Rendezvous can be solved in ASYNC with the optimal number of colors of lights if Non-Rigid($+\delta$) movement is assumed. We have also shown that Rendezvous can be solved by an $\mathcal{L}$-algorithm in ASYNC and Non-Rigid with the optimal number of colors of lights if ASYNC is LC-atomic. Interesting open problems are whether can Rendezvous be solved in ASYNC and Non-Rigid with 2 colors or not,[6] and what condition of ASYNC can $\mathcal{L}$-algorithms be solved in Non-Rigid with 2 colors?

Acknowledgment. This work is supported in part by KAKENHI no. 17K00019 and 15K00011.

References

1. Agmon, N., Peleg, D.: Fault-tolerant gathering algorithms for autonomous mobile robots. SIAM J. Comput. **36**, 56–82 (2006)
2. Bouzid, Z., Das, S., Tixeuil, S.: Gathering of mobile robots tolerating multiple crash faults. In: Proceedings of 33rd ICDCS (2013)
3. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Distributed computing by mobile robots: gathering. SIAM J. Comput. **41**(4), 829–879 (2012)
4. Das, S., Flocchini, P., Prencipe, G., Santoro, N., Yamashita, M.: Autonomous mobile robots with lights. Theor. Comput. Sci. **609**, 171–184 (2016)
5. Défago, X., Potop-Butucaru, M.G., Clément, J., Messika, S., Raipin Parvédy, P.: Fault and byzantine tolerant self-stabilizing mobile robots gathering - feasibility study - CoRRabs/1602.05546 (2016)
6. Degener, B., Kempkes, B., Langner, T., Meyer auf der Heide, F., Pietrzyk, P., Wattenhofer, R.: A tight runtime bound for synchronous gathering of autonomous robots with limited visibility. In: Proceedings of 23rd ACM SPAA, pp. 139–148 (2011)
7. Dieudonné, Y., Petite, F.: Self-stabilizing gathering with strong multiplicity detection. Theor. Comput. Sci. **428**(13), 47–57 (2012)
8. Flocchini, P., Prencipe, G., Santoro, N.: Distributed Computing by Oblivious Mobile Robots, Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, San Rafael (2012)
9. Flocchini, P., Santoro, N., Viglietta, G., Yamashita, M.: Rendezvous with constant memory. Theor. Comput. Sci. **621**, 57–72 (2016)
10. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of asynchronous robots with limited visibility. Theor. Comput. Sci. **337**(1–3), 147–168 (2005)
11. Heriban, A., Défago, X., Tixeuil, S.: Optimally gathering two robots. Research Report HAL Id: hal-01575451. UPMC Sorbonne Universités, August 2017
12. Izumi, T., Bouzid, Z., Tixeuil, S., Wada, K.: Brief announcement: the BG-simulation for byzantine mobile robots. In: Peleg, D. (ed.) DISC 2011. LNCS, vol. 6950, pp. 330–331. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24100-0_32

[6] Very recently it has been solved affirmatively [11].

13. Izumi, T., Katayama, Y., Inuzuka, N., Wada, K.: Gathering autonomous mobile robots with dynamic compasses: an optimal result. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 298–312. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75142-7_24
14. Izumi, T., et al.: The gathering problem for two oblivious robots with unreliable compasses. SIAM J. Comput. **41**(1), 26–46 (2012)
15. Kamei, S., Lamani, A., Ooshita, F., Tixeuil, S.: Asynchronous mobile robot gathering from symmetric configurations without global multiplicity detection. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 150–161. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22212-2_14
16. Lin, J., Morse, A.S., Anderson, B.D.O.: The multi-agent rendezvous problem. Parts 1 and 2. SIAM J. Control Optim. **46**(6), 2096–2147 (2007)
17. Prencipe, G.: Impossibility of gathering by a set of autonomous mobile robots. Theor. Comput. Sci. **384**(2–3), 222–231 (2007)
18. Souissi, S., Défago, X., Yamashita, M.: Using eventually consistent compasses to gather memory-less mobile robots with limited visibility. ACM Trans. Auton. Adapt. Syst. **4**(1), 1–27 (2009)
19. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: formation of geometric patterns. SIAM J. Comput. **28**, 1347–1363 (1999)
20. Terai, S., Wada, K., Katayama, Y.: Gathering problems for autonomous mobile robots with lights. Technical report of Wada Labo., Hosei University, TRW-16-1 (2016)
21. Viglietta, G.: Rendezvous of two robots with visible bits. Technical report arXiv:1211.6039 (2012)

On the Advice Complexity of Online Edge- and Node-Deletion Problems

Peter Rossmanith(✉)

Department of Theoretical Computer Science,
RWTH Aachen University, Aachen, Germany
rossmani@informatik.rwth-aachen.de

Abstract. We consider a model of online graph modification problems where the input graph is read piecewise in an adversarial order and the algorithm has to modify the graph by deleting vertices or edges in order to keep it inside some fixed graph class Π. These deletions cannot be taken back later. We analyze the least number of advice bits that enable an online algorithm to perform the same number of deletions as an optimal offline algorithm. We consider only hereditary properties Π, for which optimal online algorithms exist and which can be characterized by a set of forbidden subgraphs $\mathcal{F}$. It is clear that, if $\mathcal{F}$ is finite, then the number of required advice bits is at most linear in the size of an optimal solution. We prove lower and upper bounds based on the structure of the graphs in $\mathcal{F}$, often determining the complexity exactly or nearly exactly. The techniques also work for infinite $\mathcal{F}$ and we can determine whether then the advice complexity can be bounded by a function in the optimal solution or not. For node-deletion problems we characterize the advice complexity exactly for all cases and for edge-deletion problems at least for the case of a single forbidden induced subgraph.

1 Introduction

Online algorithms do not see their whole input at once, but receive it in pieces and have to react each time they receive a new one. This is a very natural setting for situations where the future is unknown. A typical example are strategies for caches: If the cache is full and a page fault occurs, a page has to be discarded without knowing which pages will be accessed in the (near) future. One way to analyze online optimization problems is to investigate their *competitive ratio*, which tells us how much worse an online algorithm performs relative to an offline algorithm that has access to the whole input from the beginning [3].

A different way to analyze online problems is to look at their *advice complexity*. In this model the algorithm receives an advice in the form of a bit string from an all knowing oracle helping the algorithm to compute an optimal or approximate solution with a given competitive ratio [5]. Lower bounds on the advice complexity are very strong as they tell us that an online problem cannot be solved well even if we know something about the future.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 449–462, 2018.
https://doi.org/10.1007/978-3-319-98355-4_26

Online Graph Modification Problems

In an online graph problem a graph G is revealed as a sequence of growing induced subgraphs $G[v_1], G[v_1, v_2], \ldots, G[v_1, \ldots, v_n]$. Whenever a new vertex or edge is presented, the algorithm has to make a decision about the vertex or edge that cannot be taken back later. For example, if the problem consists of finding a minimum vertex cover, it has to be decided for every new vertex whether it is in- or excluded from the vertex cover. There are several results known about the advice complexity of such online graph problems. For example, Bianchi, Böckenhauer, Hromkovič, Krug, and Steffen show that linear advice is necessary for optimal or nearly-optimal $L(2,1)$-coloring on paths [2] and Steffen's whole PhD thesis is dedicated among others to various online graph problems in this setting [7]. He also considers the online vertex cover problem that can also be seen as a graph modification problem: Finding a vertex cover is equivalent to deleting vertices in order to get an empty graph. Komm et al. [6] study the advice complexity of a class of graph problems where a largest (resp. smallest) induced subgraph of the input graph with a certain property has to be chosen in an online fashion.

In this paper we consider a different model for online graph modification problems: For a Π-modification problem the graph is again presented as a sequence of growing induced subgraphs $G_1, G_2, \ldots$, but the algorithm does not have to do anything as long as $G_i \in \Pi$. If, however, $G_i \notin \Pi$, the algorithm has to perform modifications on G_i in order to make it a member of Π.

In the offline world graph modification problems are well studied. Already a long time ago Yannakakis proved that node-deletion problems are NP-hard for every non-trivial hereditary graph property and that many edge-deletion problems are NP-hard, too [8]. Cai analyzed the parameterized complexity of graph modification problems [4]. All variants are fixed-parameter tractable with respect to the solution size if the graph property can be characterized by a finite set of forbidden induced subgraphs. In this paper we are not concentrating on the running time, but as we will be considering advice given by an oracle it has to be noted that the oracle will usually be solving NP-hard problems when preparing the advice string while the online algorithm itself usually performs only simple calculations.

In the special case of node- or edge-deletion problems we can more specifically think of maintaining a set D of deleted nodes or edges. Whenever $G_i[V-D] \notin \Pi$ (resp. $G_i[E-D] \notin \Pi$), nodes or edges have to be inserted into D such that then $G_i[V-D] \in \Pi$ (resp. $G_i[E-D] \in \Pi$). Remember that this model is less restricted than the one in [2] because nodes or edges that have been presented in the past can still be deleted. It is nevertheless an online problem because decisions cannot be taken back and have to be made without knowing the future. The motivation for this model is that often problems have to be solved as soon as they arise, but not sooner.

Let us look at a simple introductory example: Cluster deletion. A cluster graph is a collection of disjoint cliques. Given a graph G the cluster deletion problem asks for a minimum set D of edges whose deletion turns G into a cluster graph. In our model we receive the graph G piecewise vertex by vertex. Each time we receive a new vertex that turns the graph into a non-cluster graph, we have to insert edges into D in order to ensure that $G_i[E(G_i) - D]$ is a cluster graph. It is clear that in the worst case we have no chance to compute an optimal D in this way. If we denote the size of an optimal solution by *opt*, it turns out that we can find an optimal solution of the same size online if we are given *opt* advice bits: Whenever we find an induced P_2 in our graph we have to delete at least one of its edges. We can read one advice bit to find out which one is the right one. As a graph is a cluster graph iff it does not contain P_2 as an induced subgraph the algorithm is correct.

It is also easy to see that this simple algorithm is optimal: An adversary can present k times a P_2 which is in the next step expanded into a P_3 on either side. To be optimal the algorithm has to choose the correct edge to delete each time of the k times. This makes 2^k possibilities and the algorithm cannot act identically for any pair of these possibilities. Hence, the algorithm needs at least k advice bits.

In this paper we consider similar problems and find ways to compute their exact advice complexity.

2 Preliminaries

We will use the usual notation for graphs, which will always be simple, undirected, and loop-free. We write $H \subseteq G$ if H is a subgraph of G. The size of a graph is its number of vertices and denoted by $|G|$. The number of edges is denoted by $||G||$. An edge between nodes x and y is called xy. If $\mathcal{F}$ is a set of graphs and none of them is an induced subgraph of G we say that G is $\mathcal{F}$-free. If X is a set of edges or vertices we denote the subgraph of G induced by X by $G[X]$. Sometimes we will just write $G - u$ to denote the graph we get from G by deleting the vertex u. A path and circle with k edges are denoted by P_k and C_k, respectively. We will use extensively a nonstandard operation where we glue two graphs together along an edge:

Definition 1. *Let G and H be graphs and $xy \in E(G)$, $uv \in E(H)$. We define two operations that glue G and H together along their edges xy and uv. First, $G\,{}_{xy}{\oplus}_{uv}\,H$ is the graph that we get by identifying u with x and v with y (and replacing the double edge by a single one). If G and H are not vertex-disjoint we replace them by disjoint copies first. We say that $G\,{}_{xy}{\oplus}_{uv}\,H$ consists of two parts, one is the induced subgraph by $V(G)$ and the other by $V(H)$. The two parts overlap in x and y. Second, $G_{xy}{\ominus}_{uv}$ is the same except that the edges xy and uv are removed completely.*

We abbreviate $G \oplus_{xy} G := G\,{}_{xy}{\oplus}_{xy}\,G$ and $G \ominus_{xy} G := G\,{}_{xy}{\ominus}_{xy}\,G$. Figure 1 shows an example.

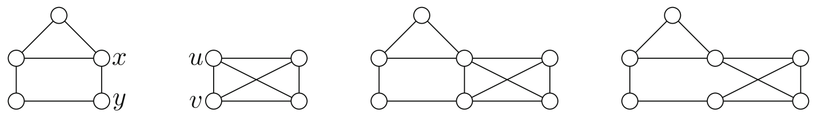


Fig. 1. From left to right: G, H, $G\ {}_{xy}{\oplus}_{uv}\ H$, $G\ {}_{xy}{\ominus}_{uv}\ H$

3 Some General Results

If we are facing a Π-graph modification problem for a graph class Π there are special cases we can consider for Π. If we know nothing about Π we can still show that $opt \log n$ advice bits are sufficient to solve the Π-edge-deletion problem optimally on a graph with n nodes.

Theorem 1. *Let Π be a hereditary graph property.*

(1) The Π-node-deletion problem can be solved optimally with $\lceil opt \log n \rceil$ advice bits.
(2) The Π-edge-deletion problem can be solved optimally with at most $2 opt \log n$ advice bits.

Proof

(1) Whenever the algorithm detects that the graph is not in Π, a node has to be deleted. There are at most n nodes to choose so the correct one can be encoded with $\log n$ bits and there are at most n^{opt} possibilities to choose opt nodes. Such a number can be encoded with $\lceil opt \log n \rceil$ bits.
In that way an optimal set of vertices is deleted. As Π is hereditary, all induced subgraphs seen by the algorithm in-between also belong to Π if the same optimal solution is deleted from them.
(2) If the algorithm detects that the graph is not in Π, one or more edges have to be deleted. There are at most $n^2/2$ edges in total to choose from. There are only $(n^2/2)^{opt}$ possibilities to choose a set of opt edges. We need only $\lceil \log((n^2/2)^{opt}) \rceil \le 2 opt \log n$ bits. □

While Theorem 1 gives us a simple upper bound on the advice complexity, it is often too pessimistic and we can find a better one. On the other hand, it will turn out that there are very hard edge-deletion problems where the bound of Theorem 1 is almost optimal.

One ugly, but sometimes necessary, property of the bound in Theorem 1 is that the number of advice bits can get arbitrarily large even if the size of the optimal solution is bounded by a constant. Let us look at some special cases, where this is not the case and the number of advice bits is bounded by a function of the solution size.

One important case are hereditary properties Π, i.e., properties that are closed under taking induced subgraphs. It is well known that such properties can be characterized by a set of forbidden induced subgraphs. If $\mathcal{F}$ is such a set we can always assume that it does not contain two graphs H_1 and H_2 such that H_1 is an induced subgraph of H_2 because H_2 would be redundant. Under this assumption $\mathcal{F}$ is determined completely by Π and can be finite or infinite and we say that $\mathcal{F}$ is *unordered*.

Moreover, it is also clear that if a hereditary class Π contains at least one graph then it also contains the null graph with no vertices (because that is an induced subgraph of any graph). There is a vast number of important hereditary graph properties, for example planar graphs, outerplanar graphs, forests, genus-bounded graphs, chordal graph, bipartite graphs, cluster graphs, line graphs, etc.

Hereditary graph properties are *exactly* those properties that can be solved optimally in this model if "optimal" means that the offline solution is not larger than the best online solution that has to modify only one graph G (while the online algorithm has to modify a sequence of induced subgraphs leading to G).

Theorem 2. *The online Π-edge and Π-node-deletion problems can be solved optimally with respect to the smallest offline solution if and only if Π is hereditary, even if arbitrarily many advice bits can be used.*

Proof. Theorem 1 already shows that an optimal solution can be found for hereditary properties.

Let now Π be a graph property that is not hereditary. Then there are graphs G_1 and G_2 such that (1) $G_1 \notin \Pi$, (2) $G_2 \in \Pi$, and (3) G_1 is an induced subgraph of G_2.

The adversary can present first G_1 and later G_2. Any correct algorithm has to delete something from G_1, but the optimal offline solution is to delete nothing. □

Because of Theorem 2 we will look only at hereditary graph properties in this paper. It should be noted, however, that a sensible treatment of non hereditary graph properties is possible if the definition of online optimality is adjusted in the right way.

4 $\mathcal{F}$-Node Deletion Problems

Let us first look at node-deletion problems because they are much easier than edge-deletion problems in the online setting. The general ideas on the lower bounds are the same.

Definition 2. *Let $\mathcal{F}$ be an unordered set of graphs. The online $\mathcal{F}$-node-deletion problem is an online graph modification problem where the algorithm chooses nodes that have to be deleted in order for the input graph to stay $\mathcal{F}$-free. Once a node is chosen for deletion this decision cannot be taken back.*

The following theorem states that the number of necessary advice bits can easily be computed by looking at $\mathcal{F}$. Essentially it depends on the size of the largest graph in $\mathcal{F}$. We will see later that in edge-deletion problems something similar, but more complicated, holds as well. The proofs will also be more involved.

Theorem 3. *Let $\mathcal{F}$ be a set of connected graphs. The online $\mathcal{F}$-node-deletion problem can be solved optimally with $\lceil opt \log s \rceil$ many advice bits, where s is the size of the largest graph in $\mathcal{F}$. This bound is tight if $s < \infty$.*

If $s = \infty$, then no algorithm can be optimal with $f(opt)$ advice bits for any function f.

Proof. Let H be the largest graph in $\mathcal{F}$ if $\mathcal{F}$ is finite and u be an arbitrary node in H. If we glue two disjoint copies of H together at u by identifying the respective nodes, the resulting graph H_u contains H as an induced subgraph and there is exactly one node (i.e., u) that we can delete in order to make H_u H-free. However, when deleting u from H_u the remaining graph becomes disconnected and both components are proper induced subgraphs of H. Hence, the components are $\mathcal{F}$-free and therefore $H_u - u$ is also $\mathcal{F}$-free.

An adversary can simply present first H and then adding nodes to get H_u for $u \in V(H)$. In this way the adversary has $|H|$ possibilities to continue and to be optimal when seeing only H the algorithm has to delete the correct u. By repeating this k times there are $|H|^k$ different possibilities and they have all to be distinguished. Hence we need at least $\log(|H|^k) = k \log(|H|)$ advice bits. It is easy to see that $\lceil k \log s \rceil$ bits are also sufficient if $k = opt$ because the algorithm has to pick the right edge of H when finding $H \in \mathcal{F}$ as an induced subgraph. □

5 $\mathcal{F}$-Edge Deletion Problems

Let $\mathcal{F}$ be a collection of graphs pairwise independent with regard to being induced subgraphs. We try to emulate results for node-deletion problems. For those we glued two graphs together at a node. Now we have to glue graphs together at an edge. A crucial difference between these two ways of glueing is that basically "nothing can go wrong" when glueing graphs at a node and then deleting the node because the graph becomes disconnected and the parts are induced subgraphs of the original graphs. This is no longer true when glueing along edges. The next definition tries to capture the idea of "nothing can go wrong" by labeling edges as *critical* if we can use them without producing a graph that contains graphs from $\mathcal{F}$.

Definition 3. *Let $\mathcal{F}$ be a collection of graphs and $H \in \mathcal{F}$. We classify all edges in H as* critical *or* non-critical. *An edge $xy \in E(H)$ is* critical *iff there is an $H' \in \mathcal{F}$ and an edge $uv \in E(H')$ such that $H\ {}_{xy}{\ominus}_{uv}\ H'$ does not contain any graph from $\mathcal{F}$ as an induced subgraph.*

Let $\#crit_{\mathcal{F}}(H)$ denote the number of critical edges in H and $\#crit(\mathcal{F}) = \max\{\, \#crit_{\mathcal{F}}(H) \mid H \in \mathcal{F}\,\}$.

If we would define "critical nodes" in a similar way for node-deletion problems it would turn out that *all* nodes are critical. In the following we will establish that the number of critical edges plays a crucial role in the advice complexity of online edge-deletion problems and ways how to compute the number of critical edges for special families $\mathcal{F}$. Please note first that it is quite easy to compute

$\#crit(\mathcal{F})$ if given a finite $\mathcal{F}$. A simple algorithm can achieve this by a polynomial number of subgraph isomorphism tests, which are of course by themselves NP-complete. The graphs in typical families $\mathcal{F}$ are usually small, so long running times are not a practical issue here.

Let us look at a very simple example. Let $\mathcal{F} = \{P_2, C_3\}$. Due to symmetry we have to look only at three different glueing operations: P_2 to P_2, C_3 to C_3, and P_2 to C_3. Whenever C_3 is involved, the resulting graph contains P_2 as an induced subgraph. For example, glueing C_3 to itself results in $C_4 \simeq C_3 \ominus_e C_3$. This means that the edges in C_3 are not critical. Glueing P_2 to itself can be done in two ways. The first result are two disjoint edges and the second is again P_2. Hence, the edges in P_2 are critical. In total, $\#crit(\mathcal{F}) = 2$.

Lemma 1. *Let G be a two-connected graph and $e \in E(G)$. Then G is two-connected iff $G \ominus_e G$ is two-connected.*

Proof. A connected graph is two-connected iff it has no cut-vertex. Let $e = xy$. Assume first that G has a cut-vertex. If it is x or y then clearly $G \ominus_e G$ is not even connected. Otherwise both parts of $G \ominus_e G$ have the corresponding vertex as a cut-vertex. The other direction of the proof is similar. $\square$

Lemma 2. *Let G be a graph that contains vertices x, y, u, v and these four conditions hold:*

1. *x and y are connected by an edge.*
2. *There are two vertex-disjoint paths from u to x and from u to y.*
3. *There are two vertex-disjoint paths from v to x and from v to y.*
4. *The edge xy is not on any of those four paths.*

Then u and v are two-connected in G.

Proof. The vertices u, x, and y are on a cycle and therefore in the same two-connected component. The same holds for v, x, and y. $\square$

The next lemma is technically the most complicated one in this paper. What it states about all two-connected graphs is also true for many other graphs and it can be checked easily for a concrete graph G. It opens a path to proving lower bounds on the advice complexity of edge-deletion problems in a way similar to how Theorem 3 works for node-deletion problems.

Lemma 3. *Let G be a two-connected graph and $e = xy \in E(G)$ one of its edges. Then $G \ominus_e G$ does not contain a subgraph that is isomorphic to G.*

Proof. Let us assume the contrary and that G is a minimal counterexample with respect to taking subgraphs. So we assume that indeed there is a $H \subseteq G \ominus_e G$ and $H \simeq G$. Because of its number of edges, H contains edges in both parts of $G \ominus_e G$ as one part has only $||G|| - 1$ edges. Hence, H contains x_1 or y_1. Since H (being isomorphic to G) is two-connected it must contain both x_1 and y_1; otherwise H would contain a cut-vertex ($\{x_1, y_1\}$ is a separator in $G \ominus_e G$).

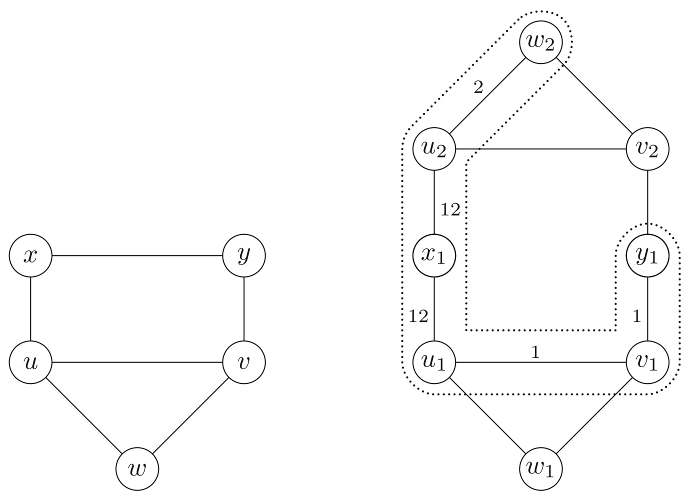


Fig. 2. A graph G is depicted left. The corresponding graph $G \ominus_{xy} G$ is on the right (here $x_1 = x_2$ and $y_1 = y_2$). Note that the edge corresponding to xy is missing in the right graph. An induced subgraph H is indicated by the dotted outline. (Please note that H is not isomorphic to G, which would be impossible by the very lemma we are about to prove.)

In the following we establish some notation. If $u \in V(G)$, then let u_1 and u_2 be the respective copies of u in the upper and lower part of $G \ominus_e G$. In particular $x_1 = x_2$, $y_1 = y_2$ and $u_1 \neq u_2$ if $u \notin \{x, y\}$.

If $u_1v_1 \in E(H)$, then we say that

- u_1v_1 is a 12-edge if $u_2v_2 \in E(H)$.
- u_1v_1 is a 1-edge if $u_2v_2 \notin E(H)$.

If $u_2v_2 \in E(H)$, then we say that

- u_2v_2 is a 12-edge if $u_1v_1 \in E(H)$.
- u_2v_2 is a 2-edge if $u_1v_1 \notin E(H)$.

In Fig. 2 the graph H has five edges. The 12-edges are x_1u_1 and x_2u_2, u_2w_2 is a 2-edge, and u_1v_1, v_1y_1 are 1-edges.

From H we construct a new graph H' (which will be a subgraph of G) as follows: The vertices of H' will be a subset of $V(G)$. Let $V(H') = \{\, u \in V(G) \mid u_1 \in V(H) \text{ or } u_2 \in V(H) \,\}$. The graph H' contains the edge uv iff $u_1v_1 \in E(H)$ or $u_2v_2 \in E(H)$ and additionally the edge xy. Then H' is clearly a subgraph of G.

What happens if we consider $H' \ominus_{xy} H'$? It must be a supergraph of H, which is isomorphic to G, which again contains H' as a subgraph:

$$H' \ominus_{xy} H' \supseteq H \simeq G \supseteq H' \tag{1}$$

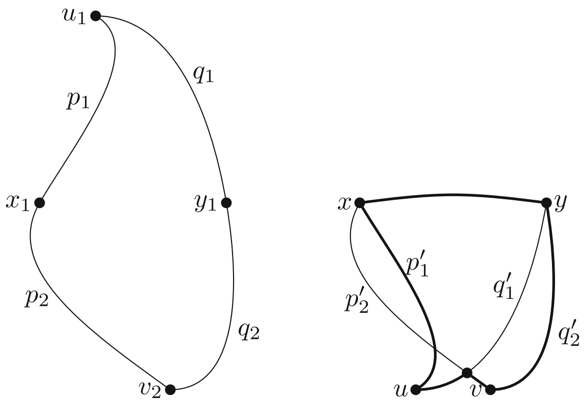


Fig. 3. The vertices u_1 and v_2 are two-connected in H (left). The resulting vertices u and v in H' are then also two-connected (right).

As H contains x_1 and y_1, H' contains x and y by definition. As H is two-connected there must be at least two vertex disjoint paths between any pair of vertices in H. Let u_1 and v_2 be such a pair. One of the paths between them has to go through x_1 and the other through y_1. Both paths start with a subpath consisting of only 1-edges and then ending with a subpath of 2-edges. Let us call these subpaths p_1, q_1, p_2, q_2 where p_1 connects u_1 with x_1 and p_2 connects x_1 with v_2. In H' there are isomorphic copies of these paths that need no longer be disjoint. There are, however, two disjoint paths p'_1 and q'_1 connecting u to x and y, and two other disjoint paths p'_2 and q'_2 connecting to v to x and y (see Fig. 3). By Lemma 2 then u and v are also two-connected in H'. If u and v do not originate from u_1 and v_2 in H, but, say, from u_1 and v_1, the situation is simpler because then all paths are automatically disjoint. Altogether, this means that H' is two-connected.

We have established the following three facts:

- $e \in E(H')$ (by construction of H'),
- H' is two-connected (previous paragraph),
- $H' \ominus_e H'$ contains a subgraph that is isomorphic to H', see (1).

If we look at the preconditions of Lemma 3 we see that H' is another counterexample. Moreover, $H' \subseteq G$. We have assumed that G is a minimal counterexample, so it cannot contain another smaller subgraph that is also a counterexample. Hence, $H' = G$.

As $H \simeq G$, the graphs H and H' must have the same number of vertices. This, however, cannot be the case if there is at least one 12-edge in H. One endpoint of a 12-edge has to be outside $\{x_1, y_1\}$ because $x_1y_1 \notin E(H)$. Let us assume x_1u_1 is such a 12-edge. Then there exists also the 12-edge $x_2u_2 = x_1u_2$. The graph H' contains the vertex u iff H contains u_1 or u_2. As H contains both u_1 and u_2, the number of vertices in H is larger than that in H'.

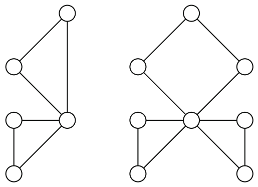


Fig. 4. Lemma 3 does not hold for single-connected graphs.

As this is impossible, the only remaining possibility is that there is not even a single 12-edge in H. The number of edges in H is $h_1 + h_2 + h_{12}$ if we denote the number of 1-, 2-, and 12-edges by h_1, h_2, and h_{12}. Then the number of edges in $G = H'$ is $h_1 + h_2 + h_{12}/2 + 1$ (note that h_{12} is always an even number). If indeed G and H are isomorphic, these counts must coincide, which is impossible if $h_{12} = 0$. This contradiction shows that our first assumption must have been wrong and the assumption was simply that Lemma 3 is wrong. □

The condition that G is two-connected Lemma 3 is necessary. Figure 4 shows that glueing an arbitrary connected graph to itself on a vertex yields a graph that is connected, but not two-connected and is a counterexample to the statement in Lemma 3. The next lemma shows that if we use $G\ {}_{yx}\ominus_{xy}\ G$ instead, a similar statement holds for graphs that are just connected.

Lemma 4. *Let G be a connected graph and $e = xy \in E(G)$ one of its edges. Then $G\ {}_{yx}\ominus_{xy}\ G$ does not contain a subgraph that is isomorphic to G.*

Proof. The outline of the proof is similar to the proof of Lemma 3. First we assume that there is a connected graph G and an edge $xy \in G(H)$ such that $H \subseteq G_{yx}\ominus_{xy}$ and H is isomorphic to G. Moreover, we assume without loss of generality that G is a minimal counterexample with respect to taking subgraphs. From this assumption we will derive a contraction.

We define H' analogous to H' in Lemma 3 and the only difference is that now $x_1 = y_2$ and $y_1 = x_2$. This time H' "automatically" contains both x and y: Since H has too many edges to fit in one part of $G_{yx}\ominus_{xy}$ it has to use x_1 or y_1. In either case H' contains both x and y.

As before we get a contradiction if there is no 12-edge:

$$||G|| = ||H||,\ ||H|| = h_1 + h_2 + h_{12},\ h_1 + h_2 \leq ||G|| - 1.$$

On the other hand, if there is at least one pair of 12-edges, say u_1v_1 and u_2v_2, then H' must be connected: G is connected therefore ${}_{yx}\oplus_{xy}$ is also connected. If ${}_{yx}\ominus_{xy}$ is disconnected then only the missing edge x_1y_1 can be responsible. Then H would consist of the connected components, one in the upper part and connected to x_1, the other in the lower part and connected to $y_1 = x_2$ (or the other way around). The corresponding parts in H' share the node u and therefore H' is connected after all. As above this shows that H' is another counterexample. Because of the assumption that G is minimal and $H' \subseteq G$ we get $H' = G$. On the other hand $h_{12} > 1$ implies that H' has fewer edges than G, a contradiction. □

Please note that Lemmata 3 and 4 have several consequences. First, it means that $\#crit(\{G\}) = ||G||$. It is also the key to the next theorem, which gives tight bounds for the $\mathcal{F}$-edge-deletion problem for the case that $\mathcal{F}$ consists of one connected graph.

Theorem 4. *Let H be a connected graph and G be an arbitrary graph. If opt is the minimal number of edges you have to delete from G to make it H-free, then every online algorithm requires $\lceil opt \log(||H||)\rceil$ advice bits to solve the $\{H\}$-edge-deletion problem optimally on input G. There is a deterministic algorithm that matches this bound.*

Proof. The adversary prepares a set $\mathcal{I}$ of $||H||^k$ different instances. We will show that every fixed algorithm (deterministic and using no advice) can solve at most one instance in $\mathcal{I}$ correctly.

Let $\{e_1, \dots, e_m\} = E(H)$ and $H_i = H\ {}_{xy}{\oplus}_{yx}\ H$ where $xy = e_i$. Each instance is a graph that has k components that are presented one after the other by the adversary. Each component is some H_i.

It is clear that making H_i H-free requires removing at least one edge and can be accomplished by removing exactly edge e_i by Lemma 3: Removing e_i from $H_{xy}{\oplus}_{yx}$ leaves $H\ {}_{xy}{\ominus}_{yx}\ H$, which does not contain H as an (induced) subgraph. On the other hand, removing any other edge will result in a graph that still contains H as an induced subgraph.

If there are less than $\lceil k \log(||H||)\rceil$ advice bits, the algorithm will react in the same way for two different instances. At least one of them will then be solved in a non-optimal way.

To show a matching upper bound is relatively easy. Whenever some H is found as an induced subgraph, one edge of it belongs to some optimal solution known to the oracle. This edge can be communicated by a number between 1 and $m = ||H||$. As only k times an edge is selected, k such numbers suffice and they can all be encoded as a single number between 1 and $||H||^k$. $\square$

The following theorem generalizes the lower bound from Theorem 4 to arbitrary sets $\mathcal{F}$. Its small disadvantage is that it uses $\#crit(\mathcal{F})$, which has to be established for each $\mathcal{F}$ individually, which is a lot of work by hand, but easy with the help of a computer. A greater disadvantage is the missing matching upper bound, which we discuss after stating and proving the theorem.

Theorem 5. *Let $\mathcal{F}$ be an unordered set of graphs. Then solving the online $\mathcal{F}$-edge-deletion problem requires at least $\lceil opt \log(\#crit(\mathcal{F}))\rceil$ advice bits if opt is the optimal solution size.*

Proof. The proof is very similar to the proof of Theorem 4. The adversary chooses $H \in \mathcal{F}$ such that $\#crit_{\mathcal{F}}(H) = \#crit(\mathcal{F})$. Let C be the critical edges in H. Now again $|C|^k$ different instances are generated. For $xy \in C$ let $H_{xy} \in \mathcal{F}$ be another graph and $uv \in H_{xy}$ be an edge such that $H\ {}_{xy}{\ominus}_{uv}\ H_{xy}$ is $\mathcal{F}$-free. Such a graph and edge exist by the definition of a critical edge. The instance presented by the adversary is $H\ {}_{xy}{\oplus}_{uv}\ H_{xy}$.

An optimal online algorithm has to delete exactly the edge xy when confronted with $H\ {}_{xy}\!\oplus_{uv} H_{xy}$ as the deletion of any other edge either leaves H or H_{xy} as an induced subgraph. Deleting, however, xy turns $H\ {}_{xy}\!\oplus_{uv} H_{xy}$ into $H\ {}_{xy}\!\ominus_{uv} H_{xy}$, which is $\mathcal{F}$-free.

Again, there are $|C|^k$ different instances by repeating such choice k times. □

It would be nice to have a matching upper bound for Theorem 5, too, but it is easy to see that Theorem 5 is not always optimal. For example, consider claw- and diamond-free graphs. The edges in the diamond are not critical and Theorem 5 gives us only $\mathit{opt} \log(3)$ as a lower bound on the advice complexity. However, you can find five different induced supergraphs of the diamond that all require a different edge of the diamond to be removed in order to make it claw- and diamond-free.

Nevertheless, the following algorithm provides a matching upper bound in many, but not all, cases. The algorithm actually consists of the online algorithm and the behavior of the oracle providing the advice string.

The online algorithm proceeds as follows: It waits until the graph is no longer $\mathcal{F}$-free and then identifies one graph $H \in \mathcal{F}$ that is present as an induced subgraph. If there is no $H' \subseteq H$ such that $H' \in \mathcal{F}$ and H' has a critical edge, then it deletes an arbitrary edge from H. Otherwise it asks the oracle to name one critical edge in an appropriate subgraph H' and deletes it. It is easy to see that only $\lceil \log(\#\mathit{crit}(\mathcal{F})^{\mathit{opt}}) \rceil$ advice bits are used.

The oracle provides the following advice when the algorithm asks to identify a critical edge. The oracle identifies the edge in H that would be deleted by an optimal offline algorithm. If e is critical, it provides its number as advice. Otherwise it provides an arbitrary number.

It turns out that this algorithm is correct even for some extreme cases. For example, if $\mathcal{F}$ consists of all cycles, then the problem is to delete the minimum number of edges to make the graph acyclic. This is a very simple problem that can be solved greedily without any advice. It turns out that here all edges are non-critical. The online algorithm would just delete an arbitrary edge in a cycle it finds.

Another example is $\mathcal{F} = \{P_2, C_3\}$. Remember that P_2 contains critical and C_3 non-critical edges. Hence, as $P_2 \subseteq C_3$ the algorithm would ask the oracle which of two given edges in a triangle has to be deleted. With correct advice this yields an optimal solution and the algorithm matches the bound of Theorem 5.

Unfortunately, the algorithm does not work in all cases and indeed such an algorithm cannot exist because the lower bound in Theorem 5 is not optimal. Figure 5 shows a family $\mathcal{F}$ with $\#\mathit{crit}(\mathcal{F}) = 3$. The first graph has no critical edges at all. An adversary can, however, present the first graph and any algorithm has to delete one edge. It is not hard to see that the adversary can force the algorithm to make a non-optimal decision if the algorithm is not able to choose the right petal from which to delete the edge.

An important open question left is to find a construction for an optimal algorithm for every family $\mathcal{F}$.

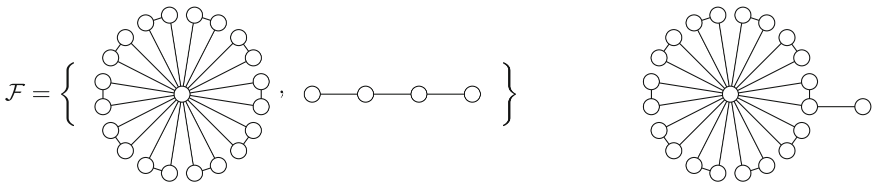


Fig. 5. A counterexample to the optimality of Theorem 5. The adversary presents the large graph in $\mathcal{F}$. The algorithm has to delete an edge e. Then the adversary adds another vertex as shown on the right side. The optimal solution is to delete two edges. The algorithm is not optimal if it did not choose e from the correct petal out of ten. Repeating this scheme for k rounds leads to $opt = 2k$ and at least $\log(10^k) = \log(10)/2 \cdot opt > 1.66 opt$ advice bits to achieve optimality. Theorem 5 provides a lower bound of only $\lceil opt \log 3 \rceil \leq \lceil 1.59 opt \rceil$.

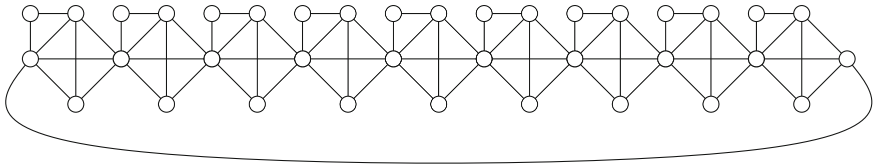


Fig. 6. Gadget for DH-edge-deletion.

The last example we consider shows a problem whose advice complexity is not bounded by a function of *opt*.

A graph is *distance hereditary* if the distance between two vertices does not change if we delete other vertices as long as they stay connected [1]. Surprisingly many graph classes are distance hereditary although it seems to be a severe restriction (which it is!). Among those graph classes are, e.g., ptolemaic graphs and cluster graphs. Let us call the corresponding problem the DH-edge-deletion problem. Its complexity is at least very close to the trivial upper bound from Theorem 1 and it is not bounded by function of the size of the optimal solution.

Theorem 6. *DH-edge-deletion requires at least* $opt(\log(n-1) - 2)$ *advice bits.*

Proof. We construct k graphs of size $4t + 1$ depicted in Fig. 6 for $t = 9$. The adversary presents first the cycle that consists of the long lower edge and the path going through the middle of the gadget. When the cycle is completed the algorithm is for the first time confronted with a graph that is not distance hereditary. Hence, the algorithm has to delete an edge. The only optimal choice is to delete the long edge making the graph distance-hereditary. Because of the rotation symmetry the adversary can construct t such gadgets and for each of them a different edge has to be deleted. Since there are k such gadgets arriving one after the other, the adversary can choose between t^k instances in total. The optimal solution deletes only $k = opt$ edges. The online algorithm requires therefore $k \log t$ advice bits. The graph has size $n = 4t + 1$, so $t = (n-1)/4$, making the number of advice bits at least $k \log((n-1)/4) = k(\log(n-1) - 2)$. $\square$

Acknowledgement. I would like to thank Walter Unger, Janosch Fuchs, Ling-Ju Hung, and Li-Hsuan Chen for stimulating discussions about the proof of Lemma 3 and an anonymous referee for insightful feedback.

References

1. Bandelt, H.-J., Mulder, H.M.: Distance-hereditary graphs. J. Comb. Theory, Ser. B **41**(2), 182–208 (1986)
2. Bianchi, M.P., Böckenhauer, H.-J., Hromkovič, J., Krug, S., Steffen, B.: On the advice complexity of the online $L(2,1)$-coloring problem on paths and cycles. Theor. Comput. Sci. **554**, 22–39 (2014)
3. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
4. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Inf. Process. Lett. **58**, 171–176 (1996)
5. Komm, D.: An Introduction to Online Computation - Determinism, Randomization, Advice. Texts in Theoretical Computer Science. An EATCS Series. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42749-2
6. Komm, D., Královič, R., Královič, R., Kudahl, Ch.: Advice complexity of the online induced subgraph problem. In: Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2016), Leibniz International Proceedings in Informatics, vol. 58, pp. 59:1–59:13 (2016)
7. Steffen, B.: Advice complexity of online graph problems. Ph.D. thesis (2014)
8. Yannakakis, M.: Node- and edge-deletion NP-complete problems. In: Proceedings of the 10th ACM Symposium on Theory of Computing, pp. 253–264. ACM (1978)

Second Thoughts on the Second Law

Stefan Wolf[1,2(✉)]

[1] Faculty of Informatics, Università della Svizzera italiana, 6900 Lugano, Switzerland
wolfs@usi.ch
[2] Facoltà indipendente di Gandria, 6978 Gandria, Switzerland

Abstract. We speculate whether the *second law of thermodynamics* has more to do with Turing machines than steam pipes. It states the *logical reversibility* of reality as a computation, i.e., the fact that no information is forgotten: nature computes with Toffoli, not NAND gates. On the way there, we correct Landauer's erasure principle by directly linking it to *lossless data compression*, and we further develop that to a lower bound on the energy consumption and heat dissipation of a general computation.

1 Prologue

A few years ago, our group had the great pleasure and privilege to receive *Juraj Hromkovič* as a guest in *Ticino*. He had announced a discourse on the topic "*What is information?*" [16]. It was a fascinating lecture in which Juraj was advocating to view information as *complexity*. This has been inspiring for me, and it is probably not a coïncidence that soon after that I realized that the use of *Kolmogorov complexity* [18][1] instead of probability distributions in the context of the fascinating but strange "non-local" correlations that quantum physics comes with — and that had been my main object of study for over a decade already at that time — offers a significant conceptual advantage: *Non-counterfactuality*, i.e., no necessity to talk about the outcomes of unperformed experiments.

John Stewart Bell showed in 1964 [6] that quantum theory predicts correlations between measurement outcomes that are too strong to be explained by shared classical information. This is as if identical twins did not only *look* alike — such correlations can easily be explained by their identical DNA sequences and do not confront us with a "metaphysical" problem — but also *behaved* in ways so strongly correlated that genetic explanations fail. Bell's result was a late reply to an attack to quantum theory by *Einstein, Podolsky, and Rosen* ("EPR") in 1935 [11] who remarked that if the outcomes of measurements on quantum systems are correlated, then they cannot be spontaneously random as predicted

[1] The Kolmogorov complexity $K_{\mathcal{U}}(x)$ of a string x with respect to a universal Turing machine $\mathcal{U}$ is the length of the shortest program for $\mathcal{U}$ that outputs x. The *conditional* complexity of x given y, $K_{\mathcal{U}}(x|y)$, is the length of the shortest program outputting x upon input y. The quantities depend on the choice of the specific machine $\mathcal{U}$ only through an additive constant.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 463–476, 2018.
https://doi.org/10.1007/978-3-319-98355-4_27

by the theory, but they must already have been determined beforehand, at the occasion of the creation of the *entangled pair.* To stay with our analogy: If twins look the same, their looks must be determined by their genes; if they also behave the same, then so must their behavior. This is a natural thought — but it is insufficient, and to realize this is Bell's breakthrough: *"Genetic" explanations are too weak for quantum correlations.* But then, where *do* they come from? What mechanism establishes them?

The basis of *EPR*'s argument has later been called "Reichenbach's principle" [22]: A correlation in a causal structure is established either by a *common cause* in the common past or a *direct influence* from one event to the other. Bell's celebrated argument rules out the first possibility if that common cause is to be a piece of classical information. *Influence* stories for establishing quantum correlations cannot be ruled out entirely, but they require the speed of that influence to be infinite, and they are unnatural in other respects expressing the fact that explaining a *non*-signaling phenomenon with a *signaling* mechanism is shooting sparrows with cannons. Eventually, *the fundamentality of the causal structure is in question*[2] — the only assumption for Reichenbach's principle. So motivated, models of *relaxed causality* have been studied [21] and disclosed an unexpectedly rich world [3] between fixed causal orders and logical inconsistency (*à la* "grandfather paradox" — you travel to the past and kill your own grandfather — etc.), much like *Bell world* between locality and signaling.

The fall of rigid causality comes with a further victim, *randomness*: Common physical definitions of freeness of randomness [9] are based on that very structure and they fall with it. One way out is to consider freeness of randomness as fundamental and causality as emerging from it, via: *"What is correlated with a perfectly free bit must be in its future."* For single bits, this is the best we can hope for. For bit *strings*, however, there can be an *intrinsic randomness notion* depending only on the data itself but not (otherwise) on the process leading up to them. In our search for such a non-contextual view, we land in a field traditionally related to probability distributions and ensembles but that knows a "non-counterfactual" viewpoint as well: *Thermodynamics.*

2 From the Steam Pipe ...

The story[3] of the *second law of thermodynamics* starts with *Sadi Carnot* (1796–1832) and his study of heat engines such as *James Watt*'s steam pipe. To conclude that the law manifests itself only for such engines and their circular processes means to underestimate a general *combinatorial fact.* The second law was discovered through steam engines because they first permitted a precise unclouded

[2] The idea of dropping causality may sound radical but is not new; see *Bertrand Russell, 1913* [23]: "The law of causality [...] is a relic of a bygone age, surviving, like the monarchy, only because it is erroneously supposed to do no harm."

[3] Most of the following historical facts are drawn from the article "Bluff your way in the second law of thermodynamics" by *Jos Uffink*, 2001 [30].

view on it — but the law is restricted to combustion engines as little as Jupiter's moons depend on telescopes.[4]

Carnot discovered that the maximal efficiency of a heat engine between two heat baths depended solely on the two temperatures involved. This was his only publication; it appeared when he was 28 and was entitled: *"Réflexions sur la puissance motrice du feu et sur les machines propres à développer cette puissance."*

Rudolf Clausius' (1822–1888) version of the second law reads: *"Es kann nie Wärme aus einem kälteren in einen wärmeren Körper übergehen, ohne dass eine andere damit zusammenhängende Änderung eintritt."* — "No process can transport heat from cold to hot and do no further change."

William Thomson (Lord Kelvin) (1824–1907) formulated his own version of the second law and then concluded that the law may have consequences more severe than what is obvious at first sight: *"Restoration of mechanical energy without dissipation [...] is impossible. Within a finite period of time past, the earth must have been, within a finite time, the earth must again be unfit for the habitation of man."*

Also for Clausius, it was only a single thinking step from his version of the law to conclude that all temperature differences in the entire universe will vanish — the *"Wärmetod"* — and that then, no change will happen anymore. He speaks of a general tendency of nature for change into a specific direction: *"Wendet man dieses auf das Weltall im Ganzen an, so gelangt man zu einer eigentümlichen Schlussfolgerung, auf welche zuerst W. Thomson aufmerksam machte, nachdem er sich meiner Auffassung des zweiten Hauptsatzes angeschlossen hatte.*[5] *Wenn [...] im Weltall die Wärme stets das Bestreben zeigt, [...] dass [...] Temperaturdifferenzen ausgeglichen werden, so muss es sich mehr und mehr dem Zustand annähern, wo [...] keine Temperaturdifferenzen mehr existieren."* — "When this is applied to the universe as a whole, one gets to the strange conclusion that already W. Thomson had pointed out after having taken my view of the second law. If heat always tends towards reducing temperature differences, then the universe will approximate more and more the state in which no temperature differences exist anymore."[6]

Ludwig Boltzmann (1844–1906) brought our understanding of the second law closer to combinatorics and probability theory: The second law was for him the

[4] A symptom of the law's generality is that it has advanced to becoming pop culture in the meantime: see, e.g., Allen, W., *Husbands and Wives* (1992) — The protagonist Sally is explaining why her marriage did not work out. First she does not know, then she realizes: "It's the second law of thermodynamics: sooner or later, everything turns to shit. That's my phrasing, not the *Encyclopedia Britannica*." The second law is less popular than Einstein's elegant relativity or Bennett et al.'s sexy teleportation since it does not have any glamour, fascination, or promise attached to it: Quite on the contrary, it stands for facts we usually deny or try to avoid.

[5] Roughly: "He had an interesting view on the second law after having adopted mine."

[6] It seems that his faithful pupil *Max Planck* believed that claim was untenable — Clausius finally erased, last-minute, all remarks concerning "the entropy of the universe as a whole" from his collected works — by hand.

expression of the fact that it is more likely to end up in a *large* set of possible states than in a small one: The higher the number of *particular* situations (microstates) which belong to a *general* one (macrostate), the more likely it is to be in that general situation. In other words, time evolution does not decrease a closed system's *entropy*, which is proportional to the logarithm of the number of corresponding microstates: Things do not get "more special" with time.[7]

Boltzmann's notions of *macrostate* and *entropy* are subjective, and it is not obvious how to define them in general, e.g., for non-equilibria. We propose instead a version of the second law that is broader and more precise at the same time, avoiding probabilities and ensembles. Crucial steps in that direction were made by *Wojciech Zurek* [32]. We follow *Rolf Landauer* [19] whose choice of viewpoint about thermodynamics can be compared with *Ernst Specker*'s [24] about quantum theory: *Logic.*

3 ...to the Turing Machine

Landauer [19] investigated the thermodynamic price of logical operations. He was correcting a belief by *John von Neumann* that every bit operation required free energy $kT \ln 2$:[8] According to Landauer — as confirmed by *Fredkin and Toffoli's* "ballistic computer" [14] —, that price is unavoidable only for *logically irreversible* operations such as the AND or the OR. On the positive side, it has been observed [7] that every function, bijective or not, can in principle be evaluated in a *logically reversible way, using only "Toffoli gates," i.e., made-reversible-and-then-universal AND gates*; this computation can then be thermodynamically neutral, i.e., it does not dissipate heat.

Landauer's principle states that the erasure (setting the corresponding binary memory cells to 0) of N bits costs $kTN \ln 2$ free energy which must be dissipated as heat to the environment, a thermal bath of temperature T. The *dissipation* is crucial in the argument: Heating up the environment compensates for the

[7] The corresponding formula $S = k \ln W$ is written in golden letters on Boltzmann's gravestone at the *Zentralfriedhof* — which is, as the word goes in Vienna, "halb so gross und doppelt so lustig wie Zürich." Boltzmann imagined that the universe had started in a completely "uniform" state. So the diverse reality now perceived would be a mere fluctuation. Note that the fact that this fluctuation is extremely unlikely is irrelevant if you can *condition on our existence*, given your thinking all this (this thought is sometimes called "the anthropic principle"). He was probably aware that this way of thinking may lead into *solipsism:* "My existence alone, simply imagining all that, is much more likely than the actual existence of all people around me, let alone all the visible galaxies, etc." He eventually hung himself in a hotel room in Duino, Italy. It has been colported in Vienna that this may have been due to "mobbing" at the university by Ernst Mach. Anyhow, today we prefer to comfort ourselves with the contrary belief that the universe initiated in a low-entropy state, and we call this assumption "the big bang".

[8] Here, k is Boltzmann's constant connecting the micro- and macroscopic realms, T is the environmental temperature — and the factor $\ln 2$ is a common sight at the border between logic and physics with their respective basic constants 2 and e.

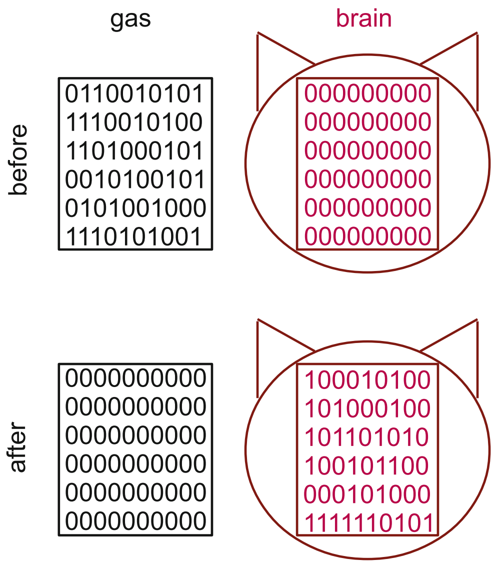


Fig. 1. Bennett's resolution of the Maxwell-demon paradox.

loss of entropy within the memory cell which is materialized by some physical system (spin, gas molecule, etc.). Landauer's principle is a direct consequence of Boltzmann's view of the second law.

Charles Bennett [8] used Landauer's slogan "Information is Physical" for making the key contribution to the resolution of the paradox of "Maxwell's demon" (see, e.g., [25]). That demon had been thought of as violating the second law[9] by adaptively handling a frictionless door with the goal of "sorting a gas" in a container. Bennett took the demon's memory (imagined to be in the all-0-state before sorting) into account, which is in the end filled with "random" information remembering the original state of the gas: The growth of disorder *within* the demon compensates for the order she creates *outside* (i.e., in the gas) — the second law is saved. The initial 0-string is the demon's resource allowing for her order creation (see Fig. 1): If we break Bennett's argument apart in the middle, we end up with the *converse* of Landauer's principle: The all-0-string has *work*

[9] Also other authors noted the strange dependency of the law on the absence of certain life forms — Kelvin wrote: "When light is absorbed *other than in vegetation*, there is dissipation [...]"). To make things worse, there is always a non-zero probability (exponentially small with respect to the strength of entropy decrease though) of exceptions, where the law fails to hold; we are not used to this from other laws of physics (outside quantum theory). This *fragility* of the second law is weirdly contrasted by its being, at the same time, *more robust* than others, such as Bell violations only realizable in extreme lab conditions: We do not need to trust experimentalists to be convinced that "it" is there — in fact, it is everywhere.

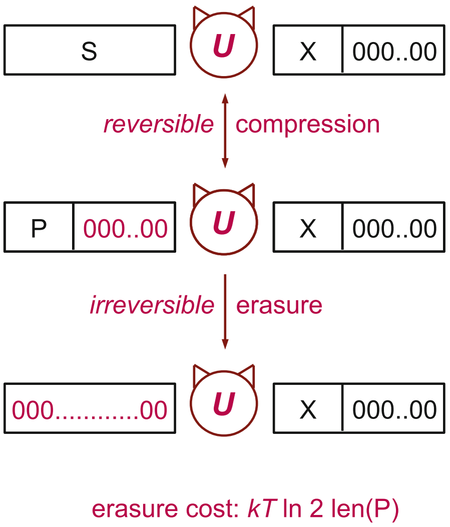


Fig. 2. Erasing S with catalyst X at cost $\mathrm{EC}_{\mathcal{U}}(S|X)$: First, X is reversibly compressed to the shorter string P for free, then that "program" P is erased at cost $kT\ln 2 \cdot \mathrm{len}(P)$.

value, i.e., if we accept the respective memory cells to become "randomized," we can extract $kTN\ln 2$ free energy from the environment (of temperature T).

Already Bennett [7] had implied that for some strings S, the *erasure cost* is less than Landauer's $\mathrm{len}(S)\cdot kT\ln 2$: Besides the obvious $00\ldots 0$ and $11\ldots 1$, this is also true, e.g., for the string formed by the first N digits of the binary expansion of π: The reason is that there is a *short program* generating the sequence or, in other words, a logically *reversible* computation between (essentially) 0^N and π^N that can be carried out thermodynamically reversibly [14]. Generally, if the string can be *compressed* in a lossless fashion, then the erasure cost shrinks accordingly.

Let us consider a *model for the erasure process* (see Fig. 2) in which there is, besides the string S to be erased, another string X on the tape as a "catalyst" summarizing possible a priori "knowledge" about S, so helping to compress it but remaining itself unchanged in the process. The universal Turing machine $\mathcal{U}$'s tape is assumed to be finite, but can be extended arbitrarily as needed — there would be infinite "work value" on it otherwise —, and the resulting erasure cost is $\mathrm{EC}_{\mathcal{U}}(S|X)$.

Bennett [7] claims that the erasure cost of a string is, actually, its Kolmogorov complexity times $kT\ln 2$; this would in our model translate to the erasure cost equalling the conditional complexity of the string to be erased, given the side information: $\mathrm{EC}_{\mathcal{U}}(S|X) = K_{\mathcal{U}}(S|X)$. Unfortunately, this is in general not achievable due to the *uncomputability* of Kolmogorov complexity and the corresponding compression transformation, and since we assume the extraction to be carried out by a Turing machine, in the spirit of the *Church/Turing thesis* [17]. It is, however, true that Kolmogorov complexity leads to a *lower bound* on the erasure cost since it represents the ultimate limit on the lossless compressibility of the string by $\mathcal{U}$. In the same way, we can argue that any *concrete* and *computable* compression algorithm (with side information) C, e.g., Lempel/Ziv [31], leads to an *upper bound* on the erasure cost: First, we reversibly compress (at no energy cost) and then erase the compression.[10]

Landauer's principle, revisited. *Let C be a computable function,*

$$C\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \ ,$$

such that

$$(V, W) \mapsto (C(V, W), W)$$

is injective. Then the cost of the erasure of S with catalyst X, carried out by the universal Turing machine $\mathcal{U}$, is bounded by

$$K_{\mathcal{U}}(S|X) \cdot kT \ln 2 \ \le\ \mathrm{EC}_{\mathcal{U}}(S|X) \ \le\ \mathrm{len}(C(S, X)) \cdot kT \ln 2 \ .$$

The principle can be extended to an *arbitrary* computation starting with input A and leading up to output B with side information X, where (A, X) and (B, X) are the only contents of the tape, besides 0s, before and after the computation, respectively. Our result is an algorithmically constructive modification of entropic results [12] and a generalization of less constructive but also complexity-based claims [33].

Landauer's principle, generalized. *Let C be a computable function,*

$$C\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \ ,$$

such that

$$(V, W) \mapsto (C(V, W), W)$$

is injective. Assume that the Turing machine $\mathcal{U}$ carries out a computation such that A and B are its initial and final configurations. Then the energy cost of this computation with side information X, $\mathrm{Cost}_{\mathcal{U}}(A \to B \,|\, X)$, is at least

$$\mathrm{Cost}_{\mathcal{U}}(A \to B \,|\, X) \ \ge\ [K_{\mathcal{U}}(A|X) - \mathrm{len}(C(B, X))] \cdot kT \ln 2 \ .$$

[10] Our result is in a certain contrast with claims by Dahlsten et al. [10] who take an *entropic* stand: If a "demon" knows perfectly the string to be erased, this can be done for free. In our algorithmic view, this is not reproduced for the case where that knowledge is *non-constructive*, e.g., "S consists of the first N bits Ω^N of the *halting probability* Ω of $\mathcal{U}$." Given that particular knowledge, the entropy of S is zero, but still it does not help for erasing S since no algorithm can generate or "uncompute" S with the help of that knowledge. In contrast, as long as the knowledge is a copy of S or a program that allows for computing S, the results of [10] and our own match.

Proof. The erasure cost of A, given X, is at least $K_{\mathcal{U}}(A|X) \cdot kT \ln 2$ according to the above. *One* possibility of realizing this complete erasure of A is to first transform it to B (given X), and then erase B — at cost at most $\mathrm{len}(C(B,X)) \cdot kT \ln 2$. Therefore, the cost to get from A to B given X cannot be lower than the difference between $K_{\mathcal{U}}(A|X) \cdot kT \ln 2$ and $\mathrm{len}(C(B,X)) \cdot kT \ln 2$. $\square$

The complexity reductions in these statements quantify the "amount of logical irreversibility" inherent in the respective process, and the quantity of required work — the price for this *non-injectivity* — is proportional to that. The picture is now strangely *hybrid*: The environment must pay a *thermodynamic (macroscopic)* price for what happens *logically (microscopically)* in one of its parts. What prevents us from looking at the environment with a microscope? If we let ourselves inspire by *John Archibald Wheeler*'s [27] "It from Bit," we see the possibility of a compact argument: The price that the environment has to pay in compensation for the irreversibility of the computation in one of its parts is such that the *overall computation is reversible.*

Second law of Thermodynamics, logico-computational.
Time evolutions are logically reversible: No information gets erased.

Let us first note that this condition on time evolutions is, like traditional "second laws," asymmetric in time: Logical reversibility only requires the future to uniquely determine the past, *not vice versa*: So if "reality" is such an injective computation, its reverse can be injective as well (e.g., computation of a *deterministic* Turing machine, Everett/Bohm interpretation of quantum theory) or this fails to hold since the forward direction has splitting paths (e.g., computation of a *probabilistic* Turing machine).

The second law has often been linked to the emergence of an *arrow of time.* How is that compatible with our view? In *determinism*, logical reversibility holds *both ways*. What could then be possible origins of our ability to distinguish past and future? (Is it the limited precision of our sense organs and the resulting *coarse-graining*?) *Indeterminism* is easier in that sense since it comes with objective asymmetry in time: *Randomness* points to the future.

4 Consequences

Logical Reversibility Implies Quasi-Monotonicity

The logical reversibility of a computation implies that the overall complexity of the Turing machine's configuration at time t can be smaller than that at time 0 by *at most* $K(C_t) + O(1)$ if C_t is a string encoding the time span t. The reason is that one possibility of describing the configuration at time 0 is to give the configuration at time t, plus t itself; the rest is exhaustive search using only a constant-length program simulating forward time evolution.

Logical Reversibility Implies a Boltzmann-Like Second Law

The notion of a *macrostate* can be defined in an objective way, using a "structure" function based on the *Kolmogorov sufficient statistics* [1,4]. Roughly speaking, the macrostate is the structure or "compressible part" of a microstate: Given the macrostate — being a set of microstates that is, ideally, small, and its description at the same time short (such as: "a gas of volume V, temperature T, and pressure p in equilibrium") —, the particular microstate is a "typical element" of it, specifiable only through stubborn binary coding. This notion of a macrostate is typically unrelated to the second law except when the initial and final macrostates both have very short descriptions, like for equilibria: Then, logical reversibility implies that their size is essentially non-decreasing in time. This is Boltzmann's law.

Logical Reversibility Implies a Clausius-Like Second Law

If we have a circuit — the time evolution — using only logically reversible Toffoli gates, then it is *impossible* that this circuit computes a transformation of the following nature: Any pair of strings — one with higher Hamming weight than the other — is mapped to a pair of equally long strings where the heavy string has become heavier and the light one lighter. Such a mapping, which *accentuates* differences, cannot be injective for counting reasons. We illustrate this with a toy example (see Fig. 3).

Example 1. Let a circuit consisting of Toffoli gates map an $N(=2n)$-bit string to another string — which must then be N-bits long as well, due to reversibility. We consider the mapping separately on the first and second halves of the full string. We assume the computed function to be conservative, i.e., to leave the Hamming weight of the full string unchanged. We look at the excess of 1's in one of the halves (which equals the deficit of 1's in the other). We observe that the probability (with respect to the uniform distribution over all strings of some Hamming-weight couple $(wn,(1-w)n)$) of the *imbalance substantially growing* is exponentially weak. The key ingredient in the argument is the function's injectivity. Explicitly, the probability that the weight couple goes from $(wn,(1-w)n)$ to $((w+\varDelta)n,(1-w-\varDelta)n)$ — or more extremely —, for $1/2 \le w < 1$ and $0 < \varDelta \le 1-w$, is

$$\frac{\binom{n}{(w+\varDelta)n}\binom{n}{(1-w-\varDelta)n}}{\binom{n}{wn}\binom{n}{(1-w)n}} = 2^{-\Theta(n)}:$$

Logical reversibility is incompatible with the tendency of *amplification* of differences.

Logical Reversibility Implies a Kelvin-Like Second Law

Finally, logical reversibility also implies statements resembling Kelvin's version of the second law: "A single heat bath alone has no work value." This, again, follows from a counting argument: There exists no reversible circuit that concentrates redundancy to some pre-chosen bit positions.

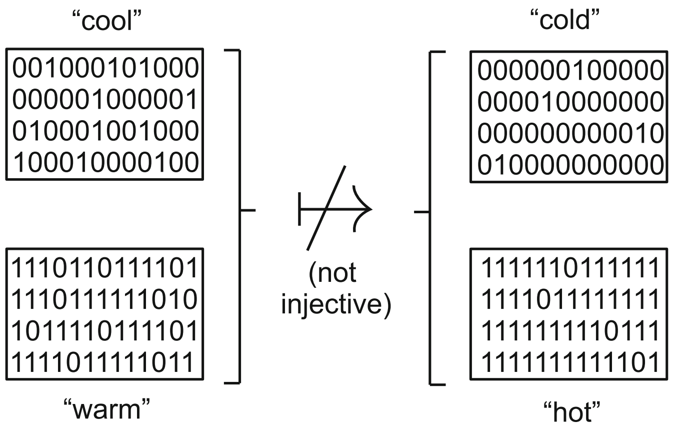


Fig. 3. Logical reversibility does not accentuate differences: Clausius.

Example 2. The probability that a fixed circuit maps a string of length N and Hamming weight w to another such that the first n positions contain only 1's, and such that the Hamming weight of the remaining $N-n$ positions is $w-n$, is

$$\frac{\binom{N-n}{w-n}}{\binom{N}{w}} = 2^{-\Theta(n)} .$$

In a sense, Kelvin's law is a special case of Clausius' formulation.

5 Epilogue

A priori, the relation between physical reality and computation is two-fold: Turing's famous machine model is physical — besides the tape's infiniteness —, and the generalized Church/Turing thesis suspects physical processes to be simulatable on a Turing machine (see Fig. 4). This *circulus* has been the background of our considerations.

In the "Church/Turing view," a physical law is a property of a Turing machine's computation: The *second law of thermodynamics* is *logical reversibility.*[11]

[11] The quantum-physical-interpretational reading thereof is the Everettian *relative-state* view: In contrast to collapse interpretations, it *is* logically reversible due to its unitarity. — There is a vivid dispute on such readings of quantum theory that is sometimes more fiery than exact, more passionate than argumentational: The nature of the debate is ignoring *Paul Feyerabend*'s remark that in science just as well as in art, a particular *style* can merely be judged from the point of view of another, never objectively. This call to modesty culminated in seeing *science as one style among many*: "Man entscheidet sich für oder gegen die *Wissenschaften* genau so, wie man sich für oder gegen *punk rock* entscheidet" [13].

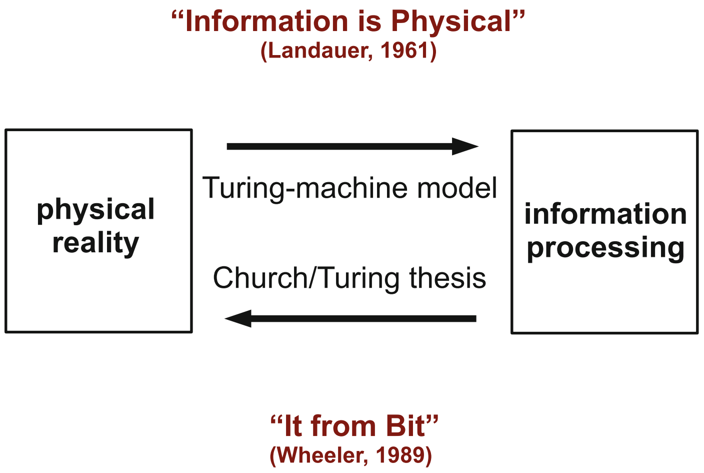


Fig. 4. Physics and information: Scenes from a marriage.

What can be said about the validity of the Church/Turing thesis? Nothing absolute, just as on "determinism." But exactly like for that latter problem, an *all-or-nothing* statement creates, at least, a clear dichotomy [2,28].

All-or-Nothing Feature of the Church/Turing Thesis.
Either no entity can generate non-Turing-computable sequences, or even single photons can.

This results when we pursue the idea that *Kolmogorov complexity measures intrinsic randomness* and apply it to *Bell correlations*: If we have access to some "super-Turing machine," we let that machine choose the particular measurement bases for the two parts of an entangled quantum system. The correlations then imply the sequence of measured values to be Turing-uncomputable as well. This analysis [29] resembles well-known arguments but replaces randomness by complexity. The new approach has two conceptual advantages. The first is its *non-counterfactuality*: We do not talk about the outcomes of *unperformed* measurements. (Any Bell inequality does.) The second is *context-independence*: No process-based notion of randomness is required. Such definitions are typically embedded into a *causal structure* — but the non-local correlations *themselves*, with their inexplicability by a "reasonable" mechanism within such a structure, are among the strongest arguments *against* fundamental causality. The alternative is to consider "free will" (randomness, if you will) as more fundamental and causality as emerging from it through: "If Y is correlated with X, and X is *freely* *random*, then Y must be in X's future."[12] Then the arrow of time appears as an accumulation of irreversible binary decisions. The reverse transformation of such splits is not logically reversible and, hence, violates the second law:

$$\text{Logical Reversibility} + \text{Randomness} = \text{Thermodynamic Irreversibility}\ .$$

This equation suggests how it comes that a law which we read here as reversibility is often linked to the exact opposite.

Acknowledgments. The author thanks Mateus Araújo, Veronika Baumann, Ämin Baumeler, Charles Bédard, Claus Beisbart, Gilles Brassard, Harvey Brown, Xavier Coiteux-Roy, Caslav Brukner, Harry Buhrman, Matthias Christandl, Fabio Costa, Bora Dakic, Frédéric Dupuis, Paul Erker, Adrien Feix, Jürg Fröhlich, Nicolas Gisin, Arne Hansen, Juraj Hromkovič, Marcus Huber, Dennis Komm, Alberto Montina, Samuel Ranellucci, Paul Raymond-Robichaud, Louis Salvail, L. Benno Salwey, Andreas Winter, and Chris Wüthrich for inspiring discussions. — *Grazie mille!*

Our work is supported by the Swiss National Science Foundation (SNF), the National Centre of Competence in Research "Quantum Science and Technology" (QSIT), and the *Hasler Foundation*.

[12] Already in the early modern period there has been a vivid debate about the fundamentality of absolute spacetime, for which *Newton* was advocating; *Leibniz*, on the other hand, rejected the notion as absurd and understood spacetime as merely relational, emergent, and of logical nature. The corresponding debate [26] has been decided by the course of history in favor of Newton — with certain exceptions, notably *Ernst Mach*. We are not the only ones to suggest that the choice taken then should be reconsidered today in the light of new insight.

Although Einstein's relativity's crystallization point was *Mach's principle* — "inertial forces are *relational*, and not with respect to absolute spacetime" — it does not obey it: For Einstein, spacetime *is* fundamental and even in the massless universe, there is the flat spacetime of special relativity. However, an inherent refutation of rigidly causal thinking *is* contained in the field equations of general relativity — having solutions in the form of *closed spacetime curves* [15]. When causality is dropped, one risks antinomies. Ruling out the latter does, however, not throw us back to causality [3]. This observation has recently been extended to *computational complexity* [5]: Non-causal circuits avoiding antinomies are strictly stronger than rigidly causal circuits.

The opposition between Newton and Leibniz can be seen as the modern version of the tension between the thinking styles of the pre-Socratic philosophers *Parmenides* and *Heraclitus*: For the former, all time and change are pure *illusions*, whereas for the latter, *only change is real*: "You cannot step into the same river twice." Nietzsche [20] compared the "cold logician" Parmenides to *ice,* whereas Heraclitus is the *fiery* physicist. The postmodern manifestation of the tension is the gap between Landauer's "Information is Physical" and Wheeler's "It from Bit." In this note, we play on just that opposition for obtaining our reading of the second law.

References

1. Aaronson, S.: (2012). http://www.scottaaronson.com/blog/?p=762
2. Baumeler, Ä., Bédard, C., Brassard, G., Wolf, S.: Kolmogorov amplification from Bell correlation. In: International Symposium in Information Theory (ISIT) (2017)
3. Baumeler, Ä., Feix, A., Wolf, S.: Maximal incompatibility of locally classical behavior and global causal order in multiparty scenarios. Phys. Rev. A **90**(4), 042106 (2014)
4. Baumeler, Ä., Wolf, S.: Causality–complexity–consistency: can space-time be based on logic and computation? In: Renner, R., Stupar, S. (eds.) Time in Physics. TSWMS, pp. 69–101. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68655-4_6
5. Baumeler, Ä., Wolf, S.: Non-causal computation. Entropy **19**(7), 326 (2017)
6. Bell, J.S.: On the Einstein-Podolsky-Rosen paradox. Physics **1**, 195–200 (1964)
7. Bennett, C.H.: The thermodynamics of computation. Int. J. Theor. Phys. **21**(12), 905–940 (1982)
8. Bennett, C.H.: Notes on Landauer's principle, reversible computation and Maxwell's demon. Stud. Hist. Philos. Mod. Phys. **34**, 501–510 (2003)
9. Colbeck, R., Renner, R.: Free randomness can be amplified. Nat. Phys. **8**, 450–454 (2012)
10. Dahlsten, O., Renner, R., Rieper, E., Vedral, V.: The work value of information. New J. Phys. **13** (2011)
11. Einstein, A., Podolsky, B., Rosen, N.: Can quantum-mechanical description of physical reality be considered complete? Phys. Rev. **41** (1935)
12. Faist, P., Dupuis, F., Oppenheim, J., Renner, R.: The minimal work cost of information processing. Nat. Commun. **6**, 7669 (2015)
13. Feyerabend, P.: Wissenschaft als Kunst. Inaugural Lecture, ETH Zurich (1980)
14. Fredkin, E., Toffoli, T.: Conservative logic. Int. J. Theor. Phys. **21**(3–4), 219–253 (1982)
15. Gödel, K.: An example of a new type of cosmological solutions of Einstein's field equations of gravitation. Rev. Mod. Phys. **21**(3), 447–450 (1949)
16. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15155-2_3
17. Kleene, S.C.: Mathematical Logic. Wiley, Hoboken (1967)
18. Kolmogorov, A.: Three approaches to the quantitative definition of information. Problemy Peredachi Informatsii **1**(1), 3–11 (1965)
19. Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. **5**, 183–191 (1961)
20. Nietzsche, F.: Die Philosophie im tragischen Zeitalter der Griechen. Werke in drei Bänden, vol. 3, Carl Hanser Verlag, München (1954)
21. Oreshkov, O., Costa, F., Brukner, C.: Quantum correlations with no causal order. Nat. Commun. **3**, 1092 (2012)
22. Reichenbach, H.: Chapter 19, The principle of the common cause. In: The Direction of Time, pp. 157–167. California Press, Berkeley (1956)
23. Russell, B.: On the notion of cause. In: Proceedings of the Aristotelian Society. New Series, vol. 13, pp. 1–26 (1912)
24. Specker, E.: Die Logik nicht gleichzeitig entscheidbarer Aussagen. Dialectica **14**, 239–246 (1960)

25. Szilárd, L.: Über die Entropieverminderung in einem thermodynamischen System bei Eingriffen intelligenter Wesen. Zeitschrift für Physik **53**, 840–856 (1929)
26. Vailati, E.: Leibniz & Clarke—A Study of Their Correspondence. Oxford University Press, New York (1997)
27. Wheeler, J.A.: Information, physics, quantum: the search for link. In: Proceedings III International Symposium on Foundations of Quantum Mechanics, pp. 354–368 (1989)
28. Wolf, S.: An all-or-nothing flavor to the church-turing hypothesis. In: Gopal, T.V., Jäger, G., Steila, S. (eds.) TAMC 2017. LNCS, vol. 10185, pp. 39–56. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55911-7_4
29. Wolf, S.: Non-locality without counterfactual reasoning. Phys. Rev. A **92**(5), 052102 (2015)
30. Uffink, J.: Bluff your way in the second law of thermodynamics. Stud. Hist. Philos. Sci. Part B: Stud. Hist. Philos. Mod. Phys. **32**(3), 305–394 (2001)
31. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. IEEE Trans. Inf. Theory **24**(5), 530 (1978)
32. Zurek, W.H.: Algorithmic randomness and physical entropy. Phys. Rev. A **40**(8), 4731 (1989)
33. Zurek, W.H.: Thermodynamic cost of computation, algorithmic complexity and the information metric. Nature **341**, 119–124 (1989)

Reoptimization of NP-Hard Problems

Anna Zych-Pawlewicz(✉)

University of Warsaw, Warsaw, Poland
anka@mimuw.edu.pl

Abstract. A reoptimization problem, given two similar instances of an optimization problem and a good solution to the first instance, asks for a solution to the second. In this paper we propose general approximation algorithms applicable to a wide class of reoptimization problems.

1 Introduction

In this paper we propose general approximation algorithms applicable to a wide class of reoptimization problems. We can build a reoptimization problem on top of any optimization problem. An input instance to the reoptimization problem is an instance of the underlying optimization problem, a good solution to it, and a local modification to that instance. We ask for the solution to the modified instance.

As an example imagine a train station where the train traffic is regulated via a certain time schedule. The schedule is computed when the station is ready to operate and, provided that no unexpected event occurs, there is no need to compute it again. Unfortunately an unexpected event, such as a delayed train, is in a long run inevitable. Reoptimization addresses this scenario, asking whether knowing a solution for a certain problem instance is beneficial when computing a solution for a similar instance. When dealing with relatively stable environments, i.e., where changing conditions alter the environment only for a short period of time, it seems reasonable to spend even a tremendous amount of time on computing a good solution for the undisturbed instance, and profit from it when confronted with a temporal change.

Due to its practical impact, a lot of work has been dedicated to reoptimization. The term reoptimization was mentioned for the first time in [16]. The authors studied the problem of scheduling with forbidden sets in the scenarios of adding or removing a forbidden set. Subsequently various other NP-hard problems in reoptimization settings have been successfully approached: the knapsack problem [2], the weighted minimum set cover problem [15], various covering problems [8], the shortest common superstring problem [7], the Steiner tree problem [6,11,13,19,20] and different variations of the traveling salesman problem [1,3,5,9,12,14]. We refer to [10,14,18,21] for the surveys, where [21] is the most

This work was partially conducted as part of a PhD programme at ETH Zürich and later supported by NCN grant UMO-2014/13/B/ST6/01811.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 477–494, 2018.
https://doi.org/10.1007/978-3-319-98355-4_28

up to date one. Most of the proposed reoptimization algorithms are based on certain general problem patterns. Hence, an attempt to abstract the patterns from the specific problems and state them on a general level appears desirable. One such generalization was presented in [14], where the class of hereditary reoptimization problems under the modification of vertex insertion was observed to be a subject to a general pattern. This was an inspiration to writing this paper.

Our aim is to abstract the general patterns and properties of the problems that lead to good reoptimization algorithms. For the characterized reoptimization problems we propose general algorithms with provable approximation ratios. Our characteristics are very general and apply to a variety of seemingly unrelated problems. Our algorithms achieve the approximation ratios in [2,8,15] obtained for each problem separately. Moreover, as discussed further in the paper, the recent advances in reoptimization of the minimum Steiner tree problem are heavily based on our general methods, see [18]. In [8], reoptimization variants of some covering problems were considered for which the general approximation ratios proposed in this paper were proven tight. This indicates that our methods are unlikely to be improved on a level as general as presented here.

2 The Basics

In this section we model the concept of reoptimization formally and provide some preliminary results. We observe that, in principle, reoptimization of NP-hard problems is NP-hard. Most of the times approximation becomes easier, i.e., a reoptimization problem admits a better approximation ratio than its optimization counterpart. In principle, if the modification changes the optimal cost only by a constant, the reoptimization problem admits a PTAS. The more interesting situation when the optimal cost can change arbitrarily is addressed in subsequent sections.

We now introduce the notation and definitions used further in the paper.

Definition 1 (Vazirani [17]). *An NP* optimization *(NPO) problem $\Pi = (\mathcal{D}, \mathcal{R}, cost, goal)$ consists of*

1. *A set of valid instances $\mathcal{D}_\Pi$ recognizable in polynomial time. We will assume that all numbers specified in an input are rationals. The size of an instance $I \in \mathcal{D}_\Pi$, denoted by $|I|$, is defined as the number of bits needed to write I under the assumption that all numbers occurring in the instance are written in binary.*
2. *Each instance $I \in \mathcal{D}_\Pi$ has a set of feasible solutions, $\mathcal{R}_\Pi(I)$. We require that $\mathcal{R}_\Pi(I) \neq \emptyset$, and that every solution* Sol *$\in \mathcal{R}_\Pi(I)$ is of length polynomially bounded in $|I|$. Furthermore, there is a polynomial-time algorithm that, given a pair $(I, \textsc{Sol})$, decides whether* Sol *$\in \mathcal{R}_\Pi(I)$.*
3. *There is a polynomial-time computable objective function, $cost_\Pi$, that assigns a nonnegative rational number to each pair $(I, \textsc{Sol})$, where I is an instance and* Sol *is a feasible solution to I.*

4. *Finally, Π is specified to be either a minimization problem or a maximization problem: $goal_\Pi \in \{\min, \max\}$.*

An optimal solution to a problem Π is a feasible solution that minimizes or maximizes the cost, depending on $goal_\Pi$. We denote the set of optimal solutions to an instance I by $\textsc{Optima}_\Pi(I)$ and the optimal cost by $\mathrm{Optcost}_\Pi(I)$. Typically, we write $\textsc{Opt} \in \textsc{Optima}_\Pi(I)$ to denote an optimal solution to an instance I of a problem Π. To denote any solution, we write $\textsc{Sol} \in \mathcal{R}_\Pi(I)$. The costs are referred to as Opt and Sol, respectively. We omit the index and/or the argument if it is clear from the context.

We view an algorithm *Alg* solving an *NPO* problem $\Pi = (\mathcal{D}, \mathcal{R}, cost, goal)$ as a mapping from the instance set to the solution set satisfying $Alg(I) \in \mathcal{R}(I)$. We denote the asymptotic running time of an algorithm *Alg* by $\mathrm{Time}(Alg)$, whereas $\mathrm{Poly}(n)$ stands for a function polynomial in n. For the sake of simplicity, we define an order $\succeq$ on the values of *cost* which favors better solutions: $\succeq$ equals $\leq$ if $goal = \min$, and $\geq$ otherwise. An algorithm *Alg* is exact if, for any instance $I \in \mathcal{D}$,

$$cost(I, Alg(I)) = \max_{\succeq}\{cost(I, \textsc{Sol}) \mid \textsc{Sol} \in \mathcal{R}(I)\}.$$

Alg is a σ-approximation, if for any instance $I \in \mathcal{D}$,

$$cost(I, Alg(I)) \succeq \sigma \max_{\succeq}\{cost(I, \textsc{Sol}) \mid \textsc{Sol} \in \mathcal{R}(I)\},$$

where $\sigma \leq 1$ if $goal = \max$, and $\sigma \geq 1$ otherwise.

We model the scenario where an instance and a corresponding solution are known, and one needs to find a solution after the instance is modified. Hence, we introduce a binary modification relation on the set of instances, which determines which modifications are allowed. Additionally, the parameter ρ measures the quality of the input solution. Formally, a reoptimization problem is defined as follows.

Definition 2. *Let $\Pi = (\mathcal{D}_\Pi, \mathcal{R}_\Pi, cost, goal)$ be an NPO problem and $\mathcal{M} \subseteq \mathcal{D}_\Pi \times \mathcal{D}_\Pi$ be a binary relation (the modification). The corresponding reoptimization problem $\mathfrak{Re}^\rho_\mathcal{M}(\Pi) = (\mathcal{D}_{\mathfrak{Re}^\rho_\mathcal{M}(\Pi)}, \mathcal{R}_{\mathfrak{Re}^\rho_\mathcal{M}(\Pi)}, cost, goal)$ consists of:*

- *a set of feasible instances: $\mathcal{D}_{\mathfrak{Re}^\rho_\mathcal{M}(\Pi)} = \{(I, I', \textsc{Sol}) \mid (I, I') \in \mathcal{M}, \textsc{Sol} \in \mathcal{R}_\Pi(I)$ and $\mathrm{Sol} \succeq \rho \mathrm{Optcost}(I)\}$; we refer to I as the original instance and to I' as the modified instance*
- *a feasibility relation: $\mathcal{R}_{\mathfrak{Re}^\rho_\mathcal{M}(\Pi)}((I, I', \textsc{Sol})) = \mathcal{R}_\Pi(I')$.*

For the sake of simplicity, we denote $\mathfrak{Re}^1_\mathcal{M}(\Pi)$ by $\mathfrak{Re}_\mathcal{M}(\Pi)$.

We illustrate this concept on the following example.

Example 1. Consider the weighted maximum satisfiability problem (*wMaxSAT*), where clauses are assigned weights and one seeks for assignment that satisfies a set of clauses of maximum weight. In the reoptimization scenario, a ρ-approximate assignment $\textsc{Sol}$ to formula Φ is given. The given formula Φ is altered for instance by adding one of its literals to one of its clauses. This is captured by a modification relation: the original instance I and the modified instance I' are a valid part of the input if and only if $(I, I') \in \mathcal{M}$. The modification is defined as follows: $(I, I') \in \mathcal{M}$ if and only if all the clauses but one are the same in $\varPhi$ and $\varPhi'$ and the only different clause in $\varPhi'$ contains one additional literal. The reoptimization problem $\mathfrak{Re}^{\rho}_{\mathcal{M}}(wMaxSAT)$, given $(\varPhi, \varPhi', \textsc{Sol})$ such that $(\varPhi, \varPhi') \in \mathcal{M}$, asks for an assignment $\textsc{Sol}'$ for $\varPhi'$ maximizing the total weight of satisfied clauses. Let us denote the set of satisfied clauses by $Sats(\varPhi', \textsc{Sol}')$.

The two consecutive lemmas show both the limits and the power of reoptimization.

Lemma 1 (Böckenhauer et al. [10]**).** *Reoptimization problems of NP-hard problems, where applying the modification a polynomial number of times can arbitrarily change an input instance, are NP-hard, even if all input instances contain an optimal solution.*

Lemma 2 (Bilò et al. [8]**).** *If, for every instance $(I, I', \textsc{Sol})$ of $\mathfrak{Re}^{\rho}_{\mathcal{M}}(\varPi)$ and any constants b and b', we can compute in polynomial time a feasible solution $\textsc{Sol}'$ to I', such that $|\mathrm{Sol}' - \mathrm{Optcost}_{\varPi}(I')| \leq b$, and a best feasible solution among the solutions to I' whose cost is bounded by b', then $\mathfrak{Re}^{\rho}_{\mathcal{M}}(\varPi)$ admits a* PTAS.

Proof. Let $\varPi$ be a maximization problem (for minimization problems the proof is analogous). For a given $\varepsilon > 0$, we compute two solutions to I': $\textsc{Sol}'$ as in the claim of the lemma, and $\textsc{Sol}''$, a best among the feasible solutions with cost bounded by $\frac{b}{\varepsilon}$. If $\mathrm{Optcost}_{\varPi}(I') \leq \frac{b}{\varepsilon}$, then $\textsc{Sol}''$ is optimal. Otherwise $\mathrm{Sol}' \geq \mathrm{Optcost}_{\varPi}(I') - b \geq (1 - \varepsilon)\mathrm{Optcost}_{\varPi}(I')$. $\square$

Example 2. We illustrate Lemma 2 on the example of the maximum satisfiability problem under clause addition: $\mathfrak{Re}_{\mathcal{M}}(MaxSAT)$, where $(\varPhi, \varPhi') \in \mathcal{M}$ if and only if $\varPhi' = \varPhi \wedge C_{\mathrm{new}}$ for $C_{\mathrm{new}} \notin \varPhi$. Due to Lemma 2, $\mathfrak{Re}_{\mathcal{M}}(MaxSAT)$ admits a PTAS. Clearly, if C_{new} contains new variables, the reoptimization is trivial, so we focus on the case when C_{new} contains only variables from $Var(\varPhi)$. Optimal assignments: $\textsc{Opt}$ to $\varPhi$ and $\textsc{Opt}'$ to $\varPhi'$ differ at most by one (the contribution of C_{new}). Hence, $\textsc{Opt}$ as a solution to $\varPhi'$ has a cost greater or equal to $\mathrm{Opt}' - 1$. To satisfy the second condition of the lemma, we need to compute a best among the solutions with cost bounded by a constant b in polynomial time. To that end, we exhaustively search for at most b to be satisfied clauses. For each choice of $b' \leq b$ clauses we verify if the selected set of clauses is satisfiable. Note that b' variables suffices to satisfy b' clauses, so the satisfying assignment, if it exists, can be found in polynomial time via exhaustive search. The PTAS for $\mathfrak{Re}_{\mathcal{M}}(MaxSAT)$ follows by Lemma 2.

Lemma 2 typically applies to unweighted reoptimization problems. It implies a PTAS for *MaxSAT* (maximum satisfiability) if the modification alters only a constant number of clauses, a PTAS for maximum independent set (*MaxIndSet*), maximum clique (*MaxCli*), minimum vertex cover (*MinVerCov*) and minimum dominating set (*MinDomSet*) if the modification alters a constant number of vertices/edges in the graph [8], a PTAS for minimum set cover (*MinSetCov*) if the modification alters a constant number of sets [8], and a PTAS for β-minimum Steiner tree (β-*SMT*) under changing the weight of an edge and under changing the status of a vertex [13].

3 Self-Reducibility

In the previous section we addressed the reoptimization problems where the modification alters the optimal cost only by a constant. Should that not be the case, other measures need to be developed. As we show in the next section, typically an adjustment of the solution given in the input provides a greedy reoptimization algorithm. To go beyond the approximation ratio of the greedy algorithm, we compute a solution alternative to the greedy one and return a better among them. We propose the self-reduction method for computing the alternative solution and show how it applies to self-reducible problems. We explain the concept of self-reducibility before moving on to the method.

The self-reducibility explains why, for many *NPO* problems, we can construct a recursive algorithm, which reduces an input instance to a few smaller instances and calls itself on each of them. Perhaps the best problem to explain what we have in mind is *MaxSAT*. For a given a formula Φ in n variables, a feasible solution is a partial assignment that assigns to each variable $x_i \in \mathit{Var}(\Phi)$ either 0 or 1 or nothing. The reason why it is convenient for us to allow partial assignments will become clear in Sect. 4. For a partial assignment $\textsc{Sol}$ to Φ, we set $\mathit{cost}(\Phi, \textsc{Sol})$ to be the total number of clauses satisfied by $\textsc{Sol}$. The recursive algorithm takes the first unassigned variable $x_1 \in \mathit{Var}(\Phi)$ and sets $x_1 = 1$. It then reduces Φ to a smaller formula $\Phi_{x_1=1}$, the solutions for which are the partial assignments on the remaining $n-1$ variables. The reduced formula $\Phi_{x_1=1}$ is obtained by removing all the literals x_1 and $\overline{x_1}$ and all the clauses containing x_1 from Φ. The algorithm calls itself recursively on $\Phi_{x_1=1}$. The recursive call returns a partial assignment $\textsc{Sol}_{x_1=1}$ to $\Phi_{x_1=1}$. Then, a solution $\textsc{Sol}$ to Φ is computed by assigning $x_1 = 1$ and the values of $\textsc{Sol}_{x_1=1}$ to the remaining variables. The algorithm analogously computes solution $\textsc{Sol}'$ where $x_1 = 0$. Finally, it returns a better solution among $\textsc{Sol}$ and $\textsc{Sol}'$.

This algorithm finds an optimum for the following reasons. Firstly, it stops after at most $2^{|\Phi|}$ recursive calls, as $|\Phi_{x_1=j}| < |\Phi|$. Secondly, there is a one-to-one correspondence between feasible assignments to Φ setting x_1 to 1 and the feasible assignments to $\Phi_{x_1=1}$. Observe that $\mathit{cost}(\Phi, \textsc{Sol})$ is equal to the number of removed clauses plus $\mathit{cost}(\Phi_{x_1=1}, \textsc{Sol}_{x_1=1})$, so the aforementioned correspondence preserves the order on the assignments induced by their cost. We need that property to find the optimum. In what follows we formalize the above and abstract it from *MaxSAT*.

We assume that the solutions to instances of *NPO* problems have a certain granularity, i.e., they are composed of smaller pieces called atoms. In the above example with *MaxSAT*, an atom is an assignment on a single variable, for example $x_1 = 1$ or $x_1 = 0$. We denote the set of atoms of solutions to I by $\mathit{Atoms}(I)$.

We assume that the size of $\mathit{Atoms}(I)$ is polynomially bounded in the size of I, i.e., $|\mathit{Atoms}(I)| \leq \mathrm{Poly}(|I|)$. The set of atoms of solutions to I is formally defined by $\mathit{Atoms}(I) = \bigcup_{\textsc{Sol} \in \mathcal{R}(I)} \textsc{Sol}$. Note that $\mathit{Atoms}(I)$ is entirely determined by the set of feasible solutions for I. For a problem $\mathit{\Pi} = (\mathcal{D}, \mathcal{R}, \mathit{cost}, \mathit{goal})$, the set of all atoms in all instances of $\mathit{\Pi}$ is denoted by $\mathit{Atoms}_{\mathit{\Pi}} = \bigcup_{I \in \mathcal{D}} \mathit{Atoms}(I)$ and we omit the index $\mathit{\Pi}$ if it is clear from the context. We assume that $\mathit{cost}(I, A)$ is defined for any set of atoms $A \subseteq \mathit{Atoms}(I)$. We extend the function $\mathit{cost}(I, \cdot)$ to any set of atoms in $\mathit{Atoms}_{\mathit{\Pi}}$ so that the atoms do not contribute unless they are valid atoms to I. More formally, we denote $A \cap \mathit{Atoms}(I)$ by $[A]_I$ and set $\mathit{cost}(I, A) = \mathit{cost}(I, [A]_I)$. We assume that we can extract $[A]_I$ from A in polynomial time, i.e., we can recognize which atoms are the atoms of I for any problem instance I. We impose a special form of the correspondence function, i.e., a solution to the reduced instance corresponds to the union of itself and the set containing the atom used for the reduction. Clearly, the cost should carry over accordingly.

Definition 3. *We will say that a problem $\mathit{\Pi}$ is* self-reducible *if there is a polynomial-time algorithm, $\mathit{\Delta}$, satisfying the following conditions.*

– *Given an instance I and an atom α of a solution to I, $\mathit{\Delta}$ outputs an instance I_α. We require that the size of I_α is smaller than the size of I, i.e., $|I_\alpha| < |I|$. Let $\mathcal{R}(I|\alpha)$ represent the set of feasible solutions to I containing atom α. We require that every solution* $\textsc{Sol}$ *of I_α, i.e.,* $\textsc{Sol} \in \mathcal{R}(I_\alpha)$, *has a corresponding solution* $\textsc{Sol} \cup \{\alpha\} \in \mathcal{R}(I|\alpha)$ *and that this correspondence is one-to-one.*
– *For any set $A \subseteq \textsc{Sol}_\alpha \in \mathcal{R}(I_\alpha)$ it holds that $\mathit{cost}(I, A \cup \{\alpha\}) = \mathit{cost}(I, \alpha) + \mathit{cost}(I_\alpha, A)$.*

Definition 3 is an adaptation of the corresponding definition in [17]. We illustrate it on a few examples.

Example 3 (Self-Reducibility of the Weighted Maximum Satisfiability). For a given instance $I = (\mathit{\Phi}, c)$, the set of feasible solutions $\mathcal{R}(I)$ contains truth assignments on the subsets of variables. A feasible truth assignment assigns either 0 or 1 but not both to each variable in some set $X \subseteq \mathit{Var}(\mathit{\Phi})$. We represent such assignments as a set of pairs in $X \times \{0, 1\}$. For example, an assignment that assigns 1 to x_1 and 0 to x_2 is represented as $\{(x_1, 1), (x_2, 0)\}$. Such a representation determines the set of atoms for an instance $(\mathit{\Phi}, c)$ as $\mathit{Atoms}((\mathit{\Phi}, c)) = \mathit{Var}(\mathit{\Phi}) \times \{0, 1\}$. The cost of a subset of atoms $A \subseteq \mathit{Atoms}((\mathit{\Phi}, c))$ is $\mathit{cost}(\mathit{\Phi}, A) = \sum_{C \in \mathit{Sats}(\mathit{\Phi}, A)} c(C)$. The reduction algorithm $\mathit{\Delta}$ on atom (x, i), $i \in \{0, 1\}$, returns the instance $(\mathit{\Phi}_{(x,i)}, c)$ obtained by removing all literals x and $\bar{x}$ and all clauses containing x from $\mathit{\Phi}$. Clearly, $\mathit{Var}(\mathit{\Phi}_{(x,i)}) \subseteq \mathit{Var}(\mathit{\Phi}) \setminus \{x\}$, so any assignment $\textsc{Sol}_{(x,i)}$ to $\mathit{\Phi}_{(x,i)}$ can be extended to an assignment $\textsc{Sol}$ to $\mathit{\Phi}$ by setting $\textsc{Sol} = \textsc{Sol}_{(x,i)} \cup \{(x, i)\}$. Then, $\mathit{cost}((\mathit{\Phi}, c), \textsc{Sol}) = c(\mathit{Sats}(\mathit{\Phi}, \{(x, i)\})) + \mathit{cost}((\mathit{\Phi}_{(x,i)}, c), \textsc{Sol}_{(x,i)})$. It may happen that, if during the reduction step from I to $\mathit{\Delta}(I, (x, i))$ two variables x and y are lost, then the solutions to I assigning 1 or 0 to y cannot be reached from solutions to $\mathit{\Delta}(I, (x, i))$ by adding (x, i). They are, however, equivalent with solutions to I assigning nothing to y, and these can be reached. There is a way to define Δ to meet the conditions in Definition 3 exactly, for the sake of clarity however we do not describe it here.

The next few examples are built on graph problems. For such we adopt the following notation. The neighborhood of a vertex $v \in V(G)$ is denoted by $\Gamma_G(v)$. Note that if G is a simple graph then $v \notin \Gamma_G(v)$. The degree of a vertex $v \in V(G)$ is defined as $deg_G(v) = |\Gamma_G(v)|$. For a set of vertices $V' \subseteq V(G)$, by $G - V'$ we denote the subgraph of G induced on $V(G) \setminus V'$.

Example 4 (Self-Reducibility of the Weighted Maximum Independent Set Problem). For a given input instance $G = (V, E, c)$ of *wMaxIndSet*, the set $\mathcal{R}(G)$ of feasible solutions contains all sets of vertices that are independent in G: $\textsc{Sol} \in \mathcal{R}(G) \iff (\textsc{Sol} \subseteq V(G)$ and $\textsc{Sol}$ is independent in $G)$. The set of atoms for G, according to the definition, is the union of atoms over all the solutions in $\mathcal{R}(G)$. Since any vertex in $V(G)$ is in some independent set, we obtain $\mathit{Atoms}(G) = V(G)$. Given a set of atoms $A \subseteq \mathit{Atoms}(G)$ (not necessarily independent in G), we set $\mathit{cost}(G, A) = \sum_{v \in A} c(v)$. We argue that this representation of *wMaxIndSet* is self-reducible. The reduction algorithm Δ on G and $v \in \mathit{Atoms}(G)$ returns a graph G_v, obtained by removing v and its neighborhood from G: $\Delta(G, v) = G - (\Gamma_G(v) \cup \{v\})$. Clearly, any solution $\textsc{Sol}_v$ to G_v maps to the solution to G given by $\textsc{Sol} = \textsc{Sol}_v \cup \{v\}$. Clearly, $\mathit{cost}(G, \textsc{Sol}) = \mathit{cost}(G_v, \textsc{Sol}_v) + \mathit{cost}(G, v)$. Hence the conditions of Definition 3 are satisfied.

Example 5 (Self-Reducibility of the Minimum Steiner Tree (SMT) Problem). For a given instance (G, S), the feasible solutions in $\mathcal{R}((G, S))$ are the subgraphs of G spanning S. We represent solutions as sets of edges, i.e., $\textsc{Sol} \subseteq E(G)$ is a feasible solution to (G, S) if the edges in $\textsc{Sol}$ span S. This determines the set of atoms: $\mathit{Atoms}((G, S)) = E(G)$. The cost function is defined as the sum of costs of edges in the solution: $\mathit{cost}((G, S), \textsc{Sol}) = \sum_{e \in \textsc{Sol}} c(e)$. The reduction function $\Delta((G, S), e)$ contracts edge e to a vertex and includes that vertex in the terminal set. Clearly, this might create parallel edges and self-loops. Nevertheless, any solution $\textsc{Sol}_e \in \mathcal{R}(\Delta((G, S), e))$ spans S and the vertex corresponding to e in the contracted graph. Moreover, all edges of $\textsc{Sol}_e$ are edges in G. As a set of edges in G, $\textsc{Sol}_e$ forms two connected components attached to the endpoints of e, and the component $\textsc{Sol}_e \cup \{e\}$ spans S in G.

Example 6 (Self-Reducibility of the Maximum Cut with Required Edges Problem). For a given instance (G, R), feasible solutions are sets of edges that extend R to a cut in G. This determines the set of atoms: $\mathit{Atoms}((G, R)) = E(G) \setminus R$. The cost function is, as in the previous example, defined as the sum of costs of edges in the solution: $\mathit{cost}((G, R), \textsc{Sol}) = \sum_{e \in \textsc{Sol}} c(e)$. The reduction function $\Delta((G, R), e)$ reduces the cost $c(e)$ to 0 and includes e in R. It is easy to see that the conditions of Definition 3 hold.

The method we now introduce improves the approximation ratio of a self-reducible optimization problem Π if every problem instance admits an optimal

Algorithm 1: $\mathcal{S}_{Alg}$ employing an approximation algorithm *Alg*

```
   input  : Instance $I$
1  forall the $\alpha \in Atoms(I)$ do
2      $\textsc{Sol}(\alpha) := Alg(\Delta(I, \alpha))$;
3  end
   output: The best among $\textsc{Sol}(\alpha) \cup \{\alpha\}$
```

solution containing an expensive atom. In essence, we exhaustively search for the expensive atom through all atoms. For each atom we reduce the instance, approximate the solution of a reduced instance using an approximation algorithm *Alg* for Π and add the missing atom (See Algorithm 1 for the corresponding algorithm $\mathcal{S}_{Alg}$). The next lemma and corollary show what we gain by using $\mathcal{S}_{Alg}$ as compared to using *Alg* only.

Lemma 3. *If* $\Pi = (\mathcal{D}, \mathcal{R}, cost, goal)$ *is self-reducible and* Alg *is a* σ*-approximation algorithm for* Π*, then, for every* $\textsc{Sol} \in \mathcal{R}(I)$ *and every atom* $\gamma \in \textsc{Sol}$*, it holds that* $cost(I, \mathcal{S}_{Alg}(I)) \succeq \sigma cost(I, \textsc{Sol}) - (\sigma - 1) cost(I, \gamma)$*.*

Proof. Fix $\textsc{Sol} \in \mathcal{R}(I)$ and $\gamma \in \textsc{Sol}$. The algorithm returns $\textsc{Sol}(\alpha) \cup \{\alpha\}$ for some atom α which gives the best cost. This solution is feasible for I due to the first condition of Definition 3. Further, since $\textsc{Sol} \in \mathcal{R}(I|\gamma)$, there is $\textsc{Sol}_\gamma$ such that $\textsc{Sol} = \textsc{Sol}_\gamma \cup \{\gamma\}$. The estimation of $cost(I, \mathcal{S}_{Alg}(I))$ follows:

$$\begin{aligned}
&cost(I, \mathcal{S}_{Alg}(I)) \\
&\quad = cost(I, \textsc{Sol}(\alpha) \cup \{\alpha\}) \succeq cost(I, \textsc{Sol}(\gamma) \cup \{\gamma\}) \\
&\quad = cost(I, \gamma) + cost(I_\gamma, \textsc{Sol}(\gamma)) \succeq cost(I, \gamma) + \sigma cost(I_\gamma, \textsc{Sol}_\gamma) \\
&\quad = cost(I, \gamma) + \sigma(cost(I, \textsc{Sol}) - cost(I, \gamma)) \\
&\quad = \sigma cost(I, \textsc{Sol}) - (\sigma - 1) cost(I, \gamma)
\end{aligned}$$

□

Corollary 1. *If every instance* I *of* Π *admits an optimal solution containing an atom* α *with* $cost(I, \alpha) \succeq \delta \mathrm{Optcost}(I)$*, then* $\mathcal{S}_{Alg}$ *is a* $\sigma - \delta(\sigma - 1)$*-approximation.*

In what follows, we generalize the idea by introducing the concept of *guessable* sets of atoms. We say that a set of atoms $A \subseteq \textsc{Sol} \in \mathcal{R}(I)$ is $\mathrm{F}(n)$-guessable for some instance I of a self-reducible problem Π, $|I| = n$, if we can determine a collection of sets of atoms $\mathcal{G}$ with $|\mathcal{G}| \in \mathcal{O}(\mathrm{F}(n))$ such that $A \in \mathcal{G}$. We then call $\mathcal{G}$ a set of guesses for A. Guessing A boils down to an exhaustive search through $\mathcal{G}$. This is useful when we can prove the existence of a certain set A with some characteristic, but we have no knowledge what the elements of A are. If A is $F(n)$-guessable, it is enough to iterate through $\mathcal{O}(F(n))$ candidates in $\mathcal{G}$ to be guaranteed that at some point we look at A. For example, a subset of $Atoms(I)$ of size b of maximum cost among the subsets of size b is n^b-guessable: $\mathcal{G}$ is in that case a collection of subsets of $Atoms(I)$ containing exactly b atoms. Hence,

Algorithm 2: $\mathcal{S}^{\mathcal{G}}_{Alg}$ employing an approximation algorithm *Alg*

```
   input : Instance $I$
1  forall the $A = \{\alpha_1, \ldots, \alpha_{|A|}\} \in \mathcal{G}$ do
2      $I_0 := I$;
3      for $(j = 1, \ldots, |A|)$ do
4          $I_j := \Delta(I_{j-1}, \alpha_j)$;
5      end
6      $\widetilde{\text{Sol}}_{|A|} := Alg(I_{|A|})$;
7      for $(j = 0, \ldots, |A| - 1)$ do
8          $\widetilde{\text{Sol}}_{|A|-j-1} := \widetilde{\text{Sol}}_{|A|-j} \cup \{\alpha_{|A|-j}\}$;
9      end
10 end
   output: The best $\widetilde{\text{Sol}}_0$ over all considered $A \subseteq Atoms(I)$
```

it is Poly(n)-guessable if b is constant (recall that we assume that $|Atoms(I)| \leq$ Poly(n)). Another example of a Poly(n)-guessable set applies to problems where the instances are graphs and the solutions are the sets of vertices in the graph. In that case, we let $A \subset Atoms(I)$ be the neighborhood of a vertex that belongs to some optimal solution (any such vertex). We know that such a set A exists if optimum is not empty. The size of the neighborhood may not be constant as in the previous example, however, the neighborhood itself is determined by a single vertex. Setting $\mathcal{G} = \{\Gamma_G(v)\}_{v \in V(G)}$ brings us to the conclusion that A is n-guessable as $|\mathcal{G}| = |V(G)| < n$. Note that every set of atoms (including the optimal solution) is $2^{|Atoms(I)|}$-guessable.

We modify $\mathcal{S}_{Alg}$ to handle guessable sets. Let $\mathcal{G}$ be the set of valid guesses for a set of atoms. The modified algorithm $\mathcal{S}^{\mathcal{G}}_{Alg}$ runs through all sets $A \in \mathcal{G}$ and reduces the instance using the atoms in A one by one. It applies *Alg* on the obtained instance and combines the solution with the atoms in A. The resulting algorithm $\mathcal{S}^{\mathcal{G}}_{Alg}$ is presented in Algorithm 2.

Algorithm $\mathcal{S}^{\mathcal{G}}_{Alg}$ runs in time $\mathcal{O}(|\mathcal{G}|(|Atoms(I)|\text{Time}(\Delta) + \text{Time}(Alg)))$, so it is a polynomial-time algorithm for $|\mathcal{G}| = \text{Poly}(n)$.

Lemma 4. *If $\Pi = (\mathcal{D}, \mathcal{R}, cost, goal)$ is self-reducible and Alg is a σ-approximation algorithm for Π then, for every* Sol $\in \mathcal{R}(I)$ *and every set $A \in \mathcal{G}$ such that $A \subseteq$* Sol, *it holds that $cost(I, \mathcal{S}^{\mathcal{G}}_{Alg}(I)) \succeq \sigma cost(I, \text{Sol}) - (\sigma - 1) cost(I, A)$.*

Proof. Let $A = \{\alpha_1, \ldots, \alpha_{|A|}\}$ and a solution Sol to instance I be as in the claim of the lemma. Due to Definition 3, Sol corresponds to a solution Sol_{α_1} to instance $I_1 = \Delta(I, \alpha_1)$ such that $\text{Sol} = \text{Sol}_{\alpha_1} \cup \{\alpha_1\}$. Thus all atoms of Sol, except of possibly α_1, are contained in $\text{Sol}_{\alpha_1} \in \mathcal{R}(I_{\alpha_1})$. Hence, we can reduce I_1 further using α_2 as a reduction atom. The argument carries on to all the instances constructed by $\mathcal{S}^{\mathcal{G}}_{Alg}$. Again due to Definition 3, $\widetilde{\text{Sol}}_j = \widetilde{\text{Sol}}_{j+1} \cup \{\alpha_{j+1}\}$ is feasible to I_j for $j = 0, \ldots, |A|$, and the following estimation of the cost of the output holds:

$$\begin{aligned} cost(I, \mathcal{S}^{\mathcal{G}}_{Alg}(I)) &\succeq cost(I, \widetilde{\textsc{Sol}}_0) \succeq cost(I, \widetilde{\textsc{Sol}}_1 \cup \{\alpha_1\}) \\ &= cost(I, \alpha_1) + cost(I_1, \widetilde{\textsc{Sol}}_1) \\ &\succeq \cdots \succeq \sum_{j=1}^{|A|} cost(I_{j-1}, \alpha_j) + cost(I_{|A|}, \widetilde{\textsc{Sol}}_{|A|}) \end{aligned}$$

Now let $\textsc{Sol}_0 := \textsc{Sol}$ and $\textsc{Sol}_{j+1}$ be the solution for I_{j+1} which satisfies $\textsc{Sol}_j = \textsc{Sol}_{j+1} \cup \{\alpha_{j+1}\}$ for $j := 0, \ldots, |A| - 1$. We continue our estimation of the cost. Based on the second condition of Definition 3 and the fact that $\widetilde{\textsc{Sol}}_{|A|}$ is a σ-approximation the following holds.

$$\begin{aligned} cost(I, \mathcal{S}^{\mathcal{G}}_{Alg}(I)) &\succeq cost(I, A) + \sigma cost(I_{|A|}, \textsc{Sol}_{|A|}) \\ &= cost(I, A) + \sigma(cost(I_{|A|-1}, \textsc{Sol}_{|A|-1}) - cost(I_{|A|-1}, \alpha_{|A|})) \\ &= cost(I, A) + \sigma(cost(I, \textsc{Sol}) - \sum_{j=1}^{|A|} cost(I_{j-1}, \alpha_j)) \\ &= cost(I, A) + \sigma(cost(I, \textsc{Sol}) - cost(I, A)) \end{aligned}$$

□

Corollary 2. *If every instance I admits an optimal solution containing a set $A \in \mathcal{G}$ with $cost(I, A) \succeq \delta \mathrm{Optcost}(I)$, then $\mathcal{S}^{\mathcal{G}}_{Alg}$ is a $(\sigma - \delta(\sigma - 1))$-approximation.*

4 Modifications

The main problem with the algorithm $\mathcal{S}^{\mathcal{G}}_{Alg}$ introduced in the previous section is that it does not help if the optima do not contain an efficiently guessable expensive set of atoms. Here, the reoptimization comes into play. Knowing the solution to a similar problem instance, we can construct a greedy solution which is an alternative to the solution computed by $\mathcal{S}^{\mathcal{G}}_{Alg}$. Luckily, the bad instances for $\mathcal{S}^{\mathcal{G}}_{Alg}$ are exactly the good instances for the greedy algorithm. For the remainder of this section, let $\Pi = (\mathcal{D}, \mathcal{R}, cost, goal)$ be a self-reducible optimization problem and let $\mathfrak{Re}^{\rho}_{\mathcal{M}}(\Pi)$ be the corresponding reoptimization problem. Also, for the remainder of this section, let Alg be a σ-approximation algorithm for Π.

The greedy algorithm depends on the type of modification. For example, if the input solution $\textsc{Sol}$ happens to be feasible to the modified instance I', we can use it as the greedy solution. We classify in this section three types of modifications by a combination of two classification criteria and provide general approximation algorithms for them. For the so-called *progressing* modifications, feasible solutions to the original instance I remain feasible to the modified instance I'. For example, after removing an edge from a graph, independent sets remain independent. This is not the case when adding an edge. Then, however, an independent set in the modified instance I' is also independent in the original instance I. The latter is how we define *regressing* modifications.

Table 1. Different types of local modifications with the corresponding approximation ratios for $\mathfrak{Re}_{\mathcal{M}}(\Pi)$

Modification		
Progressing		Regressing
Subtractive	Additive	Additive
$\frac{1}{2-\sigma}$	$\frac{2\sigma-1}{\sigma}$	$\frac{2\sigma-1}{\sigma}$

Once we figured out if the modification is progressing or regressing, we can use the solutions to one of the input instances as the solution to the other one. Unfortunately, this does not work in the opposite direction. Our second classification criterion describes how to modify a solution to this other instance in order to make it feasible for the instance it is not feasible for. As a result, progressing and regressing modifications are subdivided into two further types: *subtractive* and *additive*. They strictly correspond to whether the problem is a maximization or a minimization problem. The classification together with the corresponding approximation ratios is shown in Table 1. For our approximation algorithms to work, feasibility preservation must be accompanied by cost preservation. We start with a few definitions that let us preserve the cost of feasible solutions in a convenient way.

Definition 4. *We say that a cost function cost is:*

- pseudo-additive *if the sum of the costs of two sets of atoms is greater than or equal to the cost of the union of the sets. Formally, for any* $A_1, A_2 \subseteq Atoms(I)$ *such that* $A_1 \cap A_2 = \emptyset$ *it holds that* $(cost(I, A_1 \cup A_2) \leq cost(I, A_1) + cost(I, A_2)$ *and, if* $A_1 \subseteq A_2$*, then* $cost(I, A_1) \leq cost(I, A_2))$*,*
- additive *if the cost of a set of atoms is the sum of the measures of its disjoint subsets. Formally, for any* $A_1, A_2 \subseteq Atoms(I)$ *such that* $A_1 \cap A_2 = \emptyset$ *it holds that* $cost(I, A_1 \cup A_2) = cost(I, A_1) + cost(I, A_2)$*.*

Definition 5. *A modification* $\mathcal{M}$ *is cost-preserving if, for all* $(I, I') \in \mathcal{M}$ *and all* $\textsc{Sol} \in \mathcal{R}(I)$*, it holds that* $cost(I, \textsc{Sol}) \preceq cost(I', \textsc{Sol})$*.*

Definition 6. *A modification* $\mathcal{M}$ *is reversely cost-preserving if, for all* $(I, I') \in \mathcal{M}$ *and all* $\textsc{Sol}' \in \mathcal{R}(I')$*, it holds that* $cost(I', \textsc{Sol}') \preceq cost(I, \textsc{Sol}')$*.*

Definition 7. *A modification* $\mathcal{M}$ *is:*

1. progressing *if feasible solutions to the original instance* I *are feasible also to the modified instance: for every pair* $(I, I') \in \mathcal{M}$ *it holds that* $[\mathcal{R}(I)]_{I'} \subseteq \mathcal{R}(I')$*,*
2. regressing *if feasible solutions to the modified instance* I' *remain feasible for the original instance* I*: for every* $(I, I') \in \mathcal{M}$ *it holds that* $[\mathcal{R}(I')]_I \subseteq \mathcal{R}(I)$*.*

Definition 8. *A progressing modification $\mathcal{M}$ is:*

1. subtractive *(in maximization problems) if there is an optimal solution to the modified instance I' which, after removing a part of it, becomes feasible for I and not less valuable:*

$$\begin{array}{r|c} \exists \textsc{Opt}' \in \textsc{Optima}(I') & \textsc{Opt}' \setminus A' \in \mathcal{R}(I) \\ \exists A' \subseteq \textsc{Opt}' & \mathit{cost}(I, \textsc{Opt}' \setminus A') \geq \mathit{cost}(I', \textsc{Opt}' \setminus A'), \end{array}$$

2. additive *(in minimization problems) if, for every optimal solution* Opt *to the original instance I, there is an optimal solution* Opt$'$ *to the modified instance I' such that one can be transformed into the other as follows*
   - Opt$'$ *(minus possibly some atoms) becomes feasible to I after adding to it a part of* Opt
   - *If we remove this part from* Opt, *it may not be feasible anymore. It becomes feasible to I' after adding a part of* Opt$'$. *Formally:*

$$\begin{array}{r|c} \forall \textsc{Opt} \in \textsc{Optima}(I) & \\ \exists \textsc{Opt}' \in \textsc{Optima}(I') & \textsc{Opt}' \setminus A'' \cup A \in \mathcal{R}(I) \\ \exists A \subseteq \textsc{Opt} & \mathit{cost}(I, \textsc{Opt}' \setminus A'' \cup A) \leq \mathit{cost}(I', \textsc{Opt}' \setminus A'' \cup A) \\ \exists A', A'' \subseteq \textsc{Opt}' & (\textsc{Opt} \setminus A) \cup A' \in \mathcal{R}(I'). \end{array}$$

We now introduce the general approximation algorithms.

Theorem 1. *Let cost be pseudo-additive, $\mathcal{M}$ be cost preserving and progressing subtractive and let A' be as in Definition 8.1. Then, there is a $\frac{1}{2-\sigma}$-approximation algorithm for $\mathfrak{Re}_{\mathcal{M}}(\Pi)$ that runs in time $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(Alg) + \mathrm{Poly}(n)))$ if A' is $\mathrm{F}(n)$-guessable.*

Proof. Let $(I, I', \textsc{Opt})$ be an input instance in $\mathcal{D}_{\mathfrak{Re}^1_{\mathcal{M}}(\Pi)}$. The approximation algorithm for $\mathfrak{Re}^1_{\mathcal{M}}(\Pi)$ returns the better one of Opt and $\mathcal{S}^{\mathcal{G}}_{Alg}(I')$. Since $\mathcal{M}$ is progressing, $\textsc{Opt} \in \mathcal{R}(I')$. Let Opt$'$ be an optimal solution for I'. Then

$$\begin{aligned} \mathit{cost}(I', \textsc{Opt}) &\underset{\text{cost pres.}}{\geq} \mathit{cost}(I, \textsc{Opt}) &(1)\\ &\underset{\text{prog. subt., optimality}}{\geq} \mathit{cost}(I, \textsc{Opt}' \setminus A') &(2)\\ &\underset{\text{prog. subt.}}{\geq} \mathit{cost}(I', \textsc{Opt}' \setminus A') &(3)\\ &\underset{\text{ps. add.}}{\geq} \mathit{cost}(I', \textsc{Opt}') - \mathit{cost}(I', A'). &(4) \end{aligned}$$

By Lemma 4, $\mathit{cost}(I', \mathcal{S}^{\mathcal{G}}_{Alg}(I')) \geq \sigma \mathit{cost}(I', \textsc{Opt}') - (\sigma - 1)\mathit{cost}(I', A')$. Simple calculations show that at least one of the two provided solutions gives an approximation ratio of $\frac{1}{2-\sigma}$. □

Note that among (1) to (4), only (4) is crucial for the theorem to hold. Hence,

Remark 1. If we are able by some means to obtain a solution with a cost bounded from below as (4) dictates, then the claim of Theorem 1 holds.

We next show how to generalize Theorem 1 to the case where we can guess a larger part of $\textsc{Opt}'$ than the part that we need for estimating the cost of $\textsc{Opt}$.

Definition 9. *A progressing modification is α-subtractive, $\alpha \leq 1$, if*

$$\begin{array}{r|c} \exists \textsc{Opt}' \in \textsc{Optima}(I') & \textsc{Opt}' \setminus A'' \in \mathcal{R}(I) \\ \exists A' \subseteq \textsc{Opt}' & cost(I, \textsc{Opt}' \setminus A'') \geq cost(I', \textsc{Opt}' \setminus A'') \\ \exists A'' \subseteq A' & cost(I', A'') \leq \alpha\, cost(I', A') \end{array}$$

Note that a 1-progressing subtractive modification is simply the progressive subtractive modification from Definition 8.1.

Corollary 3. *Let cost be pseudo-additive, $\mathcal{M}$ be cost-preserving and α-progressing subtractive and let A' be as in Definition 9. Then, there is a $\frac{1-\sigma(1-\alpha)}{1-\sigma+\alpha}$-approximation algorithm for $\mathfrak{Re}_{\mathcal{M}}(\mathit{\Pi})$ that runs in time $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(\mathit{Alg}) + \mathrm{Poly}(n)))$ if A' is $\mathrm{F}(n)$-guessable.*

Proof. Analogous to the proof of Theorem 1. In (2) we substitute A' with A''. Then (4) gives $cost(I', \textsc{Opt}) \geq cost(I', \textsc{Opt}') - cost(I', A'')$. Using the fact that $cost(I', A'') \leq \alpha\, cost(I', A')$, we obtain a lower bound on $cost(I', \textsc{Opt})$ given by $cost(I', \textsc{Opt}) \geq cost(I', \textsc{Opt}') - \alpha\, cost(I', A')$. The alternative solution, $\mathcal{S}^{\mathcal{G}}_{\mathit{Alg}}(I')$, by Lemma 3 satisfies $cost(I', \mathcal{S}^{\mathcal{G}}_{\mathit{Alg}}(I')) \geq \sigma\, cost(I', \textsc{Opt}') - (\sigma - 1) cost(I', A')$. Again simple calculations lead to the approximation ratio as claimed. □

Corollary 4. *Let the assumptions be as in Corollary 3. Then, there is a $\rho\frac{1-\sigma(1-\alpha)}{1-\sigma+\alpha\rho}$-approximation algorithm for $\mathfrak{Re}^{\rho}_{\mathcal{M}}(\mathit{\Pi})$ that runs in time $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(\mathit{Alg}) + \mathrm{Poly}(n)))$ if A' is $\mathrm{F}(n)$-guessable.*

Proof. We plug in a multiplicative factor of ρ in (2). The rest of the proof is analogous to the proof of Corollary 3. □

The use of Corollary 4 is illustrated by the following example.

Example 7. Consider *wMax-k-SAT* under clause addition: $\mathfrak{Re}^{\rho}_{\mathcal{M}}(\mathit{wMax\text{-}k\text{-}SAT})$ where $(\mathit{\Phi}, \mathit{\Phi}') \in \mathcal{M}$ if and only if $\mathit{\Phi}' = \mathit{\Phi} \wedge C_{\text{new}}$ for $C_{\text{new}} \notin \mathcal{C}(\mathit{\Phi})$. Recall that solutions to $\mathit{\Phi}$ are partial assignments and $cost(\mathit{\Phi}, \textsc{Sol}) = \sum_{C \in \mathit{Sats}(\textsc{Sol})} c(C)$. Observe that the cost function is pseudo-additive (as in Definition 4). Since $\mathit{Var}(\mathit{\Phi}) \subseteq \mathit{Var}(\mathit{\Phi}')$, a feasible assignment to $\mathit{\Phi}$ is feasible to $\mathit{\Phi}'$, and its cost does not decrease, hence $\mathcal{M}$ is progressive and cost-preserving (as in Definitions 5 and 7). To prove that it is also additive, let A' be the set of atoms in $\textsc{Opt}'$ that satisfy C_{new}, i.e., the atoms of form $(x, 1)$ for $x \in C_{\text{new}}$ and $(x, 0)$ for $\overline{x} \in C_{\text{new}}$. Clearly, $|A'| \leq k$, so $|A'|$ is n^k-guessable if n is the size of the instance. The conditions of Definition 8.1 are satisfied, as $\textsc{Opt}' \setminus A'$ is feasible to $\mathit{\Phi}$ and its cost is the same with respect to both $\mathit{\Phi}'$ and $\mathit{\Phi}$. Thus, Corollary 4 applies with $\alpha = 1$ and we obtain a $\frac{\rho}{1-\sigma+\rho}$-approximation algorithm that runs in polynomial time if k is constant. For instance, for $\mathfrak{Re}_{\mathcal{M}}(\mathit{wMax}\text{-}3\text{-}\mathit{SAT})$ this gives an approximation ratio of 0.83, whereas *wMax*-3-*SAT* admits an approximation ratio of 0.8. As a matter of fact, following Remark 1 we can apply our

Table 2. Some reoptimization problems where Corollary 4 applies. In the remarks column we state if we meet the earlier approximation ratio for the same problem. Corollary 4 gives polynomial-time algorithms even if a constant number of items is added, removed, or change their cost.

Problem	Modification	Remarks
wMaxIndSet	Edge removal	Corollary 4 applies with $\alpha = 0.5$ for edge removal; implies ratio from [8]
	Vertex addition	
wMaxCli	Edge addition	Analogous to *wMaxIndSet*; implies ratio from [8]
	Vertex addition	
wMax-k-SAT, const. k	Clause addition	See Example 7
wMaxSAT	Addition of a variable to arbitrary number of clauses	
Knapsack	Item addition	Implies ratio from [2]
ReqMaxCut	Edge cost increase	See Example 6 for self-reducibility; implies approximation algorithm for *MaxCut* under edge cost increase with the same ratio

algorithm to the *wMaxSAT* problem under addition of a clause as well. It suffices to observe that, for any atom $x_i = j \in \textsc{Opt}'$ satisfying C_{new}, it holds that $cost(I', \textsc{Opt}) \geq cost(I', \textsc{Opt}') - c(C_{\text{new}}) = cost(I', \textsc{Opt}') - cost(I', (x_i, j))$.

Table 2 shows the power of Corollary 4, i.e., we list there the problems, where Corollary 4 directly applies.

Theorem 2. *Let cost be additive, $\mathcal{M}$ be cost preserving and progressing additive and A, A' be as in Definition 8.2. Then there is a $\frac{2\sigma-1}{\sigma}$-approximation algorithm for $\mathfrak{Re}_{\mathcal{M}}(\Pi)$ that runs in time $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(Alg) + \mathrm{Poly}(n)))$ if A and A' are $\mathrm{F}(n)$-guessable.*

Proof. Let $\textsc{Opt}$ and $\textsc{Opt}'$ be the optima of I and I'. The algorithm computes two solutions $(\textsc{Opt} \setminus A) \cup A'$ and $\mathcal{S}^{\mathcal{G}}_{Alg}(I')$, and returns the better one. Let us first estimate the cost of $(\textsc{Opt} \setminus A) \cup A'$:

$$\begin{aligned}
&cost(I', (\textsc{Opt} \setminus A) \cup A') \\
&\underset{\text{ps. add.}}{\leq} cost(I', \textsc{Opt} \setminus A) + cost(I', A') \\
&\underset{\text{additivity}}{\leq} cost(I', \textsc{Opt}) - cost(I', A) + cost(I', A') \\
&\underset{\text{cost pres.}}{\leq} cost(I, \textsc{Opt}) - cost(I', A) + cost(I', A') \\
&\underset{\text{optimality}}{\leq} cost(I, (\textsc{Opt}' \setminus A'') \cup A) - cost(I', A) + cost(I', A') \\
&\underset{\text{additivity}}{\leq} cost(I', (\textsc{Opt}' \setminus A'') \cup A) - cost(I', A) + cost(I', A')
\end{aligned}$$

$$\underset{\text{ps. add.}}{\leq} \quad cost(I', \textsc{Opt}') + cost(I', A').$$

By Lemma 4, $cost(I', \mathcal{S}^{\mathcal{G}}_{Alg}(I')) \leq \sigma cost(I', \textsc{Opt}') - (\sigma - 1)cost(I', A')$. Putting the inequalities together gives the desired result. □

Theorem 2 does not generalize to the case where an approximate solution is given as a part of the input, see [8] for the proof of this fact. A straightforward example of a problem where Theorem 2 applies is the weighted minimum vertex cover problem under edge removal.

Example 8. We illustrate Theorem 2 on the example of the weighted minimum vertex cover problem under edge removal, i.e., $\mathfrak{Re}_{\mathcal{M}}(wMinVerCov)$, where $(G, G') \in \mathcal{M}$ if and only if $V(G) = V(G')$ and $E(G') = E(G) \setminus \{e\}$ for some $e \in E(G)$. For an instance G of *wMinVerCov*, a feasible solution is a set of vertices $\textsc{Sol} \subseteq V$ covering all the edges in G. This defines the set of atoms of G as $Atoms(G) = V(G)$. The cost function $cost(G, \textsc{Sol}) = \sum_{v \in \textsc{Sol}} c(v)$ is clearly additive. The self-reducibility manifests itself with the reduction function Δ which removes a vertex from the graph together with the adjacent (covered) edges: $\Delta(G, v) = G - v$. A vertex cover remains feasible when an edge is removed from the graph, hence $\mathcal{M}$ is progressing. The cost of any set of vertices is the same for G and G', hence $\mathcal{M}$ is cost-preserving. Let $\textsc{Opt}$ be an optimal solution for the original graph and $\textsc{Opt}'$ for the modified graph. We expose the sets A, A' and $A'' = \emptyset$ satisfying Definition 8.2. Adding any endpoint of the removed edge e to $\textsc{Opt}'$ makes it feasible for the original instance, as it covers e. One of the endpoints of e however, say v, must be in $\textsc{Opt}$. We set $A = \{v\}$. Observe that $v \notin \textsc{Opt}'$ implies $\Gamma_G(v) \subseteq \textsc{Opt}'$ and set $A' = \Gamma_G(v)$ to be the second augmenting set. Since both A and A' are Poly(n)-guessable, there is a polynomial-time 1.5-approximation algorithm for this variant of *wMinVerCov* reoptimization by Theorem 2 (the approximation ratio for *wMinVerCov* is $\sigma = 2$ [4]).

Other reoptimization problems where Theorem 2 applies are the minimum Steiner tree problem under removing a vertex from the terminal set and under decreasing the cost of an edge. The direct application of Theorem 2 results in approximation algorithms with $\mathcal{O}(n^{\log n + b})$ running time for both these problems where b is a constant, however parametrization allows reducing the running time to polynomial by an arbitrary small constant increase in the approximation ratio, see [18] for the details.

Definition 10. *A regressing modification $\mathcal{M}$ is* additive *(in minimization problems) if every feasible solution (possibly without some atoms) of the original instance I becomes feasible by adding a part of an optimal solution for the modified instance I':*

$$\begin{array}{r|l} \forall \textsc{Sol} \in \mathcal{R}(I) & \\ \exists \textsc{Opt}' \in \textsc{Optima}(I') & \quad (\textsc{Sol} \setminus A) \cup A' \in \mathcal{R}(I') \\ \exists A \subseteq \textsc{Sol} & cost(I', \textsc{Sol} \setminus A) \leq cost(I, \textsc{Sol} \setminus A) \\ \exists A' \subseteq \textsc{Opt}' & \end{array}$$

Theorem 3. *Let cost be pseudo-additive, $\mathcal{M}$ be reversely cost-preserving (Definitions 5 and 6) and regressing additive, and A' be as in Definition 10. Then there is a $\frac{2\sigma-1}{\sigma}$-approximation algorithm for $\mathfrak{Re}_{\mathcal{M}}(\Pi)$ with $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(Alg)+\mathrm{Poly}(n)))$ running time if A, A' are $\mathrm{F}(n)$-guessable.*

Proof. The approximation algorithm for reoptimization receives $(I, I', \textsc{Opt})$ as input. Let $\textsc{Opt}' \in \textsc{Optima}(I')$. The algorithm returns the better of $(\textsc{Opt} \setminus A) \cup A'$ and $\mathcal{S}^{\mathcal{G}}_{Alg}(I')$. Observe that, by Definitions 6 and 7.2,

$$\begin{aligned} cost(I', \textsc{Opt} \setminus A) &\underset{\text{reg. add.}}{\leq} cost(I, \textsc{Opt} \setminus A) \\ &\underset{\text{ps. add.}}{\leq} cost(I, \textsc{Opt}) \\ &\underset{\text{regressing}}{\leq} cost(I, \textsc{Opt}') \\ &\underset{\text{rev. cost p.}}{\leq} cost(I', \textsc{Opt}'). \end{aligned} \tag{5}$$

We estimate what follows based on pseudo-additivity and the above observation:

$$\begin{aligned} cost(I', \textsc{Opt} \setminus A \cup A') &\underset{\text{ps. add.}}{\leq} cost(I', \textsc{Opt} \setminus A) + cost(I', A') \\ &\underset{\text{obs.}}{\leq} cost(I', \textsc{Opt}') + cost(I', A'). \end{aligned}$$

By Lemma 3, $cost(I', \mathcal{S}^{\mathcal{G}}_{Alg}(I')) \leq \sigma cost(I', \textsc{Opt}') - (\sigma - 1) cost(I', A')$. Simple calculations show that one of the provided solutions gives an approximation ratio of $\frac{2\sigma-1}{\sigma}$. □

Corollary 5. *Let the assumptions be as in Theorem 3. Then $\mathfrak{Re}^{\rho}_{\mathcal{M}}(\mathcal{D}_{\Pi})$ for any $\rho \leq \sigma$ admits a $\frac{\sigma-\rho+\sigma\rho}{\sigma}$-approximation algorithm with $\mathcal{O}(\mathrm{F}(n)(\mathrm{Time}(Alg)+\mathrm{Poly}(n)))$ running time if A' is $\mathrm{F}(n)$-guessable.*

Proof. Plug in a factor of ρ in (5) in the proof of Theorem 3. The rest of the proof is the same. □

Below we provide an example application of Theorem 3. In addition to that Table 3 lists other problems where Corollary 5 applies.

Example 9. We illustrate Theorem 3 with the weighted minimum vertex cover problem under edge addition: $\mathfrak{Re}_{\mathcal{M}}(wMinVerCov)$. The self-reducible representation of the *wMinVerCov* problem was introduced in Example 8. The addition of an edge can make a vertex cover infeasible, but the solution in the modified instance must cover this edge with one of its endpoints. Therefore, adding this endpoint (A' satisfying Definition 10 contains only this endpoint) to the solution for the original instance makes it feasible for the modified one. Hence, Theorem 3 implies a 1.5-approximation algorithm that runs in polynomial time.

Table 3. The lists of the reoptimization problems where Corollary 5 applies. In case of the *SMT* problem, the resulting running times are sub-exponential, but can be reduced to polynomial by a simple parametrization. For the other problems, the corollary gives polynomial-time algorithms even if a constant number of items is altered.

Problem	Modification	Remarks
wMinVerCov	Edge addition	Implies ratios from [8]
	Vertex addition	
wMinDomSet	Edge removal	Implies ratio from [8]
wMinSetCov	Removal of an element from an arbitrary number of sets	See [18]
	Removal of a set of a bounded size	
	Removal of a bounded number of elements from a set	Implies ratios from [8,15]
SMT	A non-terminal becomes a terminal	See [18,19]
	Edge weight increase	
wMinFvs	Edge addition	

References

1. Archetti, C., Bertazzi, L., Speranza, M.G.: Reoptimizing the traveling salesman problem. Networks **42**(3), 154–159 (2003)
2. Archetti, C., Bertazzi, L., Speranza, M.G.: Reoptimizing the 0–1 knapsack problem. Technical report 267, University of Brescia (2006)
3. Ausiello, G., Escoffier, B., Monnot, J., Paschos, V.T.: Reoptimization of minimum and maximum traveling salesman's tours. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 196–207. Springer, Heidelberg (2006). https://doi.org/10.1007/11785293_20
4. Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem. In: Analysis and Design of Algorithms for Combinatorial Problems, vol. 109, pp. 27–45 (1985)
5. Berg, T., Hempel, H.: Reoptimization of traveling salesperson problems: changing single edge-weights. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 141–151. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00982-2_12
6. Bilò, D., et al.: Reoptimization of steiner trees. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 258–269. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69903-3_24
7. Bilò, D., et al.: Reoptimization of the shortest common superstring problem. Algorithmica **61**(2), 227–251 (2011)

8. Bilò, D., Widmayer, P., Zych, A.: Reoptimization of weighted graph and covering problems. In: Bampis, E., Skutella, M. (eds.) WAOA 2008. LNCS, vol. 5426, pp. 201–213. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-93980-1_16
9. Böckenhauer, H.-J., et al.: Reusing optimal TSP solutions for locally modified input instances. In: Navarro, G., Bertossi, L., Kohayakawa, Y. (eds.) TCS 2006. IIFIP, vol. 209, pp. 251–270. Springer, Boston, MA (2006). https://doi.org/10.1007/978-0-387-34735-6_21
10. Böckenhauer, H.-J., Hromkovič, J., Mömke, T., Widmayer, P.: On the hardness of reoptimization. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 50–65. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77566-9_5
11. Böckenhauer, H.-J., Hromkovič, J., Královič, R., Mömke, T., Rossmanith, P.: Reoptimization of steiner trees: changing the terminal set. Theor. Comput. Sci. **410**, 3428–3435 (2009)
12. Böckenhauer, H.-J., Komm, D.: Reoptimization of the metric deadline TSP. J. Discrete Algorithms **8**(1), 87–100 (2010)
13. Böckenhauer, H.-J., Freiermuth, K., Hromkovič, J., Mömke, T., Sprock, A., Steffen, B.: The steiner tree reoptimization problem with sharpened triangle inequality. In: Calamoneri, T., Diaz, J. (eds.) CIAC 2010. LNCS, vol. 6078, pp. 180–191. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13073-1_17
14. Escoffier, B., Ausiello, G., Bonifaci, V.: Complexity and approximation in reoptimization. In: Computability in Context: Computation and Logic in the Real World. Imperial College Press (2011)
15. Mikhailyuk, V.A.: Reoptimization of set covering problems. Cybern. Syst. Anal. **46**, 879–883 (2010)
16. Schäffter, M.W.: Scheduling with forbidden sets. Discrete Appl. Math. **72**(1–2), 155–166 (1997)
17. Vazirani, V.V.: Approximation Algorithms (2001)
18. Zych, A.: Reoptimization of NP-hard problems. PhD thesis, ETH Zürich (2012)
19. Zych, A., Bilò, D.: New reoptimization techniques applied to steiner tree problem. ENDM **37**, 387–392 (2011). LAGOS 2011
20. Goyal, K., Mömke, T.: Robust reoptimization of steiner trees. In: 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015) (2015)
21. Böckenhauer, H.J., Hromkovič, J., Komm, D.: Reoptimization of hard optimization problems. In: AAM Handbook of Approximation Algorithms and Metaheuristics, 2nd edn, chap. 25 (2018, to appear)

Computer Science Education

CS Unplugged—How Is It Used, and Does It Work?

Tim Bell[1(✉)] and Jan Vahrenhold[2]

[1] University of Canterbury, Christchurch, New Zealand
tim.bell@canterbury.ac.nz

[2] Westfälische Wilhelms-Universität Münster, Münster, Germany
jan.vahrenhold@uni-muenster.de

Abstract. Computer Science Unplugged has been used for many years as a way to communicate concepts from computer science to audiences in a variety of settings. Despite its widespread use, there is relatively little systematic evaluation of its effectiveness. In this paper we review what (Computer Science) Unplugged means, and the many contexts in which it has been used, as it started as an outreach tool, and then found its way into other contexts such as teacher training, and more recently into the classroom to support a formal curriculum. Formal evaluations of using Computer Science Unplugged as an approach to teaching the subject of computer science are reviewed, and we also reflect on the complex considerations that lie behind the development of activities and puzzles that are simple enough for school students to use.

Keywords: CS education · CS Unplugged · Pedagogy

1 Introduction

Computer Science Unplugged (CS Unplugged) is a widely used collection of activities and ideas to engage a variety of audiences with great ideas from computer science, without having to learn programming or even use a digital device [20]. CS Unplugged is mentioned in hundreds of papers about CS education, and appears regularly in curriculum recommendations, teacher forums, and social media.

It originated as an outreach program to engage primary school students with these ideas and especially to help them understand what computer science might involve other than programming [20]. The activities published through the csunplugged.org site have since ended up being widely used for classroom and out-of-school instruction [38], and have been translated into over 20 languages. It is frequently mentioned in books on teaching computer science [26,61], and it is used as a pedagogical technique on "coding" sites such as code.org.[1] The CS Unplugged approach appears in curriculum recommendations and designs

[1] https://code.org/educate/curriculum/cs-fundamentals-unplugged.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 497–521, 2018.
https://doi.org/10.1007/978-3-319-98355-4_29

including example activities in the 2003 ACM K-12 recommendations [108], as part of the design of a middle-years school curriculum [95], as a component of the "Exploring Computer Science" course [53], and as a resource to support the Australian Digital Technologies curriculum [43].

But being popular does not necessarily mean that it is effective as an educational tool, and some research around its effectiveness in school settings has raised doubts about whether or not it should even be used. In this paper we review the considerable literature that has developed around using and evaluating the CS Unplugged approach.

We begin in Sect. 2 by exploring what "Unplugged" has come to mean, and then in Sect. 3 we review a variety of contexts in which it is used, including outreach, broadening participation, and teacher professional development. Section 4 reviews the literature on the effectiveness of CS Unplugged when it is used for the specific purpose of teaching topics in computer science in a general classroom setting. In Sect. 5 we then consider the broader picture of how a deeper view of the CS Unplugged topics can inform those who create examples for students, and also underpin teaching the material.

2 What Is CS Unplugged?

We first need to define what we mean by "Computer Science Unplugged." The term is used in several slightly different ways, and as we analyze its effectiveness, it's important to be clear how the term is being used. The term originated with a collection of activities that were shared online from the early 1990s in various formats, culminating in 1999 with a free online book co-authored by Tim Bell, Mike Fellows and Ian Witten, called "Computer Science Unplugged: Off-line activities and games for all ages" [13]. Some formal studies refer to this original book (the "classic" edition), which was intended for academics involved in outreach; a "Teachers' edition" was published in 2000, and revised in 2002 [14]; this version was re-written in a way that classroom teachers could use it, although it was still intended as enrichment and extension exercises, and did not assume that computer science would be part of the curriculum. The material can be adapted for classroom use, but this requires consideration of issues such as linking the activities to curriculum objectives and broader issues in computing, and assessing student progress [19]. Recently the material has been developed into lesson plans for use within curricula, and this includes links to "plugging it in" activities, which are ideas for related programming exercises. Unlike the other versions, this latest version is published as a website rather than a book, although all of the versions mentioned above are available through the main CS Unplugged site.[2]

The CS Unplugged website articulates a number of principles that it is based on, including no computers required, real computer science, learning by doing, fun, and resilient activities. The "Unplugged" material is not intended to be

[2] https://csunplugged.org.

used as a curriculum or program of study, but as a form of pedagogy that has several potential benefits:

- the barrier of learning programming, which can be seen as an insurmountable hurdle by some, is removed as a prerequisite to learning about great ideas in computer science [16];
- students can engage in a meaningful way with the broader and lasting issues tackled by computer science [67], and avoid the misconception that the subject is only about programming [91], which supports a spiral curriculum where students are able to maintain an overview of where their learning is heading, rather than getting bogged down in the details of one aspect of the subject;
- it can be used in situations where computers are not available, or if they are, they can have other issues such as distracting students or causing technical issues as software must be installed and deployed in the classroom situation (e.g., teaching where access to devices is limited [73]);
- if a short time slot is available and/or there is a large audience (e.g., a 15-min outreach talk to a school or a show in a science center [11]), it is easier to engage students with a CS Unplugged exercise rather than trying to have students do programming in that setting.

New activities have been developed by other educators; many have been shared on the CS Unplugged website, but many also exist independently, including books by other authors that pick up the "Unplugged" theme [24,110].

This means that the CS Unplugged approach has become more than a particular collection of activities (which themselves are evolving anyway). The general idea of using an "Unplugged" approach for making a discipline approachable is discussed in [18]. Some specific guidelines on how CS Unplugged activities can be designed are given in [88], who identify some key principles that underpin the CS Unplugged approach, particularly:

- avoiding using computers and programming,
- a sense of play or challenge for the student to explore,
- being highly kinesthetic,
- a constructivist approach,
- short and simple explanations, and
- a sense of story.

With the variety of material appearing, the term "Unplugged" has come to be used for computer science teaching activities that do not involve programming, and so the term has come to refer to a general pedagogical approach. Due to the subtle differences between each of these forms of "Unplugged," it is important to be aware of which context is being referred to when a single activity or an event is evaluated.

3 Survey of Applications of CS Unplugged

As noted above, CS Unplugged was initially intended for use as an outreach tool, to communicate what the subject involves to those not engaged in computer science and help them become engaged with it. This has been important as new curricula are being implemented internationally that include computer science, and those implementing the new material, e.g., officials, school management, and teachers, may not have experience of it from their own education.

In this section, we review how this has led to "Unplugged" approaches being used in a variety of situations *other than* directly in classroom teaching; a review of its use in the classroom is given in the next section. This review presents a sample of reports in the literature of how CS Unplugged has been used. These papers do not usually evaluate CS Unplugged *per se*, although they often evaluate an overall program in which is was used. Thus we cannot rely on this as evidence of the effectiveness of the "Unplugged" approach, although it does expose a variety of situations where teachers and presenters have adopted the approach, which implies that their professional view was that it held promise in that situation.

CS Unplugged for Outreach. CS Unplugged was originally intended for outreach, and there are many reports in the literature of it being used for camps, special events, after-school programs, and extra-curricula activities. An "Unplugged" approach can be used in a variety of outreach situations, including contests, shows, and magazine articles [16]. A common scenario is using it as part of an event for school-age children. An "Unplugged" approach gives the opportunity to cover topics of some substance in relatively little time, which is often needed in an outreach situation; while simple steps in programming can also be covered quickly, getting to the point where a student has access to the big ideas of programming takes some time. Lonati et al. [79] raise the concern that as an initial experience, programming can be negative for students, particularly if covered in a short time—they point out that the "hour of code" risks giving an "incorrect first impression," and instead propose an "Unplugged" approach. For example, Alamer et al. [5] use CS Unplugged in a camp where there is limited time, to introduce programming concepts, Khoja et al. [71] use CS Unplugged in the introductory part of a camp for girls, and Sysło and Kwiatkowska [103] use it as part of a "children's university." The "Abenteuer Informatik" project [50] has even been used as an interactive display in shopping malls, where passers-by might only spend a matter of minutes engaged with the display. Overdorf and Lang [89] report that a group of undergraduate women designing an outreach program chose CS Unplugged activities over teaching a programming language such as Scratch or Logo, Henderson [63] reports using CS Unplugged activities successfully for enrichment courses, and Rosamond et al. [94] suggest CS Unplugged as a possible element of presenting scientific results in the media, where the reader may only spend limited time reading the article.

CS Unplugged and Magic. A common misconception is that young people know a lot about "computer science." However, it contains many unexpected elements [15], and this can be used to present magic tricks based on ideas that students do not know, yet are right in front of them in their digital world. A number of tricks based around an "Unplugged" approach have been reported [31,54]. The use of magic shows to teach CS principles is another form of Unplugged outreach [46,51]; Curzon and McOwan [30] use magic tricks to explain algorithms and even introduce algebraic proofs for verification. One of the first CS Unplugged activities developed was the "Parity" magic trick, which uses error correction to make the audience think that the presenter is a mind-reader; Hromkovič et al. [65] discuss how this can be extended into a detailed discussion of fundamental underpinnings of Computer Science, while Greenberg [54] analyzes this quintessential Unplugged magic trick in considerable detail, and comes up with variations.

CS Unplugged and Broadening Participation in CS. There is considerable interest in attracting and retaining those who are under-represented in computer science, including girls, special-needs students, ethnic minorities and otherwise under-served K-12 Students [29]. Lau and Yuen [76] suggest CS Unplugged as one of three approaches that is promising for "nurturing a gender and style sensitive classroom." Cohoon et al. [27] use CS Unplugged as part of a workshop for teachers, which resulted in teachers becoming very active in recruiting girls and minority students (although of course this effect cannot be linked to CS Unplugged alone). Stewart-Gardiner et al. [100] and Sullivan [102] report on using Unplugged-based activities as the basis of games for an after-school program for girls. CS Unplugged activities can help students with disabilities engage quickly with the subject, too; for example, activities have been adapted for visually impaired students [23,69,70]. Manabe et al. [80] report on using CS Unplugged ideas for students with disabilities (for example, the sorting network is a challenge for students in wheelchairs), while Marghitu et al. [81,82] use CS Unplugged activities as part of a camp also attended by special-needs children.

"Unplugged" Approaches to Measuring Achievement. Assessing student achievement becomes important as the idea of computational thinking (CT) enters the curriculum. Assessing CT using programming implies that students must first develop that skill, and the concern remains that the assessment will be affected by other issues such as access to devices and detailed knowledge of syntax. Several authors have proposed an "Unplugged" approach to assessing CT. One of the most widely used forms of this is the "Bebras" challenge,[3] which is a test of computational problem solving that doesn't require the use of a programming language [34,57]. Hubwieser and Mühling [68] explore the idea of using the Bebras challenge as an internationally comparable measure in the style of PISA surveys. Bell et al. [12] report student explanations in high school assessment based on an Unplugged approach, which personalizes the work (for example, a

[3] http://bebras.org.

student explaining a comparison of sorting algorithms using their own examples based on the balance scale activity from CS Unplugged). Dwyer et al. [40] use CS Unplugged to measure students' ability to work with step-by-step instructions in algorithms, and Voigt et al. [111] consider the use of CS Unplugged activities as the basis of programming competitions.

CS Unplugged Helping with Integration with Other Subjects. Since CS Unplugged activities are fast to deploy and do not require expensive specialist equipment, they have also been used as the basis of connecting computer science with other disciplines. At primary school level the connections with maths are easy to see, but teachers have also connected the activities to topics as diverse as physical education, literacy, creative writing, and art [17]. Dwyer et al. [41] used an "Unplugged" approach to connect computational thinking to physics, Carruthers et al. [25] integrated algorithms with biology, and CS Unplugged ideas have also been used to integrate computational thinking and music [9,18].

Using "Unplugged" Approaches to Make Specialised CS Topics Accessible. Sometimes traditional introductions to topics have been difficult to access for those with a weak background, particularly in programming or mathematics. A number of authors have adapted or created CS Unplugged activities to overcome these barriers. For example, Sivilotti and Pike [98] consider CS Unplugged activities in the context of distributed algorithms, and point out the value of kinesthetic activities for "increasing student engagement and promoting open classroom discussion." Heintz et al. [62] developed a course that includes the use of a CS Unplugged activity to teach HCI, and Ford et al. [47] report positive results from using an "Unplugged" approach for teaching students about cybersecurity, to make it more accessible to those with a limited technical background. Hara et al. [60] use an "Unplugged" approach to demonstrate Page Rank (as well as teaching HTML) for a writing-intensive liberal arts course.

CS Unplugged for Teacher Professional Development. Computational thinking and computer science have been introduced in various forms to curricula in many countries, and this has meant that teachers have needed professional development to be able to deliver the new subject, particularly in primary schools, where teachers are generalists and are required to cover most subjects [32]. It is not uncommon that those delivering the subject did not encounter it as part of their own study [66], and this creates a significant challenge. Being able to help teachers understand key ideas in the subject and allow them to have early success in their classroom is important. If teachers find the subject hard to become enthused about, then having it in the curriculum can backfire since students might receive a negative message about the subject in school [59]. Since learning to teach programming can be daunting for teachers with no experience in this area, a CS Unplugged introduction can help to break down defenses and overcome teachers' fears. Using an "Unplugged" approach means that relatively deep ideas can be covered fairly quickly in a playful and empowering way, using familiar materials (paper, string, chalk and so on), rather than having to install and learn to use complex software. This provides quick success and a positive experience for teachers who might otherwise find it difficult to become engaged with the subject. For example, Curzon et al. [32] used an "Unplugged" approach combined with story-telling for teacher professional development, and reported that it was "inspiring, confidence building and gave [the teachers] a greater understanding of the concepts involved." Cutts et al. [33] used an "Unplugged" approach with new material that they developed for "Enthusing and informing potential computer science students and their teachers," while Smith et al. [99] found that Master teachers (who were helping other teachers) included CS Unplugged in what they used to provide professional development for colleagues. The "MyCS" program for middle-schools, which was designed for teachers "without prior CS training" [36] drew on CS Unplugged ideas; Liu et al. [78] report CS Unplugged activities being used for teacher training in three Universities (Purdue, Carnegie Mellon, and Marquette), Pokorny and White [90] used it for a teacher workshop, and Gutiérrez and Sanders [59] suggest it as a way to break down teachers' concerns. Duncan et al. [39] found that teachers responded positively to CS Unplugged activities, and also saw curriculum integration opportunities that helped them to become engaged, but also revealed that misconceptions can arise, and that it is important to have mechanisms in place to help non-specialist teachers avoid these. Bellettini et al. [21] report that using an activity "inspired by" CS Unplugged received positive feedback from teachers, and Smith et al. [99] found that teachers valued sharing alternative ways of teaching, including CS Unplugged. Morreale and Joiner [87] surveyed which tools teachers used after attending their workshop, and found that CS Unplugged was one of the most widely adopted, particularly the lessons on binary number representations; Morreale et al. [86] included CS Unplugged in the resources that they introduced to teachers, and highlighted the importance of ideas that teachers "were able to immediately use on their return to the classroom." Sentance and Csizmadia [96] also found that when they followed up on what teachers took away from a workshop, "a significant proportion of teachers mentioned, unprompted, that they try to support students' understanding by using physical, or unplugged-style activities in the classroom."

The examples above show that an "Unplugged" approach is popular in outreach and recommended for teacher professional development, and that teachers report using it when they have found out about it. But this raises the question of whether or not it is effective in the classroom, and this question is explored in the following section.

4 Effectiveness of CS Unplugged for Teaching

The kinesthetic and playful nature of CS Unplugged activities makes them obvious candidates for outreach settings such as camps or short training events, which is reflected in the large number of experience reports regarding the effectiveness of CS Unplugged for these situations. However, despite the fact that CS Unplugged activities were suggested in the CSTA Standards for K-12 computer science [108], and that teachers report using CS Unplugged activities in class [96,107,112], surprisingly few empirical studies about the use of CS Unplugged activities in a regular classroom setting have been conducted. In this section we review the studies that exist, and reflect on how an "Unplugged" approach can be used in the classroom.

Taub et al. [104] investigated the use of CS Unplugged activities (which were designed for primary school children) in middle-school classrooms and after-class activities. Using a mixed-methods approach, they collected data regarding views, attitudes, and intentions of computer science, in particular as a field of work. The main insights from this classroom-based study was that CS Unplugged activities indeed changed the students' view of computer science towards a more mathematical thinking. However, at the same time, these students still considered computers essential to computer science and, alarmingly, were found to become less attracted to the field.

Taub et al. conjectured that these effects were due to the CS Unplugged activities being only loosely related to "central concepts" in computer science and not explicitly linked to students' prior knowledge. The latter, in fact, is a direct consequence of the activities having been developed for very young children who—by definition—cannot be expected to have much prior knowledge in computer science or mathematics to build upon. Taub et al. suggested that one should revise CS Unplugged activities to be used with older children to include the following four processes described by Linn and Eylon [77, p. 523f.]: eliciting the students' existing ideas, introducing new ideas, developing criteria for evaluating new ideas acquired, and sorting out ideas. For example, the CS Unplugged sorting activity is interesting in itself, but some effort is needed to draw out the idea that sorting algorithms are used in real life, and that the time taken by algorithms is not usually linearly proportional to the amount of data, which has a significant impact on real users when the wrong algorithm is used in a program.

The study by Feaster et al. [44] seems to confirm the above observations: in their evaluation of a one-semester course using CS Unplugged activities for even older students, i.e., students in high-school, they found that CS Unplugged materials were not suited to improving content understanding, possibly because the students viewed themselves as already expert in the subject, and also were less interested in kinesthetic activities. However, they reported "a number of experimental factors" including "adjusting for preexisting differences in student attitudes and content understanding, [and] adjusting for instructor-induced impacts" [44, p. 252] to be outside the researchers' control.

Thies and Vahrenhold [105] examined the CS Unplugged activities with respect to learning objectives for students in lower secondary education. They found that the union of the derived learning objectives did not fully cover all dimensions of Bloom's revised taxonomy [6]. However, their classification also formally showed that CS Unplugged activities indeed address objectives well suited for outreach *and* for introducing new topics in class, thus showing the applicability of CS Unplugged beyond reasons of playfulness or creating intrigue. Building upon these results, Thies and Vahrenhold investigated the effectiveness of using CS Unplugged for introducing new topics in class, comparing three activities (Binary Numbers, Binary Search, Sorting Networks) with conventional instructional methods. In a controlled experimental setting that measured short- and mid-term understanding of the concepts taught, no statistically significant differences between the two groups could be found [106]. The experiment was revalidated across different instructors, schools, and age groups; also the Treasure Hunt activity (covering finite state automata) was examined in a high-school course. Again, no statistically significant differences could be found between CS Unplugged and conventional methods [107].

Thies and Vahrenhold [107] also surveyed teachers who had attended a professional development workshop on CS Unplugged. Based upon their classroom experiences, the teachers that had used CS Unplugged in their classroom after the workshop agreed, at times very strongly, that CS Unplugged material made students curious to learn more about computer science, did not subchallenge students, and could be used across a variety of ages. Those who had not used the material in class stated that they felt uncomfortable teaching kinesthetically, feared that preparing CS Unplugged lessons would take too much time, and also that their classrooms did not allow for teaching kinesthetically; only a few explicitly mentioned that their students were either too young or too old for CS Unplugged.

Rodriguez et al. [93] followed up on the classification of the learning objectives in CS Unplugged activities [105]. They investigated which activities could be used to foster computational thinking as expressed in the following aspects: (1) Data Representation, (2) Decomposition, (3) Abstraction, (4) Algorithmic Thinking, and (5) Pattern Recognition. These activities were then reworked, for example, by adding worksheets or introducing new supplementary activities, to explicitly address computational thinking. Among other insights, they found priming activities that address naive or pre-existing ideas to be very helpful in fostering understanding, thus confirming Taub et al.'s [104] suggestion to explicitly elicit students' existing ideas. Rodriguez et al. [92] report on a project in which they adjusted not only the CS Unplugged activities but also assessment instruments to measure aspects of computational thinking. In their discussion of the results obtained, however, the authors caution against "forcing every assessment of computing skills to fit neatly within CT" [92, p. 506].

The study by Wohl et al. [114] employed a rather different assessment approach. In a comparative study on Cubelets, CS Unplugged, and Scratch in three primary schools, pupils received roughly two hours of instruction time using the respective methodology or tool. They then were asked to draw figures and to build paper models of the procedures developed. The authors found that these "could also be used as a proxy for understanding" [114, p. 57]. Combining these findings with the results from an interview study, the authors conclude that "[CS Unplugged] appeared to generate the highest level of understanding of the concepts of algorithms, logical predictions and debugging; while Cubelets proved one of the most engaging methods; and Scratch generated the most 'tool'-based questions" [114, p. 60].

Some of the negative results about the use of CS Unplugged in the classroom have been based on using it as a self-contained curriculum that completely rejects the use of computers. However, while acknowledging that sometimes it is necessary to teach regular classes without computers,[4] in a contemporary classroom setting the CS Unplugged activities are intended to be used in conjunction with conventional programming lessons.

This duality has led to discussions regarding whether or not CS Unplugged activities should be part of programming lessons. Faber et al. [42, p. 22] report that "the unplugged aspect of the lesson materials seems to elicit positive reactions from both teachers and students" and thus, even in the absence of statistical analyzes, suggest CS Unplugged activities "as a valuable alternative to regular, online programming lessons." In contrast, Grover and Pea point out that we do need to be careful with the tools that we use, and that an "Unplugged" approach "may be keeping learners away from the crucial computational experiences [such as programming] involved in CT's common practice" [56, p. 40]. A recent study, however, by Hermans and Aivaloglou [64] used a controlled-study design between two groups of students, both learning Scratch programming, but with one group spending some of the time using Unplugged material as well. They compared the groups' mastering of programming concepts, the used vocabulary of Scratch blocks, and self-efficacy. After eight weeks of instruction, the two groups of 35 elementary students in total were found to have progressed in the same way regarding mastery of programming concepts. However, the group taught using CS Unplugged material showed higher self-efficacy and used a wider vocabulary of Scratch blocks. Similarly, Gaio [49] concludes from his study of roughly 250 pupils in grades 3 and 4 that "obsessing over using just one [approach, i.e., Scratch or CS Unplugged] is not the correct decision" [49, p. 7]. This conclusion is based on the observation that CS Unplugged activities seem to come more naturally to children and lead to more "Aha! Moments," whereas connections "from [CS Unplugged] real-world-oriented tasks to computer science" seem to be difficult to establish at times.

Summing up the above discussion about whether, and if so, how, CS Unplugged can be used to teach computer science, we recall a formative statement in a paper by Fellows and Parberry at the time that Mike Fellows was developing CS Unplugged activities: "Computer science is no more about computers than astronomy is about telescopes, biology is about microscopes or chemistry is about beakers and test tubes" [45].[5] This reflects a concern at the time that the device became the object of a cargo cult mentality, but to continue the analogy, presumably one would not advocate that astronomers should not use telescopes, but simply that they treat it as an essential part of the work, but not an end in itself. More recently, as computers have become ubiquitous and inexpensive,

[4] CS Unplugged has been used in schools where computers or Internet access are not available, or where there is limited time, and at the time the activities were developed it was unusual for programming to be part of the junior curriculum [20].

[5] Versions of this quote are often attributed to Dijkstra [20].

things have moved to another extreme, where students do not understand the great ideas in the software that makes digital devices so popular [15].

In fact, while teaching computer science without using a machine is possible, the reality of modern classrooms is that good physical computational tools may well be available, and that the physical device is key to digital technology becoming such a significant part of daily life, even though it is the algorithms and data on the device that bring it to life [28]. Reflecting this new environment, the CS Unplugged material is being converted to lesson plans, which in turn are being augmented with "plugging it in" exercises to provide a way to use it with programming classes, and this is supported by the research reviewed above. The "programming vs. Unplugged" discussion should not be an "either-or" debate, but working out how to combine the two to bring about valuable learning, where each approach is used where it is most effective.

Taking this a step further, there are numerous reports on using an "Unplugged" approach as a way to teach programming, where an Unplugged explanation or walk-through is used before getting on the computer to help students understand the design of a computer program or language elements [5,37,48,97]. This is a promising pedagogical approach, as it gives students the opportunity to design their program away from the computer, rather than launching into writing code before thinking through the requirements of the whole task. It has also been used for exploring particular programming concepts; for example, Gunion et al. [58] use an "Unplugged" approach to successfully teach recursion to middle-school students.

We conclude this section with a quote from a recent literature review on the pedagogy of teaching computer science in schools: "Despite mixed evidence of the impact of unplugged activities on student learning, the approach appears to be popular with teachers as a pedagogical approach. Much more research is needed to determine how best to use it effectively" [112, p. 31].

5 Beyond Playfulness and K-12 Computer Science

Many CS Unplugged activities are appealing to young children as they can connect to the material as a Kindergarten activity, such as coloring with crayons. However, in line with the suggestion by Taub et al. [104], the activities should be embedded into the larger context of computer science. In this section, we exemplify how two topics touched upon in CS Unplugged activities can be linked to non-trivial concepts of computer science beyond the topic that is presented to the students. These relationships, in particular when used in teacher training or certification courses, can be built upon to relate K-12 computer science to more advanced concepts, thus presenting an intellectual counterpart to the perceived playfulness of the activities. Teachers can be aware of the depth of the subject they are teaching, even if the material has been simplified to a format that young students can work with.

5.1 Graph Coloring

When developing any activity or task for learners, educators need to ensure that the problems presented actually have a well-defined solution. For example, unless exploratory learning is employed, mathematics teachers in primary school need to make sure that exercises involving addition do not result in numbers outside the range students are familiar with; similarly, quadratic equations in high-school usually are required to have real-valued roots. It is easy to generate questions for students in these domains; it requires broader understanding to generate problems that are pedagogically valuable!

The playful nature of CS Unplugged activities often hides the reality that developing or extending these activities may require non-basic concepts of computer science and discrete mathematics to ensure that the examples for the children are simple or illustrate the key ideas suitably. The example we discuss in this subsection is the "Poor Cartographer" activity in which colorings of graphs are reasoned about using ideas well beyond a Kindergarten student, yet result in activities suitable for a young child. The setting of the activity is to help a cartographer color the regions of a map (planar graph) using as few colors as possible while making sure that no two regions sharing a border receive the same color.

The activity starts out by presenting maps that can be colored using exactly two colors and then touches upon the fact that any map can be colored using at most four colors. This is an obvious, non-trivial connection to advanced topics in Discrete Mathematics: in 1977, Appel, Haken, and Koch [7,8] showed by exhaustive enumeration that every planar map is four-colorable. In fact, their proof is said to be the first exhaustive proof done by a computer.[6]

However, even when starting to work with the "Poor Cartographer" activity in class, teachers need to prepare maps (graphs) that can be colored with two colors such that no two regions (faces) that share a border (edge) have the same color. The following problem in graph theory naturally arises:

> How can we construct a graph whose faces can be colored using only two colors such that no two faces that share an edge have the same color?

Surprisingly, there is a construction simple enough to be taught in class; this construction relies on the following lemma:

Lemma 1. *Any planar graph that is induced by a collection of closed curves is two-colorable.*

As a consequence of this lemma, teachers (and students) can construct graphs that are two-colorable by simply drawing overlapping circles, rectangles, or other closed curves. Thus, students can take ownership of the process by creating additional "challenges" for their peers or family at home.

[6] The field of graph coloring is still very active; indeed, the recent handbook by Gross, Yellen, and Zhang [55] contains two extensive chapters on this topic.

The proof of this lemma can be presented easily in teacher training courses. It starts by constructing the dual graph of the graph induced by the collection of closed curves. It then shows that each face of this graph has an even number of edges and completes the proof by induction on the number of faces. This proof can be used in teacher training courses to connect the formal concepts taught in undergraduate computer science courses for prospective teachers to a topic that can be taught in school using CS Unplugged.

One possible extension of this activity would be to ask whether the vertices of a given graph can be colored using a certain number of colors. Exploring this in class would also directly lead into the "Tourist Town" activity in which the focus is not on faces but on (dominating sets of) vertices. By duality, Lemma 1 carries over to the coloring of vertices of planar graphs, i.e., to color the vertices of a graph using two colors. If teachers want to give visually more challenging problems, non-planar graphs could be an option. The following is known about two-colorable graphs:

Theorem 1 (Kocay and Kreher [74] and König [75]). *A graph is two-colorable if and only if every cycle has an even number of edges.*

As there may be an exponential number of graph cycles [3,113], this theorem lends itself to nice connections from CS Unplugged to combinatorics, but enumerating and checking all cycles is infeasible in practice. However, a simple greedy algorithm based on depth-first search [2] is sufficient to check whether any given graph is two-colorable. This algorithm is exactly the greedy algorithm suggested in the CS Unplugged activity for the students to perform when coloring a graph.

For constructing a two-colorable graph, neither the above theorem nor the algorithm are helpful. However, there is a simple observation which can be used to construct a graph that is guaranteed to be two-colorable.

Observation 1. *Any bipartite graph is two-colorable.*

To construct a two-colorable graph, we thus simply take any connected bipartite graph $G = (V_1 + V_2, E)$ in which both V_1 and V_2 are non-empty, embed the graph in any way we wish, and uncolor the vertices (see Fig. 1). If the bipartite graph is drawn using a graph editor or vector graphics program, it is easy to move the vertices so that the two subsets of vertices aren't obvious visually. As $K_{3,3}$ is known to be non-planar, this method is not restricted to creating planar graphs; Fig. 1 shows a graph which contains $K_{3,3}$ and, hence, is non-planar. Incidentally, $K_{3,3}$ is also one of the few graphs for which the Ahrens' upper bound [3] on the number of cycles is tight [83].

As seen above, two-colorability can be verified (relatively) easily, and four-colorability is guaranteed for planar graphs [7,8]. It thus seems natural to also consider three-colorability. However, recognizing three-colorable graphs, i.e., ensuring that the vertices of a graph to be handed out to pupils can be colored using at most three colors, is substantially harder:

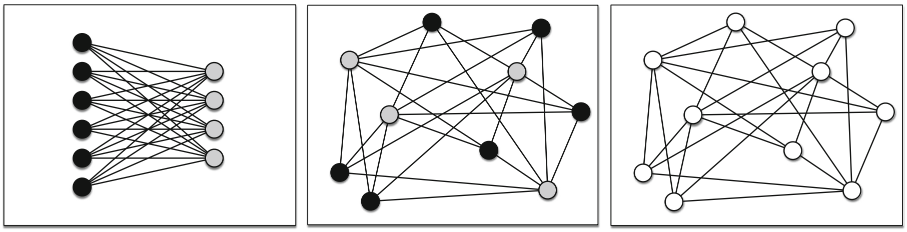


Fig. 1. Constructing a graph whose vertices are two-colorable.

Theorem 2 (Stockmeyer [101]). *Deciding whether a given graph is three-colorable, is $\mathcal{NP}$-complete.*

Fortunately, we can again use a simple construction. We start with an n-vertex simple polygon P, i.e., with an n-vertex polygon whose interior is connected and does not contain holes. We then triangulate the polygon by adding $n-3$ diagonals that partition the interior of P into disjoint triangles, which is done by connecting vertices in a way such that no diagonals intersect (see Fig. 2). This process, while very involved if one aims at an efficient solution [109], can be done easily with a greedy algorithm that iteratively cuts off "ears" from the polygon [85].

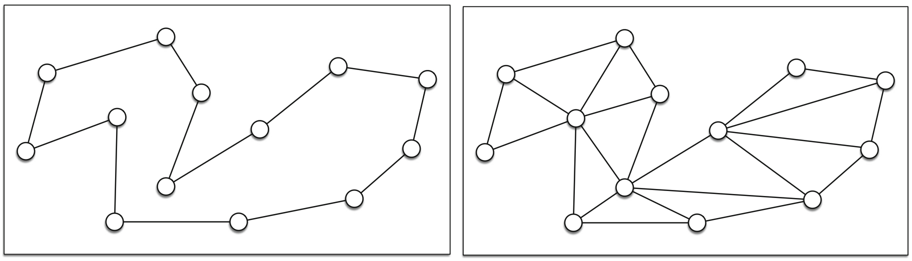


Fig. 2. A simple polygon and its triangulation.

The textbook by de Berg et al. [35] derives the following lemma from Meister's *two-ear* theorem [85]:

Lemma 2. *Any planar graph that is induced by a triangulation of a simple polygon is three-colorable.*

The idea behind the proof is to show that the dual graph of the triangulation (excluding the outer face) is a tree. Based upon this, the correctness of the following procedure can be established [35]: Start at any triangle of the triangulation and color its vertices using exactly three colors. Perform a depth-first exploration of the tree rooted at the vertex dual to this face. When visiting a new vertex of this tree, i.e., a triangle of the triangulation, color the one uncolored vertex using the one color not used for coloring the vertices of the triangulation edge dual to the tree edge just traversed.

To construct visually "challenging" graphs whose vertices are guaranteed to be three-colorable, teachers can use the algorithm implied by the above proof to construct a three-coloring of a polygon's triangulation. They then can add arbitrary edges connecting vertices of different colors to obfuscate the original polygonal structure and then uncolor all vertices. By construction, the resulting graph is three-colorable. An example is shown in Fig. 3.

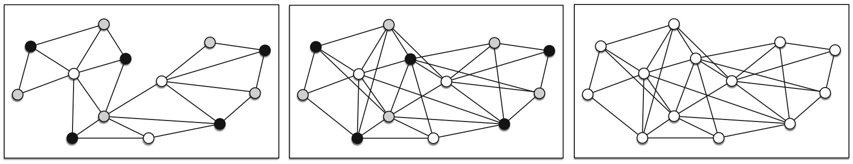


Fig. 3. Three-coloring and augmenting the triangulation of a simple polygon.

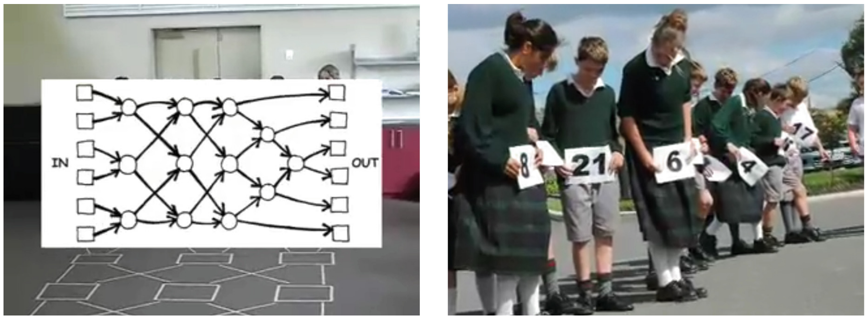


Fig. 4. The "Beat the Clock" activity in a classroom (left) and outdoors (right).

In conclusion, the "Poor Cartographer" activity, while seemingly playful, requires a relatively deep understanding of formal concepts if teachers want to generate interesting maps, and even slightly expand the topics presented on the worksheet. If they chose to do so, however, they can open a window into much richer topics in computer science and discrete mathematics than can be covered in K-12 computer science, thus relating the activity to tertiary education and research.

5.2 Sorting Networks

The second example we discuss is the "Beat the Clock" activity. This activity allows children to explore small sorting networks by enacting a sorting algorithm using a sorting network drawn on the floor. The students' excitement when performing this activity has been captured prominently in the CS Unplugged video clips on the main website, showing the activity both in a classroom and outdoors.

The sorting network has become a popular activity with teachers, as it combines many key elements including simplicity of the explanation, physical movement, competition to complete it quickly, and intrigue for the students. It has been used for many purposes other than illustrating parallel algorithms, as it sets up a structure where students are motivated to compare values repeatedly. It can be used for any binary relation that is a total order, including musical notes (both written and sounded), text such as names, calculated numeric values such as fractions or arithmetic combinations of numbers, or sequences in nature (such as an egg transforming to a butterfly, or growth of a seed into a plant). It can even be used to memorize sequences; for example, we have seen it used to memorize a whakataukī (Māori proverb) by breaking the proverb into six elements, and having the group recite the entire whakataukī, with the students making the pairwise comparisons, noting in which order their components come in during the recitation.

Interestingly, the "Beat the Clock" activity is the only activity of those investigated by Thies and Vahrenhold [106,107] for which no statistically significant difference between traditional and unplugged teaching could be found, neither short-term nor long-term. Thies and Vahrenhold attribute this to the visual representation used in the assessments serving as a sub-conscious reminder of the procedure. A closer look, however, reveals that an additional ceiling effect might have also played a role, i.e., that the principles behind using a sorting network were both easy to grasp and easy to remember. Thus the value of this activity for integrated learning seems to be high, including the physical exercise that students get on a large network, and it is also useful as an outreach tool for quickly helping students to understand that computer science explores concepts that they may not be aware of, even if the fundamental concept can be grasped without the physical activity.

The relevance of this activity for connecting K-12 computer science and teacher training to computer science research is three-fold. The obvious connection is to the construction of sorting networks. While it is possible to construct an optimal sorting network, i.e., a sorting network that is guaranteed to use no more than $\mathcal{O}(\log n)$ parallel steps to sort n numbers [4], this algorithm is almost exclusively used as a black box to prove optimality of certain other constructions, e.g., ε-nets [84], or protection against learning a software program from its execution [52], and (conference and journal versions combined) it has been cited over 1,500 times according to Google Scholar. The construction and analysis of sorting networks is still an active research area—see, for example, Bundala et al. [22] and the references therein.

In recent years sorting networks have become immensely relevant in practice. As sorting networks are data-oblivious in the sense that the sequence of steps taken does not depend on the input data, they are used as building blocks of sorting algorithms for general-purpose graphic-processing units (GPUs). These chips have multiple processors (cores) accessing a shared memory in parallel; for reasons of efficiency, these parallel accesses should avoid simultaneously working on the same memory bank. To obtain such *bank-conflict free* access patterns, it is most helpful to know that an algorithm is guaranteed to perform a certain step at a certain time, and data-oblivious algorithms such as those induced by sorting networks have exactly this property. Batcher's *even-odd mergesort* [10] laid the foundation for fast sorting algorithms used on current general-purpose GPUs [1]. In fact, Batcher already envisioned back in 1968 the use of his algorithm on "a switching network with buffering, a multiaccess memory, [and] a multiaccess content-addressable memory" [10, p. 307]. His paper gives an inductive construction which may be used as a source for additional examples in class or as an example for formal reasoning about the correctness of sorting networks in teacher training. Hence, by discussing the "Beat the Clock" activity, teachers can relate to both theoretically and practically relevant aspects of computer science far beyond the classroom.

The final connection that can be made, at least in teacher training, is to the verification of sorting networks. In principle, one would need to verify the correctness of any given sorting network for n input lines by checking against all $n! \in \Theta(n^n)$ permutations of a set of n distinct values. However, the following Zero-One Principle shows that it is sufficient to check against all 2^n inputs consisting of zeros and ones only.

Theorem 3 (Knuth [72, **Theorem Z, p. 223]).** *If a network with n input lines sorts all 2^n sequences of 0s and 1s into nondecreasing order, it will sort any arbitrary sequence of n numbers into nondecreasing order.*

The proof of this theorem can be used, for example, in teacher training, to show an elegant use of contradiction: Assuming that the output contains an inversion $x_i > x_{i+1}$, one can construct a $\{0, 1\}$-valued indicator function f signaling whether a given input value is smaller than x_i. As f is monotonic, this leads to the existence of a $\{0, 1\}$-valued input that is not sorted correctly and, thus, to the desired contradiction.

While we might not use this proof with school students, they can reason about simpler issues, such as whether the minimum value is guaranteed to find its way to the left-hand end (and vice versa for the maximum) for the six-way sorting network (Fig. 4) by exhaustively checking the six possible starting positions.

Most examples in this section are too involved to be discussed in a standard classroom. However, they show that much more theory hides behind the playfulness of CS Unplugged than is visible at first glance, especially if teachers choose for didactic reasons to provide additional exercises or extensions. Among other implications, this prominently underlines the value of a thorough formal training of prospective teachers not only in educational and didactic matters but also in the scientific foundations of the discipline.

6 Conclusions

The CS Unplugged approach has become a pedagogical method that has been used for many purposes. The value of CS Unplugged for students is that it engages them with lasting ideas in our field, although the research shows that it needs to be linked to current technology and should not just be used in isolation. This also makes sense in modern curricula where programming is increasingly expected to be taught in the classroom, and the CS Unplugged approach gives opportunities to have a spiral approach rather than waiting until students have the ability to, say, program a GPU before exploring and implementing parallel sorting algorithms. Some of the literature treats the question of using CS Unplugged as an "either/or" issue, but experience is showing that if it is integrated with teaching programming then it can assist with both the motivation to explore the field as well as helping students with their "plugged-in" learning.

Focusing exclusively on whether CS Unplugged is good for students also misses an important point: it seems that it is good for teachers. It is widely used in teacher professional development, and teachers seem to find it helpful to get engaged with this topic that they are not otherwise familiar with. When computer science was not part of school curricula it was generally delivered in extra-curricula events by enthusiasts, but now that it is becoming ubiquitous, especially in primary schools, the fact that CS Unplugged is widely reported as being embraced by teachers is significant, given the influence of having an enthusiastic teacher on students' attitude towards a subject. As recent results suggest that some parts of the curriculum can be taught "Unplugged" without detrimental effects on the thoroughness of understanding, having a more enthusiastic teacher is an added benefit that can make a real difference.

Furthermore, the ideas that can be explored in an "Unplugged" mode have considerable depth, and teachers can delve into them well beyond what they might use in the classroom with students; and those developing CS Unplugged activities may find that they need to address deeper questions about the topic in order to generate examples that come across as simple explanations. For example, being sure of the chromatic number of a graph given as an example to students involves considerably more understanding than the rules that the students are given to understand the problem.

The CS Unplugged approach is used in many settings and like all approaches, if used inappropriately, it can be ineffective or even cause harm. This review, however, has identified many reports on ways that it can be used to achieve positive results, helping both students and teachers to explore computer science in a meaningful and engaging way.

References

1. Afshani, P., Sitchinava, N.: Sorting and permuting without bank conflicts on GPUs. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 13–24. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_2
2. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
3. Ahrens, W.: Ueber das Gleichungssystem einer Kirchhoff'schen galvanischen Stromverzweigung. Math. Ann. **49**(2), 311–324 (1897)
4. Ajtai, M., Komlós, J., Szemerédi, E.: Sorting in $c \log n$ parallel steps. Combinatorica **3**(1), 1–19 (1983)
5. Alamer, R.A., Al-Doweesh, W.A., Al-Khalifa, H.S., Al-Razgan, M.S.: Programming unplugged: bridging CS unplugged activities gap for learning key programming concepts. In: Proceedings - 2015 5th International Conference on e-Learning, ECONF 2015, pp. 97–103 (2016)
6. Anderson, L.W., et al.: A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Abridged edn. Addison Wesley Longman, Boston (2001)
7. Appel, K.I., Haken, W.: Every planar map is four colorable. I. Disch. Ill. J. Math. **21**, 429–490 (1977)
8. Appel, K.I., Haken, W., Koch, J.: Every planar map is four colorable. II. Reduc. Ill. J. Math. **21**, 491–567 (1977)
9. Baratè, A., Ludovico, L.A., Malchiodi, D.: Fostering Computational Thinking in Primary School through a LEGO®-based Music Notation. Procedia Comput. Sci. **112**, 1334–1344 (2017)
10. Batcher, K.E.: Sorting networks and their applications. In: Proceedings of the Spring Joint Computer Conference. American Federation of Information Processing Societies, vol. 32, pp. 307–314 (1968)
11. Bell, T.: A low-cost high-impact computer science show for family audiences. In: Proceedings - 23rd Australasian Computer Science Conference, ACSC 2000, pp. 10–16 (2000)
12. Bell, T., Newton, H., Andreae, P., Robins, A.: The introduction of computer science to NZ high schools: an analysis of student work. In: Proceedings of the 7th Workshop in Primary and Secondary Computing Education - WiPSCE 2012, pp. 5–15 (2012)
13. Bell, T., Witten, I.H., Fellows, M.: Computer Science Unplugged: Off-Line Activities and Games for All Ages (Original Book) (1999). http://csunplugged.org
14. Bell, T., Witten, I.H., Fellows, M., McKenzie, J., Adams, R.: Computer Science Unplugged: An Enrichment and Extension Programme for Primary-Aged Children (2002). http://csunplugged.org
15. Bell, T.: Surprising computer science. In: Brodnik, A., Vahrenhold, J. (eds.) ISSEP 2015. LNCS, vol. 9378, pp. 1–11. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25396-1_1
16. Bell, T., Curzon, P., Cutts, Q., Dagiene, V., Haberman, B.: Overcoming obstacles to CS education by using non-programming outreach programmes. In: Kalaš, I., Mittermeir, R.T. (eds.) ISSEP 2011. LNCS, vol. 7013, pp. 71–81. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24722-4_7
17. Bell, T., Duncan, C., Atlas, J.: Teacher feedback on delivering computational thinking in primary school. In: Proceedings of the 11th Workshop in Primary and Secondary Computing Education - WiPSCE 2016, pp. 100–101 (2016)

18. Bell, T., Fellows, M., Rosamond, F., Bell, J., Marghitu, D.: Unplugging education: removing barriers to engaging with new disciplines. In: Proceedings SDPS Transdisciplinary Conference on Integrated Sytems, Design, and Process Science, 10–14 June, Berlin, Germany, SDPS, pp. 168-1–168-8 (2012)
19. Bell, T., Newton, H.: Unplugging computer science. In: Angeli, C., Kadijevich, D., Schulte, C. (eds.) Improving Computer Science Education. Routledge, New York (2013)
20. Bell, T., Rosamond, F., Casey, N.: Computer science unplugged and related projects in math and computer science popularization. In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday. LNCS, vol. 7370, pp. 398–456. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30891-8_18
21. Bellettini, C., et al.: Informatics education in Italian secondary schools. ACM Trans. Comput. Educ. **14**(2), 1–6 (2014)
22. Bundala, D., Codish, M., Cruz-Filipe, L., Schneider-Kamp, P., Závodný, J.: Optimal sorting networks. J. Comput. Syst. Sci. **84**(C), 185–204 (2017)
23. Burgstahler, S.E., Ladner, R.E., Bellman, S.: Strategies for increasing the participation in computing of students with disabilities. ACM Inroads **3**(4), 42–48 (2012)
24. Caldwell, H., Smith, N.: Teaching Computing Unplugged in Primary Schools. SAGE Publications, Thousand Oaks (2016)
25. Carruthers, S., Gunion, K., Stege, U.: Computational biology unplugged! In: Proceedings of the 14th Western Canadian Conference on Computing Education - WCCCE 2009, p. 126 (2009)
26. Clarke, B.: Computer Science Teacher. British Computer Society, Swindon (2017)
27. Cohoon, J., Cohoon, J., Soffa, M.: Focusing high school teachers on attracting diverse students to computer science and engineering. In: ASEE/IEEE Frontiers in Education Conference, pp. 1–5 (2011)
28. Connor, R., Cutts, Q., Robertson, J.: Keeping the machinery in computing education. Commun. ACM **60**(11), 26–28 (2017)
29. Cortina, T.J.: Reaching a broader population of students through "unplugged" activities. Commun. ACM **58**(3), 25–27 (2015)
30. Curzon, P., McOwan, P.: Teaching formal methods using magic tricks. In: Fun with Formal Methods: Workshop at the 25th International Conference on Computer Aided Verification, number 122 (2013)
31. Curzon, P., McOwan, P.W.: Engaging with computer science through magic shows. In: Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE 2008, pp. 179–183. ACM, New York (2008)
32. Curzon, P., McOwan, P.W., Plant, N., Meagher, L.R.: Introducing teachers to computational thinking using unplugged storytelling. In: Proceedings of the 9th Workshop in Primary and Secondary Computing Education - WiPSCE 2014, pp. 89–92 (2014)
33. Cutts, Q., Brown, M., Kemp, L.: Enthusing and informing potential computer science students and their teachers. ACM SIGCSE Bull. **39**, 196–200 (2007)
34. Dagienė, V., Stupurienė, G., Vinikienė, L.: Promoting inclusive informatics education through the Bebras Challenge to all K-12 students. In: Proceedings of the 17th International Conference on Computer Systems and Technologies 2016 - CompSysTech 2016, pp. 407–414. ACM (2016)

35. de Berg, M., Cheong, O., van Kreveld, M.J., Overmars, M.H.: Computational Geometry: Algorithms and Applications. Springer, Berlin (2008). https://doi.org/10.1007/978-3-540-77974-2
36. Dodds, Z., Erlinger, M.: MyCS: building a middle-years CS curriculum. In: Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2013, p. 330 (2013)
37. Dorling, M., White, D.: Scratch: a way to logo and python. In: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, pp. 191–196 (2015)
38. Duncan, C., Bell, T.: A pilot computer science and programming course for primary school students. In: WIPSCE 2015, pp. 39–48 (2015)
39. Duncan, C., Bell, T., Atlas, J.: What do the teachers think? Introducing computational thinking in the primary school curriculum. In: Proceedings of the Nineteenth Australasian Computing Education Conference, pp. 65–74 (2017)
40. Dwyer, H., Hill, C., Carpenter, S., Harlow, D., Franklin, D.: Identifying elementary students' pre-instructional ability to develop algorithms and step-by-step instructions. In: Proceedings of the 45th ACM Technical Symposium on Computer Science Education - SIGCSE 2014, pp. 511–516 (2014)
41. Dwyer, H.A., Boe, B., Hill, C., Franklin, D., Harlow, D.: Computational thinking for physics: programming models of physics phenomenon in elementary school. In: 2013 Physics Education Research Conference Proceedings, pp. 133–136 (2013)
42. Faber, H.H., Wierdsma, M.D.M., Doornbos, R.P., Van Der Ven, J.S., De Vette, K.: Teaching computational thinking to primary school students via unplugged programming lessons. J. Eur. Teach. Educ. Netw. **12**, 13–24 (2017)
43. Falkner, K., Vivian, R.: A review of computer science resources for learning and teaching with K-12 computing curricula: an Australian case study. Comput. Sci. Educ. **25**(4), 390–429 (2015)
44. Feaster, Y., Segars, L., Wahba, S.K., Hallstrom, J.O.: Teaching CS unplugged in the high school (with limited success). In: Rößling, G., Naps, T.L., Spannagel, C. (eds.) Proceedings of the 16th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2011, Darmstadt, Germany, 27–29 June 2011, pp. 248–252. ACM (2011)
45. Fellows, M., Parberry, I.: SIGACT trying to get children excited about CS. Comput. Res. News, 7 (1993)
46. Ferreira, J.F., Mendes, A.: The magic of algorithm design and analysis: teaching algorithmic skills using magic card tricks. In: Proceedings of the 19th ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2014, pp. 75–80 (2014)
47. Ford, V., Siraj, A., Haynes, A., Brown, E.: Capture the flag unplugged. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education - SIGCSE 2017, pp. 225–230 (2017)
48. Fowler, A.: Engaging young learners in making games. In: Proceedings of the International Conference on the Foundations of Digital Games - FDG 2017, pp. 1–5 (2017)
49. Gaio, A.: Programming for 3rd graders, scratch-based or unplugged? In: 10th Congress of European Research in Mathematics Education (CERME), TWG 22 (2017)
50. Gallenbacher, J.: Abenteuer informatik: hands-on exhibits for learning about computational thinking. In: Proceedings of the 7th Workshop in Primary and Secondary Computing Education - WiPSCE 2012, p. 149 (2012)

51. Garcia, D.D., Ginat, D.: Demystifying computing with magic, part III. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE 2016, pp. 158–159 (2016)
52. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM **43**(3), 431–473 (1996)
53. Goode, J., Margolis, J.: Exploring computer science. ACM Trans. Comput. Educ. **11**(2), 1–16 (2011)
54. Greenberg, R.I.: Educational magic tricks based on error-detection schemes. In: Proceedings of the 22nd ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2017, pp. 170–175 (2017)
55. Gross, J.L., Yellen, J., Zhang, P.: Handbook of Graph Theory, 2nd edn. Chapman and Hall/CRC Press, Boca Raton (2018)
56. Grover, S., Pea, R.: Computational thinking in K-12: a review of the state of the field. Educ. Res. **42**(1), 38–43 (2013)
57. Gujberova, M., Kalas, I.: Designing productive gradations of tasks in primary programming education. In: Proceedings of the 8th Workshop in Primary and Secondary Computing Education, pp. 108–117. ACM (2013)
58. Gunion, K., Milford, T., Stege, U.: The paradigm recursion: is it more accessible when introduced in middle school? J. Probl. Solving **2**(2), 142–172 (2009)
59. Gutiérrez, J.M., Sanders, I.D.: Computer science education in Perú: a new kind of monster? ACM SIGCSE Bull. **41**(2), 86–89 (2009)
60. Hara, K.J.O., Burke, K., Ruggiero, D., Anderson, S.: Linking language and thinking with code: computing within a writing-intensive introduction to the liberal arts. In: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, pp. 269–274. ACM (2017)
61. Hazzan, O., Lapidot, T., Ragonis, N.: Guide to Teaching Computer Science: An Activity-Based Approach. Springer, London (2011). https://doi.org/10.1007/978-0-85729-443-2
62. Heintz, F., Mannila, L., Färnqvist, T.: A review of models for introducing computational thinking, computer science and computing in K-12 education. In: Proceedings - Frontiers in Education Conference (FIE), pp. 1–9 (2016)
63. Henderson, P.B.: Computing unplugged enrichment. ACM Inroads **2**(3), 24–25 (2011)
64. Hermans, F., Aivaloglou, E.: To scratch or not to scratch?: a controlled experiment comparing plugged first and unplugged first programming lessons. In: Proceedings of the 12th Workshop on Primary and Secondary Computing Education, WiPSCE 2017, pp. 49–56. ACM, New York (2017)
65. Hromkovič, J., Keller, L., Komm, D., Serafini, G., Steffen, B.: Fehlerkorrigierende Codes - Ein Unterrichtsbeispiel zum gelenkten entdeckenden Lernen. LOG IN: Informatik Sch. Ausbildung **168**, 50–55 (2011)
66. Hromkovič, J., Lacher, R.: How to convince teachers to teach computer science even if informatics was never a part of their own studies. Bull. Eur. Assoc. Theor. Comput. Sci. (BEATCS) **123**, 120–125 (2017)
67. Hromkovič, J., Lacher, R.: The computer science way of thinking in human history and consequences for the design of computer science curricula. In: Dagiene, V., Hellas, A. (eds.) ISSEP 2017. LNCS, vol. 10696, pp. 3–11. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71483-7_1
68. Hubwieser, P., Mühling, A.: Playing PISA with bebras. In: Proceedings of the 9th Workshop in Primary and Secondary Computing Education, WiPSCE 2014, pp. 128–129. ACM, New York (2014)

69. Jašková, L.: Blind pupils begin to solve algorithmic problems. In: Diethelm, I., Mittermeir, R.T. (eds.) ISSEP 2013. LNCS, vol. 7780, pp. 68–79. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36617-8_6
70. Jašková, L., Kaliaková, M.: Programming microworlds for visually impaired pupils. In: Proceedings of the 3rd International Constructionism Conference, pp. 1–10 (2014)
71. Khoja, S., Wainwright, C., Brosing, J., Barlow, J.: Changing girls' attitudes towards computer science. J. Comput. Sci. Coll. **28**(1), 210–216 (2012)
72. Knuth, D.E.: Sorting and Searching. The Art of Computer Programming, vol. 3. Addison-Wesley, Reading (1998)
73. Koblitz, N.: Crypto galore! In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday. LNCS, vol. 7370, pp. 39–50. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30891-8_3
74. Kocay, W., Kreher, D.L.: Graphs, Algorithms, and Optimization. Discrete Mathematics and Its Applications. Chapman & Hall/CRC, Boca Raton (2005)
75. König, D.: Theorie der endlichen und unendlichen Graphen: Kombinatorische Topologie der Streckenkomplexe. Akademische Verlagsgesellschaft, Leipzig (1936)
76. Lau, W.W.F., Yuen, A.H.K.: Gender differences in learning styles: nurturing a gender and style sensitive computer science classroom. Australas. J. Educ. Technol. **26**(7), 1090–1103 (2010)
77. Linn, M.C., Eylon, B.S.: Science education: integrating views of learning and instruction. In: Alexander, P.A., Winne, P.H. (eds.) Handbook of Educational Psychology, pp. 511–544. Routledge, New York (2006)
78. Liu, J., Hasson, E.P., Barnett, Z.D., Zhang, P.: A survey on computer science K-12 outreach: teacher training programs. In: Proceedings - Frontiers in Education Conference, FIE, pp. 11–16 (2011)
79. Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A.: Is coding the way to go? In: Brodnik, A., Vahrenhold, J. (eds.) ISSEP 2015. LNCS, vol. 9378, pp. 165–174. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25396-1_15
80. Manabe, H., Kanemune, S., Namiki, M., Nakano, Y.: CS unplugged assisted by digital materials for handicapped people at schools. In: Kalaš, I., Mittermeir, R.T. (eds.) ISSEP 2011. LNCS, vol. 7013, pp. 82–93. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24722-4_8
81. Marghitu, D., et al.: Auburn university robo camp K12 inclusive outreach program: a three-step model of effective introducing middle school students to computer programming and robotics. In: Society for Information Technology and Teacher Education International Conference, Association for the Advancement of Computing in Education (AACE), pp. 58–63 (2013)
82. Marghitu, D., Zylla-Jones, E.: Educational and technological computer literacy program for typical and special needs children: an auburn university case study. In: Crawford, C.M., Carlsen, R., McFerrin, K., Price, J., Weber, R., Willis, D.A. (eds.) Proceedings of Society for Information Technology and Teacher Education International Conference 2006, Orlando, Florida, USA, Association for the Advancement of Computing in Education (AACE), pp. 2326–2331, March 2006
83. Mateti, P., Deo, N.: On algorithms for enumerating all circuits of a graph. SIAM J. Comput. **5**(1), 90–99 (1976)
84. Matoušek, J.: Construction of ε-nets. Discrete Comput. Geom. **5**, 427–448 (1990)
85. Meisters, G.H.: Polygons have ears. Am. Math. Mon. **82**(6), 648–651 (1975)

86. Morreale, P., Jimenez, L., Goski, C., Stewart-Gardiner, C.: Measuring the impact of computational thinking workshops on high school teachers. J. Comput. Sci. Coll. **27**(6), 151–157 (2012)
87. Morreale, P., Joiner, D.: Reaching future computer scientists. Commun. ACM **54**(4), 121 (2011)
88. Nishida, T., Kanemune, S., Idosaka, Y., Namiki, M., Bell, T., Kuno, Y.: A CS unplugged design pattern. In: Fitzgerald, S., Guzdial, M., Lewandowski, G., Wolfman, S.A. (eds.) Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2009, Chattanooga, TN, USA, pp. 231–235. ACM (2009)
89. Overdorf, R., Lang, M.: Reaching out to aid in retention. In: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education - SIGCSE 2011, p. 583 (2011)
90. Pokorny, K.L., White, N.: Computational thinking outreach: reaching across the K-12 curriculum. J. Comput. Sci. Coll. **27**(5), 234–242 (2012)
91. Prieto-Rodriguez, E., Berretta, R.: Digtial technology teachers' perceptions of computer science: it is not all about programming. In: Frontiers in Education Conference (FIE), pp. 1–5 (2014)
92. Rodriguez, B., Kennicutt, S., Rader, C., Camp, T.: Assessing computational thinking in CS unplugged activities. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2017, pp. 501–506. ACM, New York (2017)
93. Rodriguez, B., Rader, C., Camp, T.: Using student performance to assess CS unplugged activities in a classroom environment. In: Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2016, pp. 95–100. ACM, New York (2016)
94. Rosamond, F., et al.: Reaching out to the media. Commun. ACM **54**(3), 113 (2011)
95. Schofield, E., Erlinger, M., Dodds, Z.: MyCS: a CS curriculum for middle-years students. J. Comput. Sci. Coll. **29**(4), 145–155 (2014)
96. Sentance, S., Csizmadia, A.: Computing in the curriculum: challenges and strategies from a teacher's perspective. Educ. Inf. Technol. **22**(2), 469–495 (2017)
97. Settle, A., et al.: Infusing computational thinking into the middle- and high-school curriculum. In: Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE 2012, p. 22 (2012)
98. Sivilotti, P.A.G., Pike, S.M.: A collection of kinesthetic learning activities for a course on distributed computing. SIGACT News **38**(2), 57–74 (2007)
99. Smith, N., Allsop, Y., Caldwell, H., Hill, D., Dimitriadi, Y., Csizmadia, A.: Master teachers in computing: what have we achieved? In: Proceedings of the Workshop in Primary and Secondary Computing Education, pp. 21–24 (2015)
100. Stewart-Gardiner, C., Carmichael, G., Gee, E., Hopping, L.: Girls learning computer science principles with after school games. In: Proceedings of the Third Conference on GenderIT, pp. 62–63. ACM (2015)
101. Stockmeyer, L.: Planar 3-colorability is polynomial complete. ACM SIGACT News **5**(3), 19–25 (1973)
102. Sullivan, K., Byrne, J.R., Bresnihan, N., O'Sullivan, K., Tangney, B.: CodePlus - designing an after school computing programme for girls. In: 2015 IEEE Frontiers in Education Conference (FIE), pp. 1–5 (2015)
103. Sysło, M.M., Kwiatkowska, A.B.: Playing with computing at a children's university. In: Proceedings of the 9th Workshop in Primary and Secondary Computing Education - WiPSCE 2014, pp. 104–107. ACM Press, New York, November 2014

104. Taub, R., Armoni, M., Ben-Ari, M.: CS unplugged and middle-school students' views, attitudes, and intentions regarding CS. Trans. Comput. Educ. **12**(2), 8:1–8:29 (2012)
105. Thies, R., Vahrenhold, J.: Reflections on outreach programs in CS classes: learning objectives for "unplugged" activities. In: SIGCSE 2012: Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, pp. 487–492 (2012)
106. Thies, R., Vahrenhold, J.: On plugging "unplugged" into CS classes. In: SIGCSE 2013: Proceedings of the 44th ACM Technical Symposium on Computer Science Education, pp. 365–370 (2013)
107. Thies, R., Vahrenhold, J.: Back to school: computer science unplugged in the wild. In: Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2016, pp. 118–123. ACM Press, New York (2016)
108. Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., Verno, A.: A model curriculum for K-12 computer science: final report of the ACM K-12 task force curriculum committee. ACM, New York (2003)
109. Vahrenhold, J.: Polygon triangulation. In: Kao, M.Y. (ed.) Encyclopedia of Algorithms, 2nd edn. Springer, New York (2015)
110. Vöcking, B., et al. (eds.): Algorithms Unplugged. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-15328-0
111. Voigt, J., Bell, T., Aspvall, B.: Competition-style programming problems for Computer Science Unplugged activities. In: Verdu, E., Lorenzo, R., Revilla, M., Regueras, L. (eds.) A New Learning Paradigm: Competition Supported by Technology, pp. 207–234. CEDETEL, Boecillo (2009)
112. Waite, J.: Pedagogy in teaching computer science in schools: a literature review, London (2017)
113. Weisstein, E.W.: Graph cycle. From MathWorld-A Wolfram Web Resource. http://mathworld.wolfram.com/GraphCycle.html
114. Wohl, B., Porter, B., Clinch, S.: Teaching computer science to 5–7 year-olds: an initial study with scratch, cubelets and unplugged computing. In: Proceedings of the Workshop in Primary and Secondary Computing Education, pp. 55–60 (2015)

Resurgence of Informatics Education in Schools
A Way to a Deeper Understanding of Informatics Concepts

Valentina Dagienė(✉)

Vilnius University, Vilnius, Lithuania
valentina.dagiene@mii.vu.lt

Abstract. Five decades ago, computing at several high schools was consistent with the academic and professional world. Nowadays, many countries have started to think about how to establish or re-establish informatics education in schools and how to attract children to learn informatics. Although informatics is not taught as a discipline in many countries, children are invited to participate in different contests and competitions in informatics, programming, or robotics organized all over the world. Educators are interested in motivating students to learn more about informatics, to understand informatics concepts, and to develop computational thinking. This paper discusses informatics education through the recent decades with its main focus on Logo activities and other strategies which support deeper understanding and promote attractive approaches of learning informatics in school. The worldwide Bebras informatics challenge is presented and discussed as an example of connecting formal and non-formal informatics education by using thousands of tasks based on informatics concepts and applying problem-solving strategies. As a case study, the statistical data from the 2017 Lithuanian Bebras challenge is presented and discussed.

1 Introduction

Significant changes in society do not begin at one particular day or even in a particular year. Changes come slowly, especially in school education. Teachers, policy makers, and researchers should work continuously for decades in order to gain significant results on children's achievements in informatics.

Teaching informatics (computer science or computing) in high schools started about four decades ago with the introduction of programming. Machines at that time were complicated to handle and programming was the exceptional possibility to manage them. The goal of teaching programming at schools was to show how a computer can help in routine work, mainly doing huge amounts of calculations. Computer programming is the best way to build a language for instructing (communicating with) a machine. Later, Juraj Hromkovič and others went deeper and declared: "We have to teach programming as a skill to

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 522–537, 2018.
https://doi.org/10.1007/978-3-319-98355-4_30

describe possibly complex behaviors by a sequence of clear, simple instructions" [11]. Then, Avi Cohen and Bruria Haberman promoted informatics (computer science) as a language of technology [4]. The history of informatics at school started with programming. However, programming is only a part of informatics. Why do we need to teach informatics at schools? What is informatics? What should we teach and how? These and similar questions are the core problems to everybody who has been thinking on bringing informatics to any school level.

The educational achievements that were obtained in the 1970s and 1980s might be explained by the implementation of computers in schools and by their impact to general education. In the beginning of the 1980s, informatics education, although to a different extent, was established in many schools across the USA and Europe, e.g., Austria, Bulgaria, Lithuania, and Slovakia.

One of the early Russian pioneers in the field of theoretical and systems programming, Andrei Ershov, said: "Programming is the second literacy", which became a slogan for several years [10]. Politicians and educators in the industrialized countries proclaimed "computer literacy" as an essential part of education, and they demanded the integration of new technologies into the school curriculum.

A significant role in designing a methodology for teaching programming has been played by researchers in various countries. More and more countries have been reconsidering the role of informatics in general education, discussing the curricula for teaching informatics in secondary schools, and developing new courses for teacher training in informatics. Informatics was significantly supported in countries which had a strong academic way of teaching. Lack of access to computers was compensated by providing theoretical knowledge on programming and the working principles of computer hardware. A lot of attention was paid to learn the syntax of a programming language, to understand commands, etc. These countries had implemented informatics education in secondary school education as a discipline and the main focus was on teaching programming.

Bringing informatics to schools through the curriculum in a formal track is quite important, but it is also necessary to support the informal ways of introducing students to informatics. Contests and olympiads on programming are informal ways of introducing informatics to students. There are some discussions and disagreements on the role of contests in education. An important aspect is how the contests are introduced: are they tools for attracting more students and to motivate them to participate and solve problems by collaborating and discussing results? In many cases, contests make learning and teaching programming more attractive for students. Furthermore, computer programming is one of the most appropriate and effective ways to develop problem-solving skills for students of computer science.

2 Logo

In 2006, when Lithuania celebrated twenty years of informatics in schools, it was written: "At the beginning there was Logo. Then everything happened" [5]. In many countries, teaching informatics started with Logo, which was developed by Seymour Papert, the "father of Logo".

Logo is an educational language for children. Today the language is remembered mainly for its use of "turtle graphics", in which commands for movement and drawing produce line graphics. The language was originally conceived to teach concepts of programming.

For most people, learning Logo is not an end in itself, and programming tends to have a higher goal for the learner. Logo is accessible to novices, including young children, and also supports complex explorations and sophisticated projects by experienced users.

Seymour Papert's Logo offers a window into a time when computers and Logo had been in widespread use in schools for just a few years, while also delving into issues about informatics and education that remain relevant today.

An important event occurred in 1980—the publication of Seymour Papert's *Mindstorms* [13]. Educators throughout the world became excited by the intellectual and creative potential of Logo. Their enthusiasm fueled the Logo boom of the early 1980s. The book begins with an affirmation of the importance of making a personal connection with one's own learning and ends with an examination of the social context in which learning occurs.

In 1985, Logo Computer Systems corporation developed *LogoWriter*, which was a novel computer program in several ways. First, it included word processing capabilities. Second, the user interface was simplified and made more intuitive. *LogoWriter* also included, similarly as the earlier "sprite" Logos, multiple turtles that could take on different shapes.

Another innovation of the mid-eighties was Lego/Logo. M. Resnick and S. Ocko developed a system which interfaced Logo with motors, lights, and sensors that were incorporated into machines built out of Lego bricks and other elements.

Then more and more interesting programs appeared: Scratch, SNAP!, NetLogo, Comenius Logo, and so on. Scratch became a block-language metaphor and was widely used all over the world.

Many tasks for learning Logo have been developed, and many books have been published on Logo-inspired ideas to introduce children to programming. The ideas from Logo reoccur time after time; through the development of learning tools in many countries when new educational products are developed.

There have been many important events in connection with teaching Logo and algorithms all over the world (see Fig. 1). A detailed Logo Tree Project is provided by Boytchev [2].

3 Pascal as a Language to Introduce Algorithms

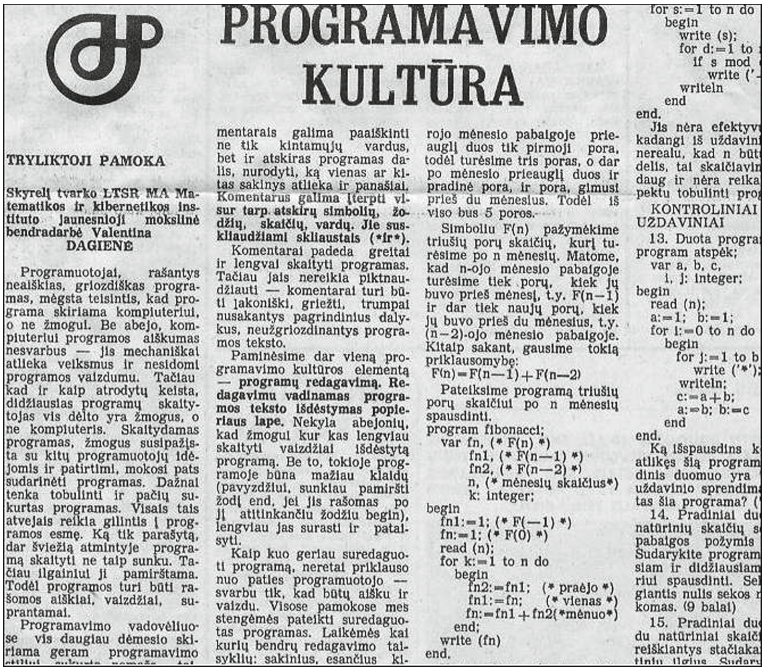
Pascal, the great programming language that served in education for many years, was designed in 1968 at ETH Zurich, and implemented and published in 1970. Pascal was designed by Niklaus Wirth with the goal to encourage good programming practices to novices by using structured programming and data structuring.

XLogoOnline		ABZ of ETH Zurich
Code.org		
Librelogo		
Turtle Academy		
Blocky		
SNAP!-BYOB	**2010**	B. Harvey, J. Mönig
		J. Hromkovič: *Algorithmic Adventures: From Knowledge to Magic*, 2009.
StarLogo TNT		
Computational Thinking		
Khan Academy		
Bebras		
Scratch		MIT Media Lab
	2000	
Netlogo		U. Wilensky
		B. Harvey: *CS Logo Style*, 1997.
CS Unplugged		Tim Bell
Programmable Bricks		
Logo Blocks		
Micro Worlds		M. Resnick: *Turtles, Termites and Traffic Jams: Exploration in Massively Parallel Microworlds*, 1994.
Python	**1990**	G. van Rossum
IOI		
EuroLogo conference (now Constructionism)		
Lego Logo		M. Resnick, S. Ocko
Object Pascal		
Logo Writer		
Logo Visualization	**1980**	S. Papert: *Mindstorms: Children, Computers, and Powerful Ideas*, 1980.
Constructionist approach		
		N. Wirth: *Algorithms + Data Structures = Programs*, 1976.
Pascal		N. Wirth
Logo	**1970**	S. Papert

Fig. 1. Influential contributions to school informatics education based on tools and approaches for learning algorithms and core concepts

In lectures, articles, and books, Wirth describes the history of (and road to) Pascal [18,19]. Structured programming and strict type checking were two of the innovative ideas proposed by programming researchers at this time.

Pascal had big influence on programming education in many countries. In Lithuania, Pascal was chosen as the language to communicate to big machines and express algorithms especially for secondary school students. It was a way



PROGRAMAVIMO
KULTŪRA

TRYLIKTOJI PAMOKA

Skyrelį tvarko LTSR MA Matematikos ir kibernetikos instituto jaunesnioji mokslinė bendradarbė Valentina DAGIENĖ

Programuotojai, rašantys
neaiškias, griozdiškas progra-
mas, mėgsta teisintis, kad pro-
grama skiriama kompiuteriui,
o ne žmogui. Be abejo, kom-
piuteriui programos aiškumas
nesvarbus — jis mechaniškai
atlieka veiksmus ir nesidomi
programos vaizdumu. Tačiau
kad ir kaip atrodytų keista,
didžiausias programų skaity-
tojas vis dėlto yra žmogus, o
ne kompiuteris. Skaitydamas
programas, žmogus susipažįs-
ta su kitų programuotojų idė-
jomis ir patirtimi, mokosi pats
sudarinėti programas. Dažnai
tenka tobulinti ir pačių su-
kurtas programas. Visais tais
atvejais reikia gilintis į prog-
ramos esmę. Ką tik parašytą,
dar šviežią atmintyje progra-
mą skaityti ne taip sunku. Ta-
čiau ilgainiui ji pamirštama.
Todėl programos turi būti ra-
šomos aiškiai, vaizdžiai, su-
prantamai.

Programavimo vadovėliuo-
se vis daugiau dėmesio ski-
riama geram programavimo

mentarais galima paaiškinti
ne tik kintamųjų vardus,
bet ir atskiras programas da-
lis, nurodyti, ką vienas ar ki-
tas sakinys atlieka ir panašiai.
Komentarus galima įterpti vi-
**sur tarp atskirų simbolių, žo-
džių, skaičių, vardų. Jie sus-
kliaudžiami skliaustais (*ir*).**

Komentarai padeda greitai
ir lengvai skaityti programas.
Tačiau jais nereikia piktnau-
džiauti — komentarai turi bū-
ti lakoniški, griežti, trumpai
nusakantys pagrindinius daly-
kus, neužgriozdinantys progra-
mos teksto.

Paminėsime dar vieną pro-
gramavimo kultūros elementą
— **programų redagavimą. Re-
dagavimu vadinamas progra-
mos teksto išdėstymas popie-
riaus lape.** Nekyla abejonių,
kad žmogui kur kas lengviau
skaityti vaizdžiai išdėstytą
programą. Be to, tokioje prog-
ramoje būna mažiau klaidų
(pavyzdžiui, sunkiau pamiršti
žodį end, jei jis rašomas po
jį atitinkančiu žodžiu begin),
lengviau jas surasti ir patai-
syti.

Kaip kuo geriau suredaguo-
ti programą, neretai priklauso
nuo paties programuotojo —
svarbu tik, kad būtų aišku ir
vaizdu. Visose pamokose mes
stengėmės pateikti suredaguo-
tas programas. Laikėmės kai
kurių bendrų redagavimo tai-
syklių: sakinius, esančius ki-

rojo mėnesio pabaigoje prie-
auglį duos tik pirmoji pora,
todėl turėsime tris poras, o dar
po mėnesio prieauglį duos ir
pradinė pora, ir pora, gimusi
prieš du mėnesius. Todėl iš
viso bus 5 poros.

Simboliu F(n) pažymėkime
triušių porų skaičių, kurį tu-
rėsime po n mėnesių. Matome,
kad n-ojo mėnesio pabaigoje
turėsime tiek porų, kiek jų
buvo prieš mėnesį, t.y. F(n−1)
ir dar tiek naujų porų, kiek
jų buvo prieš du mėnesius, t.y.
(n−2)-ojo mėnesio pabaigoje.
Kitaip sakant, gausime tokią
priklausomybę:

F(n) = F(n—1) + F(n—2)

Pateiksime programą triušių
porų skaičiui po n mėnesių
spausdinti.

```
program fibonacci;
   var fn, (* F(n) *)
         fn1, (* F(n—1) *)
         fn2, (* F(n—2) *)
         n, (* mėnesių skaičius*)
         k: integer;
begin
   fn1:=1; (* F(—1) *)
   fn:=1; (* F(0) *)
   read (n);
   for k:=1 to n do
      begin
         fn2:=fn1;   (* praėjo *)
         fn1:=fn;    (* vienas *)
         fn:=fn1 + fn2(*mėnuo*)
      end;
   write (fn)
end.
```

```
for s:=1 to n do
   begin
      write (s);
      for d:=1 to
         if s mod
            write ('-
      writeln
   end
end.
```

Jis nėra efektyv
kadangi iš uždavin
nerealu, kad n būt
delis, tai skaičiavin
daug ir nėra reika
pektu tobulinti prog

KONTROLINIAI
UŽDAVINIAI

13. Duota progra
program atspėk;

```
  var a, b, c,
     i, j: integer;
begin
  read (n);
  a:=1;  b:=1;
  for i:=0 to n do
     begin
        for j:=1 to b
           write ('*');
        writeln;
        c:=a+b;
        a:=b; b:=c
     end
end.
```

Ką išspausdins k
atlikęs šią program
dinis duomuo yra
uždavinio sprendim
tas šia programa? (

14. Pradiniai du
natūrinių skaičių s
pabaigos požymis
Sudarykite program
siam ir didžiausiam
riui spausdinti. Se
giantis nulis sekos
komas. (9 balai)

15. Pradiniai du
du natūriniai skaič
reiškiantys stačiaka

Fig. 2. Programming lessons were printed in the largest national daily newspaper from 1981 to 1983, from January to May, once a week. In the upper left corner appears the logo of the School—JPM: letters J, P, and M from the Lithuanian name of the school (Jaunųjų programuotojų mokykla).

to algorithmic thinking. Pascal was the backbone for the Young Programmers' School by Correspondence established in 1981 in Lithuania [9]. The goal of the school was to teach the main concepts of procedural programming and basic algorithms using Pascal notation. The first programming lessons were published in the largest daily newspaper over nearly half of a page a few times per month for a number of years, see Fig. 2.

All the teaching materials of the Young Programmers' School consisted of several chapters of lessons in a cyclic manner: (1) identifiers, variables, constants, assignment statements and sequence of statements, (2) conditional statements, (3) repetitions of actions, (4) programs and their execution by a computer, (5) logical values, (6) functions and procedures, (7) recursion, (8) discrete data types, (9) real numbers and records, (10) arrays, (11) programming style, (12) program design. Theoretical knowledge was delivered only as supplementary material for understanding informatics concepts and program design assignments.

Actually the Young Programmers' School laid the foundations to teach informatics in all upper secondary schools in Lithuania starting from 1986, and later, in 2005, in all lower secondary schools.

Informatics education in Lithuanian schools (grades 10–12) was based on Pascal, the language that fit perfectly for thinking about and developing algorithms. Pascal's advantage was its simple syntax and logical structure. Also, we can use Pascal to develop algorithms using pen and paper and later run them on a computer, which was an important aspect in the early eighties when schools had very limited access to machines. We taught algorithms to solve tasks, and we taught and learned how to think, both constructively and critically.

4 Programming, Children, and Bebras Tasks

Logo and Pascal have influenced informatics education in schools all over the world. A whole generation grew up programming with the Logo turtle as a visual tool and with Pascal for text-based algorithms. A group of informatics researchers in Lithuania developed many interesting tasks for learning programming and used them in school informatics curricula and after-school activities.

Researchers have shown that learning to program is a complex cognitive activity [16]. For example, the features of Logo as a programming language, like interactivity, modularity, extensibility, and flexibility of data types, are very powerful. Modular design requires children to organize their programs into conceptually independent units. This modularization process is the core to teaching informatics. First, it helps to approach challenging problems by breaking them down into smaller tasks, which can be solved independently by applying top-down or bottom-up problem-solving strategies. Second, thinking about problem instances in an abstract way helps identifying structural properties inherent to the problem, and therefore allows children to classify them into abstract problem classes. This results in a high intrinsic cognitive load.

Developing abilities to master modern technologies and skills for solving problems is among the most important capabilities of an educated future citizen of the information society, and these abilities can be directly connected with informatics education. Problem solving by means of programming does not lose its importance in a contemporary school equipped with modern information technologies, and it will remain a very important part of understanding the information processing and running of a computer.

Programming is an activity composed of several components: comprehension of the problem, choosing and encoding an algorithm, debugging, testing, and optimizing. Since many of the skills required for successful programming are similar to those required for effective problem solving, computer programming, particularly choosing one of several possible solutions and later debugging it within a short period of time, provides a fertile field for developing and practicing problem-solving skills in an environment that is engaging for young students [3].

When students begin to learn the basics of programming, they soon try to find a place where they can demonstrate their skills and projects, share their interests or compare themselves with others. This might explain the reasons why many students, soon after they have started learning programming, choose one of the areas where they are able to demonstrate their work immediately, e.g., creation of web pages or computer graphics. For some areas, e.g., developing algorithms, it is not easy to find a practical demonstration. The most powerful means that endorse the students' motivation are competitions and contests.

Nowadays, England has become the country to mandate computer science classes for all children between the ages of 5 and 16. Their age will determine exactly what they will learn, with topics ranging from algorithms to debugging code and lessons in programming languages.

4.1 Worldwide Challenge on Informatics Concepts

Pasi Sahlberg, a world-renowned Finnish educator, indicated playful learning, games, and gamification as one of the success factors of Finnish education [15]. Playful learning activities can attract the children's attention to learn various subjects.

Olympiads and contests in informatics focus mainly on developing algorithms and programming. The programming course can be followed by the introduction of some basic concepts of algorithms, data structures, recursion, procedures, and fundamental methods for designing algorithms. These and other basic informatics concepts can be introduced to students by contests or other activities in an attractive way.

In 2003, the idea of the Bebras contest on informatics was proposed (the name Bebras—in English *beaver*—is connected with a hard-working, intelligent, goal-seeking and lively animal). It took almost a year to create sample tasks and to prepare the technology for implementation. The contest started with a few countries and focused on school students divided into a few age groups.

Later, many other activities have been developed under the Bebras umbrella: hands-on seminars for students and teachers, discussions for deepening informatics knowledge and teamwork on developing Bebras tasks, so that the contest idea was changed to a broader Bebras challenge on informatics and computational thinking. In the past years, the number of Bebras participants has been notably growing. In 2007, the Bebras contests took place in seven countries, with about 50 000 participants in total. In 2017, more than 2 million students from over 50 countries were involved in solving Bebras tasks world-wide. Slovenia had the strongest relative participation with nearly 30 000 contestants, whereas France had the highest total number of participants, over half a million.

The crucial point of the Bebras challenge are the tasks based on informatics concepts [1,14]. The challenge needs to have a challenging set of tasks. The developers of Bebras tasks are seeking to choose interesting tasks (problems) to motivate students to deal with computer science and to think deeper about what constitutes the core of informatics. Some agreements on task development criteria have to be settled. Initially, six task topics (types) were proposed: (1) Information comprehension, (2) algorithmic thinking, (3) using computer systems, (4) structures, patterns, and arrangements, (5) social, ethical, cultural, international, and legal issues, as well as (6) puzzles [6,17]. During the last few years, a two-dimensional system for categorizing tasks was elaborated [7]. The suggested categorization system incorporates both informatics concepts and computational thinking skills in the classification of tasks.

Algorithms, data, data structures, data representation, and abstraction are the most important areas of informatics for schools. Creating an interesting task (problem) that is based on informatics concepts requires a lot of intellectual effort. Collaboration with different cultures and researchers, as well as peer-review and discussions, are very important for developing tasks which promote and introduce core concepts of informatics.

4.2 Bebras Tasks

An international Bebras workshop on creating tasks (problems) based on informatics concepts for all age groups is organized annually in different countries. The main goals of the workshops are to develop a set of tasks for the upcoming challenge, to discuss them and to come to an agreement among the countries with different (or without) curricula and traditions of teaching informatics in general education. All submitted tasks, including graphics, are developed under the Creative Commons Attribution-ShareAlike international license.

Several types of tasks (problems) have been used in the challenge: interactive (dynamic) tasks, open-ended tasks, and multiple-choice tasks are the most common groups. Motivation and attraction of students to take part in the challenge and to think deeper about informatics concepts are based on context-rich and powerful tasks, whose development and introduction are undoubtedly very challenging for researchers as well as for teachers.

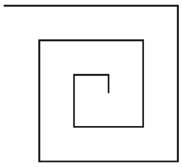
Multiple-choice tasks have four non-trivial and well-defined answer choices with only one correct solution. For dynamic tasks, interactivity is defined as a two-way transfer of information between a user and a machine. Thus an interactive task provides a specification of the problem, and, in solving it, a student needs to interact directly with the computer: drag and drop objects, click spots on pictures, manipulate objects using a keyboard, select list elements, etc. Many countries have established networks and teams of researchers, teachers, and educators for creating and discussing tasks. They usually come with new proposals every year. Below, three examples of Bebras tasks are provided and discussed.

Task Example 1: Drawing a Spiral. Ada has drawn a rectangular spiral with the help of a dynamic object, a turtle, and two commands:

forward 10 — the turtle moves forward drawing a line which is 10 steps (dots) long;

left 90 — the turtle turns left making an angle of 90°.

Which of the following numbers expresses the length of the whole spiral in the number of dots?



A 550
B 170
C 300
D 250

This example is a task from a decade ago. It introduces two commands: move forward and rotate. Additionally, the task requires calculations based on some understanding of what is happening.

Task Example 2: Beaver Code. Beaver Barbara has two rubber stamps. With the first one, she can produce a little flower and with the second one a little sun. Being a clever girl she thinks of a way to write her own name with these two stamps by encoding each letter with a sequence of flowers and suns as follows:

Letter	B	A	R	E	Y
Code	✿	✸✸	✸✿✸	✸✿✿✿	✸✿✿✸

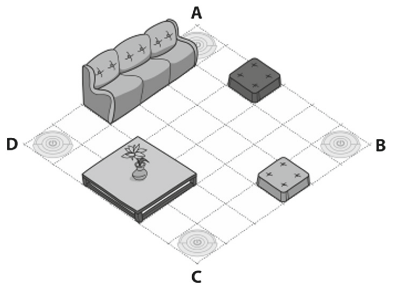
The name "Barbara" would be encoded as: ✿✸✸✸✿✸✿✸✸✸✿✸✸✸.

She then goes on to also write down the names of her friends. But, unfortunately, they got all mixed up (see the right column below). Drag the corresponding sun-flower-codes to the names of Barbara's four friends.

Abby		✸✸✸✿✸✸✿✿✸✸✸
Arya		✿✸✸✸✿✸✸✿✸✸✿✿✸
Barry		✸✸✿✿✸✿✿✸
Ray		✸✿✸✸✸✸✿✿✸

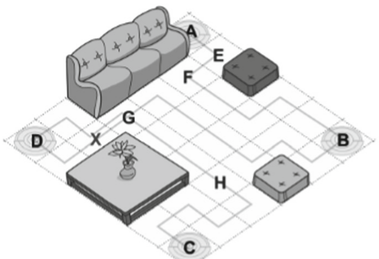
This task was developed by a group of educators from Switzerland in 2016 (U. Hauser, J. Hromkovič, T. Kohn, I. Kosírová, B. Steffen, Ch. Datzko). Often in informatics, instead of storing data in a simple and straightforward way, we can devise a scheme to store it more efficiently, using less space. Letters have different frequencies in texts (depending on the given language), and we can use these frequencies to improve our encoding. Specifically, we encode frequent letters with smaller codes. In this task, B takes one symbol, A takes two, and the other letters take more. However, you have to make sure that the code is unambiguous. Huffman encoding is a widely used algorithm.

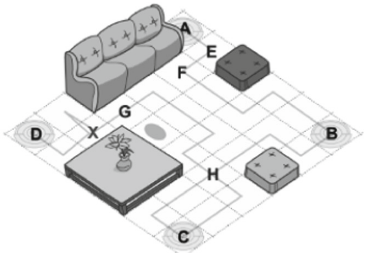
Task Example 3: Robot Cleaner (proposed by V. Dagienė, Lithuania). A robot washes a square-tiled floor by using the following commands: F—move forward one tile (takes 1 min), R—turn 90° right (performed instantly), W—wash a tile (takes 1 min). The robot starts and ends on one of the four corner tiles (A, B, C, D) — but not necessarily on the same tile.



What is the minimum number of minutes the robot needs for cleaning all the available tiles on the floor?

For a solution strategy, the main issue is to minimize the number of tiles which are passed twice. There is a tile X which must be passed twice in every chosen way (as seen in the two figures below). Tiles on the "bottleneck" to corner tiles A, B, C, and D are described by letters E, F, G, H, respectively. Tiles E and F are at the A bottleneck, tile G at the D bottleneck and H at the C bottleneck. Corner B does not contain any bottleneck tile. If a corner tile is neither the starting nor the finishing point, the robot must pass through bottleneck tiles of this corner twice. If the starting and finishing corner are different, bottleneck tiles of these two corners need to be passed through only once. Thus, we can try to arrange the path such that the robot goes through as few bottleneck tiles as possible. If the robot starts and finishes at the same corner tile, it must pass all of the bottleneck tiles twice. Since the bottleneck at A contains 2 tiles, namely E and F, a good strategy is to choose the point A as a starting (or finishing) tile. Then, we have to decide between finishing points C and D (in both cases we avoid going twice through another bottleneck point, G or H).





The trajectory in the picture to the right above shows that, for moving from A to D, the robot needs an odd number of moves so that, after using 28 moves forward, at least one tile is not passed yet, so it is longer than from A to C (trace in the left picture above). The minimum time for cleaning is thus 27 (washing) + 28 (moving) minutes = 55 min by going from A to C.

This is a special case of the traveling salesman problem—finding the shortest path when we want to visit every graph node, which transforms to the extended Hamiltonian path problem if there is a path in the graph to visit every node exactly once. There is no efficient algorithm to solve either of these two problems. For small cases, students can investigate the situation and come up with a strategy for solving the problem in a systematic way.

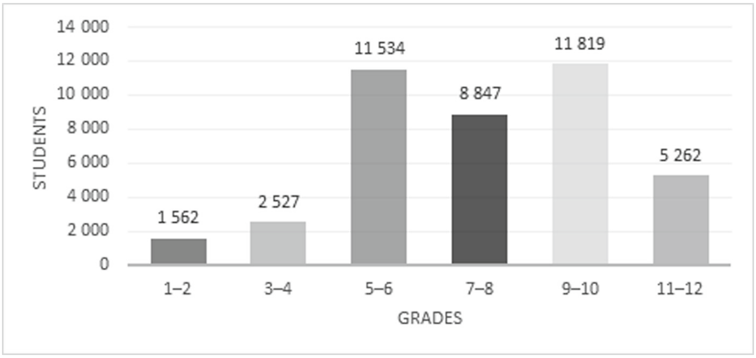


Fig. 3. Number of participants distributed by age groups (school grades)

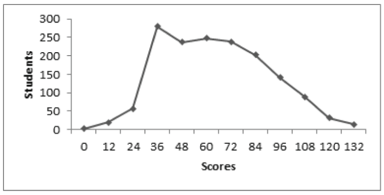
4.3 Solving Tasks: A Lithuanian Case

In November 2017, over 2 million school students from 40 countries took part in the Bebras challenge. Countries in the Southern Hemisphere are going to run the contest later in March.

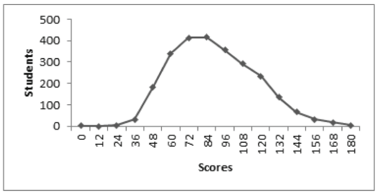
As a case study, we will discuss data about the Lithuanian participants in the 2017 challenge. From an organizational viewpoint, most countries are running the contest and supplementary activities in a similar way. The countries choose tasks on their own from a Bebras task pool—there is a large amount of reviewed and well documented tasks for various age groups. However, we have noticed that many countries have agreed to use the same (or almost the same) set of tasks. The set of tasks chosen in Lithuania is overlapping with the selection in many countries: Austria, Germany, the Netherlands, Switzerland, Finland, Hungary, Sweden, etc. (all age groups except grades 1 and 2). Lithuania had over 41 000 participants in the 2017 autumn Bebras challenge. The distribution of participants by grades is presented in Fig. 3.

The Bebras community has reflected on the involvement of girls in informatics education. An investigation on gender issues was done during previous years, for example, in the UK and Lithuania [8]. The Bebras challenge can be seen to be an event that attracts girls' attention to informatics education: worldwide, more than 40% of participants are girls (we cannot estimate the number exactly because some students have not indicated their gender). There is evidence from several countries that relatively more girls participate at a younger age and more boys take part in the higher grades [12].

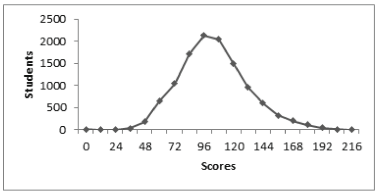
The data shown in Fig. 4 breaks down the scores per age group in Lithuania Bebras 2017. Lithuania has all six age groups covered now—the youngest grades 1 and 2 joined the contest last year. Most countries have organized the challenge based on 4 or 5 age groups in primary schools as well as lower and upper secondary schools. Scores are grouped and the number of students scoring a range of marks is shown for each age group. As we can see, the figures show a normal



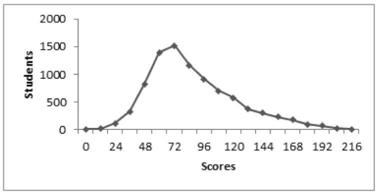
(a) Grades 1–2



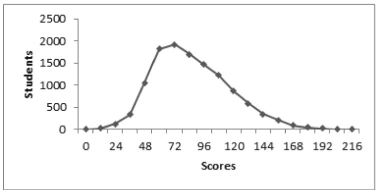
(b) Grades 3–4



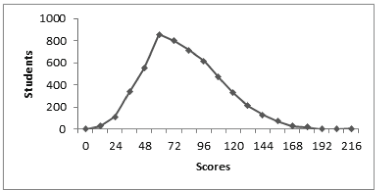
(c) Grades 5–6



(d) Grades 7–8



(e) Grades 9–10



(f) Grades 11–12

Fig. 4. Distribution of scores from tasks in all age groups in Lithuania (grades from 1 to 12)

distribution of scores in solving tasks across all age groups (similar results were obtained in the 2015 challenge in both Lithuania and the UK [8]).

We selected 67 different tasks distributed among six age groups. Some tasks were the same for 2–3 age groups. Tasks are categorized according to informatics concepts and computational thinking skills. For this study, we are interested in tasks related to algorithm concepts and algorithmic thinking. Usually, tasks cover more than one concept of informatics, but then we can usually estimate what the main concept behind the task is. 37 tasks were chosen for detailed investigation in this study. Eight tasks were solved by more than half of the students and six tasks were very hard—less than 10% solved them. This year, the chosen algorithm tasks were really challenging for the students—nobody obtained full scores. For the hardest tasks like “Pizzeria Biberia”, “Arabot’s walks” and “Book-Sharing Club”, the most common format is a long text with several detailed pictures. We have observed that students, especially in grades 7–10, which were the youngest students to whom these tasks were given, do not like reading long explanations and usually skip such tasks.

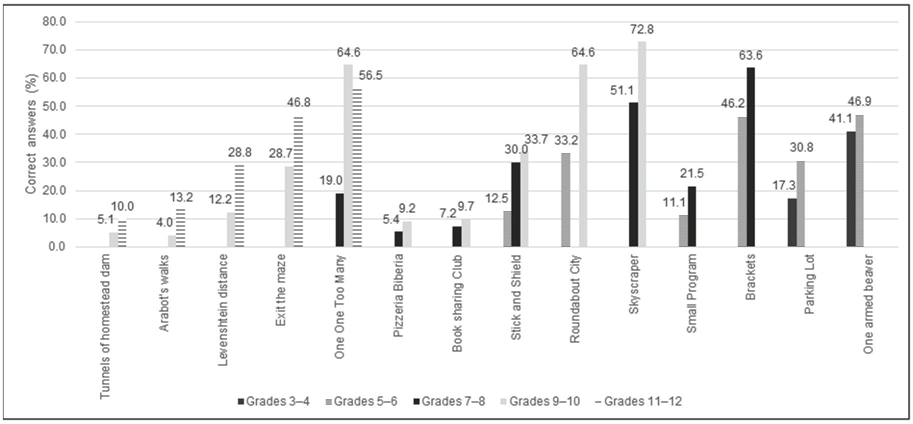
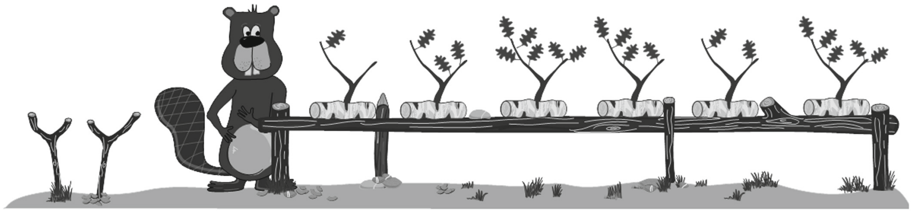


Fig. 5. Percentage of students that solved correctly the algorithms tasks that were given to more than one age group

As mentioned above, some tasks were given to 2–3 age groups. In the algorithmic concepts category, there were 13 tasks chosen for two age groups and two tasks for three age groups. We observed that students in the older age groups are doing better than expected. For most of the tasks, the differences in solutions between age groups are quite large, except for “Stick and Shield” (30% solved by grades 7–8 and 33.7% by grades 9–10) and “One-Armed Beaver” (41.1% solved by grades 3–4 and 46.9% by grades 7–8), see Fig. 5.

Students had the possibility to comment on the tasks in the Lithuanian contest management system. The task “One-Armed Beaver” (proposed by Christian and Susanne Datzko, Switzerland), given to students in grades 3–4 and grades 5–6, raised most excitement and got the most positive feedback.

One-Armed Beaver. The more leaves a branch has, the tastier it is. The beaver intends to sort the branches based on their taste using the temporary storage beside him. Please help the beaver sort the branches by taste so that the tastiest branch is closest to him.



Variables are a core concept in informatics: This task focuses on swapping the value of two variables. Swapping two pieces of wood in two places by using a temporary new storage—this is about variables and their values. Also a sorting algorithm is needed for this task. For sorting a list while having a limited amount of memory, we use a sorting algorithm such as selection sort: Going from the

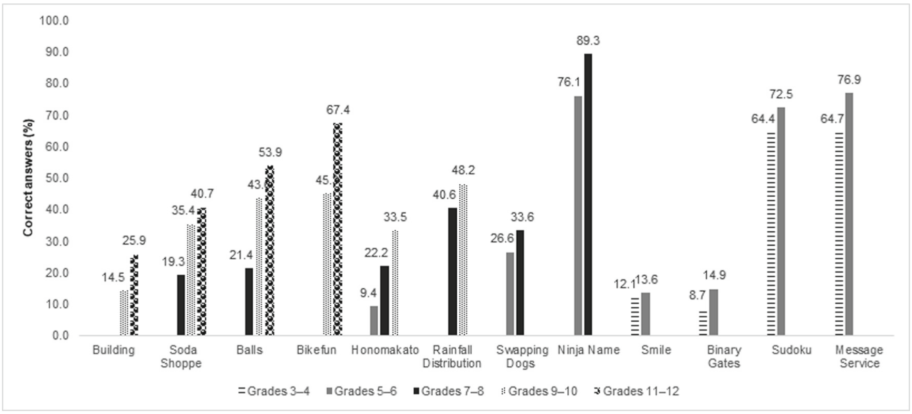


Fig. 6. Percentage of students that solved correctly the tasks related to data structures and data representation that were given for more than one age group

first to the last element, swap the current element with the smallest one in the remaining list using a temporary variable.

The statistical data (see Fig. 5) of the task "One Too Many" (Janez Demšar, Slovenia) shows that the juniors solved it better than the oldest students (64.6% of students in grades 9–10 solved this task correctly and the result is better than the 56.5% for grades 11–12). Core concepts in this task include handling a text editor and how to follow a sequence of instructions. The task is also about swapping values of two variables by using a temporary variable, but in a more complex situation than in "One-Armed Beaver" above.

The concept of data, data structures, and data representation is a separate category within the categorization of Bebras tasks. However, that category is very tightly related to algorithms and computational thinking. Observing the results in this category, we have noticed that there is not a big difference between scores gained by adjacent grades for the two oldest age groups solving a given task; when 3 groups solved the same task, the youngest group has much lower results for each task (see Fig. 6). This indicates that data structures and data representation concepts are more easily accepted by different age groups than algorithmic concepts.

The participants of the Bebras challenge liked the task "Ninja Name": It is short and clearly formulated. Understanding a string is a core concept of this task, and it involves coding as an attractive element as well. Letter replacements in strings are often used in informatics tasks. These tasks serve to provide an easier visualization of what is happening when elements are replaced within complex rows of data such as strings of signs.

5 Conclusions and Challenges

We need to work on balancing continuity and innovativeness in informatics education. Is continuity boring? The sun comes up every morning, we take the same route to work every day, and through the window we see the same landscape. We often do not even notice these things but there is a place for them in our minds and so we feel well and peaceful. A similar understanding of continuity can be applied to learning informatics at schools for students—lessons, everyday problem solving, focus on core concepts, and a few contests at various levels per year as motivational tools can be a comfortable continuity.

I ask myself, what has been the most important phenomenon during all those years of introducing informatics in schools? Probably—the relationship among people. While sitting for long hours at your workplace, in different meetings, no longer knowing if you are thinking in the right direction, not knowing how to move on, you suddenly receive a message from a person, then from another one, and then from yet another one. You feel connected and see many others who search, doubt, discover, and rejoice. We must not stop dreaming, searching and communicating: about everyday life events and informatics education among them.

Involving students in the recognition of informatics as a science discipline should be our target, and we should try to achieve it more successfully. Well-organized activities with interesting, playful, and exciting tasks will introduce students to the essence of the computer science world and help them to understand core concepts of informatics.

Acknowledgements. I want to acknowledge my doctoral students Gabrielė Stupurienė and Lina Vinikiene for taking care of the Bebras challenge in Lithuania and sharing their ideas on how to improve Bebras tasks based on informatics concepts.

References

1. Bellettini, C., Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A., Torelli, M.: How challenging are Bebras tasks? An IRT analysis based on the performance of Italian students. In: Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, pp. 27–32. ACM (2015)
2. Boytchev, P.: Logo Tree Project (2014). http://www.elica.net/download/papers/LogoTreeProject.pdf. Accessed 5 Feb 2018
3. Casey, P.J.: Computer programming: a medium for teaching problem solving. In: Computers in the Schools, vol. XIII, pp. 41–51. The Haworth Press Inc., New York (1997)
4. Cohen, A., Haberman, B.: Computer science: a language of technology. ACM SIGCSE Bull. **39**(4), 65–69 (2007)
5. Dagienė, V.: The Road of Informatics. TEV, Vilnius (2006)
6. Dagienė, V., Futschek, G.: Bebras international contest on informatics and computer literacy: criteria for good tasks. In: Mittermeir, R.T., Sysło, M.M. (eds.) ISSEP 2008. LNCS, vol. 5090, pp. 19–30. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69924-8_2

7. Dagienė, V., Sentance, S., Stupurienė, G.: Developing a two-dimensional categorization system for educational tasks in informatics. Informatica **28**(1), 23–24 (2017)
8. Dagienė, V., Sentance, S.: It's computational thinking! Bebras tasks in the curriculum. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 28–39. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_3
9. Dagys, V., Dagienė, V., Grigas, G.: Teaching algorithms and programming by distance: quarter century's activity in Lithuania. In: Proceedings of the 2nd International Conference on Informatics in Secondary Schools: Evolution and Perspectives (ISSEP 2006), pp. 402–412. Institute of Mathematics and Informatics, Vilnius (2006)
10. Ershov, A.P.: Programming, the second literacy. Microprocess. Microprogr. **8**(1), 1–9 (1981)
11. Hromkovič, J.: Contributing to general education by teaching informatics. In: Mittermeir, R.T. (ed.) ISSEP 2006. LNCS, vol. 4226, pp. 25–37. Springer, Heidelberg (2006). https://doi.org/10.1007/11915355_3
12. Hubwieser, P., Hubwieser, E., Graswald, D.: How to attract the girls: gender-specific performance and motivation in the Bebras challenge. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 40–52. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_4
13. Papert, S.: Mindstorms: Children, Computers, and Powerful Ideas. Basic Books Inc., New York (1980)
14. Pohl, W., Hein, H.W.: Aspects of quality in the presentation of informatics challenge tasks. In: ISSEP 2015. LNCS, vol. 9378, pp. 21–32. Springer, Cham (2015)
15. Sahlberg, P.: Finnish lessons 2.0: what can the world learn from educational change in Finland? Teachers College, Columbia University (2015)
16. Sweller, J.: Cognitive load theory. In: Psychology of Learning and Motivation, vol. 55, pp. 37–76. Academic Press (2011)
17. Vaníček, J.: Bebras informatics contest: criteria for good tasks revised. In: Gülbahar, Y., Karataş, E. (eds.) ISSEP 2014. LNCS, vol. 8730, pp. 17–28. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09958-3_3
18. Wirth, N.: Recollections about the development of Pascal. In: Bergin, T.J., Gibson, R.G. (eds.) History of Programming Languages II, pp. 97–120. Addison-Wesley (1996)
19. Wirth, N.: Pascal and its successors. In: Broy, M., Denert, E. (eds.) Software Pioneers, pp. 108–119. Springer, Heidelberg (2002). https://doi.org/10.1007/978-3-642-59412-0_8

The Adventure of Computer Science

Understanding the Impact of Informatics on General Education and Designing the Living Environment

Jens Gallenbacher[1,2](✉)

[1] Technische Universität Darmstadt, Darmstadt, Germany
jg@di.tu-darmstadt.de

[2] Johannes Gutenberg University Mainz, Mainz, Germany

Abstract. In the 20th century, a paradigm shift started regarding the acquisition of knowledge. Our living environment is more and more determined by engineering-driven construction. This paper explains how the tangible living environment of pupils and the models of explanation used in general school education drift further and further apart. Introducing computer science as a nationwide mandatory subject in schools offers a chance to take this paradigm shift into account. In this context, didactic approaches like CS Unplugged or "Abenteuer Informatik" (the adventure of computer science, a German project) can help to implement school education that is really general.

Keywords: Abenteuer Informatik · CS Unplugged
Acquisition of knowledge · Peirce · General education
Models of explanation · Construction · Design
Mandatory subject computer science

1 Of Peirce and the Natural Sciences

In human history preceding the age of the Enlightenment, one had an exact idea of how new knowledge can be gathered: above all, there was study. This primarily meant the study of scientific (and religious) literature. The main ways to yield new knowledge were *deduction* – implying special knowledge from general knowledge – and *induction* – achieving general knowledge from special cases. Since all implications had to be strictly related to some knowledge that was already present, introducing new ideas required detours.

Thus, also natural scientists were not yet recognized as such. Galileo Galilei, who formulated principles of scientific research by experiments and observation already at the beginning of the 17th century, was far ahead of his time. Even a century later, Isaac Newton was recognized as a philosopher, not as a scientist. The living environment in those times was shaped by faith. Design was largely related to art and architecture, much less to profane things of every-day life.

In the 19th century, the Industrial Revolution created a new point of view on the living environment: Steam engines provided independence from rivers as

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 538–548, 2018.
https://doi.org/10.1007/978-3-319-98355-4_31

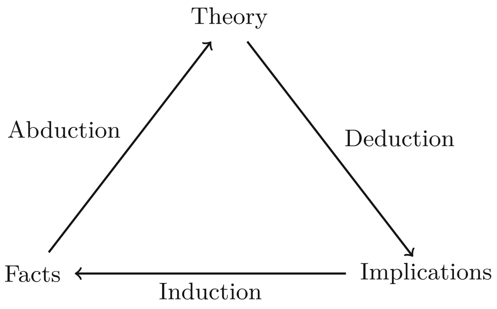


Fig. 1. The Peirce cycle of abduction, deduction, and induction

a means for driving the mills, and chemical fertilizer revolutionized agriculture and, very concretely, reduced starvation. Understanding the working principles of nature now was an important foundation of society. "Discovering" new physical or chemical connections was increasingly viewed as reasonable.

Taking this development into account, around 1900, Charles Sanders Peirce introduced the notion of *abduction*. This can be seen as a formal model for generating hypotheses from new ideas that are inspired by observations. From such a hypothesis, one can then use deduction to derive possible consequences, and can scientifically verify or falsify these in experiments using the principle of induction. The relations between abduction, deduction, and induction can be visualized in a cycle diagram as shown in Fig. 1, which goes back to the work of Peirce [11]. Hence, abduction is no revolutionary idea, but the mere acknowledgment of a scientific principle that was in practice used by mankind "all along" to acquire new knowledge about the surrounding nature. Following this acknowledgment, natural sciences were also introduced into the educational system. At universities, faculties of natural sciences were established and partitioned into single disciplines. For instance, during the 19th century, chemistry emancipated itself from medicine and became a separate discipline at many places.

Due to the needs of the industrialization, also the school system could no longer avoid the respective lessons in natural sciences, but these were much under-represented with 2 out of 20 lessons in the high-school ("Gymnasium") curriculum of the 19th and beginning 20th century. However, there existed some special schools ("Realschulen" and, to some extent, "Realgymnasien") that put much more weight onto these subjects. The transition to specialized subjects within the natural sciences at high school happened, depending on the region, only a hundred years after the differentiation at university level. But the proportion with respect to the complete curriculum did not rise significantly, with a percentage of, e.g., 12.5% in Hesse in 2018.

Abduction, as the possibility to build hypotheses about already present material or immaterial artefacts in an explanatory, deducing way, is already included in the school curriculum, although probably not under this name.

THE
Fun
SCIENCE NEWS COMPANY
Alarming:
Desolate amongst
friends!
Scientific proof: 84% of the
population has less friends than
the average of their friends!

Fig. 2. Fictitious headline

Until the end of the last century, this scientific, explanatory approach was at large consistent with the living environment of the pupils. Although the environment was more and more influenced by human-made artefacts, the laws of nature were also valid for those. Euler's implication of Newton's second law "force equals mass times acceleration" is also valid for a car, for which one has to estimate whether it is possible to cross the street in front of it unharmed.

2 The Living Environment in the 21st Century

Due to the increasing virtualization, the shift of human communication to social networks, and the all-time-everywhere reachability, the living environment is increasingly dominated by human-made artefacts that do not necessarily follow the laws of nature.

Of course, such artefacts can in the first place be viewed and analyzed in the traditional way. Besides a manganese nodule found close to a volcano, or a water sample from a local source, also a desktop computer can be (disassembled and) analyzed. The cycle of abduction, deduction, and induction also works for this, and the pupils "learn" to understand their living environment. However, a very relevant element is missing, which constitutes the key to a full-fledged participation in a modern living environment: the ability to participate in actively designing it.

We want to illustrate this with a striking example: The fictitious headline from Fig. 2 is no "fake news", but is based on a study by Ugander et al. from May 2011 [12], which analyzed 721 million people with 69 billion friendship relations

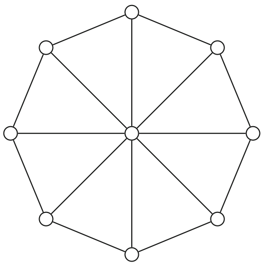


Fig. 3. Friendship graph

– an order of magnitude that is extremely remarkable for an empiric study. A basic hypothesis of this study was that most people have less friends than their friends do.

Included in this study were all Facebook users which were signed in for at least four weeks and had at least one friend relation there. The results indeed showed that about 84% of the "study participants" had less friends than the average of their friends.

An important question that remains, however, is how relevant this result actually is. As computer scientists, we know the intellectual tool of modeling for analyzing and solving problems. If we use this tool on a "normal" example with, say, 9 friends, we arrive at a graph like the one shown in Fig. 3. Here, vertices represent people and edges represent friendship relations. A very sociable person is present in the center and has a large number of friends. Surrounding this person, there are several cliques of people that know only two more friends besides her or him.

In this example, we have one person with eight friends and three friends of friends on average, and eight persons with three friends, but 4.67 friends of friends on average. This means that "provably" eight out of nine people (thus, even almost 89%) have less friends than the average of their friends. On the one hand, this example supports the findings of the study, but on the other hand, it is easy to see that people with many friends contribute more to the friends-of-friends statistics than those with less friends. This insight makes the result of the Facebook study much less surprising.

The same result was indeed also obtained 20 years before the study of Ugander et al. by Feld [3], who calls this the "friendship paradox". More precisely, he proved the following result: If there exist, in a group of people with friendship relations, at least two people having a different number of friends, then there is always a majority of people in the group that have less friends than their friends on average.

Here, we can take two different points of view: One is the view from the outside, where we can analyze the situation with the Peirce cycle. On the other hand, the artefact to be analyzed is a social network, and thus something

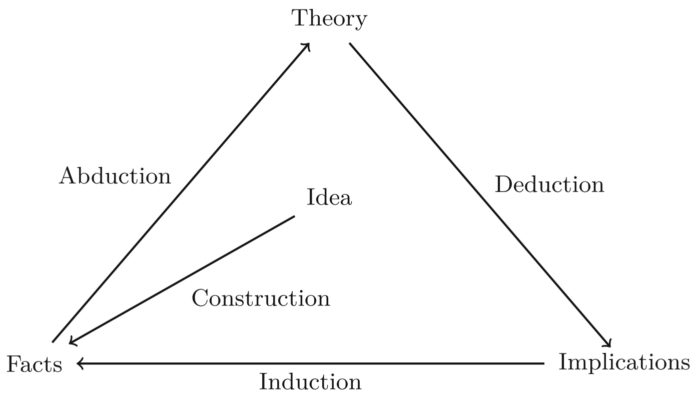


Fig. 4. The Peirce model with construction

human-made. Hence, it seems to be much more direct and reasonable, and in particular much less tedious, to incorporate this design into the study.

3 Taking Peirce to the 21st Century

This human-made design in our living environment does not only become visible in social networks: car-sharing apps rough up the transport market, Industry 4.0 promises cooperation on a completely new level, insurance companies can use "big data" for much more individualized offers, and deep learning comes up with results that by now really remind us of artificial intelligence.

At universities, the masterminds of such designs can mostly be found in engineering and computer science. These are scientific fields that often have emancipated themselves from natural sciences and mathematics in the middle of the 20th century. The knowledge generated by them can in fact be analyzed using the Peirce cycle, but not the generating process itself. They are more than hypotheses: Our living environment is permanently enriched and extended by human-made design. Hence, Fig. 4 shows the proposal to add the principle of *construction* to Peirce's model.

Construction creates new facts which then – as already shown in an example – can be analyzed using abduction, deduction, and induction.

Construction is not new to the school system! It can be found in classical subjects such as art or literature. Nobody would expect pupils to analyze a painting by Picasso before having tried out a pencil or a brush themselves. An interpretation of Goethe's work is done only after gaining own experience with writing simple essays. But in the area of artefacts, which have an immense impact on our living environment, one usually immediately starts with the analysis: media-pedagogic articles eloquently explain, with the use of quantitative empirical research, the implications of social networks. The contribution to "big data" edited by the "Bundeszentrale für politische Bildung" (federal central office for political education), which is an important source for teaching materials for many schools in Germany, is comprised of six articles written by journalists, lawyers, sociologists, politologists, and physicians, who view and analyze this "phenomenon" from the outside.

How can and must school carry out its mission of general education with respect to this? For sure by acknowledging that construction is an element that massively, and possibly soon in a vital way, shapes our living environment. Today's pupils are no longer only asked to passively analyze the respective artefacts or – in an also passive way – to operate the resulting systems. Today's pupils will design the artefacts and systems of a future living environment! General education means that, on the one hand, all should have the chance to participate in this design – actively by constructing themselves – or also passively by participating in decisions regarding the possible application of newly constructed systems, based on their deeper insight into the basic working principles and models of the systems. On the other hand, general education means to preserve cultural coherence, thus very explicitly not to create a society of "nerds" who are able to design systems and a society of the "others" who view the products of these nerds like natural phenomena. This legitimates mathematics as mandatory subject for the whole school period, but also a special subject addressing the constructive approach – like computer science.

Also the remaining tasks of general school education as described by Heymann [8] – preparation for life, practicing communication and cooperation, orientation towards the world, critical thinking, development of the willingness to take up responsibility, and strengthening self-esteem – suggest to take construction at school seriously [5].

4 Structural Chances of Teacher Education

Models of knowledge acquisition are central competencies in all school subjects, just as, e.g., considering tasks of general education or scientific working techniques. Hence, within the scope of a quality enhancement program for teacher education, Technische Universität Darmstadt has established a network for didactics students, where the respective competencies are developed in an interdisciplinary way, among others within an integrated lecture about "central ideas and tools".

After short inputs in the form of a classical lecture, project teams consisting of four or five students from different disciplines independently develop teaching sequences under a very broad common theme that varies from year to year. For instance, the theme of the first course in the summer term of 2017 was the "intelligent refrigerator", which was aimed at stimulating thoughts about sustainability, functioning principles, and communication abilities of home devices from the point of view of the living environment of the pupils. The theme in the forthcoming summer term 2018 will be "To pixel or not to pixel", following a presentation by the mathematician Sir Michael Francis Atiyah, who talked about discrete and continuous models of our world at TU Darmstadt in 2017 [1].

The main task of the students is, given this common theme, to set particularly central ideas and tools into the context of school and to connect them to the living environment of the pupils. The course of action follows research-centered learning according to Huber [9]. The best results are presented at an (internal) peer-reviewed conference [6].

Consequently, the goal is not the unilateral focus of teachers on construction and engineering-like thinking, but an integration of this new viewpoint on knowledge acquisition, as we discussed it here, into the previous context, also providing the chance to strengthen the connections and cooperation between different scientific disciplines.

The student feedback for the first course was consistently positive, apart from some comments on initial organizational difficulties; the self-estimated increase of competencies was high.

5 Implications for the Content of the Curriculum

Finally, we want to illustrate with an example, what implications the extended view on construction can have on teaching contents, and in particular on learning goals.

Today, the binary system for the representation of numbers is without doubt firmly connected to computers and informatics; thus, it is contained in almost all curricula. It can be easily explained using binary cards or a combination of binary clock and binary cards [7], and there exists a multitude of activities concerning binary numbers; see also [2].

But how do we motivate in class why we deal with this number system that might appear cumbersome at first glance? The explanations commonly used can be mapped to the following two approaches.

> In computer systems, signals have to be transmitted, and this signal transmission is more secure having two states, as the distance between signal and noise becomes larger.

Yes, this might be true. Nevertheless, modern data transmission protocols work with significantly more states to enhance efficiency, e.g., with 256 states in the case of VDSL. This explanatory model is thus not convincing.

> Zuse had relays at his disposal, and these have two switching states.

Although, strictly speaking, Konrad Zuse did not use relays for his Z1, but home-made logical units constructed from small metal plates, the first completely functional models indeed used relays. Nevertheless, Zuse would have had also the rotary selectors from telephone engineering at his disposal – a kind of relay with 10 switching states. Thus, also this explanation does not work.

To find a reasonable explanation, we follow a historic-genetic approach [13] and reconstruct a traditional algorithm from ancient Egypt, which is also known as "Ethiopian multiplication" or "Russian rural multiplication". Already in those

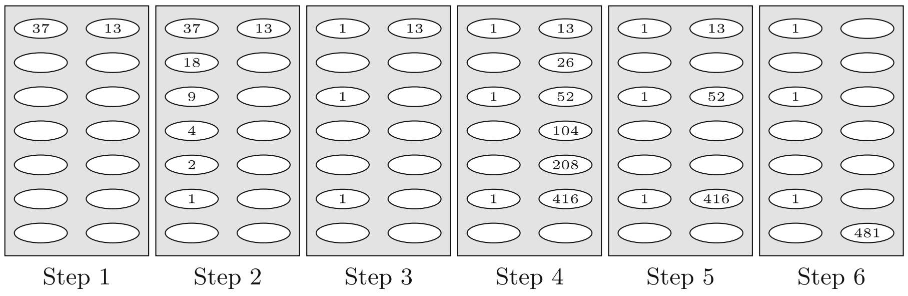


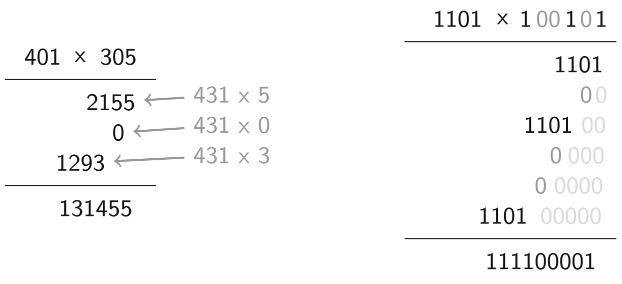
Fig. 5. Egyptian multiplication with the multiplicands 37 and 13

times, a trustworthy method was needed for, e.g., determining the total price of 37 amphorae at 13 dinar each, without buyer or seller being able to calculate.

The calculation was carried out with the help of hollows in the sand and small stones, which we represent here by their numbers per hollow. The calculation now strictly follows the rules of the following algorithm.

1. Fill the two uppermost hollows with the multiplicands.
2. In the left column, do the following: Into every still empty hollow, put half of the content (rounded down) of the preceding hollow.
3. In the left column, do the following: Remove pairs of stones from each hollow as long as possible.
4. In the right column, do the following: Into every still empty hollow, put double of the content of the preceding hollow.
5. In the right column, do the following: Empty every hollow that is neighboring to an empty hollow from the left column.
6. In the right column, do the following: Count all remaining stones.

Figure 5 shows an example with the multiplicands 37 and 13. The Egyptian multiplication is a method which transforms a rather complicated operation



(a) decimal (b) binary

Fig. 6. Written multiplication

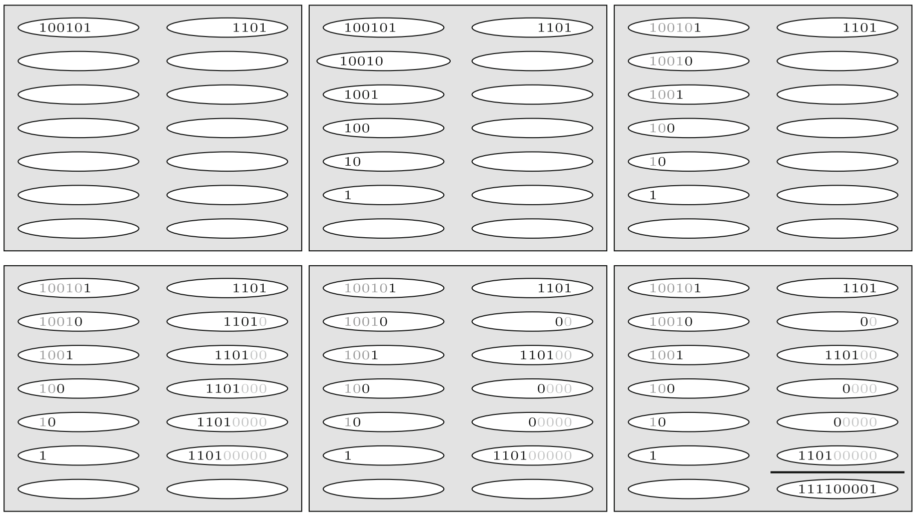


Fig. 7. Egyptian multiplication with binary representation

– multiplying two arbitrarily large numbers – to a never-changing sequence of simple actions. These actions are "halving", "doubling", and "adding". This kind of transformation is called *structured partitioning*, and it constitutes the basic idea of an algorithm and is thus a fundamental idea within computer science.

To find out what this has to do with the binary number system, we take another detour and look at the written multiplication as taught in primary school. An example of multiplying 431 and 305 is shown in Fig. 6a. Also here, we transform something complicated into simpler calculations – in this case into the basic multiplication table and written additions for adding the intermediate results. Now we look at exactly the same method applied to numbers written in binary. The result for the numbers 1101 and 100101 is shown in Fig. 6b. For better readability, the intermediate results are not only indented, but also the correct number of zeros is appended.

Already at this point, the pupils should recognize that the written multiplication in binary was rather easy. The principle becomes even more obvious if we now show the connection to the Egyptian multiplication, namely that both methods are in principle identical. The Egyptian multiplication with the binary representation of the numbers 37 and 13 is shown in Fig. 7. A more detailed teaching instruction can be found in [4,7].

In principle, the ancient Egyptians multiplied in the same way as every computer does it nowadays – based on binary number representations. In this context, it is irrelevant whether the ancient Egyptian mathematicians were conscious about this representation. It is more important that the motivation for this was the same then as it is now: it is amazingly simple! It is based on the "very small" multiplication table for binary numbers as shown in Fig. 8. How much easier would primary school have been for us if we had have to learn only this. . .

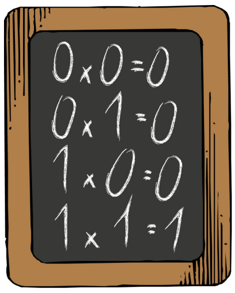


Fig. 8. Multiplication table for binary numbers

Thus, our whole computer technology is not based on complexity, but on the opposite: on simplicity. Together with structured partitioning, this principle can be used to solve a vast majority of all problems effectively. Specific artefacts of our age – like the popularity and importance of the binary number system – can be analyzed using induction and deduction. Since these are completely intellectual creations, not even explainable by physiognomy (like the decimal system is by humans having ten fingers), they can only be explained by competencies in construction. Taking part in designing our living environment is only possible with constructive competencies.

In the above sketch, no construction happens, though – it is just reproduced. But using the derived competence of structured partition, many more principles can be rediscovered and newly derived, e.g., binary search or sorting algorithms, but also principles outside computer science such as modular construction.

6 Conclusion

Informatics is part of general education. This claim has been established and proven in many articles in the past, e.g., in [14]. The paradigm shift of the living environment discussed here, and the resulting necessity to accept construction as an important element of knowledge acquisition and to teach it in school implies a stronger version of the claim:

General school education is not possible today without a mandatory subject with appropriate time slots for all pupils that focuses on the – very relevant with respect to the living environment – principle of construction! The only established school subject that is able to provide this is computer science.

References

1. Sir Michael Francis Atiyah. The discrete and the continuous from James Clerk Maxwell to Alan Turing. http://www.heidelberg-laureate-forum.org/blog/video/lecture-thursday-september-29-2017-sir-michael-francis-atiyah/. Accessed 15 Mar 2018

2. Bell, T., Witten, I.H., Fellows, M.: CS Unplugged (2015). https://classic.csunplugged.org/wp-content/uploads/2015/03/CSUnplugged_OS_2015_v3.1.pdf. Accessed 15 Mar 2018
3. Feld, S.L.: Why do your friends have more friends than you do? Am. J. Sociol. **96**(6), 1464–1477 (1991)
4. Gallenbacher, J.: IT-Grundlagen für Kinder. Mini Series in c't Magazin, vol. 7/2016, p. 144, vol. 10/2016, p. 158, vol. 12/2016, p. 158
5. Gallenbacher, J.: Allgemeinbildung in der digitalen, gestalteten Lebenswelt. In: Diethelm, I. (ed.) Informatische Bildung zum Verstehen und Gestalten der digitalen Welt. Lecture Notes in Informatics (LNI), pp. 19–28. Gesellschaft für Informatik (2017)
6. Gallenbacher, J.: Zentrale Ideen und Werkzeuge MINTplus. In: TU Darmstadt (ed.) MINTplus - Systematischer und vernetzter Kompetenzaufbau in der Lehrerbildung, pp. 22–23 (2017). http://www.zfl.tu-darmstadt.de/media/zfl/alle_medien_in_struktur/projekte_medien/projekte_mintplus/projekte_mintplus_anmeldung_zur_tagung_mintplus_am_5__oktober_2017/MintPlus_Broschure.pdf. Accessed 15 Mar 2018
7. Gallenbacher, J.: Abenteuer Informatik - IT zum Anfassen für alle von 9 bis 99, 4th edn. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-53965-1. ISBN 978-3-662-53964-4
8. Heymann, H.W.: Allgemeinbildung und Mathematik. Studien zur Schulpädagogik und Didaktik, vol. 13. Beltz, Weinheim/Basel (1996). New edition 2013
9. Huber, L.: Warum Forschendes Lernen nötig und möglich ist. In: Huber, L., Hellmer, J., Schneider, F. (eds.) Forschendes Lernen im Studium, pp. 9–35. Universitätsverlag Webler, Bielefeld (2009)
10. Minnameier, G.: Tightening the Peirce-Strings forms of abduction in the context of an inferential taxonomy. In: Magnani, L., Bertolotti, T. (eds.) Springer Handbook of Model-Based Science. Springer, Berlin (2015)
11. Peirce, C.S.: The Collected Papers of Charles Sanders Peirce, vols. I–VI, C. Hartshorne and P. Weiss (ed.) (Harvard University Press, Cambridge (1931–1935)), vols. VII–VIII, A.W. Burks (ed.) (same publisher (1958)). Electronic Edition (1994)
12. Ugander, J., Karrer, B., Backstrom, L., Marlow, C.: The Anatomy of the Facebook Social Graph. eprint arXiv:1111.4503, November 2011
13. Wagenschein, M., Berg, H.C., von Hentig, H.: Verstehen lehren - Genetisch - Sokratisch - Exemplarisch, 2nd ed. Beltz, Langensalza (2010)
14. Witten, H.: Allgemeinbildender Informatikunterricht? Ein neuer Blick auf H. W. Heymanns Aufgaben allgemeinbildender Schulen. In: Informatische Fachkonzepte im Unterricht, Proceedings of the 10. GI-Fachtagung Informatik und Schule (INFOS 2003), pp. 59–75 (2003)

Toward Scenario-Based Algorithmics

David Harel and Assaf Marron(✉)

Weizmann Institute of Science, Rehovot, Israel
{david.harel,assaf.marron}@weizmann.ac.il

Abstract. We propose an alternative approach to the classical way of specifying algorithms, inspired by the scenario-based paradigm for reactive systems. Rather than being presented as a carefully ordered sequence of instructions, an algorithm is formalized as an unordered collection of rules or scenarios, specifying actions that must or must not be taken when certain conditions hold or after certain sequences of events. A successful implementation of such a methodology, which can be aligned with a natural language specification, can have many advantages, including naturalness, comprehensibility and incrementality. We believe that our approach can also accelerate the acquisition of problem-solving and analytical skills by children and students. This is because by writing (and reading) computer programs written in this way, people would have access to a broad base of instructions on how to solve problems, stated and organized in a way that can be readily understood and used in practice also by humans. We describe the principles of the approach, *scenario-based algorithmics* (SBA), provide some examples, and compare it to other techniques for algorithm specification and to human algorithmic or computational thinking.

1 Introduction

Ask a student, instructor or researcher of computer science, about a particular algorithm or protocol, be it bubble-sort, quicksort, depth-first search of a tree structure, or two-phase commit, and you are likely to get in response a short list of basic principles—the key points that distinguish this algorithm from an arbitrary or brute force or random approach to the problem. Similarly, text books often precede or summarize the actual details of such algorithms with such a list of principles. The algorithm itself is most often described in some sort of code—either pseudo-code or in some particular programming language, like C, or C++—providing the detailed step-by-step instructions for carrying out the prescribed process. While these instructions can be readily followed (and executed) by a human or a computer, the basic principles of the algorithm are often only implicit, reflected in the names of various methods and subroutines, or in the physical structure of the code. In fact, most often they appear explicitly only in text comments that explain the otherwise-arcane lines of code.

In this paper we propose an approach, termed *scenario-based algorithm specification* or *scenario-based algorithmics* (SBA), for specifying algorithms. Rather

© Springer Nature Switzerland AG 2018
H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 549–567, 2018.
https://doi.org/10.1007/978-3-319-98355-4_32

than describing the algorithm in a form that explicitly prescribes a carefully ordered sequence of step-by-step instructions, we propose to create a collection of often very brief self-standing rules (or, more precisely, scenarios), which specify actions that must or must not be taken when certain conditions hold and/or when certain sequences of events or other actions take place. Humans and computers would execute these instructions by repeatedly considering all rules, and progressing when one of these, or several of them, instructed them to do so.

The advantages of this approach include ease of understanding by humans of the actual steps that an algorithm would call for,[1] allowance of runtime recognition and tackling of conditions that were not in the specification, and amenability to incremental enhancement of the specification itself for accommodating new or refined requirements. In addition, and most importantly, the approach provides a unique opportunity to introduce human-understandable idioms for algorithmic self-reflection and meta operations, in which the scenarios and the instructions therein become part of the data visible to the algorithm.

As SBA is intended to align with how people often describe their thinking about problems and solutions, causality, planning, and design, we argue that a successful implementation and adoption of the method can have a profound effect on many facets of education in science and in human thinking skills in general.

The remainder of the paper is structured as follows. In Sect. 2, we present the research background that inspired the paper, and in Sect. 3 we provide more details on SBA and how it works. Section 4 presents technical and methodological considerations relative to the SBA approach to specification. In Sect. 5, we discuss the software engineering benefits we anticipate stemming from a successful implementation and adoption of SBA. In Sect. 6 we elaborate on why we believe that adopting SBA can contribute to education and general skill acquisition. In Sect. 7 we propose research and development actions that can contribute to broad usage of SBA and to empirical confirmation of the expected benefits, followed by a variety of research areas that can further serve to expand SBA concepts and their impact.

2 Rationale

Regardless of whether an algorithm is specified in code, pseudo code or natural language, working with such specifications is hard. First, points that are critical to understanding and implementing the algorithm may be hidden. These can include specific innovative steps (as distinguished from mundane bookkeeping steps), or complex, sensitive computations, where every typo in the code or

[1] In [17], Juraj Hromkovič quotes Max Planck: *"Science is an innerly compact entity. Its division into different subject areas is conditioned not only by the essence of the matter but, first and foremost, by the limited capability of human beings in the process of getting insight."* To this profound observation by a giant, may we modestly add that perhaps such a division might also be helpful to humans' effort to understand ever-more-refined concepts and entities, algorithms included.

error in reading could have far-reaching consequences. Furthermore, the essence of simulation results and run traces that are meant to highlight such points may be cluttered by numerous distracting and less interesting steps. Classical code would also be sensitive to hidden assumptions, such as certain properties of a data structure or of the data stored in it, and it may be difficult to identify all places in the code that should be changed when such an assumption is modified or when new requirements are introduced. In addition, much time may be spent on understanding the reason for including a line of code that is of little importance, merely handling some highly atypical or even obsolete case.

Granted, several common software development practices have been devised to alleviate some of these concerns. They include encapsulating and abstracting certain sets of steps in designated methods, giving meaningful names to program entities, writing comments in the code and in external documentation and presentations, and running selective or abstracted simulations and traces. However, the above issues in understanding and maintaining algorithms still persist.

Over the last three decades much research in computer science has been dedicated to methodologies for dealing with reactive systems [14]. A particular question is how to best describe the full behavior of a system that has to constantly react to stimuli and inputs from its environment. The meaning of the adjective *best* in this context usually reflects some balance of expressive power, intuitiveness to humans, executability, compositionality, and amenability to (formal) analysis. Of particular relevance to the present paper is the introduction of Statecharts [9] and of scenario-based programming (SBP) [4,10,13,16]. This paper concentrates mainly on the latter, though in Sect. 7 there is a comment about the potential of the orthogonality feature of Statecharts for the present context.

2.1 Scenario-Based Specifications

In this section we provide a brief introduction to the development paradigm of *scenario-based specification* (or *modeling*, or *programming*, all of which we abbreviate as SBP). We focus on the principles and current research in SBP, emphasizing the capabilities and promise of the approach, towards our quest of using it in the context of algorithm specification. For more background and details we refer the reader to [4,7,8,10,11,13,16,19] and references therein.

SBP was introduced in [4,13], using the *live sequence charts* (*LSC*) formalism. The approach aims to streamline and simplify the development of executable models of reactive systems, by shifting the focus from describing individual objects and components into describing behaviors of the overall system. The basic building block in this approach is the *scenario*: an artifact that describes a single behavior of the system, possibly involving multiple different components thereof. Scenarios are multi-modal: they can describe desirable behaviors of the system or undesirable ones, and combinations thereof. A set of user-defined scenarios can then be interwoven and executed, yielding cohesive, potentially complex, system behavior.

In SBP, a specification, a model, or a program, is a set of scenarios, and an execution is a sequence of points in time, in which all the scenarios synchronize. At every such behavioral-synchronization point (abbreviated *bSync*) each scenario pauses and declares events that it *requests* and events that it *blocks*. Intuitively, these two sets encode desirable system behaviors (the requested events) and undesirable ones (the blocked events). Scenarios can also declare events that they passively *wait-for*—stating that they wish to be notified if and when these events occur. The scenarios do not communicate their event declarations directly to each other; rather, all event declarations are collected by a central *event selection mechanism* (*ESM*). Then, during execution, at each synchronization point the ESM selects for execution (*triggers*) an event that is requested by some scenario and is not blocked by any scenario. Every scenario that has a pending request for, or is waiting for, the triggered event is then informed, and can update its internal state, proceeding to its next synchronization point. Figure 1 (borrowed from [12]) demonstrates a simple behavioral model.

Sensor and actuator scenarios tie events to physical interfaces. Sensor scenarios can access the physical environment or other components of the software system through specialized devices and the software interfaces that the respective devices or components offer. They report the information to the SBP system by translating it into requested behavioral events. Actuator scenarios wait for the SBP system to trigger behavioral events that signify desired changes to the environment or to other components, and call the respective interfaces to bring about these changes.

Several facets and generalizations of scenario-based modeling and programming (also termed *behavioral programming*) have been discussed and handled in different ways. Scenarios can be specified and represented graphically, as in the original LSC approach (see, e.g., Fig. 2), in natural language (see, e.g., Fig. 3), by two-dimensional blocks that are dragged-and-dropped on the programming canvas (see, e.g., Fig. 4), in specially-designed textual languages (see, e.g., Fig. 5) or in standard programming languages like Java or C++ (see Fig. 7).

Scenario-based models can be executed by naïve *play-out*, by smart playout with model-checking based lookahead, or via controller synthesis. The modeling process can be augmented by a variety of automated verification, synthesis and repair tools. However, from this and other research it seems that the basic principles at the core of the approach, shared by all flavors, are *naturalness* and *incrementality*—in the sense that scenario-based modeling is easy to learn and understand, and that it facilitates the incremental development of complex models [2,6]. These properties stem from the fact that modeling and programming are carried out similarly to the way humans explain complex phenomena to each other, detailing the various steps, rules and behaviors one at a time.

Years ago, as part of the work in our group on SBP for reactive systems, Nimrod Talmon raised the question of whether SBP techniques could be applied to other forms of processing, such as database management or classical operations on data structures. This of course immediately points to the issue at the center of this paper—can the specification and development of general algorithms benefit

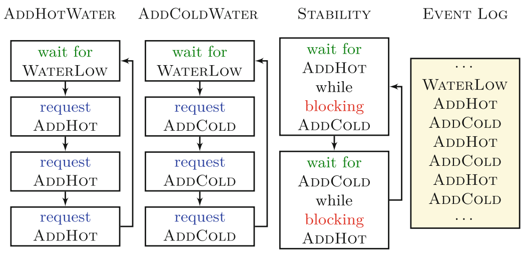


Fig. 1. Incrementally modeling a controller for the water level in a tank. The tank has hot and cold water sources, and either one may be turned on and off in order to control the temperature and quantity of water in the tank. Each scenario is given here as a transition system, where the nodes represent synchronization points. The labels on the edges are omitted here as they can be readily inferred from the context of requested or waited-for events, The scenario AddHotWater repeatedly waits for WaterLow events and requests three times the event AddHot. Scenario AddColdWater performs a similar action with the event AddCold, capturing a separate requirement, which was introduced when adding three (hot) water quantities for every sensor reading proved to be insufficient. When a model with scenarios AddHotWater and AddColdWater is executed, the three AddHot events and three AddCold events may be triggered in any order. When a new requirement is introduced, to the effect that the water temperature should be kept stable, the scenario Stability is added, enforcing the interleaving of AddHot and AddCold events by alternately blocking AddHot and AddCold events. The execution trace of the resulting enriched model is depicted in the event log.

from the developments in approaches to specification, execution and analysis of reactive systems. In this paper, we argue that although the algorithm itself need not be reactive in the sense of being heavily characterized by interactions with its environment, the answer is a resounding *yes*, partly due to our observation that the interaction of the algorithm's control with its data structures has the main characteristics of classical reactivity, with the data structures 'playing the role' of the environment. In addition, we may draw the following analogies:

1. The computational problem that an algorithm has to solve, and/or properties of its results align well with the concept of *requirements* (as provided by users, customers, system engineers and other stakeholders) in system specifications.
2. Descriptive texts that commonly accompany algorithms, which cover design principles, operational highlights, and delicate or innovative points, also align well both with the concept of requirement specification and with detailed operational scenarios of reactive systems.
3. The incremental development and refinement of an algorithm, and the reprogramming of established algorithms, which are often narrowly scoped and are done 'one step at a time', also suggest a behavioral description of those constituent steps: "when condition C1 holds, and/or after actions A1, A2 and A3 take place, the algorithm must always carry out (also) action A4".

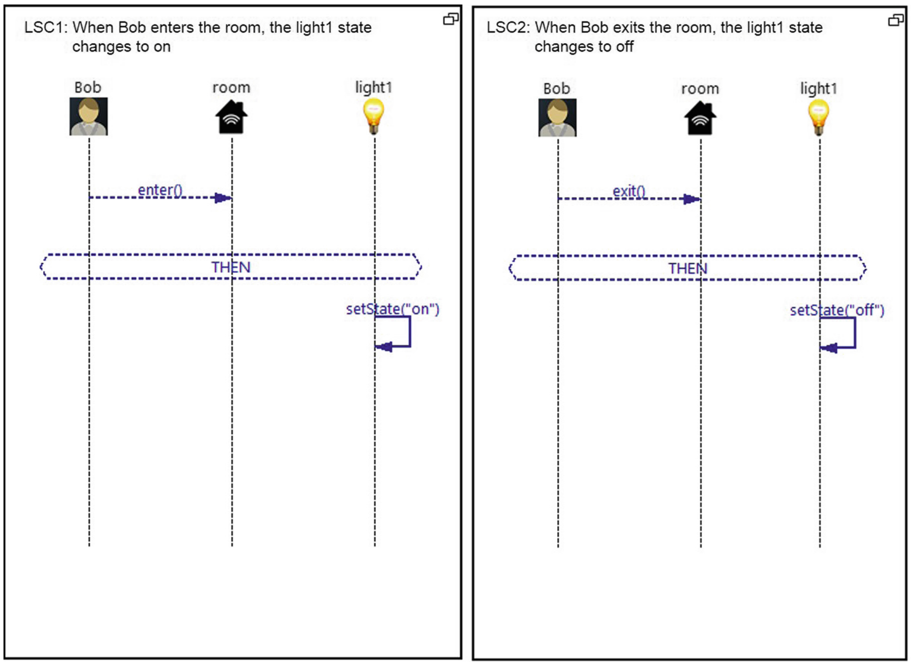
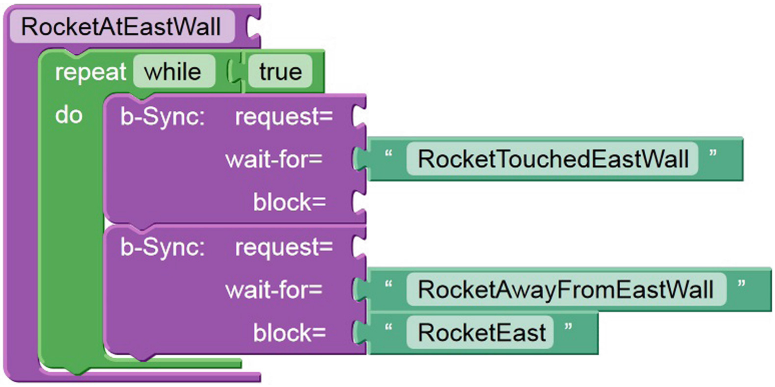

LSC1: When Bob enters the room, the light1 state changes to on
Bob
room
light1
enter()
THEN
setState("on")
LSC2: When Bob exits the room, the light1 state changes to off
Bob
room
light1
exit()
THEN
setState("off")

Fig. 2. LSC scenarios specifying automated light behavior in a smart home. The function of LSC1 is described by the comment *"When Bob enters the room light1 state changes to on"*. This scenario waits for the event of Bob entering the room, and then causes the triggering of the event of the light (in this case - the one named light1) turning on. The semantics of the LSC implies that time flows from top to bottom; participating objects are represented by vertical 'lifelines'; arrows represent events or messages and are annotated as monitored or executed, mandatory or optional, and desired or forbidden.

```
"When Bob enters the room light1 state changes to on."
```

Fig. 3. A scenario specified completely in natural language. The text shown is after interactive disambiguation for specific object identification and for resolving, if needed, of whether a particular word refers to an object, a method, a property, a property value, etc. This natural-language specification is then translated automatically into an LSC (shown in Fig. 2) to be executed in the LSC development environment PlayGo.

Thus, if indeed many aspects of traditional algorithm specification can be implemented using the methods that apply to reactive systems, one would gain the benefits of the latter methods such as intuitiveness and ease of understanding by humans, direct executability, and compositional formal analyzability.



```
RocketAtEastWall
  repeat while true
  do b-Sync: request=
             wait-for= "RocketTouchedEastWall"
             block=
     b-Sync: request=
             wait-for= "RocketAwayFromEastWall"
             block= "RocketEast"
```

Fig. 4. A scenario in the BP-for-Blockly language. This is the code for the behavior of a wall in a 3D game, in which the user attempts to land a rocket using mouse and keyboard commands. This scenario waits for the rocket to reach the wall, and then blocks further movement in the same direction until the rocket moves back, away from the wall. In the Blockly language (from Google), commands are organized as puzzle pieces that fit together to form program modules. In BP for Blockly each scenario is a separate set of commands, as shown here.

```
assumption scenario DriverObeysStopSignal {
 car->dashboard.showStop()
 car->dashboard.showGo()
} constraints [
 forbidden env -> car.carMovesToNextArea()
 forbidden env -> car.carMovesToNextAreaOnOvertakingLane()
]
```

Fig. 5. A scenario specified in Scenario Modeling Language. The context is that of automated control of cars moving in a two-lane road, and it indicates the fact (in this case, an environment assumption rather than a system behavior), that the human driver obeys the signals of the car's dashboard and that once the dashboard displays a STOP command (which triggers this scenario in the first place), the car will not move forward and will not move to the next lane either.

2.2 A Note on Predictability and Executability

While SBP claims to be usable for creating executable code for the final operational system, at this point we do not (yet) claim that playing out a scenario-based specification of a classical algorithm would be an efficient way to execute the algorithm. Instead, we claim that it will provide a highly understandable way to *specify* an algorithm and to follow its runs. In particular, playing out the specification with particular inputs, subject to SBP principles, can yield executable or predictable step-by-step instructions equivalent to those provided, say, by pseudo-code. We realize that this paragraph is intended to be something of a teaser, but it will become clearer as we proceed.

3 Scenario-Based Algorithmics

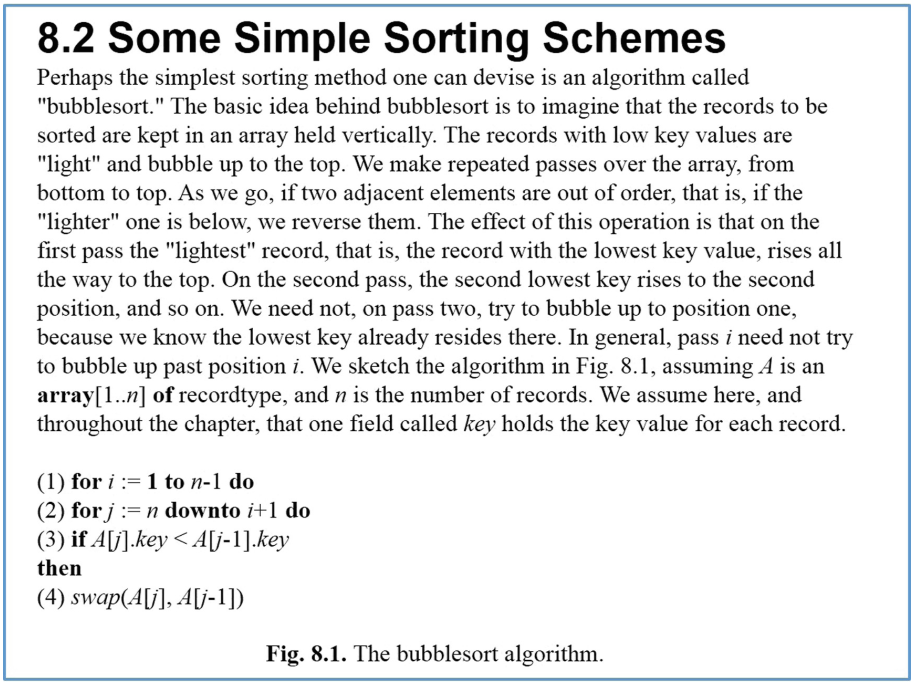
3.1 A Motivating Example: The Bubble-Sort Algorithm

To explain the concepts of SBA we first consider an example. Figure 6, taken from [1] contains program code for the bubble-sort algorithm, preceded by some textual introduction.

While the algorithm is simple and well known, we can still imagine an expert programmer explaining to a novice some delicate points and emergent concepts, which, in this short program with no additional comments, are only implicit. Such an expert explanation would attempt to be technical and precise, and to be complementary to the bubble metaphor in the introductory text but not to replace it. In fact, with a good technical explanation the metaphor might not be needed. Below are some examples of possible observations that the expert and novice would share. Note that other than the definition of the terms and concepts that should appear before they are used, the observations can be provided in any order:

1. "Generally, when we (*we* here refers to the algorithm of course) see two adjacent records in which the keys are out of order we switch the records' locations."
2. "Swapping locations of two entries moves the one located at the higher array index to a lower index location, namely, closer to the beginning of the array."
3. "Repeating the above process in any array, moves ('bubbles-up') the record with the smallest key to the first cell of the array, possibly rearranging all other records in the array's remaining 'tail'."
4. "In this algorithm, the resulting sorted array occupies the same place as the original."
5. "The algorithm starts by bubbling up the lowest-key record in the input array."
6. "Whenever we finish bubbling up the record with the smallest key to the first location of a given array $\mathcal{A}$, the algorithm does similar bubbling-up in the sub-array $\hat{\mathcal{A}}$, which contains all the cells of $\mathcal{A}$ from the second entry to the last one."
7. "The algorithm stops after processing the array consisting of the last two cells in the original array."
8. "For the 'bubble-up' metaphor we consider low array indices as 'up' and high array ones as 'down'."

Our goal is to establish programming idioms and development methodologies that enable the working code of the algorithm to directly reflect to humans all (and only) such relevant insights. Note that the difficulties in understanding that we are trying to address are not in the apparent non-intuitiveness of programming expressions like "`for i := 1 to 10`" or of how the scope of a loop is demarcated. In fact, many would argue that these expressions *are* quite intuitive, and if not, that they can be improved by some form of one-to-one translation. However, we believe that the core issues are that (i) many insights or guidelines about an algorithm's execution involve multiple (not necessarily adjacent) lines



8.2 Some Simple Sorting Schemes

Perhaps the simplest sorting method one can devise is an algorithm called "bubblesort." The basic idea behind bubblesort is to imagine that the records to be sorted are kept in an array held vertically. The records with low key values are "light" and bubble up to the top. We make repeated passes over the array, from bottom to top. As we go, if two adjacent elements are out of order, that is, if the "lighter" one is below, we reverse them. The effect of this operation is that on the first pass the "lightest" record, that is, the record with the lowest key value, rises all the way to the top. On the second pass, the second lowest key rises to the second position, and so on. We need not, on pass two, try to bubble up to position one, because we know the lowest key already resides there. In general, pass *i* need not try to bubble up past position *i*. We sketch the algorithm in Fig. 8.1, assuming *A* is an **array**[1..*n*] **of** recordtype, and *n* is the number of records. We assume here, and throughout the chapter, that one field called *key* holds the key value for each record.

```
(1) for i := 1 to n-1 do
(2) for j := n downto i+1 do
(3) if A[j].key < A[j-1].key
then
(4) swap(A[j], A[j-1])
```

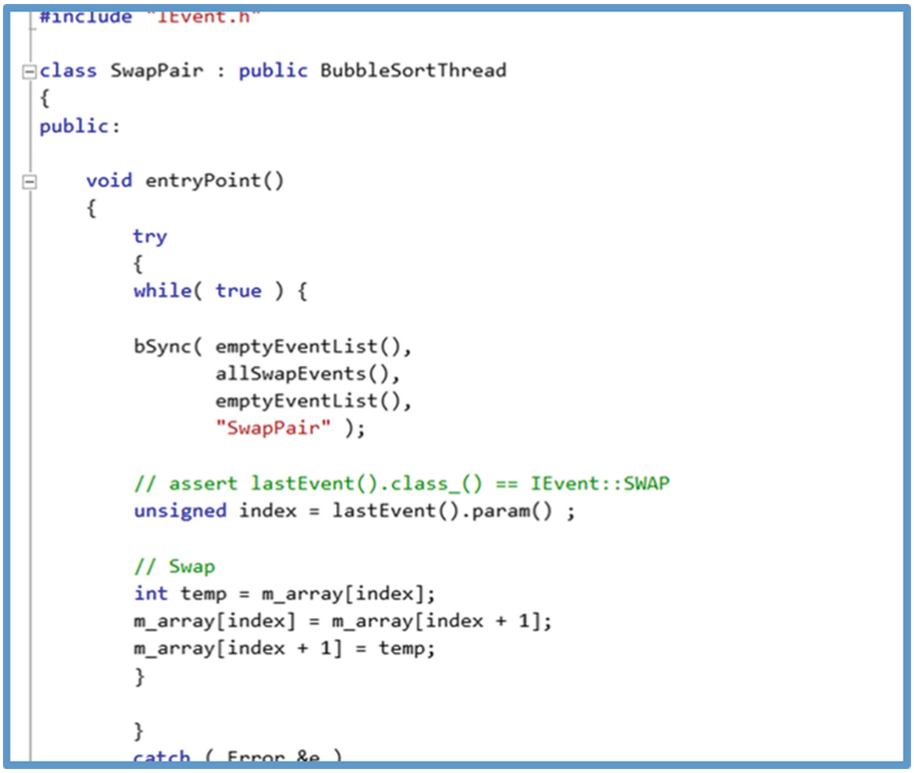
Fig. 8.1. The bubblesort algorithm.

Fig. 6. Textbook presentation of the bubble-sort algorithm: introductory text and code, taken from Aho, Hopcroft and Ullman [1].

of code, (ii) a given line of code may serve multiple guidelines, and (iii) some central insights may be hidden in a small notation artifact like a pair of parentheses, a tagged symbol, or a -1 in an expression. Hence, we are looking for a major shift in how the program is structured in the first place, and in how the developers express their thinking. Once we discover such a shift, the individual programming idioms will be gradually refined until they are sufficiently natural and intuitive for the stakeholders who need or want to read them.

3.2 A Scenario-Based Specification of Bubble-Sort

We now provide the scenario-based specification of the bubble-sort algorithm from [1]. A snippet of the behavioral programming code (in C++) appears in Fig. 7. For brevity and clarity the full list of events and scenarios is described in natural language (English) rather than in C++. Our main focus and interest at this point is the intuitive composition of scenarios, each of which performs a function that is intuitively understood, and which often relates directly to the original requirements. As discussed in Sect. 7, we do not include here means for enhancing the language used to code the individual scenario. Doing so in ways that result in scenario code that is itself shorter and/or more intuitive is something we leave as a future research objective.



```
#include "IEvent.h"

class SwapPair : public BubbleSortThread
{
public:

    void entryPoint()
    {
        try
        {
        while( true ) {

        bSync( emptyEventList(),
               allSwapEvents(),
               emptyEventList(),
               "SwapPair" );

        // assert lastEvent().class_() == IEvent::SWAP
        unsigned index = lastEvent().param() ;

        // Swap
        int temp = m_array[index];
        m_array[index] = m_array[index + 1];
        m_array[index + 1] = temp;
        }

        }
        catch ( Error &e )
```

Fig. 7. Code snippet for a scenario coded in C++. This is an actuator scenario (behavior thread) that waits for all `SwapPair` events, and then performs the actual swapping. The invocation of the `bSync` method is the synchronization point where the scenario synchronizes with all other scenarios in the specification, and some event that is requested and not blocked is selected and triggered. The three parameters of the `bSync` method are the requested, waited-for and blocked events, in that order.

The events in our system are:

- `CHECK_ORDER(n)`: A request to check if the pair of array cells indexed by (i.e., at locations) n and $n + 1$ are in the correct order. On the one hand, this is the backbone event of the algorithm and the order of its occurrence drives the order of most other events in each run. On the other hand, this can be viewed as a simple actuation of a sensor, which could be driven periodically (i.e., by the elapsing of some unit of time) or by other events.
- `UNSORTED(n)`: This is a sensor event that reports that the pair of array cells indexed n and $n + 1$ were compared and discovered to be out of order.
- `SORTED(n)`: This is a sensor event that reports that the pair of array cells indexed n and $n + 1$ were compared and discovered to be in correct order.
- `EXTERNAL_CHANGE`: A sensor event that reports that something in the array has changed (in our case, probably due to a sort-related action).

- `START`: Triggers the entire process.
- `PREFIX_IS_SORTED(n)`: The first n array cells are sorted and their values are smaller than those in all other cells.
- `ENTIRE_ARRAY_IS_SORTED`: Reports that the process is complete.

The scenarios in the specification are:

- `Sensor:` This sensor scenario is the only one that provides reports about the conditions of the environment, namely the array. It waits for any `CHECK_ORDER(n)` event, compares the values of the respective cells and then requests `SORTED(n)` if the value of the lower indexed cell in this pair is smaller than or equal to the value of the other cell, and it requests the `UNSORTED(n)` event otherwise.
- `SortPair`: This scenario is the essence of the sorting logic: it waits for any `UNSORTED(n)` event, and requests a corresponding `SWAP(n)` event. For completeness, in our implementation, this scenario also requests the `CHECK_ORDER(n)` event, to (i) make sure that the swap action completed successfully, and (ii) to facilitate the straightforward triggering of a `SORTED(n)` event from the same sensor scenario (as opposed to an insightful result report from this `SortPair` scenario or from the SwapPair scenario.
- `SwapPair`: This actuator scenario waits for all `SWAP` events, and carries out the swapping of the contents of respective cells.
- `BubblePair(n)`: This scenario controls the order of execution. When a pair of adjacent cells is discovered that needs to be sorted, it initiates the sensing and subsequent possible sorting of the next pair in the array.
- `BubbleStartNextPass`: This scenario controls the iterative processing of bubble-sort—the next bubble. Whenever an unsorted pair is sensed, it waits for the first pair in the array to be declared as sorted, and then requests that the last pair in the array be checked again.
- `PrintArray`: This is a support scenario that prints/displays the entire array after every request for comparing any two cells.
- `LogSelected`: Another support scenarios that logs to an external file all the events that were selected for triggering.
- `BubbleSortStart`: This scenario starts the entire process by simply requesting the `CHECK_ORDER` event for the last pair in the original array.

We played out the specification using the BPC execution infrastructure for behavioral programming in C++. As expected, the trace showed the events in the same order of comparisons and swaps that we would expect from the classical algorithm, and a visualization of data movements within the array showed the expected bubble-up process.

4 Methodological Notes

4.1 Creating SBA Specifications

Creating a detailed methodology for developing scenario-based algorithm specifications, either as a transformation of a classical specification, or when starting from scratch, is a topic we leave for future research. Below, we present part of our work-in-progress efforts in this area, namely, key steps in an initial methodology for converting classical algorithm specifications into scenario-based ones. The steps can be repeated until the results are satisfactory.

- Identify the main data structures manipulated by the algorithm, and see whether they can be considered as objects with which the control part of the algorithm interacts.
- Identify main conditions or steps in the process and create behavioral events whose triggering indicates that the condition holds or the step took place.
- Describe in natural language key points of the algorithm, as one would do in an introductory or summary paragraph in a textbook (e.g., "always after event E1 we take action A2"). Attempt to translate each of these into a formal scenario.
- Encapsulate specific complex computations that are *not* a part of the essence of the algorithm as sensor and actuator scenarios.
- Examine statically the step-by-step algorithm specification, identify desired elements that are not covered yet, phrase them in self standing natural language statements (e.g. "and I have forgotten to say that after we do A3, if condition C4 holds, then before we do the action A5 as is currently prescribed, we first do action A6"), and add these (eventually by automatic translation into scenarios) as additional, stand-alone formal scenarios.
- Play out the scenario-based specification. Whenever there is a choice of multiple enabled events (say, dictated by different scenarios), add a scenario that makes a deterministic choice for this case, if so desired. If a deterministic algorithm is a must, then such choices may be arbitrary at times, as is sometimes the case in ordering algorithm steps whose order does not matter in a classical specification.
- Compare the SBP playout trace with a run of the step-by-step algorithm. Whenever there is a difference, study it, and adjust the SBP specification.
- Use formal verification tools or systematic test tools to confirm that the two implementations yield the same results for large ranges of data.

4.2 'Events vs. Method Calls' or 'Dedicated Scenarios vs. Method Parameters'

Scenario-based specifications of algorithms may at first be reminiscent of ordinary programming, with methods replaced by scenarios, and method calls replaced by broadcast events. Such an event carries information about its destination scenario or object (which corresponds to a method name) as well as its execution parameters (corresponding to the method-call parameters or return values). However, the SBA approach provides a powerful alternative view:

In the extreme case there would be no parametric information passing. Instead, for each object, field, record, or relevant set(s) thereof, and for each behavior that this entity has to exhibit, the developer would create a dedicated scenario. This scenario stands as a sentinel, deliberately ignoring almost everything that is happening in the system, and if and only if certain conditions that apply to it are satisfied, or events that it is interested in occur, the scenario requests one or very few particular actions. Once these actions take place (and, of course, only when they are not blocked by other SBP scenarios) it reports this fact, with little or no knowledge or 'interest' in what will be done with the results, or which component will use them.

This view provides new dimensions of encapsulation, parallelization opportunities for more efficient execution, and opportunities for functional refinement, while allowing individual scenarios to be "taught" (explicitly by humans or via automated learning) how to handle yet new conditions as may be encountered within their purview.

For scalability purposes, e.g., when the number of behaving entities is large, and where some of which are dynamic in nature (i.e., are created and discarded at run time), common optimization techniques—like using a pool of worker threads—can be applied towards efficient utilization of computer resources, without diminishing the value of utilizing automated 'narrow-minded' processes working 'with their blinders on'.[2]

This approach to dynamic scenario instantiation also provides an approach for recursion that is presently built into standard programming languages. This 'magic' of using the same code at different states, with the relevant context kept in the computer's or the human's mental stack, will be replaced in SBA with the explicit creation of emergent objects like ever-shorter tails of an array, ever-smaller subtrees, etc.

4.3 A Note on Feasibility

An important question regarding our approach is whether many or most interesting algorithms can be specified in SBA. In two extreme senses, the answer is trivially positive: First, one could simply encapsulate the entire algorithm in a single actuator scenario, which simply waits for one `Start` event with appropriate parameters and carries out the desired computations internally. Second, the structure of the algorithm can be left unchanged, but every command in the classical specification would be converted into a scenario, and these scenarios communicate via helper events. E.g., line 17 in some sequential code segment that is contained in a loop can be specified as the following scenario "Repeatedly, wait for the event `AnotherIterationOfThisLoopStarted`; wait for the event Line16Completed; then, forbid the event `line18Started` until further notice; request (and wait for) the event `Line17Started`; carry out as an actuator what line 17 does in the original algorithm; request (and wait for) the event `Line17Completed`; and, finally, release the blocking of `line18Started`."

Such extreme cases are obviously not what we are after. Instead, the goal is to identify methods and principles for scenario-based creation of algorithms that will have nontrivial added value, enhancing understanding and simplifying the development of the algorithms.

[2] Those that prevent horses from being distracted or alarmed.

5 Advantages of Scenario-Based Algorithmics

As stated earlier, one of our goals is to exploit SBA to enrich the intuitiveness, clarity, alignment with the requirements, incrementality, executability, and amenability to formal verification of general algorithm specification and development. Though systematic empirical studies proving this are yet to be carried out, we believe that SBA can indeed contribute significantly to these.

An additional non-obvious advantage is succinctness. This claim may be surprising, since even in the relatively simple bubble-sort example, the natural-language scenarios were quite verbose as compared to the original pseudo code, and the C++ code was even more so. However, along the lines of formal succinctness proofs of SBP [11], we argue that *eliminating* from the specification many classical structural elements, such as intricate control flow, and the exact locations of method invocations, can serve to replace one large module, i.e., the entire algorithm, with multiple, smaller, more robust modules, i.e., the scenarios. Our measure of size here is not the number of characters or lines (which some languages can compress very efficiently), but the number of different states and conditions and flows that a module represents. Thus, we consider the difficulty of understanding the SBA specification to be proportional to the sum of the sizes of the scenarios, while understanding the original specification is proportional to its own size, which in some cases can be closer to the product, rather than the sum, of the sizes of the constituent scenarios.

Reflection in software is the ability of a programmed entity to look at itself and at other such entities and dynamically adjust its own behavior accordingly; e.g., checking at run time if the definition of a target object class includes a particular method call, and when this is the case calling this method on objects of this class. SBA enables extending such reflection significantly. Consider the following hypothetical excerpt from an email correspondence between a development manager (M) and a developer (D) on his or her team:

- (M): *"Please change all program modules where we do action A, to first check for condition C."*
- (D): *"But in program module P this change would be incorrect!"*
- (M): *"You are right, indeed, do not change program module P, and also do not change module P2, as it suffers from the same issue."*

Collectively these requests and decisions describe precisely the revised behavior of the system. Indeed, when the developer completes the task, the manager would know what the system does. However, we should note that this specification was not the same as the behavioral scenario *"Action A should occur only when condition C holds"*, but a structural one. As a proof, consider the fact that we do not know what is it about modules P and P2 that render the requirement inapplicable in those contexts. Such reflective capabilities are available in a variety of programming language, as well as in Statecharts, where the components can refer at development time and at run time to (states of) other components. SBA allows us to readily and incrementally enhance a specification with scenarios of this type.

SBA can also offer new insights into understanding data and behaviors. Consider for example the *Sieve of Eratosthenes* method for finding the prime numbers. The classical algorithm calls for a data structure in which cells corresponding to multiples of discovered primes are marked as non-primes, and the cells remaining at the end of the process are output as primes.

In one SBA design, we modeled the method as an iterative-cumulative process. Once a prime p is discovered (starting with $p = 2$) a scenario is added that iteratively requests *events* declaring all multiples of p as non-primes. In parallel, another scenario iteratively requests events that declare an integer to be a prime but abandons that request upon the occurrence of the event declaring the said integer as a non-prime. We then used event priorities in the Behavioral-Programming-in-Java platform (BPJ) to make the "p is not a prime" event take precedence over "p is a prime", which yields the final desired results.

In another implementation (relying also on BPJ's support for infinite sets in the declaration of blocked events and waited-for events), for each discovered prime p we replaced the requesting of events that declare multiples of p as non-primes with the creation of a scenario that blocks the event "n is a prime" for all values of n that are multiples of p. In yet another possible implementation, instead of dynamically creating scenarios associated with discovered primes, one could create a scenario for each integer in the range to block declaring its multiples as primes.

The attractiveness of these SBA implementations is not just in that they replace the need for the data structure in the Sieve of Eratosthenes algorithm (note that they clearly do not save in computer memory). While loyal to producing the desired functional results, the SBA versions, using priority and event-blocking semantics, can rely on the infrastructure for some bookkeeping that in the classical versions are specified more explicitly and prominently (as storing of interim results and evaluating conditions).

6 SBA and Human Thinking

"Thinking like a computer scientist" [20] (or "thinking like a programmer" as it is often referred to), as well as understanding the basic principles behind algorithms and computer science at large [15], are considered to be desired mental capabilities, which are valuable in many real life contexts. Much important work has been done on endowing the population at large with such skills, even from very young ages (see, e.g., [3,5,18] and references therein). To these efforts we wish to add the following observation: once we are successful in explaining to humans, in human terms and in a systematic and repeatable way, what systems and computers and algorithms do, then perhaps we will be able to distill the underlying principles of such useful human terms, and start programming systems and computers and algorithms using those very terms themselves.

Once this is achieved, it may open up new opportunities for both humans and machines. E.g., a person who is often personally baffled by complex decisions and planning challenges may acquire necessary skills not just from occasional tips from friends and books, but from the systematic understanding of the principles of how powerful computer systems (like road navigation or warehouse stock management systems) accomplish the same. With SBA, these principles will stand out better. Conversely, once systems are structured in this manner (built with SBP and SBA), humans will be able to experiment with those systems, and even enhance them with an ever-growing number of desired features and capabilities. The phrasing of an idea, like "can the system do action A1 when condition C1 holds?" will be directly translated into a component (a scenario) that can be tested independently and then be verified together with the system, in order to check whether it introduces unexpected problems or conflicts.

7 Future Research

The initial proposal described in this paper must clearly be followed by further development and research. First, existing and new integrated development environments (IDEs) should be built to support the broad application of scenario-based development in general, and SBA in particular. This should be geared to audiences in science, engineering and education, tending to the users' different needs, and with a special focus on SBA intended strengths, such as understandability and maintainability of algorithms. The expected benefits in areas such as productivity, quality and learning should then be empirically confirmed, by measuring specific results in synthetic and realistic projects carried out by many participants, and comparing the SBA-based results to the ones obtained using known techniques.

SBA also presents interesting research questions and technical challenges. Key among these is the efficient implementation of direct execution of a scenario-based specification. Work in this direction is underway along various tracks, including synthesis of an efficient monolithic automaton from a scenario-based specification, efficient approaches for distribution and parallelization, while relaxing some of the behavioral synchronization requirements, hardware circuits designed especially for assisting in the SBP event selection process, and more. Depending on the success of such solutions, especially distribution and hardware assists, we will be interested in exploring the costs and benefits of dedicating large numbers of parallel-running behavior threads (e.g., in the case of bubble-sort, one per every pair of array cells, or one per array 'tail', etc.).

For evaluating the correctness of an SBA-specified algorithm that is also available in a classic form, one should create a log for the original specification and compare the logs of the two implementations. Proving full equivalence is a separate task.

From a software engineering standpoint, it would be interesting to formalize and then discuss the syntactic and semantic differences between scenarios and method calls. In such a discussion, one would compare, e.g., the fact that each scenario can be cognizant of the global (system-wide) conditions and events under which it should be called, with the fact that in classical method calls it is the collective responsibility of all the callers to invoke the method under all relevant conditions and following all relevant event sequences. Of particular interest are the auxiliary events that are added to indicate a particular interim condition or occurrence (e.g., that a certain pair of array cells is or is not sorted). We believe that one would be able to show that while such events may seem to be merely 'helper' events, which would not be necessary in a sufficiently powerful programming language, they indeed provide a solid foundation for the specification. This would be similar to storing interim computation results (e.g., of expressions in parentheses or numerators and denominators of a division in separate well-named variables), not just for shortening and clarifying each computation, but for highlighting the existence and subsequent use of this emergent entity.

Another interesting topic for future research is the incorporation of scenario-based techniques into the Statecharts formalism. In a research project underway, we are endowing the orthogonal state components accommodated by Statecharts with the ability to declare requested, blocked and waited-for events, and are enhancing the event triggering and composition semantics of Statecharts to accommodate the SBP event selection semantics. This can allow the SBA approach to be carried out in yet another, more familiar, visual formalism.

Finally, the programming idioms at the foundation of SBP languages have in mind reactive systems operating in a real-world environment. It would be interesting to create a standard methodological mapping (or at least a consistent approach thereto) from classical entities and concepts that appear in algorithms, like complex data structures or recursion, to the scenarios and events underlying SBP. Furthermore, an SBP language may be enhanced with basic idioms geared specifically for scenario-based algorithmics. We are particularly interested in finding concise, yet explicit, idioms for cryptic or implicit notations; e.g., that in a program with an array of size n, whose indexing begins at 0, the expression $n-1$ is the index of the last item as well as the length of the 'tail' sub-array after dropping the first array cell. These can, of course, be conveyed in meaningful names for temporary variables, but perhaps some orderly methodology could be discovered in order to balance clarity and clutter.

8 Conclusion

We have described the SBA approach to algorithm specification, where each of the algorithm's steps and special properties is specified in a stand-alone manner, in any order, and for which the step-by-step execution is derived from the collective parallel execution of all these specification artifacts. Data structures play the role of the environment of a reactive system for such steps and properties. While additional experimental work with students and engineers is still needed, SBA promises several benefits: It can facilitate better human understanding of algorithms, which can ultimately contribute to the quality of computerized systems, and it can enable, and even drive, the enhancement and improvement of algorithms, since one can more naturally focus on special features while handling bookkeeping steps and special extreme cases separately.

The approach may also bring the development of reactive systems and classical computation closer—enabling a broader or deeper application of techniques developed for reactive systems in the classical areas of algorithm design and programming. Formal verification can be one such example.

Finally, the SBA approach might also contribute to the teaching and learning processes around computer science and software engineering, and perhaps even enhance the penetration of a variety of human skills associated with computational, or algorithmic, thinking [15,17,20].

Acknowledgements. We thank Nimrod Talmon for initial conversations that partly triggered this pursuit. Thanks to Ori Koren for the prototype of the scenario-based implementation of bubble-sort using behavioral programming in C++. We are also grateful to the anonymous reviewers and editors for their valuable comments. This work was supported in part by grants from the German Israeli Foundation (GIF), the Israeli Science Foundation (ISF), and the Minerva Foundation.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms. Addison-Wesley, Reading (1983)
2. Alexandron, G., Armoni, M., Gordon, M., Harel, D.: Scenario-based programming: reducing the cognitive load, fostering abstract thinking. In: Proceedings of the 36th International Conference on Software Engineering (ICSE), pp. 311–320 (2014)
3. Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K.: Developing computational thinking in compulsory education-implications for policy and practice; EUR 28295 EN (2016). http://publications.jrc.ec.europa.eu/repository/bitstream/JRC104188/jrc104188_computhinkreport.pdf
4. Damm, W., Harel, D.: LSCs: breathing life into message sequence charts. J. Form. Methods Syst. Des. **19**, 45–80 (2001)
5. Denning, P.J.: Remaining trouble spots with computational thinking. Commun. ACM **60**(6), 33–39 (2017)
6. Gordon, M., Marron, A., Meerbaum-Salant, O.: Spaghetti for the main course? Observations on the naturalness of scenario-based programming. In: Proceedings of the 17th Conference on Innovation and Technology in Computer Science Education (ITICSE), pp. 198–203 (2012)
7. Gordon, M., Harel, D.: Generating executable scenarios from natural language. In: Gelbukh, A. (ed.) CICLing 2009. LNCS, vol. 5449, pp. 456–467. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00382-0_37
8. Greenyer, J., Gritzner, D., Katz, G., Marron A.: Scenario-based modeling and synthesis for reactive systems with dynamic system structure in ScenarioTools. In Proceedings of the MoDELS 2016 Demo and Poster Sessions, Co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MoDELS). CEUR (2016)
9. Harel, D.: Statecharts: a visual formalism for complex systems. Sci. Comput. Program. **8**(3), 231–274 (1987)
10. Harel, D.: Can programming be liberated, period? IEEE Comput. **41**(1), 28–37 (2008)
11. Harel, D., Katz, G., Lampert, R., Marron, A., Weiss, G.: On the succinctness of idioms for concurrent programming. In: CONCUR, pp. 85–99 (2015)

12. Harel, D., Katz, G., Marelly, R., Marron, A.: An initial wise development environment for behavioral models. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 600–612 (2016)
13. Harel, D., Marelly, R.: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-642-19029-2
14. Harel, D., Pnueli, A.: On the Development of Reactive Systems. NATO ASI Series, vol. F-13. Springer, New York (1985). https://doi.org/10.1007/978-3-642-82453-1_17
15. Harel, D., Feldman, Y.A.: Algorithmics: The Spirit of Computing. Pearson Education, London (2004)
16. Harel, D., Marron, A., Weiss, G.: Behavioral programming. Commun. ACM **55**(7), 90–100 (2012)
17. Hromkovič, J.: Algorithmic Adventures: From Knowledge to Magic. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-85986-4
18. Kafai, Y.B.: From computational thinking to computational participation in K-12 education. Commun. ACM **59**(8), 26–27 (2016)
19. Marron, A., Weiss, A., Wiener, G.: A decentralized approach for programming interactive applications with JavaScript and Blockly. In: SPLASH Workshop on Programming Systems, Languages, and Applications based on Agents, Actors, and Decentralized Control (AGERE!) (2012)
20. Wing, J.M.: Computational thinking. Commun. ACM **49**(3), 33–35 (2006)

That Most Important Intersection

Lane A. Hemaspaandra

Department of Computer Science,
University of Rochester, Rochester, NY 14627, USA
http://www.cs.rochester.edu/u/lane

Abstract. This article describes the design and components of a Knuth-style problem-solving course for *undergraduate* students, including example problems. The hope is that the course approach is flexible and adaptable enough that it will be relatively portable, and yet will provide a course that for both teacher and students falls in that most important intersection: the intersection of what one can do quite well, and what one can do with much passion and joy.

To Juraj Hromkovič, with deepest thanks for his contributions to both theoretical computer science and computer-science education, and with best wishes for the future.

1 Preface: That Most Important Intersection

A Warning. This is not an article schooled in the rigor of or terminology of educational research, or based on well-designed surveys and studies. Rather, this article consists of some personal, informal, first-person reflections on bringing a Knuth-style research-immersion course to *undergraduates*, and is an attempt to make more generally available the course design, so that others can, if they wish, adopt it and adapt it to their own styles and their schools' needs. In doing so they may, if the stars align, create a course that is a joy for its teachers and students alike, and that helps the students realize that their abilities at problem-solving are far more powerful than they had previously realized.

1.1 In Prydain

One of my favorite book series as a child was Lloyd Alexander's *Chronicles of Prydain* series [2]. If you don't know of it, I'd urge you to find and read it. My guess is you'll be glad you did, and not just because you can then give it as a wonderful gift to your child-aged relatives.

One might at first think the series is merely yet another coming-of-age tale wrapped as is common in what at the start is a magical world. But within the series, one book really jumps out from the rest, and from the genre as whole, as having a message both quite insightful and brutally realistic.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 568–589, 2018.
https://doi.org/10.1007/978-3-319-98355-4_33

In particular, the fourth book in the series is *Taran Wanderer* [3]. In it the protagonist, Taran, starts as an assistant pig-keeper (although to an oracular pig). But Taran doesn't know who his parents are, and feels as if he doesn't even know who he is—whether (in the feudal-flavored setting of the series) high-born or low-born—or what he should be doing in life. So he goes on a journey, seeking his origins, seeking his identity, and in the process trying his hand at many occupations. And at each occupation, it turns out—lucky him—that he is remarkably talented. He turns out to be truly talented in smithing, weaving, and so on. And yet at each, he realizes it just isn't right for him—it simply isn't what he wants to do. Finally, he tries his hand at pottery, and he truly loves it. It is what he wants to do. Yet he slowly comes to realize that this is the one thing he has tried that he is not truly talented in. He could stick with it and be quite competent, but he'd never really produce work that transcends the mundane. And at the end of this whole process, he realizes (with the help of a perhaps not-so-enchanted water puddle that is claimed to show one's true self yet shows him only his own reflection) that it doesn't matter to him whether he is high-born or low-born, and that being an assistant pig-keeper is something he both is specially talented at and loves.

There is a lesson there for everyone. There is nothing better in life than spending one's time on things that one loves—things that truly resonate—and that one is lucky enough to have the talent to do excellently.

And in each of our young lives, we wander seeking that most important intersection, even if we do don't consciously think about the fact that that is what we're doing. If we're unfortunate or unwise or faced with circumstances that constrain us, perhaps we give up and settle, maybe not even realizing that that is what we have done. But if we're very fortunate indeed, perhaps we find our own personal most important intersection, where talent and passion coincide, and we can find joy.

Admittedly, this most-important-intersection view is too rosy and oversimplified. Perhaps Taran would have loved, and been even more talented at, arrow-making. And most of us don't possess utterly sublime talent at anything. Even if we did, passions and skills can shift or diminish over time.

So, perhaps, what one should more plausibly try—even while hoping for far more—is to find an area for which one's passion is great, and one's talent/ability is quite strong relative to what one's talent is in other areas where one senses one would feel passion. Even that is still a wonderful, difficult intersection to find.

1.2 In Academia: Research

Theoretical computer science, my own specialty, is an area where people relatively clearly have found so very well their most important intersection. One looks around, and pretty much everyone truly loves what they are doing. And although we all vary in our talent, and the advance of knowledge needed to solve the hardest questions of the field is a long game that may span many lifetimes, and most people are doing work that has real meaning and depth, and shows their (and their coauthors') fingerprints and flair.

I much suspect that the same can be said within academia for the other subfields of computer science, and—though these are beyond what I have extensive contact with—the other sciences and all of research-oriented academia. In some sense, the process of becoming a professor at a research-oriented school preselects people lucky enough to have found a personal most important intersection in what they are devoting their lives to.

1.3 In Academia: Teaching

However, teaching may be a different story. Especially at times when class sizes are very large, and when the pressure to bring in large amounts of external funding is great, many computer scientists in research universities may not view teaching their courses as either a joy or an area where they have exceptional talent. That is tragic not just for the professors, but also for the students. When a professor loves teaching a course, that shows through to the students, and even if the area is not initially the students' favorite, the students surely will learn far more about the area in a course passionately taught by a professor who loves the course's design and material than they will from one who does not.

The goal of this article is to present the design of a course that I think almost any computer-science professor who is a researcher would be passionate about teaching, and likely could teach very well. In particular, this article presents the flavor of a course my school offers that is an *undergraduate* version of the type of research-immersion course that Donald Knuth famously taught for many years to Ph.D.-level students at Stanford. Creating a course of this sort at one's school would, I suggest, add a class that professors will be falling over each other to teach, and that they will teach skillfully and well. I'll argue below that this course also tends to draw passion from students, and to reveal to them that their ability to study problems—even extremely hard problems—is far greater than they, or most of their professors, previously realized.

2 An Undergraduate Knuth-Style Research-Immersion Course

2.1 Introduction

In the 1970s and 1980s, Donald Knuth famously taught at Stanford's Ph.D.-program a problem-solving seminar, which in effect functioned as a sort of research-immersion course. Professor Knuth would give a problem to the students that was so open and hard that he himself did not know what the answer was. Then the class in groups would immerse itself in solving that problem, over the course of a few class sessions, with the groups reporting their progress at each class session. In a typical term, the class covered five problems. The stress was on *collaborative* problem-solving by groups, with cross-group intermediate discussions to share progress and information. It was a lovely, valuable approach. These classes were widely viewed by the students as extremely important to their careers—a trial by fire in meeting the unknown, in interacting with groupmates and with friendly-rival groups, and in doing one's best to brush the unknown back a bit. Happily, these courses were documented through a series of Stanford technical reports, each titled (the titles differ only in a hyphen) "A Programming and Problem-Solving Seminar" or "A Programming and Problem Solving Seminar," and coauthored by Professor Knuth and the TA, containing transcripts of the class sessions. Those reports make revelatory reading.[1]

The Stanford problem seminar was such a good idea that other CS departments started offering this type of course. My own department did, in the form of our "CSC 400: Graduate Problem Seminar." Of course, each school had its own taste and twists, and even in those places where the courses started out very much like the Knuth course, over time the nature of the course often shifted. My own department, pressed by the desire of many to have students start on research with their faculty advisor as early as possible, has made the main research-time focus of our graduate problem seminar course be an *individual* research project that is usually supervised by the student's advisor. That of course is valuable, but is quite different from the core focus on *collaborative* problem-solving of the Knuth course. Our graduate problem-solving course has many additional goals assigned to it, such as teaching students to avoid plagiarism. In some years, it has been used as a place to introduce all first-year Ph.D. students to the department's specialized hardware toys.

In the mid-1990s, we decided to create an *undergraduate* problem seminar, and some of us wanted to model that course closely on not our own graduate-level version but rather on the marvelous course that Professor Knuth had taught at Stanford, and in particular with a very strong stress on collaborative solving of open research questions—often questions so hard that the professor has little hope of a full "solution" being reached (rather, the hope is merely that the class will find a way to make some progress or gain some insights).

Having taken Professor Knuth's course myself, I was very enthusiastic about this approach, as were a number of my colleagues. Over the years, I have taught this course thirteen times—the first being 1997 and the most recent being 2016—the most of any of my department's faculty members. It took quite a lot of pleading to get the assignment that often, because the course is so much fun to teach that many of my colleagues also want to teach it. Each person teaches it in his or her own way and indeed to his or her own design, and that is a good

[1] However, getting to the reports can be a bit difficult. If one knows the report's number, one can find the report by digging down by year and then number within http://i.stanford.edu/pub/cstr/reports/cs/tr/. For example, the 1989 report numbered 1269 (report STAN-CS-89-1269) is available online at http://i.stanford.edu/pub/cstr/reports/cs/tr/89/1269/CS-TR-89-1269.pdf. That report contains the years and numbers of the seven previous technical reports, which are 77-606, 79-707, 81-863, 83-989, 83-990, 85-1055, and 87-1154. The reports also are generally available on archival microfiche copies at Stanford's library, e.g., search for "Programming Knuth Seminar" at https://searchworks.stanford.edu.

thing. But as colleagues we have of course shared our materials when the person coming after us wished that input. In particular, much credit is due to my dear friend and colleague, Professor Chris Brown, who taught the course the first times it was offered, and then very generously shared with me his meticulously crafted collection of course material.

In my offerings of the course, I have always tried my very best to make the experience be as close as possible to the insanely high expectations and open-research-problems flavor of Professor Knuth's graduate course. Indeed, a student who took both our graduate problem-solving course and our undergraduate problem-solving course commented that the undergraduate version is by far the more demanding.

2.2 Components: Overview

The components that my undergraduate research-immersion course has had over the years, not all being part of each year, are the following.

- The projects.
- The crazy-literature critiques.
- The Computer Science Gladiators Day and the Lightning Research Days.
- The one-on-one meetings.
- The quizzes.
- The guest lectures and the readings.

The following sections focus in turn on these components, giving actual examples for those cases where examples are important in understanding the flavor of the component.

2.3 Component: The Projects

2.3.1 Flavor

The Flow of Each Project. Each project spans four class sessions. At the first I present the project—ideally a problem that is as new and open to me as it is to the class—and divide the class into groups. The ideal number of groups is four, and the ideal size of groups is 3–4, with 5–6 being livable if the course size requires it. When the class is too large for even this, one can make 6–8 groups, and each of what usually would be class sessions 2, 3, and 4 of a given project becomes two class sessions, each with half the groups presenting; so each project then would span seven class sessions. However, for the rest of this section, I will generally assume the case of four groups and four class sessions per problem.

The group names are typically fun. For example, in many years we have gone with Gryffindor, Hufflepuff, Ravenclaw, and Slytherin [21]. More recently, the Great Houses from *A Game of Thrones* [18] have been fun for the students. They will often have the house sigil and the house words on their slides, and will work in clever plot allusions. And if a group starts piping "The Rains of Castamere" through the room's speakers, beware!

At the second and third class sessions, each group gives an informal presentation, on slides, of its progress to date, and the other groups, the TA, and I ask questions to the presenting group. This is meant to simulate real-world computer-science workshops, where researchers studying a subarea or problem meet to share progress and ideas.

The fourth class session models a computer-science conference session, complete with a session chair, strict time-keeping warnings, and so on. Each of the four groups on that day turns in a formal research paper and gives a polished, formal 18-minute (as the class time slot is 75 minutes) PowerPoint or Beamer conference talk on their paper/project.

Thus each project provides a time-compressed microcosm of what professors do in a research project.

The Time Needed Between Sessions. The spacing of the sessions is important to get right. Typically, the (three) gaps between sessions 1, 2, 3, and 4 will be at least one week each, and sometimes (especially between sessions 3 and 4) one and a half or two weeks. That gives the students enough time to actually perform their investigations and make progress. Any less time risks demoralizing them. Indeed, even at 1–2 weeks between meetings, that still is a brisk pace. Not many CS professors would want to be expected to make progress and have a new slide show on their progress every week or two!

Instilling Self-confidence in the Students. Students are often at first surprised that the course expects them to tackle open issues—that they are not sitting in a room and being lectured to and taking a midterm and a final on the content of the lectures. But they truly rise to the challenge of researching hard, open issues. It is sort of a *Field of Dreams* thing: If one lets students know that one expects hard work and creativity from them, and that one truly believes they have the ability to have insights even on hard problems, they remarkably often come through brilliantly. And I do believe in students because for more than 20 years I've seen the students truly grow and flourish, even within a single term. It is an honor and treat to be a part of that process.

But how does one convey that belief and confidence in them *to* the students, and let them know that they should share it? How does one convince them that such purported confidence is not just empty words?

A key way to let the students know that they can do far more than they might at first realize is that I explain to them at the start of the course that in traditional courses the professor tends to be an expert on the subject and has taught it many times before, but for the students the material of traditional classes is all new. So in traditional classes the playing field is extremely unlevel, and thus it is hardly surprising that students may feel intimidated by the apparently effortless command of the subject that the professor (sometimes) demonstrates. I point out to them that when the professor was sitting in classrooms 5 or 10 or 30 years ago, the then-student now-professor may well have felt just as intimidated by the command shown by the then-professor as the students themselves do now in their traditional classes. I then point out to them that in this undergraduate problem-solving course, the problems they are looking at are new and open—to them, but equally so to the TA and to me, and to the best of my knowledge to the entire world. The students, the TA, and I are partners in investigating these new questions—we are all peers in the investigation.

Relatedly, I let them know that, every year that I teach the course, I am impressed and humbled by the speed, energy, and insights of students in the class. And lest they think this implausible, I mention, early on, the outright glowing superiority of students in displaying outputs graphically and in writing simulations and experiments; the students know very well that they do these things far better than the typical faculty member, and so this rings true.

And continuing that, I often describe to them some of the most interesting projects done in previous years; and how the direction of one of those projects was so exciting that a few years later RPI was trying to capture the rights to other attacks on the problem; and how each of the class's groups brought wonderful approaches to the problem; and how in most years many students in the class put papers into the arXiv.org Computing Research Repository, and often in their papers refute previous papers in the archive that made overbold claims; and how some students have even produced field-advancing refereed journal publications from the course. In brief, I make it clear to the students that they can very reasonably believe that they can do exciting, valuable research in the course, because in previous years the students who started the course with precisely the same set of worries and self-doubts did do wonderful things.

And so the students do feel confidence, and they do excel. I delight in sharing with my colleagues what clever approaches and results the students are discovering in the course.

Of course, to help this all happen, it is important to choose the projects to be—though open and hard—ones where a variety of skills can be employed. That way, students with strengths as varied as CS theory, mathematical modeling, programming, heuristics, simulations, and visualization can plausibly each find their own way to contribute to their group's attack on the problem.

The Range of Topics. As a professor, there is a great temptation to focus the course all in one's own area, and there have been some offerings of the course that had that flavor. (I'm reminded of a professor at one of my earlier schools, who when assigned to teach the graduate complexity course spent the entire term teaching the professor's own quite different material, and then during the last week or two skimmed through all of complexity. The local joke then became—except the name has been changed here—that there was never any question what course Bill would teach... just what number it would have.)

Myself, when teaching this course I spread the project's topics over the department's three main focus areas. For example, in the first years I taught the course, we would have one project each in AI, systems, and theory. (In years when there are four groups in the class, and thus four class sessions per project, I usually have three projects plus a separate honors project—aka the crazy-literature critique—that does not itself take any class sessions.)

However, in my more recent offerings, that has changed a bit. I have the first project be math- or theory-inspired. And the second project is either AI- or systems-inspired. For the third project, I let each group choose its own project. Doing so lets the group focus on something the group itself is very excited about exploring. It also forces the group's members to negotiate among themselves, comparing skills and interests to find a project that works for the group as a whole.

For that third, self-chosen project, there are two phases. In the first phase, the students in the group formulate a project proposal, and present it to the class as a talk on slides, and also hand in the proposal on paper (and it is graded as a certain portion of that project's grade, e.g., 20% of the project's grade is due to Phase 1 and 80% is due to Phase 2). The class, the TA, and I then give immediate feedback. Sometimes a listener will point out that the proposed work already exists in the literature; or that it is already in progress at major research groups or companies that are putting in an enormous effort; or that it is just hopelessly ambitious to even get a start on within the time frame; or that the hardware required is beyond what the department can provide. After that, the group is allowed to if it wishes modify its approach, or its project, or even to completely change projects, in light of the feedback it received. (Since the Phase 1 presentations take a day or two, the self-chosen project involves five rather than four class sessions if there are four groups, and eight or nine sessions rather than seven if there are more than four groups.)

The topics the students choose are often utterly thrilling, ranging from natural language processing to video processing, to issues in systems, to algorithms challenges, to machine learning applications.

One group of students, David Klein, Kyle Murray, and Simon Weber, as their self-chosen project studied automated recognition of which programming language a code segment/fragment seen in isolation may be from. They wrote a lovely paper [15], and their work turned out also to be of interest to a group in industry, and that led to contact between that industry group and one of the students.

A Sample Twist. Sometimes, I throw in an unexpected twist. For example, a teaching newsletter to which I subscribe, *The Teaching Professor*, had an article suggesting that in the middle of group projects one should shift some students between groups to model the way technical workers move between companies in the real world. This seemed an excellent idea. However, at the time many tech companies were being *merged* in the real world, so I tweaked the idea to make it fit even better with the current news. In the course, I announced mid-project that Gryffindor and Hufflepuff had been merged and that Slytherin had bought Ravenclaw. There were both technical and cultural clashes as the groups integrated, but in the end, excellent projects emerged—and some lessons about combining approaches.

2.3.2 Project Example: 3-Dimensional Randomized Solitaire Bingo

This particular project was from a year in which the course was so large—31 students—that we had not four but eight groups on each project. This meant that the project involved not four but seven class days, since each of the two "workshops" and the "conference" needed two days, with four groups' talks each day. And so in that term, the course had just two projects during the term, plus the honors project (aka the crazy-literature critique) for those taking the honors version of the course.

And here is the project statement that was handed out on the day I introduced the project to the class, slightly edited and condensed.

Project 1: 3-Dimensional Randomized Solitaire Bingo

2014/1/29

CSC 200/200H: Undergraduate Problem Seminar Spring 2014
Instructor: Lane A. Hemaspaandra TA: Joe Izraelevitz

Due: March 3, 2014, 1:59PM (*even if your group will be one of the ones presenting on 3/5 rather than 3/3*—please note that carefully so you don't get a zero after having done lots of work!!!); also, fyi, the first intermediate workshops on this will be 2/10 and 2/12, and the second intermediate workshops on this will be 2/24 and 2/26.

The University of Rochester's (UR's) President Seligman has a problem. He has been challenged by the Rochester Institute of Technology's (RIT's) President Destler, to provide an evaluation of a certain "game" (called RITtaire3D, and which we will describe in detail below). Being no fool, President Seligman, after inspecting the description President Destler provided him of the game, suspects that in fact the RIT president has in mind some actual product in which a randomly chosen nonfailed "sector" of that product fails at each time step, and when a complete "line" of sectors has (over time) failed the product goes up in flames. (This seems a very bad feature of a product, but not all products are perfect. And apparently President Destler is trying to get a handle on how long one can expect to run the product before it self-immolates.) Nonetheless, filled with pride at the brightness of UR's students, President Seligman has agreed to undertake the analysis and has, of course, delegated it to his most powerful secret resource—the students of CSC200/200H.

Your task (each group's task, that is) is to write (following the rules of the course) a research paper exploring this game and in particular the issues mentioned below. This paper should be written in LaTeX. As mentioned above, we'll call the game RITtaire3D (which stands for 3-dimensional RITtaire).

The game RITtaire3D is played on an N by N by N board (N can be 1, 2, 3, etc.). To be clear and build the N into the name of the game, let us during this project when being careful write RITtaire3D$_N$ when speaking of the N by N by N version. The player during the first step chooses a random cell (each with equal probability, e.g., the player flips a fair N^3-sided die) and puts a marker on it. During the second step, the player chooses a random cell from among all cells that do not yet have a marker on them (each chosen with equal probability, e.g., the player flips a fair ($N^3 - 1$)-sided die), and puts a marker on it. And so on. The game ends when the player has N markers in a row—this can be in the height, width, or depth axes, or can be along any of diagonals that exist if one coordinate of one of the dimensions is fixed, or can even be one of the 4 such (well, 1 such if $N = 1$ as they degenerate there) diagonals that go between diagonally opposed corners of the overall cube-shaped board.

What we are interested in is how long it takes to win. So, in this project, you should study that issue. Note that there are many questions to potentially explore: What is the quickest possible win time? What is the longest possible win time? And, most centrally, what can you prove about the expected (average) win time?

It would be great if you can state and prove crisp theorems. Perhaps you'll completely resolve some issues. If not, perhaps you'll prove upper and lower bounds, and will work hard to make them as strong as possible. Or if on an issue you are failing to get any traction, maybe you'll program up a simulation, and see what insights that gives. Also, note that these issues can be studied in various ways. One issue is what values hold for a particular, fixed N, e.g., $N = 1$ (pretty easy!), or $N = 2007$, or $N = 2$, or etc. Very nice would be to get a precise formula that holds over all N, e.g., "For each $N \in \{1, 2, 3, \dots\}$, the quickest win time in RITtaire3D$_N$ is N" (which by the way happens to be a true theorem—so you see, "theorem" isn't as intimidating a word as one might think; go prove some!). But if you can't get a precise formula, you might want to deal with the *asymptotics* (e.g., "ExpectedWinTimeInRITtaire3D$_N$ $= \Theta(BLORT)$," where for BLORT will be some expression in terms of N that you have brilliantly proven to hold in your paper, or perhaps you'll not get a Θ result but will provide $\mathcal{O}$ and Ω results), or perhaps you will find/prove upper and lower bounds on particular Ns. Basically, poke around, do your best to converge to strong results by proving at first what you can, and then strengthening more and more and more. We'll discuss the flavor of this in class a bit on the starting date, and of course, as always, we'll before the presentation day have two intermediate day-pairs (February 10 and 12, and February 24 and 26) during which you share, present, defend, discuss, etc. your ideas/progress/etc. (so half the groups will speak on 2/10 and the other half will speak on 2/12, and similarly regarding 2/24 and 2/26).

Good luck, and do well. President Seligman has put the honor of UR in your hands!

(And a bit of help. Most research projects build on the shoulders of those who have come before. In this case, it turns out that some wonderful researchers—the students of the Spring 2007 CSC200/CSC200H class—have looked at the *2-dimensional* analog of the question we're studying. Although trying to get answers from those people is cheating, you of course may well want to look carefully at their papers on this, and any other related work you can find in the literature. I'm making one or two of their papers available, having previously checked with them that it is ok to share them to help future students, via our web page (they will appear there late this evening or early tomorrow). Of course, and this is *very* important, when your papers draw on ideas/results/etc. from an existing paper, whether theirs or any other, you are ethically obligated to clearly and openly cite the paper you are drawing on and to attribute to it whatever you are using from it. In the case of earlier CSC200 papers (except those that are published through a journal or through arXiv.org, as those should be simply cited in those forms), the right citation is to, as your bibliography entry, list the authors, title, and date of the paper, and call it "unpublished manuscript.")

2.3.3 Project Example: Restricted Domain Dialogue Agents: Pawlicki-in-a-Box

Here is a second project example. It is from 2008, and is of an AI flavor, and indeed a rather standard-ish one aside from it being quite hard to actually do as the role of an Undergraduate Program Director and the questions the system might need to respond to are not really as limited as the project's title might make them seem. Briefly, the students are asked to write a program for an agent that fulfills the advising role of the department's Undergraduate Program Director (Ted Pawlicki, who despite the lighthearted accusations of embezzlement in the assignment, is a treasure of a colleague and would never take as much as a poker chip from anyone... except perhaps at a poker table).

This project's flow was a bit different from the standard flow. As usual, there was a day for introducing the project, and then a "workshop" day of talks, and then another "workshop" day of talks. But a week after the second workshop, instead of moving right to the day of "conference" talks, we had a day where the students could not only demonstrate the software systems they had built, but could put to the test the systems the other groups had built. And in the next class after that, we had the traditional end-of-project "conference"-talk day.

Here is the project statement that was handed out on the day I introduced the project to the class, except slightly edited for clarity and space.

Project 3: Restricted Domain Dialogue Agents: Pawlicki-in-a-Box

2008/3/26 Version 1.1
CSC 200/200H: Undergraduate Problem Seminar Spring 2008
Instructor: Lane A. Hemaspaandra TA: Stan Park

Due Date Information. Due April 21, 2008, 4:49PM (that is the due date/time for the paper and for the archive including program/README/makefiles/sample runs/etc.), and (this is the due date/time for your talk slides) April 23, 2008, 4:49PM. Note very carefully the slight departure from our normal process. As usual your group's paper and program must be sent to us by 4/21/4:49pm—see the class info document for full details on how to send things in. And each group's program will be demonstrated/tested by each other group and then the writing group itself during that 4/21 class session. And for the 4/23 class session each group will in that class session give a "conference" presentation on their paper.

The two intermediate discussion dates will be April 7th and April 14th.

Project. UR's President Seligman has a problem. Again! He has proposed expansions that total the better part of a billion dollars. And he had careful financial plans as to how to achieve this. But a new audit shows that things will fall far short, money-wise.

After some energetic checking, the President has found out the key cause of the predicted shortfall. Shortly after a CS faculty member was used to help improve the payroll software (there is no record of which one helped, although the four comment lines among the 150,000 lines of new programming are each signed "TFP," surely some deep code and one that the school's best minds have so far been unable to crack), CS faculty member Thaddeus ("Ted") F. Pawlicki's salary suddenly jumped by a multiplicative factor of well over one thousand. By a complete coincidence, Ted has just announced his departure for a recently endowed chair at the very prestigious University of We-Don't-Have-an-Extradition-Treaty-with-the-USA. Unfortunately, due to remaining deficiencies in the payroll software, if the President refills Ted's position, the new hire must start at no less than Ted's most recent salary.

So the President has reluctantly decided to leave the position open. Through shifting the teaching of some of the department's less rigorous courses to the Interpretive Dance Program and some course-assignment shuffling regarding the remaining courses, the President has managed to handle the course-coverage loss. However, he is deeply worried about the loss of Ted's famed advising skills. So who does he call? Yes, the students of CSC200, who so very recently rose so well to the task of studying the counting of kings in graphs. In particular, President Seligman asks you to write a "Pawlicki-in-a-Box" program. This program must handle the advising of all freshman/freshwoman and sophomore CS premajors. Its goal is to provide a strong level of advice, support, and insight to such students, by fulfilling the conversational role that Ted fulfilled when students walked into his office.

Here is the framework. You do not have to tackle speech recognition or speech synthesis at all. We'll use a totally text-oriented interface. Also, you may assume that this version of Ted doesn't have access to students' advising records (although if you want, your programs can keep files that stay around between runs of the program, and thus allow it to become better and better as it gets more and more experience), and so during the conversation with "Ted," "Ted" will have to get from the student (whom you may assume to be a freshman/freshwoman or sophomore) who has come to see him whatever information "Ted" needs to identify what help the student wants and to provide advice to the student on that issue. In particular, when the program starts up, we'll assume that a student has just walked into "Ted"'s office. Your program should then (always) greet the visitor by printing out the formulaic two lines:

```
> Hello.  What can I do for you today?
> (over)
```

(Note: Every line "Ted" outputs must start with the > character and then a space. This will make it easier to visually scan the conversation logs.) More generally, both "Ted" and the student will signal that they are done with what they are currently saying by having a line of input that simply says "(over)" in the student's case and "> (over)" in "Ted"'s case (and the student is allowed to choose to interchangeably use "(over)" and "(o)" and "."—the student might often choose to type "." as it is faster and easier). Note that the student in reply might not give "Ted" all the information "Ted" needs, and might not even pose any coherent issue, and so "Ted" may have to enter into a dialogue with the student. As to what that dialogue might look like and how long it might be, well, as students, you'll have a general idea of that. (Among the types of issues students might naturally come to "Ted" for are issues of what courses to take, prerequisites, getting ready to declare the major, summer jobs, courses, and much more.) And in the 4/21 class you'll have a chance to show off your program's strengths—and to have your program put energetically to the test by other groups.

There will be a second set of special phrases that when on a line by themselves by convention will indicate that the student is leaving "Ted"'s office; "(over and out)" and "(oo)" and "bye" may be used interchangeably for that. "Ted" will never end a conversation on its own; only the student can invoke those session-ending key-phrases. (All of these key-phrases are case-sensitive, by the way. "(OO)" on a line by itself is not a valid sign-off but rather is just a strange parenthetical exclamation to "Ted.")

Of course, writing a perfect such program is (currently) pretty much impossible. In fact, even doing it nondisastrously is a wildly daunting task that is probably also pretty near impossible and would fill any senior AI researcher with terror. (So be very thankful that you are not senior AI researchers; no one likes being filled with terror. In fact, naïveté can be an asset: Róbert Szelepcsényi, as an undergraduate, resolved—in the affirmative—the huge open issue of whether the context-sensitive languages are closed under complementation probably largely because he didn't know enough to know that the problem was hopelessly hard... so he just went right ahead and solved it... correctly!) However, you'll want to do your best to have your program work as well as possible.

Good luck, and do well. President Seligman thanks you in advance for saving his big-chunk-of-a-billion dollar plan (and the rumor that he is kicking back a certain percentage of the savings to your course's professor is, most unfortunately, quite untrue).

2.4 Component: Crazy-Literature Critiques

In years when the course has both some students taking it as a nonhonors course and some (in the same room at the same times) taking it as an honors course, I give an extra assignment to the honors students. In particular, I break them into groups, and each group is asked to write a critique of a seriously-intended recent paper, usually from arXiv.org, that claims to resolve a major open issue in computer science, such as P versus NP.

Claiming that the students will succeed in that, and within just a few weeks, may sound like magical thinking. However, I have so far had twenty-four undergraduate students, in the course, attempt to meet the challenge of refuting a paper published at arXiv.org, and none have failed to meet the challenge; those refutations have all appeared on arXiv.org [5–7,9,11,12,20,22], and most are now cited on the field's central resource for tracking the status of attacks on the P versus NP problem, namely, the web page maintained by Prof. Gerhard Woeginger [27].

That is why I titled this component "crazy-literature critiques." The critiqued papers are, as I said, seriously intended. But often they are, well, a bit wild and hard to follow. (As students quickly learn, a crisply, clearly written, incorrect proof is a piece of cake to critique relative to critiquing a confused, unclear, handwaving paper.) And so far, none have been correct.

The students and the literature both benefit from these critiques. The critiques do the literature the service of validating (never so far) or shattering (always so far) claimed resolutions of the field's major problems. The students, in finding what the flaws are in a paper—typically by finding bright-line counterexamples to shatter algorithms claiming to show $\mathrm{P} = \mathrm{NP}$ and by finding incorrect, tacit assumptions that are used in proofs claiming to show $\mathrm{P} \neq \mathrm{NP}$—increase their own critical reading skills, and so when checking their own future papers they will be more demanding critics and better proofreaders.

2.5 Component: Computer Science Gladiators Day and Lightning Research Days

2.5.1 Computer Science Gladiators Day

I often have a Computer Science Gladiators Day (named after the old, and briefly revived, "American Gladiators" television show). On that day, I invite in two of our CS Ph.D. students who as a team compete with the entire class, which itself works as a team of the whole. I assign a challenging problem, usually in the subarea in which the invited graduate students are experts, that has never been seen before by the graduate students or the class members.

The graduate-student team and the team of undergraduates (which is the entire class) spend about two thirds of the class separately tackling the problem. Then the graduate students present their solution and the undergraduates present theirs.

The undergraduates know ahead of time that this counts as a quiz, and I guarantee that if their solution is at least as good as that of the graduate students, then they all get 100%, and if their solution is weaker than that of the graduate students, they will get a grade based on my judgment of the quality of their solution considered in light of the difficulty of the problem.

The way this most typically ends is that the undergraduates as a team do as well as or better than the graduate-student team, and they can (having heard both solutions) clearly themselves see that they did do that well. The Computer Science Gladiators Day is not just an adventure for the students, but also supports the theme mentioned earlier of instilling self-confidence in the students:

The day lets the undergraduate students know in the clearest possible way that they have tremendous talent—talent so strong that they can, when they work cooperatively, fully hold their own against a team of Ph.D. students who are working on a problem in their own area of specialization.

2.5.2 Lightning Research Days

Each term typically has just one Computer Science Gladiators Day, but has several Lightning Research Days. These are quite similar to Gladiators Day, except there are no graduate students. Instead, at the start of the class I present a challenge (usually one with an answer that is known to me but not known to the class, and which is a problem that I think the class will never have seen before), and the students study it for most of the class in their groups, and then either I have each group present its solution, or I have the groups then meet with each other and choose a "best" of the solutions as the one the class as a whole will be going with and will present to me during the final portion of that class session (and that is what the class is graded on if this is being counted as a quiz). (This is a simplified version of a wonderful approach Prof. Edith Hemaspaandra uses in her courses, and that I have incorporated in some of mine. She starts with students working on a problem in groups of size 1, and then following some timetable and pattern, merges them into bigger and bigger groups as the class session proceeds, with the group-size sequence often being the powers of two.)

2.5.3 Computer Science Gladiators Day/Lightning Research Days: Example Problems

Here are a few examples of some of my favorite problems to use on these one-class problem-solving challenges.

Apportionment Algorithms. Motivated by the fact that the US House of Representatives is reapportioned every decade based on census data, I pose to groups the problem of developing an apportionment algorithm. For the apportionment problem, the input is a set of states, the population p_i of each, and the "house" size H (the total size of the legislative body that is being elected), which currently is 435. And the students must assign to each state a natural number (one may not use fractional values) that will be that state's number of representatives, and the sum of those numbers must be exactly the house size H.

I mention to them, to keep things clean, that it—unlike the actual situation in the USA—is legal for their algorithms to assign 0 seats to some states. And I warn them not to overly focus on "tie-breaking" issues, e.g., if there are exactly two states, and they are of the same size, and the house size is odd.

I also mention that the proportional quota of each state j, namely

$$q_j = H\left(\frac{p_j}{\sum_i p_i}\right),$$

may be of interest to them.

I urge them, as they design their algorithms, to think about what properties their algorithms "should" be, or are, satisfying, and what it even may mean for an apportionment algorithm to be "fair."

I let them know who their historical competition is, namely, many of the existing algorithms were designed by the greatest figures in the USA's history. Among the classic algorithms for this problem are Jefferson's Method (the third president), Adams's Method (John Quincy Adams, the sixth president), Hamilton's Method (the first Secretary of the Treasury), and Webster's Method (yes, the dictionary guy, and US senator). The first presidential veto ever cast, namely by George Washington himself, was over the issue of apportionment methods.

The students are thrilled and amazed to hear that the founders of the nation designed and debated the performance of algorithms... and that the students will be trying their own hand at that same challenge. Since they know that the US President is chosen through the Electoral College, and that each state's number of votes in that is the sum of its number of senators (always 2) and its number of representatives in the House, the students realize that this algorithmic challenge is tied not just to the House's apportionment but also to how much influence states have in choosing the president.

The students very often rediscover one of the classic methods, almost always that of Hamilton. Hamilton's Method (we'll here ignore two special rules that apply in the US House, such as having to give each state at least one representative) gives each state, right up front, the floor of its proportional quota, q_i. And then it adds up all the chopped off fractional parts, which of course adds up to some whole number, say K, and it sorts the states by how large their floored-away fractional parts were, and gives the K states that lost the biggest fractional parts each one extra representative. This method, as the students almost always note and convey, "respects quota," i.e., the number of representatives a state is given is always strictly less than one away from its quota. Hamilton's Method in fact was used for a period of time in the USA as the apportionment algorithm for the House of Representatives.

I then explain to the students the other classic methods—all of them are so-called sliding-divisor methods, variously based on floors, ceilings, and arithmetic/geometric/harmonic means. (The USA currently and for many years has used a geometric-mean-based sliding-divisor method, known as the Huntington–Hill Method, to apportion the House, after it was supported in the first third of the 1900s by two blue-ribbon panels of mathematicians [17].) I point out to the students that none of these other methods always respect quota. Yet I explain why Hamilton's Method may not be as wonderful as it at first glance seems: We see that cases exist where if all the state sizes stay the same, and we increase the house size by one, some state can *lose* a seat. This is known in the literature as the Alabama Paradox.

So, within a class, the students have learned that some of the nation's founders were algorithm designers, have tested themselves against those founders and found themselves their equal, and have learned a bit of rather cool history. In this section, I have including almost no literature citations. Let me instead mention, as I do to the students, that there is an utterly wonderful, compellingly readable book on everything mentioned above, namely, the book *Fair Representation: Meeting the Ideal of One Man, One Vote*, by Balinski and Young [4]. The book weaves mathematics, algorithms, and history into a riveting tale, and one that remains important to this day.

Before ending the class, I turn this all a bit on its head, by introducing the notion of power indices, and giving examples showing that, due to that notion, the connection between number of representatives and power is far more subtle than one might think, and thus that trying to match votes closely to proportional quotas may not be the right goal to have (see [13] and the references therein for more background and empirical studies in this direction).

SAT Is in Probabilistic Polynomial Time. Even students who have not yet taken a Hromkovič- [14] or Sipser-type [25] course can typically enjoy this problem. For this challenge, I define what the famous NP-complete satisfiability problem, SAT, is. And I informally define what the complexity class PP is [10,24], namely, the class of (language) problems that have a polynomial-time algorithm (that is allowed unit-cost access to a fair coin) that on each input is right with probability strictly greater than 50%.

The students' challenge is to, during the class session, prove that SAT $\in$ PP, i.e., that there is a probabilistic polynomial-time algorithm for SAT. This is a lovely result of Gill from the 1970s [10] (that has since been much strengthened).

Students are a bit surprised to hear that SAT—and even if they have not seen its definition before they usually know that things that are "NP-complete" are thought hard—is in polynomial time if one gives one's computer access to a coin. But more often than not, they succeed in proving this themselves. We typically end the class session by discussing why this result cannot be used, through repeated sampling and then taking the majority, to get a killer heuristic algorithm for SAT; in particular, they note that in their algorithm, though its success probability is always strictly more than 1/2, the probability can decay toward 1/2 from above exponentially quickly, and so the type of heuristic just mentioned would have to sample so very many times that one might as well solve the problem exactly in deterministic exponential time by a brute-force approach.

One Hundred Prisoners and a Lightbulb. This is an absolutely delightful problem, in that at first it seems outright impossible that a solution exists at all. And it takes an "Aha!" moment for the groups to realize that it can indeed be solved. I won't describe the problem here, but rather will simply give the reference for the work on this by Dehaye, Ford, and Segerman [8], which can currently be accessed easily as a pdf from the web page of its final author. In most years, the students find the algorithm/approach that those authors call "Strategy 2," and the students prove that the algorithm's expected time is quadratic in the number of prisoners.

Since this problem has the flavor of a distributed systems protocol, I often use it for the Computer Science Gladiators Day, inviting systems Ph.D. students in to be the graduate team that challenges the class.

2.6 Component: One-on-One Meetings

During the first half of the term, I schedule one-on-one meetings with each student in the course. The meetings check on how the student is doing, check on whether the groups are functioning well as to sharing the work load and communicating with each other, and answer any questions the student may have. With graduating seniors, I ask about how things are looking as to jobs or graduate-school admissions. With students earlier than that in their time here, I (a) ask whether they are interested in going to graduate school in computer science, and if so, give them my best advice as to how to go about that, and if they are theory people give advice about schools, and if they focused on another area, I make sure they know what faculty member to consult with for expert advice on how schools are in that area; (b) discuss their plans for summer jobs, and (c) discuss their path to completing their CS major.

My guess is that most students think that these meetings are simply a chance to chat with a professor one-on-one about the course and about their career plans. But there is a more important purpose to the meetings. Earlier on, I mentioned the importance of instilling self-confidence in the students, and described some approaches. These meetings are another approach to supporting that. After all, students differ, and addressing the differing worries and talents of each student can't be done just in broadcast mode to the entire class. In contrast, these one-on-one meetings with each student in the class create a human connection between me and the student. The student can in a one-on-one setting get a far better reading of whether his or her impressions of the professor are for real, of what the professor really thinks of the student, and of whether the professor sincerely believes in that student's potential.

2.7 Component: Quizzes

The Gladiator and Lightning Research Days, as described above, often count as quizzes. As to project-related quizzes, most students would work hard even without quizzes. But to help inspire the few others to stay on top of the projects' material so they can pull their weight in their groups, classes sometimes start with 2–15-minute quizzes, sometimes easy ones and sometimes a bit more challenging.

For example, at the very start of class during the first intermediate workshop on the "3-Dimensional Randomized Solitaire Bingo" project above, I give the following two-question quiz.

Question 1 [50 points]. Consider the 2-dimensional version of RITtaire$_N$. For the $N = 4$ case, give a board with 12 filled squares such that there still is no N-in-a-row (of filled squares) on the board.

Question 2 [50 points]. Now, argue that for the 2-dimensional version of RITtaire$_N$, for the $N = 4$ case, every way of filling 13 squares does result in at least one N-in-a-row of filled squares. (Hint: Think of how many squares remain unfilled. Can you argue something from that?) (Note: From this question and the one above, you have now shown exactly how long the game can go for $N = 4$ in the longest case.)

The quiz is basically helping the students realize what the longest possible length is of a 2-dimensional solitaire bingo game (and in doing so, the question is quietly suggesting, as did the project statement itself, that their papers surely should look at the same issue in 3 dimensions—and almost every group did, not finding a tight bound but in most cases finding relatively close upper and lower bounds for the 3-dimensional case). In the 2-dimensional case that the above quiz was about—i.e., an $N \times N$ board—except for $N = 2$, one can always fill $N^2 - N$ squares without having N-in-a-line on any row, column, or major diagonal. That is optimal: Every way of filling $N^2 - N + 1$ squares yields a bingo (i.e., N filled squares in some row, column, or major diagonal), since with one more square filled, the average number of filled squares per row is strictly greater than $N - 1$, and so at least one row must have N filled squares in it. For $N = 2$, the most filled squares one can have without a bingo is not $2^2 - 2$, but rather is 1. The reason $N = 2$ is pathological has to do with the example that realizes $N^2 - 2$, which is simply to fill every square on the board except to leave empty the entire top-left to bottom-right major diagonal, and then if N is an even number adjust that board by rotating the 4 center-of-the-board cells by 90 degrees. That rotation is to block the other major diagonal, and works fine for all even N except 2, where the rotation would reopen the originally blocked diagonal. Pictorially, here is an 8×8 board with $8^8 - 8$ filled squares yet with no bingo existing:

	●	●	●	●	●	●	●
●		●	●	●	●	●	●
●	●		●	●	●	●	●
●	●	●	●		●	●	●
●	●	●		●	●	●	●
●	●	●	●	●		●	●
●	●	●	●	●	●		●
●	●	●	●	●	●	●	

2.8 Component: Guest Lectures and Readings

The guest lectures and the readings are listed in the same component since they share the same goal: showing students how widely varied are the ways that actual researchers approach research.

As mentioned above, there are usually one or more nonproject days between the project days. Those "open" days are used for the Gladiator and Lightning Research Days, and for guest lectures. Typically, I'll ask one to three people from each of the department's three main areas to give guest talks on their research. When there is time, I'll invite a faculty member from a related department (such as the Department of Electrical and Computer Engineering or the Department of Brain and Cognitive Sciences) to speak, or I'll invite a graduate student to speak. Once or twice, I've invited a very research-successful undergraduate student to speak.

In years when I want to (or due to class-time limitations need to) supplement or partially replace that with readings, I have the students read articles by or about the greatest research minds in computer science's history. My two favorite sources for such readings are the Turing Award Lectures (the early ones are even collected as a book [1]) and the wonderful book *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists* [23].

My goal, in having those talks and assigning those readings, is not to show the students that there is one way to do research. It is to show them that successful researchers fit no mold (aside from usually being passionate about their research). Even within a given area of the field, researchers have different views, approaches, and work habits. My hope is that some of those will resonate with the students, and that each student can enrich his or her own approach to research with those approaches that he or she hears or reads that speak to him or her. Of course, the guest talks also are about introducing the various areas of CS to the students; but to me that isn't the talks' most important function.

My very favorite reading from the above is without any doubt Ken Thompson's three-page Turing Award Lecture, "Reflections on Trusting Trust" [26], which (among other things) basically shows how to push malicious code into such a low level that recompiling programs and or even recompiling the compiler itself doesn't protect one, as the executable that is running when one calls the compiler has itself been compromised in a subtle and self-propagating way.

My favorite reading not of the above two types—Turing Award lectures and chapters from the book *Out of Their Minds*—is the breathtaking article "Cheating Husbands and Other Stories: A Case Study of Knowledge, Action and Communication," by Moses, Dolev, and Halpern [19]. Like the Thompson paper, this article is so clever, brilliant, and surprising that after one gets past feeling awed and inadequate, one squares one's shoulders and tries one's best to do work that displays at least some hint of the sparkling beauty that one finds revealed in those papers.

3 Conclusion

This article has sketched the design of a Knuth-style undergraduate research-immersion course, and provided some examples of projects and problems. The hope is that this article interests some faculty members elsewhere in creating such a course at their own schools, and if so that this article will help them in bringing such a course to life.

The course, far more flexibly and broadly than most course frameworks, has the potential of creating a class that for many faculty members will, after they resculpt it to best match their own style, be located in that most important intersection—the intersection of what they love doing and what they can do exceptionally well. And in this course, there typically is a reinforcement cycle between the professor loving and passionately leading the course, and the students loving it and excelling in it. Sometimes, things all simply come together.

Acknowledgments. My thanks to Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, who kindly invited me to participate in this project honoring the inspirational Juraj Hromkovič; to Chris Brown, who first taught my department's undergraduate problem seminar and generously shared with me his excellent materials; to Edith Hemaspaandra and an anonymous reviewer for many helpful suggestions and corrections; to my former department chair, Henry Kautz, who a few years ago suggested I write an article on my version of our undergraduate problem seminar; and to Donald Knuth, whose Autumn 1981 Programming and Problem-Solving Seminar [16] I had the treat of taking, and whose teaching of the course was a model of being in "that most important intersection."

References

1. ACM Turing Award Lectures: The First Twenty Years. ACM Press Anthology Series, Addison-Wesley, Boston (1987)
2. Alexander, L.: The Chronicles of Prydain (The Book of Three, The Black Cauldron, The Castle of Llyr, Taran Wanderer, The High King). Holt, Rinehart, and Winston (1964–1968)
3. Alexander, L.: Taran Wanderer. Holt, Rinehart, and Winston (1967)
4. Balinski, M., Young, H.: Fair Representation: Meeting the Ideal of One Man, One Vote. Yale University Press, New Haven (1982)
5. Cardenas, H., Holtz, C., Janczak, M., Meyers, P., Potrepka, N.: A refutation of the clique-based P = NP proofs of LaPlante and Tamta-Pande-Dhami. Technical report arXiv:1504.06890 [cs.CC]. Computing Research Repository, April 2015
6. Christopher, I., Huo, D., Jacobs, B.: A critique of a polynomial-time SAT solver devised by Sergey Gubin. Technical report arXiv:0804.2699 [cs.CC]. Computing Research Repository, April 2008
7. Clingerman, C., Hemphill, J., Proscia, C.: Analysis and counterexamples regarding Yatsenko's polynomial-time algorithm for solving the traveling salesman problem. Technical report arXiv:0801.0474 [cs.CC]. Computing Research Repository, January 2008
8. Dehaye, P., Ford, D., Segerman, H.: One hundred prisoners and a lightbulb. Math. Intell. **24**(4), 53–61 (2003)
9. Ferraro, F., Hall, G., Wood, A.: Refutation of Aslam's proof that NP = P. Technical report arXiv:0904.3912 [cs.CC]. Computing Research Repository, April 2009
10. Gill, J.: Computational complexity of probabilistic Turing machines. SIAM J. Comput. **6**(4), 675–695 (1977)
11. Hassin, D., Scrivener, A., Zhou, Y.: Critique of J. Kim's, "P is not equal to NP by modus tollens". Technical report arXiv:1404.5352 [cs.CC]. Computing Research Repository, April 2014
12. Hemaspaandra, L., Murray, K., Tang, X.: Barbosa, uniform polynomial time bounds, and promises. Technical report arXiv:1106.1150 [cs.CC]. Computing Research Repository, June 2011

13. Hemaspaandra, L., Rajasethupathy, K., Sethupathy, P., Zimand, M.: Power balance and apportionment algorithms for the United States Congress. ACM J. Exp. Algorithmics **3**(1), 16 (1998). http://doi.acm.org/10.1145/297096.297106
14. Hromkovič, J.: Theoretical Computer Science: Introduction to Automata, Computability, Complexity, Algorithmics, Randomization, Communication, and Cryptography. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11120-4
15. Klein, D., Murray, K., Weber, S.: Algorithmic programming language identification. Technical report arXiv:1106.4064 [cs.LG]. Computing Research Repository, June 2011
16. Knuth, D., Weening, J.: A programming and problem-solving seminar. Technical report STAN-CS-83-989 (aka STAN-B-83-989). Department of Computer Science, Stanford University, Stanford, December 1983. http://i.stanford.edu/pub/cstr/reports/cs/tr/83/989/CS-TR-83-989.pdf
17. Malkevitch, J.: Apportionment I. FEATURE COLUMN: Monthly Essays on Mathematical Topics (an AMS Web Column), May 2002. http://www.ams.org/publicoutreach/feature-column/fcarc-apportion1. Note: The part giving the history of apportionment in the USA, and mentioning the two blue-ribbon panels of mathematicians, is Section 2. http://www.ams.org/publicoutreach/feature-column/fcarc-apportion2
18. Martin, G.: A Game of Thrones. Bantam Books, New York City (1996)
19. Moses, Y., Dolev, D., Halpern, J.: Cheating husbands and other stories: a case study of knowledge, action and communication. Distrib. Comput. **1**, 167–176 (1986)
20. Richardson, A., Brown, C.: A critique of solving the P/NP problem under intrinsic uncertainty. Technical report arXiv:0904.3927 [cs.CC]. Computing Research Repository, April 2009
21. Rowling, J.: Harry Potter and the Philosopher's Stone. Bloomsbury Publishing, London (1997). American edition published under the title Harry Potter and the Sorcerer's Stone
22. Sabo, K., Schmitt, R., Silverman, M.: Critique of Feinstein's proof that P is not equal to NP. Technical report arXiv:0706.2035. Computing Research Repository, June 2007
23. Shasha, D., Lazere, C. (eds.): Out of their Minds: The Lives and Discoveries of 15 Great Computer Scientists. Springer, New York (1995)
24. Simon, J.: On some central problems in computational complexity. Ph.D. thesis. Cornell University, Ithaca, January 1975. Available as Cornell Department of Computer Science Technical Report TR75-224
25. Sipser, M.: Introduction to the Theory of Computation, 3rd edn. Cengage Learning, Boston (2013)
26. Thompson, K.: Reflections on trusting trust. Commun. ACM **27**(8), 761–763 (1984)
27. Woeginger, G.: The P-versus-NP page. http://www.win.tue.nl/~gwoegi/P-versus-NP.htm

Paving the Way for Computer Science in German Schools

Ulrik Schroeder(✉), Nadine Bergner, and Thiemo Leonhardt

Learning Technologies, RWTH Aachen University,
Ahornstraße 55, 52074 Aachen, Germany
{schroeder,bergner,leonhardt}@informatik.rwth-aachen.de

Abstract. There is an ongoing discussion in Germany, whether and to what extent computer science should be integrated into school education as a mandatory subject on the level as natural sciences. The goal of this article is to present scientific evidence of effects of computing interventions of extra-curricular services offered by universities as supplement of the currently missing school integration.

Keywords: Extra-curricular computer science education
InfoSphere · go4IT!

1 Introduction

Worldwide, there is an ongoing controversial discussion with regard to the importance of Computer Science Education (CSE) and more particularly its lack of presence in pre-college education [1]. In this article, the authors present arguments, research results and projects that (a) taking up informatics as a mandatory school subject at least from lower secondary level on is recommendable, (b) there are very good concepts and examples to teach informatics, (c) from early ages on, and (d) especially girls and young women can profit.

Computer Science (CS) has both enormous educational benefits (logical thinking and problem solving skills, understanding a world enhanced with digital technology), and economic benefits (as a modern workforce depends on far reaching IT-skills). It should be part of every student's education, just as natural sciences are, especially, because utilizing the digital world becomes more and more important in all fields of life, not only in the IT sector itself.

It is crucial to distinguish between Computer Science as a rigorous subject discipline on the one hand, and IT applications and/or digital literacy on the other. Computer Science has its own body of knowledge, established methods and techniques, and thinking skills, which will last students a lifetime [1]. The core skill-set of Computer Science is independent of new technologies and programming languages. Programming is central to computing, but the underlying principles of algorithms, data structures, and computational thinking skills are both more fundamental and more durable [2]. On the other hand, ICT education

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 590–609, 2018.
https://doi.org/10.1007/978-3-319-98355-4_34

covers knowledgeable fluency with computer tools and the internet. Due to the prevalence of informatics systems in the all-day life of children, both subjects are essential and should be taught in primary and lower secondary education.

The Joint Informatics Europe & ACM Europe Working Group on informatics education presented strong arguments for teaching informatics as early as possible, preferably in primary schools. "[...] European nations are harming their primary and secondary school students, both educationally and economically, by failing to offer them an education in the fundamentals of informatics." [2, p. 17]. Although associations like the German Association for Informatics (GI) have argued to include informatics in general education since the late 1980s and published diverse recommendations for principles and standards for CSE [3–7], the German educational system falls back behind that of other nations. In view of the widespread digitalization, in many countries, CSE in primary or secondary schools has reached a turning point, shifting its focus from ICT-oriented to rigorous computer science concepts [8–13]. One of the existing challenges in these countries though is CS teacher (further) education. In Germany, there is a quite solid CS teacher education program [14], but the broad implementation of CS as mandatory subject in lower secondary schools is still missing and thus, too few students take up the subject.

As long as informatics is not a school subject comparable to the natural sciences, there are many universities offering extra-curricular activities teaching various aspects of informatics ranging from targeting pupils from primary to lower and upper secondary education. At ETH Zürich Hromkovič has established LOGO courses [15] for primary and lower secondary students and worked out teaching materials [16] based on Papert's original ideas [17] and focusing on the essential core of what computer science means for human culture [18]. In the following, we present our own comprehensive activities at RWTH Aachen University, namely go4IT! [19,20] and InfoSphere [21,22], and the underlying research [23] which lead to their conception and design.

2 go4IT! – How to Raise Girls' Interest for Computing

2.1 Motivation for go4IT!

A positive effect of an early access to informatics could lead to increasing the interest for technology, preparing children for computational thinking and fostering problem-solving strategies in general. Interest for technics and new developments definitely exists in primary schools and should be promoted as well. Especially the girls' interest for technical subjects like informatics should be stimulated at that age, as their interest diminishes during puberty and – maybe therefore – they were often neglected in respect of technics so far [1].

To foster young girls' interest in informatics RWTH Aachen started the project go4IT! in 2009 [19,20]. Since then, 284 workshops have been conducted in which more than 3.700 girls in the age 11–13 took their first steps in programming robots and computational thinking. The concept was based on the Roberta project (2003–2006) [24]. The BMBF-funded project Roberta Project of Fraunhofer IAIS (Institute for Autonomous Intelligent Systems) focused on gender-sensitive course concepts in didactic implementation and planning. In a second funding phase from 2005 to 2008, the project was funded by the European Union under the title "Roberta goes EU" for the transfer of concepts. Within the projects Roberta Centers were established, but no further efficacy studies were carried out after the first phase of the project [25].

The following assumptions were derived from the results of the Roberta project. A sustainable change in girls' self-concept through gender-sensitive robotics courses, including the gender difference, must be geared towards offering longer courses that go beyond a typical short workshop offer, as is customary at GirlsDay events. The didactic-pedagogical concept is based on increasing difficulty of the tasks while at the same time maintaining the motivation. Consequently, a particular value is attached to a variable schedule and aids for adjusting a current course are to be provided. The voluntary nature of participation decreases with longer courses in importance for effectiveness. Longer courses are therefore also suitable for participants who are previously not interested in STEM topics.

Additionally, instead of having girls to visit Roberta centers, go4IT! teams actively visit all high schools in the greater Aachen Region to offer two-day workshops in their premises. At the schools, an enthusiasm for computer science and technology should be transferred to the pupils, the parents and the teachers. The intervention course go4IT! is anchored in school-based profiling and thus brought into the consciousness of the school community. This takes pressure off girls, who are easily exerted by the peer group, when individual girls (for example, the only one in a class) express their interest in STEM, especially in computer science.

The main results of the dissertation of Leonhardt [26] for the conceptual refinement, design, and scientific evaluation of go4IT! are summarized in the following two subsections.

2.2 Theory-Based Design and Evaluation of go4IT!

Different gender-specific behaviors of (baby) girls and boys are recognizable by parents from birth, and present themselves in more stable emotions for girls, and higher irritability and impulsivity for boys [27–29]. However, research into the differences in the development of interests, attitude, and professional orientation, or even their origin, has not yielded any unambiguous results [29]. As Mammes notes, the approaches in gender research are controversially discussed and, in the case of a comparison of purely biological causes to purely sociological causes, can even be described as antagonized [30].

Therefore, these differences lead – amongst others – to a significantly smaller representation of women in computer science and other technical disciplines. Numerous measures and initiatives try to introduce girls, adolescents and young women to computer science. Nevertheless, since the reasons for these gender-specific differences are still mostly unclear, most measures lack a scientific basis.

The analysis of the state of research on measures to increase the number of female computer science students yielded a research controversy on the causes of gender differences. An examination of these causes has been essential with regard to the legitimacy and theory-based development of gender-specific support measures in the go4IT!-project.

It is obvious that there are biological differences between the sexes and it is indisputable that different behaviors at different stages of development can be observed. However, the interpretation of an implied biological gender difference as a basic assumption for gender-specific behavior in interaction with the environment is not detectable: neither by consideration of hormonal and genetic differences nor by theories about evolutionary selection. Stereotyping and thus accepting the social gender role in adulthood and the biological effects of hormones in physical and cognitive development form a complex interactive system capable of describing the current self, but the causes of gender differences seem inadequate. The question of the causes of gender differences remains scientifically largely unanswered and does not provide a suitable basis for constructively solving the problem.

Research findings on gender differences in the cognitive area are marginal, so that currently the thesis of gender equality in the cognitive area is plausible. Crucial cognitive differences are hard to detect or, as in the case of the gender difference in mental rotation (ability to mentally rotate two- or three-dimensional objects) can be quickly adjusted through training interventions [31]. Measurably large differences can be found in the motor skills and in basic psychological constructs. For the investigation of a persistent interest of a person in computer science and technology, the self-concept, the self-efficacy and the locus of control convictions in dealing with technology have been the most relevant psychological constructs [17,32].

From the point of view of pedagogical psychology, the formation of a dispositional interest in a person is a multiphase process that presupposes repeated activities with the object of interest. A negative attitude in self-efficacy (negative assessment of one's ability to reach a certain goal) and locus of control (expectation that the outcome does not depend on one's behavior or condition) is expressed in a weak self-concept in relation to the subject matter. A weak self-concept counteracts a continuous engagement with the object. The development of a dispositional interest in the subject is thereby severely impaired.

An intervention to promote a technical and informatics-related dispositional interest should therefore first foster the development of a positive self-concept. Increasing and stabilizing the interest development can then be done secondarily or subsequently – under the more favorable condition of an already positively influenced self-concept [33].

Utilizing tangible learning objects take away the fear of not being able to understand new commands, and has a positive effect on the participants' confidence in the examined setting. Hence, in groups with little prior knowledge tangible objects are preferable over virtual [32]. The advantage of a virtual learning object is the willingness to use unknown commands and, with increasing

Fig. 1. Designing and programming robots and their environment during go4IT!-workshops

complexity of tasks, can lead participants to try new approaches and to try new commands and constructs rather than a real learning object. According to these results, virtual learning objects make sense as a subject of study for in-depth courses [34].

2.3 Resulting go4IT! Design

Taking into account the deficits and conclusions described above, a two-step approach was developed. The first phase consists of a 2-day intervention workshop in the 6th and 7th grades (late childhood, age 11–12) for 12 to 14 girls in their schools (see Fig. 1). The choice of location aims at reaching girls who would normally be shy to independently take up an extra-curricular offer. Two students (at least one female) supervise the workshops acting as assistants and role models. The supervising students are trained in the practical application of the didactic principles of a reattribution training. This includes commenting on observed performance results with desired attributions, verbalizing desired attributions through a gender-neutral model, and, to a lesser extent, reinforcing favorable cause statements. There are almost no instruction phases. Students actively explore the commands for programming the robots.

During the workshop, the girls build Lego robots and program their behavior in a textual language NXC (not eXactly C) by reading sensor values and controlling different motors. In the end, they solve self-imposed tasks such as finding a path out of a labyrinth. Utilizing a textual programming language instead of a visual programming environment like Scratch has the advantage that the girls perceive it as real programming instead of clicking and playing with robots. They are proud having achieved the programming and tell the boys in their classes.

In the second phase, a follow-up workshop deepens the first and links to everyday life of the participants, who experience technology as designable and useful. The participation is voluntary, which corresponds to the phase of an emerging individual interest according to [35]. Carrying out the construction workshop over a longer period of time (4 days) as well as external special location support the repeated occurrence of a stabilized situational interest according to [36]. The psychological construct "fun" positively influences the subject-related emotion and in turn the development of an emerging individual interest. This link to fun has a significant impact on professional decisions [37]. Promoting autonomy, competence experience, social inclusion as well as the personal significance of the subject matter all support the formation of a stable personal interest among the participants.

In order to influence the control variable environmental socialization after the two workshops have been carried out, the parents of the children are included in the final stage of the second workshop.

Currently, we offer two different follow-up workshops, one during Easter vacation, and the other during the autumn break. In the first, titled "CS designer," the girls design and program clothes and accessories with technical finesse utilizing microcontrollers (Arduino Lilypad), lots of LEDs and conductive yarn. The girls take home their unique items for further development an enhancement of their programming skill in school lessons, study groups or privately. The autumn workshop focusses on programming their first App for a smartphone as a remote control for the robots, which then can be controlled for a complex choreography such as robot group dances, a robot marriage or playing roles in modern fairy tales in the final presentation with parents and siblings.

The questionnaire-based empirical pre- and post-study of the intervention workshop, together with the pre- and post-examination of the follow-up workshop, gave insights regarding the stabilization of effects on the technical self-concept as well as the related self-efficacy expectation and the locus of control in dealing with technology. In addition, a possible effect on a future perspective in the field of computer science has been examined.

In detail the following research hypotheses are examined: 1. The locus of control in dealing with technology and the self-efficacy expectancy in dealing with computers and technology and the future perspectives in the technical and informatics area. 2. The change on these variables by the action of an Intervention workshop involving the environment of Participating. 3. The updated actual state with respect to these variables in the Participation in the follow-up construction workshop. 4. The change in terms of these variables through the Follow-up construction workshop.

The non-parametric Wilcoxon signed rank test for connected samples has been used over the parametric t-test because the differences between the post- and pre-tests are not sufficiently normally distributed and the items are ordinally scaled. To interpret the measure of a possible effect, the effect size (r) is used in relation to Cohen's criterion. The significance is given as p. The test statistic U is calculated as the minimum of the negative and the positive rank sums [38].

There is a high significant improvement in the group of participants after the treatment intervention workshop with high effect size ($U = -13.012$, $p < .001$, $r = -.55$) on the self-efficacy in the use of computers and technology. This change was found in the entire sample (medianpre = 2.3333 and medianpost = 2.0000, while strong shift of the lower and upper quartile in the same direction). On locus of control scale, a high significant positive change was seen after the treatment with mean efficacy ($U = 6.418$, $p < .001$, $r = -.29$). The shift is evident throughout the sample. The future perspectives in the technical and informatics area changed after the treatment high significantly with a mean effect size ($U = -7.512$, $p < .001$, $r = -.325$). The median remains the same and the shift is in the upper two quartiles in the intended direction. Accordingly, it has not changed the overall group's attitude in the future perspective, but above all those participants with a previously more accurate or complete rejection of a future perspective in the informatics-technical field.

On the self-efficacy in dealing with computers and technology, only a significant small effect in the intended direction could be measured in the follow-up workshop ($U = -1.746$, $p < .041$, $r = -.24$). This is explained above all by the very strong self-efficacy of the participants in the follow-up workshop. The analysis of the change in the locus of control in dealing with technology did not yield any results. The future perspective changed in the group of the post-treatment setup workshop with a highly significant mean effect in the intended direction ($U = -2.663$, $p < .01$, $r = -.369$).

Sustainability of Interest. Despite the large number of projects that seek to promote STEM, there still is no long-term effect on study choice behavior, especially for women [39]. This is a future-endangering development, especially in our increasingly technological society, since access to the design of the STEM area is thus open to only one – predominantly male – part of the population.

According to Renn [39], essential deficits are the lack of entanglement of various measures and the lack of continuity due to financial problems and didactic deficits. Since there are only few subject areas from adolescence, in which a deeper personal interest is built up and maintained, interventions for the development of technology interest have to be made early in the development of children. This applies in particular to girls with a steadily declining interest and corresponding low self-concept towards STEM in lower secondary education [40].

From a system-theoretical perspective, the two phenomena of neutralization and dyssynchrony explain the lack of long-term effects on the self-concept and the self-efficacy expectation after interventions [33]. Neutralization describes the negative effect of the permanent control environment on the effect of the intervention. The effect of an intervention can be neutralized despite the proven short-term effects of the non-environmental factor. The environment as a control variable is subdivided into the two core areas of school and family in child development. Children spend much of their time at school. In addition to school education, they get to know gender stereotypes here as well as in their private environment, go through their puberty and solidify their attitudes. The behavior of the teachers and the design as well as the perception of the lessons have an effect on the subject-specific attitudes and on the professional career taken. According to [41], the assessed teaching quality of the subject closest to the later study option has a strong impact on the likelihood of starting a corresponding course. This underlines the important role that teachers can play in students' career choices.

A partial change of a system is not enough, so that the entire system permanently shows the desired behavior. To avoid dyssynchronies, it is important to offer promotion on a continuous basis. This lead to the foundation of the extra-curricular student lab InfoSphere at RWTH Aachen University, which is described in detail in the following section.

3 InfoSphere – Extracurricular Student Laboratory for Computer Science at RWTH Aachen University

3.1 Motivation for InfoSphere

The main goal for founding InfoSphere [21,22] as the learning lab for computer science at the RWTH Aachen University in 2011 was to strengthen students' stable interest in informatics through subsequent interventions on a regular schedule. InfoSphere currently offers 35 workshops, ranging from few hours to several days and addressing children and adolescents between 7 and 19 years, or in the case of family events even younger children up to their grandparents. The workshops promote the often under-perceived aspects of computer science such as creativity, teamwork, and its impact on everyday life.

Overview. InfoSphere offers different approaches to various facets and applications of computer science for children and teens covering all school grades. There are half-day and full-day modules as well as modules that stretch over several days. Modules contain research- and 'Puzzle'-projects and, above all, hands-on experience. Many of the modules were designed for lower secondary students, so that no prior knowledge is required, and can be booked for every grade. Some of the modules can be easily integrated into high school computer science classes, to broaden the knowledge on covered topics with a field-trip or as an alternative to in-class teaching. For these all learning materials are available as open educational ressources (OER).

Goals. The main goal of InfoSphere is to strengthen students' self-efficacy and locus of control as well as concrete competences to utilize and design technology for a range of tasks. Students are to explicitly perceive their own skills in analysis and design by utilizing methods of computer science. They work on open tasks, which require creative solutions in teams. InfoSphere is one of very few student laboratories, which do not just bother with applications, but with the core of computer science. The goal is to uncover concepts, methods and tools of computer science, which are normally hidden in everyday life applications.

InfoSphere Concept. The student lab is open to every level of education (all types and classes of schools) and computer science workgroups. It picks up current topics of computing in a different setting with modern devices (tablets, smartphones, computing toys and gadgets, microcontrollers and learning games), which cannot be experienced at school. Thus, subjects are more appealing and more impressive to students. They are directly connected to their life (e.g., smartphone-programming, GPS systems, social networks). They extend their point of view towards new and fascinating subjects like visual cryptography, the limits of computer-calculable problems, and other current computer science research topics.

Most of the modules can be completed independent of a computer science course, but even greater effects can be achieved if workshops are embedded in school courses. In order to allow teachers to embed workshops into their course plan module handbooks describe the compatibility of every InfoSphere module to the official curriculum (NRW-Informatiklehrplan).

InfoSphere modules additionally strengthen media-competences. Results of projects and experiences are documented in form of a community on our website and are made available for other students. Project results can be presented to non-participants like parents, fellow students or other teachers. This is most interesting when the topics are taken up in school lessons. Pupils are given access to a new platform, on which they can continue to pursue the topic and interact with fellow interested students.

Current CS research topics are also addressed to expose students to the world of a university and give them a first impression of research.

InfoSphere has drawn more than 9.800 students, presenting various facets of informatics and computational thinking; see Fig. 2. Again, scientific studies accompanied the conception, design, and development. In this case the research focused on factors to form and develop students' notion of computer science as a discipline. The main results of the corresponding dissertation are presented in the next section.

3.2 Theory-Based Design and Evaluation of InfoSphere

In her dissertation [23], Bergner investigated the prevailing public image of computer science – especially the one of children and adolescents – and analyzed it for typical misconceptions. She designed interventions in the extra-curricular student lab InfoSphere in order to shape prevailing images towards a more realistic picture of the discipline. Having the wrong imagination of what computer science is and what competences and skills need to be learnt is seen as one obstacle for students to (a) pick up the topic as career choice, and (b) successfully finish CSE in middle schools as well as universities.

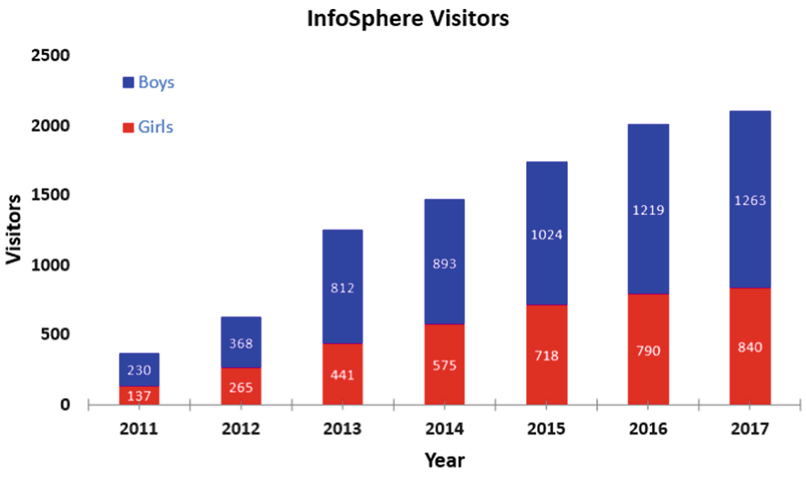


Fig. 2. InfoSphere workshops have been conducted with 9.800 students. There is an even higher demand, which is only limited by spatial and staff capacity.

Several studies concerning the overall concept of InfoSphere, design criteria for extra-curricular CSE interventions, and factors that influence the image of the discipline have been conducted. Overall, interesting differences were found between the sexes, in terms of both expectations for the visit to InfoSphere, and in terms of their ideas about the computer science discipline and IT professionals. For example, girls see the technical aspects of computer science in the foreground more than their male peers do. This statement, coupled with the fact that girls especially feel that the technical aspects are of little interest, can already be seen as a factor contributing to the low number of women interested in computer science.

Overall Goals of InfoSphere. Five main goals have been pursued when building the computer science learning lab InfoSphere at RWTH Aachen University:

1. All students have the opportunity to get to know the full spectrum computer science and its applications.
2. Visitors can actively and individually explore the world of informatics (thus its name); see Fig. 3.
3. Raising interest in informatics should occur as early as possible in order to counter public misconceptions and prejudices concerning the discipline.
4. Interventions are also addressing previously (seemingly) not-interested children (and especially girls and female adolescents) to give them a realistic insight into informatics in order to prepare well-grounded decision of whether to choose the subject, e.g., as elective in middle school.
5. Courses on all levels and for all ages are to continuously accompany students throughout their school time and lead to a sustainable development of interest and well considered decision of career choices.

Bergner could show via various qualitative and quantitative studies that the objectives 1, 4, and 5 have been achieved with InfoSphere modules. She found a greater diversity and a stronger content-related proximity to the notion of informatics concepts in the free text answers in the post-test (1) The high demand for workshops especially by primary and lower school classes, lead to the conclusion that students with previously no or few interest for computer science were reached (4) Additionally, the steadily increasing numbers of private registrations by multiply recurring students affirm that objective 5 is well addressed. Concerning (2) no conclusions can be drawn from the existing data. Only goal (3) show contradictory effects: While the interest of some students, especially the younger ones, could be increased, on average a small decrease has been perceived.

The sample of Bergner's studies included 2123 valid records, of which 1758 could be paired correctly. Thus, she evaluated pre- and post-test data of 879 visitors, of which 308 were female (35.0%) and 571 male participants (65.0%). It should be noted that among the visitors of the upper secondary level, the proportion of women drops to only 21.1% (compared to 41.1% in lower and middle grades). Furthermore, most of the visitor groups from the 8th grade visited InfoSphere as part of their computer science education. Overall, therefore, there is a dependence between age and gender as well as to having received computer science lessons in school.

Factors for Designing Extra-Curricular Interventions. Main findings show, that modules not relying on previous knowledge are favored. Modules with a strong reference to everyday life or those with materials and media not available in the average school context (i.e., smartphones, robots and microcontrollers) had a larger demand; see Fig. 3. A majority of participants reported in the post-test an increased interest in the cultural and sociological impact of modern technology and computer science methods (i.e., encryption algorithms).

The didactic design with two or three tutors (ideally with role models for both sexes) fostering individual and active exploration in teams, as well as rich mix of methods and presentations with modern devices has been evaluated very positive by students, teachers and parents. Participants acknowledged a high fun-factor for each of the 20 evaluated modules.

The difficulty level of a module had a great influence on almost all other ratings. A module rated as too difficult leads to low scores in interest for the topic, motivation to come back, perceived fun and even rating of tutors or materials. In contrast, too easy modules had nearly no loss in motivation, which leads to the conclusion that expecting too much of the students must always be avoided. Challenges on the other hand help the students to increase their performance and if applied correctly are the key to increase long term motivation.

The Prevailing Image of Computer Science. The mainly quantitative evaluation with regard to the students' conceptions about the discipline was implemented by means of an online-based pre-post-test design. After pre-tests and a first study, a second period – from November 2012 to December 2013 – 116

Fig. 3. Active exploration in teams with individually defined tasks is fostered

complete records of primary school children, and 879 responses from students of lower and upper secondary classes were evaluated. Measurements were made on the prevailing image of computer science (pre-test), as well as the changes triggered by an intervention, in particular the effect of various modules (comparison of pre- and post-test). For both evaluations, the participants were divided into subgroups of girls and boys, different age and school levels as well as those with and without previous informatics schooling.

Informatics Being Purely Concerned with Computers. The analysis of (up to) three free-text answers associated with the discipline revealed – analogous to the results of previous investigations – that computer science is often seen as only concerned with the computer. The term "computer" with 908 of the 1414 answers (64.2%) was clearly named most frequently. This is not as self-evident as it might seem, because in German the discipline is named informatics and not computer science, which is a misleading term. Also in the ranking of given terms, the term "computer literacy" has been chosen consistently for all subgroups as the top notion.

Informatics = Programming. The prejudice that computer science is synonymous with programming was also evident in a majority of students. The number of mentions of the term "programming" is highly dependent on the respective school level. While lower secondary students hardly mention programming, upper level students strongly focus it with 201 mentions (52.3%) only just behind the term "computer" with 207 votes (53.9%). The vast majority of the upper level students had CSE in school. Still, almost all visitors associate the job description of the computer scientist strongly with that of a programmer.

Girls Less Confident than Boys. Although overall it could be shown that the estimates for the confidence of the own image of the computer science are nearly normally distributed, nevertheless significant differences resulted between the sexes and also the age groups. Overall, girls felt clearly less confident with their conception. As a result of the lesser experience, younger pupils are much less confident than older students.

Interest Independent of the Subgroups. Although the gender-specific numbers of students suggest a different assumption, the preliminary survey shows that girls find computer science same as interesting and exciting as boys. In addition, the interest in computer science, as well as the assessment of the attributes "versatile" and "exciting," are independent of having been schooled in computer science. However, the school-level-specific evaluation shows that high school students (having had CSE in school) consider computer science to be less versatile than those from lower levels, which reveals that computer science education at schools does not always cover the entire spectrum of computer science.

Girls See No Future for Themselves in the Field of Computer Science. However, in terms of importance for the later professional life and the willingness to deal with computer science aspects in the future, there are significant differences between the sexes. Girls can imagine far less about dealing with computer science in the future than boys. Also, girls rank the influence of digitalization on their own later professional life far smaller than their male classmates. In order to find out the reasons for these divergent assessments, it is necessary to shed more light on the other questions on the image of computer science.

Computer Science (Still) a Male Subject. Although both sexes have a clear tendency towards CS being a male subject, it should be noted that – contrary to several studies by other authors – girls in the InfoSphere sample consider it significantly less male than boys.

Computer Scientist Are Socially Not Well Integrated. The presumed lack of social integration might explain why most students did not see their future in the field of computing. Especially girls and pupils without informatics classes doubt that computer scientists have (many) friends.

CS Is Not Creative. The same subgroups judge the profession of being not creative. Most girls do not link skills in solving complex problems with computer scientists. No groups recognize that utilizing methods of CS can create innovations.

Importance of Technology. In addition, the given free association of computer science terms revealed that female participants in particular strongly focus on the term "technology" (20.4% of girls versus 9.8% of boys). Gender comparisons show that boys give more room to aspects such as "teamwork" and "intelligence," while girls stronger emphasize the aspects of "understanding technology" and "programming."

Expectations on Computer Science Lessons (or Study Programmes). Lower secondary students who choose the elective show an interest "in computers" (probably more precisely in applications running on it), whereas counterarguments are diverse. In secondary education, on the other hand, most students take up the subject, if they plan to study CS at collegiate level.

Further Gender Differences. For the workshops girls are much more likely to desire clear guidance. This can be explained by often greater uncertainty and lower previous experience with informatics and computers. Only 46.9% of girls have previous knowledge from any kind of computer science education, whereas 81.7% of boys do. In addition, girls prefer to work with (many) different materials, whereas boys are more computer focused. This difference can also be explained by the different previous experiences with the computer medium. In order to address both genders equally, it is important to find a balance between plugged and unplugged CSE.

Impact of Computer Science Education. Students with computer science lessons rank concepts such as "logic" or "problem-solving" much higher than those without CSE. Additionally, this subgroup perceives the social relevance of CS consciously.

Age-Dependent Effects. Younger students link the ability to "explain well" with computer scientists presumably by transferring the observed competence from computer science teachers (or tutors in InfoSphere).

Overall, the analysis of the prevailing image of computer science, shows that

- the age, the school level, having received computer science education and above all the gender of the participants of the studies have a great influence on the prevailing image of computer science,
- above all the description as "science purely concerned with computers" and the equalization of computer science with programming are widespread prejudices,
- girls, even stronger than boys, are uncertain about the correctness of their image of CS,
- although there is generally no gender-specific interest in computer science, girls show less interest in almost all aspects of computer science topics,
- girls for themselves rarely see a future in or with computer science and
- the prejudices of the male computer science and the low social integration of computer scientists are still present today, albeit less pronounced than in previous studies.

Factors Leading to Transforming the Image of Computer Science. In order to evaluate the changes that were triggered by having taken at least one of the workshops in InfoSphere, on the one hand module-independent analyzes of the entire sample and on the other hand specific analyzes of the most frequently evaluated modules were carried out.

Broadened Image of Computer Science. The cross-module evaluations show strongly the image of computer science has widened. In the post-test survey, a total of 665 different terms were given, 24% more than in the preliminary survey. Most of the classical associations (e.g., "computer" or "programming") lost importance. The term "logic" was the only one of the top 10 mentions from the preliminary survey, which recorded an increase. The previously highly ranked term "technology," especially among female visitors, has been downgraded in favor of other informatics aspects.

Confidence Level. A key change, especially in terms of the objectives, is in the confidence concerning one's own image. Boys and girls feel significantly more confident in their personal image of CS.

Computer is a Tool. The role of the computer is evaluated differently. It is clearly more often perceived as a tool and not as the (sole) core of computer science. In direct connection with this, students are far more convinced that computer scientists do not only work on the computer and do not exclusively program. Furthermore, the activity of repairing devices in the view of children and adolescents decreases significantly. In the ranking of the given terms the keywords "computer literacy" and "technical understanding" lose value.

Decreasing Overall Interest in Computer Science. Negative, however, is the (slight) decrease in the values of interest. Here it is to be hoped that just those children and adolescents have lost some of their interest, which otherwise would have opted for computer science under false beliefs. Nonetheless, in the future it will be necessary to further investigate the effect and make corresponding changes to the modules.

Decrease in Relevance to Working Life. Furthermore, the perceived significance for one's later professional life has decreased. One possible explanation is that for the majority of the visitors, the modules address completely new and previously unknown areas of computer science. As these go far beyond the publicly prevailing opinion on IT competences, it is understandable that at first glance these aspects seem less relevant for later professional life.

Teamwork and Creativity. In the apparent contradiction to the previous point more advocates for a compulsory subject computer science are found after workshops. Another clear success of the InfoSphere concept is that the stereotype of the men's subject has been significantly reduced, both among girls and boys. In addition, the concept ranking shows that the relevance of the terms "creativity" and "teamwork" has increased significantly. Thus, it has been possible to bring previously unnoticed concepts into the consciousness, whereas often cited stereotypes have fallen into the background.

Social inclusion is now also viewed from a different perspective: more students are now convinced that computer scientists can have (many) friends. Overall, computer scientists are no longer viewed as loners who sit in front of the computer all day.

The *interest in computer science* has been influenced by InfoSphere workshops. Previously ignored activities (e.g., "solving puzzle problems") are attracting greater interest, whereas classical activities (e.g., "developing programs") are falling in interest. The reason for these changes could be that classical activities seem to be less desirable, or that the other activities have simply become more interesting. In addition, there is a shift in interest towards the social impact of CS.

It is surprising that despite the clear preference for partner and group work after completing a workshop the *desire for individual work* has increased. This effect might be caused by the almost continuous work with other students on the one hand or by the fact that in teams, some technical devices (smartphone, etc.) or limited hands-on materials must be shared, what occasionally led to disputes.

The module-specific evaluation yielded more detailed insights into the aforementioned findings. Especially the module "treasure hunt" was able to clarify because of its didactic structure – the computer serves here as a pure aid – that computer science is more than pure programming. Already the free association task showed that this module strongly emphasizes the meaning of the term "logic" in relation to computer science. Directly related to this is also the increased visibility of theoretical aspects of computer science. Also, the module "First Own App" shows significant effects, as the ability or activity to repair computers is less in the focus of the students. Presumably the first contact with aspects of programming for the young target group lead to the increase of the perceived importance of the term "programming." Unaffected by this, the perceived relevance of computer literacy decreases, so that it can be assumed that a new, broader picture of computer science has been conveyed.

The negative effects of the modules on the general interest of children and adolescents in computer science can be better explained by a detailed analysis of the two similar modules "InfoSphere goes Android" and "First Own App." Above all, the level of tasks is crucial. For example, the more sophisticated module "InfoSphere goes Android" saw a significant decline, while the second module remained the same. The decline can therefore mainly be explained by setbacks during workshops.

The module "First Own App" is particularly effective against the cliché of the male subject, with the other modules also showing significant changes. A possible reason for this is the chosen context. The participants develop their own painting program, which should appeal to especially artistic-creative students and especially female students. In addition, observations during workshops show that female students in particular follow the given instructions very closely and thus achieve the goal without major setbacks, whereas students who are mainly trying out things more independent have difficulties in implementing their ideas with the App Inventor.

With regard to the social integration of computer scientists, which was originally considered very negative, the module "Treasure Hunt" shows the greatest success. Through the continuous and necessary teamwork throughout the module it is conveyed that computer scientists are networking and working in teams.

In addition to an increase in the question of whether computer scientists have (many) friends, above all the ranking of given terms shows that according to this module "teamwork" and also "creativity" have gained considerable relevance. A similar result in terms of teamwork is also observed with the module "First Own App," although the teamwork there is not mandatory in terms of content, but an exchange between the participants significantly facilitates the independent development of software.

Very positive are also the other changes to the image of computer science. For example, visitors to the module "First Own App" regard computer science as more exciting and diversified. After the module "Internet Game," some students state that computer science is now easier to understand. The participants gain insights into a hitherto completely unknown technique. These first experiences of success in understanding complex relationships probably lead to this shift. The module also emphasizes that computer science plays a major role in a large number of areas.

It is also interesting to see how the students' interests regarding informatics content and activities have changed. Thus, both modules for app development and also for treasure hunting lead to a significant increase in interest in social aspects of computer science. At the same time, the "InfoSphere goes Android" module is reducing the interest in "getting to know new technologies" and "programming." This can mean two things: either the interest in programming has fallen directly due to the challenges and setbacks of the module, or it has somewhat faded into the background in favor of other aspects. The module "First Own App" increases the desire to try different methods, to give presentations and to discuss different aspects. The modules "Internet Game" and "Treasure Hunt" help the puzzle tasks to become more popular. Only the module "Internet Game" causes among its young target group, that also the classical activity – understand the functioning of programs – gains in popularity.

Finally, a few special aspects of individual modules stand out. First, the use of the App Inventor (or other English-language software) is very profitable in conveying the need for at least basic foreign language skills. On the other hand, utilizing particularly interesting devices or hands-on materials during group work phases may lead to conflicts and envy. Third, the desire for more individual work came up in the evaluation of the modules "InfoSphere goes Android" and "Internet Game." That shift did not occur in the other modules. It might be explained by the fact that teamwork was considered obligatory, otherwise achieving the goal would have been much more difficult or even impossible.

4 Conclusion

For both of the RWTH outreach activities there is scientific evidence that CSE is possible in young ages and can have the intended effects. The go4IT!-project has been shown to raise interest in technology and CS. The program runs successfully since 2009. However, it is also obvious, that singular inventions can only start the process of wakening interest, which must be stabilized over iterations of activities, which build on the previous positive experience. Of course it would be optimal, if CSE could be implemented as mandatory school subject on the same level as natural sciences.

As long as there is no such school integration, further CSE experience in lower and middle school classes must be provided by extra-curricular services such as the ones by the computer science student lab InfoSphere. Again, effects concerning the image of CS as a discipline could be shown. However, a longitudal study to investigate effects concerning career choices are missing.

References

1. Sabitzer, B., Antonitsch, P.K., Pasterk, S.: Informatics concepts for primary education: preparing children for computational thinking. In: Proceedings of the 9th Workshop in Primary and Secondary Computing Education (WiPSCE 2014), pp. 108–111 (2014)
2. Joint Informatics Europe & ACM Europe Working Group on Informatics Education: Informatics education: Europe cannot afford to miss the boat. Technical report (2013). http://europe.acm.org/iereport/ACMandIEreport.pdf. Accessed 28 Jan 2018
3. Gesellschaft für Informatik: Rahmenempfehlung für die Informatik im Unterricht der Sekundarstufe I, Informatik Spektrum, Band 9, Heft 2 (1986)
4. Gesellschaft für Informatik: Lehrerbildung im Bereich der Informatik, GI Empfehlung (1987). (Kurzfassung im Informatik Spektrum, Band 10, Heft 6, 1987; Langfassung: Beilage zu LOG IN 7, 1987, Heft 5/6)
5. Gesellschaft für Informatik: Fach Informatik in der Sekundarstufe II allgemeinbildender Schulen, Empfehlungen der GI, Informatik Spektrum, Band 16, Heft 6 (1993)
6. Puhlmann, H.: Grundsätze und Standards für die Informatik in der Schule: Bildungsstandards Informatik für die Sekundarstufe I; Empfehlungen der Gesellschaft für Informatik eV. LOG IN Verlag (2008)
7. Röhner, G., et al.: Bildungsstandards Informatik für die Sekundarstufe II. Beilage zu LOG IN, Heft 183/184 (2016)
8. The Committee on European Computing Education (CECE): Informatics education in Europe: are we all in the same boat? Technical report. ACM, New York (2017)
9. Bell, T., Andreae, P., Lambert, L.: Computer science in New Zealand high schools. In: Proceedings of the 12th Australasian Conference on Computing Education, pp. 15–22. Australian Computer Society, Brisbane (2010)
10. Duncan, C., Bell, T.: A pilot computer science and programming course for primary school students. In: Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE 2015), pp. 39–48 (2015)
11. Brown, N.C.C., Kölling, M., Crick, T., Peyton Jones, S., Humphreys, S., Sentance, S.: Bringing computer science back into schools: lessons from the UK. In: Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE 2013), pp. 269–274. ACM, New York (2013). https://doi.org/10.1145/2445196.2445277
12. Brown, N.C.C., Sentance, S., Crick, T., Humphreys, S.: Restart: the resurgence of computer science in UK schools. ACM Trans. Comput. Educ. **14**(2), 1–22 (2014). https://doi.org/10.1145/2602484. Article 9

13. Hubwieser, P., Armoni, M., Giannakos, M.N., Mittermeir, R.T.: Perspectives and visions of computer science education in primary and secondary (K-12) schools. Trans. Comput. Educ. **14**(2), 1–9 (2014)
14. Knobelsdorf, M., et al.: Computer science education in North-Rhine Westphalia, Germany—a case study. ACM Trans. Comput. Educ. (TOCE) **15**(2), 1–22 (2015). Special Issue II on Computer Science Education in K-12 Schools
15. Hromkovič, J.: Einführung in die Programmierung mit LOGO. Springer, Wiesbaden (2014). https://doi.org/10.1007/978-3-658-04832-7
16. Serafini, G.: Programmierunterricht für Kinder und deren Lehrpersonen: Unterrichtsmaterialien, didaktische Herausforderungen und konkrete Erfahrungen. In: Gallenbacher, J. (ed.) Informatik allgemeinbildend begreifen, pp. 267–272. Gesellschaft für Informatik e.V, Bonn (2015)
17. Papert, S.: Mindstorms: Children, Computers, and Powerful Ideas. Basic Books Inc., New York (1980)
18. Hromkovič, J.: Homo informaticus. Why computer science fundamentals are an unavoidable part of human culture and how to teach them. Olympiads in Informatics **10**, 99–109 (2016). https://doi.org/10.15388/ioi.2016.07
19. Leonhardt, T., Schroeder, U.: go4IT!: Initiierung und nachhaltige Förderung von Interesse an MINT-Fächern bei Mädchen. Informatische Bildung in Theorie und Praxis, Beiträge zur INFOS **13**, 132–142 (2009)
20. go4IT! Website. https://schuelerlabor.informatik.rwth-aachen.de/go4it. Accessed 31 Jan 2018
21. InfoSphere- Schülerlabor Informatik der RWTH Aachen. https://schuelerlabor.informatik.rwth-aachen.de. Accessed 31 Jan 2018
22. Bergner, N., Holz, J., Schroeder, U.: InfoSphere: an extracurricular learning environment for computer science. In: Proceedings of the 7th Workshop in Primary and Secondary Computing Education (WiPSCE 2012), pp. 22–29. ACM, New York (2012)
23. Bergner, N.: Konzeption eines Informatik-Schülerlabors und Erforschung dessen Effekte auf das Bild der Informatik bei Kindern und Jugendlichen. Dissertation, RWTH Aachen University, No. RWTH-2015-06987. Fachgruppe Informatik (2016)
24. Open Roberta Website. https://www.open-roberta.org. Accessed 31 Jan 2018
25. Kümmel, A.: Ergebnisse der wissenschaftlichen Begleitforschung des Projekts "Roberta goes EU". Berichtszeitraum 15.1.2007–31.12.2007 (2008)
26. Leonhardt, T.: Etablierung eines begabungsfördernden Lernumfeldes für Mädchen im Bereich Informatik. Dissertation RWTH Aachen, No. RWTH-2015-00826. Fachgruppe Informatik (2015)
27. Parsons, T., Bales, R. F., Olds, J., Zelditch, M., Slater, J. R.: Family, socialization and interaction process (International library of sociology, 1998. Aufl.). Routledge, London (1998)
28. Campbell, A., Shirley, L., Caygill, L.: Sex-typed preferences in three domains: do two-year-olds need cognitive variables? Br. J. Psychol. **93**(2), 203–217 (2002)
29. Hyde, J.S., Else-Quest, N.: Half the Human Experience. The Psychology of Women, 8th edn. Wadsworth Publishing Company, Belmont (2013)
30. Mammes, I.: Förderung des Interesses an Technik. Eine Untersuchung zum Einfluss technischen Sachunterrichts auf die Verringerung von Geschlechterdifferenzen im technischen Interesse (Europäische Hochschulschriften Pädagogik, Bd. 835). Lang, Frankfurt am Main (2001)
31. Feng, J., Spence, I., Pratt, J.: Playing an action video game reduces gender differences in spatial cognition. Assoc. Psychol. Sci. **18**, 850–855 (2007)

32. Brauer, P.: Konzeption, Entwicklung und Analyse eines greifbaren Turtles in Hinblick auf die Steigerung der Computerselbstwirksamkeit von Schülerinnen und Schülern: Diplomarbeit am Lehr- und Forschungsgebiet Informatik 9, RWTH Aachen (2010)
33. Ziegler, A., Schirner, S., Schimke, D., Stoeger, H.: Systematische Mädchenförderung in MINT: Das Beispiel CyberMentor. In C. Quaiser-Pohl (eds.), Bildungsprozesse im MINT-Bereich. Interesse, Partizipation und Leistungen von Mädchen und Jungen, pp. 109–126. Waxmann, Münster (2010)
34. Kammer, T., Brauner, P., Leonhardt, T., Schroeder, U.: Simulating LEGO mindstorms robots to facilitate teaching computer programming to school students. In: Kloos, C.D., Gillet, D., Crespo García, R.M., Wild, F., Wolpers, M. (eds.) EC-TEL 2011. LNCS, vol. 6964, pp. 196–209. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23985-4_16
35. Hidi, S., Renninger, K.A.: The four-phase model of interest development. Educ. Psychol. **41**, 111–127 (2006)
36. Krapp, A.: Pädagogische Psychologie. Ein Lehrbuch (Lehrbuch, 5. Aufl.). Beltz, Weinheim (2006)
37. Zwick, M.M., Renn, O.: Die Attraktivität von technischen und ingenieurwissenschaftlichen Fächern bei der Studien- und Berufswahl junger Frauen und Männer. Eine Präsentation der Akademie für Technikfolgenabschätzung in Baden-Württemberg. Akademie für Technikfolgenabschätzung in Baden-Württemberg, Stuttgart (2000)
38. Field, A.P.: Discovering statistics using SPSS. (and sex and drugs and rock 'n' roll) (Introducing statistical methods, 3. Aufl.), Los Angeles. SAGE Publications, Thousand Oaks (2009)
39. Renn, O.: Monitoring von Motivationskonzepten für den Techniknachwuchs. Herausgegeben von acatech - Deutsche Akademie der Technikwissenschaften. acatech BERICHTET UND EMPFIEHLT 5 (2011)
40. Hoffmann, L., Häußler, P., Lehrke, M.: Die Kieler Interessenstudie. IPN, Kiel (1998)
41. Heine, C., Egeln, J., Kerst, C., Müller, E., Park, S.-M. Bestimmungsgründe für die Wahl von ingenieur- und naturwissenschaftlichen Studiengängen. Ausgewählte Ergebnisse einer Schwerpunktstudie im Rahmen der Berichterstattung zur technologischen Leistungsfähigkeit Deutschlands, HIS (2006)

A Master Class on Recursion

Tom Verhoeff(✉)

Department of Mathematics and Computer Science,
Software Engineering and Technology Group,
Eindhoven University of Technology, Eindhoven, The Netherlands
t.verhoeff@tue.nl

Abstract. We elaborate some key topics for a master class on recursion. In particular, we show how to do recursion in an object-oriented programming language that does not allow recursion.

This article is dedicated to Juraj Hromkovič, who has inspired me (and many others) with his rare gift of contributing to cutting-edge research in theoretical computer science and to the didactics and popularization of informatics, and with his contagious enthusiasm for educating others. Good theoretical insights can and must be passed on to the next generations.

1 Introduction

The notion of a master class originated in the field of music, where a master musician gives a, often public, class to one or more accomplished players, usually in a one-on-one setting to help them improve and refine their skills. Likewise, the master class on recursion that is the subject of this article assumes prior experience with programming, and also with recursion. The goal is to improve and refine the skills in dealing with recursion, whether as a programmer or as a teacher. It is not aimed at the average student, but rather at the top 5%, at those who want to look beyond the International Olympiad in Informatics.

Recursion is considered a difficult but fundamental and even essential topic in informatics education. Two literature surveys [17,24] review a large number of publications on the teaching of recursion. A few additional publications are [1,3,11,12,20,25]. One of my introductions to recursion was [2], which is now outdated. For students who are not so keen on reading, the *Computerphile* videos on recursion [5,6] are informative and enjoyable. For a completely different angle on recursion, we refer to [22]. Still, we feel that several misconceptions about recursion keep popping up. This article (also) addresses these misconceptions.

This is not an article about (research on) the didactics of informatics. Rather, it constitutes a master class on recursion, focusing on what in my opinion are key ingredients. The approach and notation were heavily influenced by my teachers and mentors Edsger Dijkstra, Wim Feijen, and Netty van Gasteren, who took a formal approach to programming, referred to by some as *The Eindhoven School*.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 610–633, 2018.
https://doi.org/10.1007/978-3-319-98355-4_35

Overview

Section 2 discusses some of the preliminaries about *definitions*. The syntactic and operational aspects of recursion are addressed in Sect. 3. In particular, we introduce the notion of a *call graph*, and distinguish the *static call graph of a program* and the *dynamic call tree of its execution*. Section 4 is concerned with the *design* of recursive solutions. A key insight is that *operational reasoning* is *not* helpful, and that *contractual reasoning* makes recursion simple; so simple, in fact, that it can be applied in primary school. In Sect. 5, we look at various *program transformations* that somehow involve recursion. Among them are powerful techniques to improve the performance of recursive designs. In a way, we view Sect. 6 as the main reason for writing this article. There we explain how to do recursion in an object-oriented programming language if recursive functions are 'forbidden'. It shows how to program a *fixed-point constructor*, well-known from lambda calculus, in Java without recursive functions.

Section 7 concludes the article.

2 Preliminaries

2.1 Definitions as Abbreviations

We first look at how mathematical definitions 'work' (also see [26, Sect. 3.2]). Consider for example the following definition of $\mathit{mid}(a, b)$ for the number halfway between numbers a and b:

$$\mathit{mid}(a, b) = (a + b)/2 \tag{1}$$

It has some obvious properties:

$$\mathit{mid}(x, x) = x \tag{2}$$
$$\mathit{mid}(x, y) = \mathit{mid}(y, x) \tag{3}$$
$$\mathit{mid}(x, y) - x = y - \mathit{mid}(x, y) \tag{4}$$
$$\mathit{mid}(x - c, x + c) = x \tag{5}$$

And some maybe less obvious properties:

$$\mathit{mid}(x, y) = x \Leftrightarrow x = y \tag{6}$$
$$|z - x| = |y - z| \Leftrightarrow x = y \vee \mathit{mid}(x, y) = z \tag{7}$$
$$\mathit{mid}(\mathit{mid}(a, b), \mathit{mid}(c, d)) = \mathit{mid}(\mathit{mid}(a, c), \mathit{mid}(b, d)) \tag{8}$$
$$\mathit{mid}(\mathit{mid}(a, b), \mathit{mid}(b, c)) = \mathit{mid}(\mathit{mid}(a, c), b) \tag{9}$$

How would we prove (8)? We can calculate with these expressions and *apply* the definition of *mid*. Such an application involves a *double substitution*:

$$C(\mathit{mid}(A, B)) = C((A + B)/2) \tag{10}$$

Somewhere in an expression C (the context) occurs a usage of *mid*, with expressions A and B as its arguments. We replace that entire occurrence with the right-hand side of definition (1), and in that replacement, we replace every (free) occurrence of a by A, and of b by B, simultaneously.[1] We denote such a simultaneous substitution by $a, b \leftarrow A, B$. It is based on Leibniz's principle of *substituting equals for equals*. Here is an example calculation:

$$\begin{array}{cl}
 & mid(mid(a,b), mid(c,d)) \\
= & \{ \text{ apply definition (1) to leftmost occurrence of } mid \ \} \\
 & (mid(a,b) + mid(c,d))/2 \\
= & \{ \text{ apply definition (1) to leftmost occurrence of } mid \ \} \\
 & ((a+b)/2 + mid(c,d))/2 \\
= & \{ \text{ apply definition (1) to leftmost occurrence of } mid \ \} \\
 & ((a+b)/2 + (c+d)/2)/2 \\
= & \{ \text{ algebra } \} \\
 & (a+b+c+d)/4
\end{array}$$

In a similar way, we can calculate

$$\begin{array}{cl}
 & mid(mid(a,c), mid(b,d)) \\
= & \{ \text{ calculation above, with } a,b,c,d \leftarrow a,c,b,d \ \} \\
 & (a+c+b+d)/4.
\end{array}$$

Combining these two results yields (8).

Exercise. Using these properties, prove

$$mid(mid(x,z), mid(y,z)) = z \Leftrightarrow mid(x,y) = z. \tag{11}$$

Such mathematical definitions are just *abbreviations*, which can always be eliminated by repeated substitutions. This elimination is a mechanical process that needs to be done with care, but it requires neither intuition nor insight.

2.2 Recursive Definitions, Unfolding, and Inductive Proofs

Mathematics also allows *recursive definitions*,[2] which in general cannot be completely eliminated by substitution as described above.[3] Well known is the factorial function $n!$ (n factorial) defined for natural numbers n by

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ n \cdot (n-1)! & \text{if } n \geq 1. \end{cases} \tag{12}$$

[1] If A or B also contain free occurrences of a or b, then those must *not* be replaced. First replacing each a by b and then each b by a in $(a+b)/2$ would yield $(b+b)/2$.

[2] Also known as *inductive definitions* but see Subsect. 2.3 for a distinction.

[3] Also see Sect. 6.

This definition allows one to compute that $4! = 24$. But in the expression $(a+b)!$ it is not possible to eliminate the factorial in general. Note that definition (12) involves a case distinction. When $n = 0$, the factorial can be eliminated, but when $n \geq 1$, a single substitution will reintroduce the factorial, albeit applied to a smaller argument. Such a substitution is also known as an *unfolding* of the definition. It treats the definition as a *rewrite rule.*

Another famous recursive definition is that of the Fibonacci sequence F_n, where n is a natural number:

$$F_n = \begin{cases} n & \text{if } n \leq 1, \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases} \tag{13}$$

Here, an unfolding reintroduces two occurrences of F. It is easy to compute the first ten elements of the Fibonacci sequence:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots \tag{14}$$

But how can we hope to prove a property like

$$\sum_{i=0}^{n} F_i = F_{n+2} - 1 \quad \text{for } 0 \leq n \tag{15}$$

when the definition of F cannot be eliminated? Here are some values:

n	0	1	2	3	4	5	6	7	8	9	…
F_n	0	1	1	2	3	5	8	13	21	34	…
$\sum_{i=0}^{n} F_i$	0	1	2	4	7	12	20	33	54	88	…

It can be tackled by an *inductive proof.* Observe that (15) holds for $n = 0$ (via elimination by substitution). This is known as the *base of the induction.* To prove the property for $n \geq 1$, we now assume that it holds for $n-1$ (for which we have $n - 1 \geq 0$); this assumption is called the *induction hypothesis.* We calculate

$$\begin{array}{ll} & \sum_{i=0}^{n} F_i \\ = & \quad \{ \; n - 1 \geq 0; \text{ split off the last term, having } i = n \; \} \\ & \left(\sum_{i=0}^{n-1} F_i\right) + F_n \\ = & \quad \{ \text{ induction hypothesis: (15) with } n \leftarrow n - 1 \; \} \\ & (F_{n-1+2} - 1) + F_n \\ = & \quad \{ \text{ algebra } \} \\ & F_{n+1} + F_n - 1 \\ = & \quad \{ \text{ definition (13) of } F_n \text{ with } n \leftarrow n + 2 \; \} \\ & F_{n+2} - 1. \end{array}$$

This is called the *inductive step*, and it completes the proof by induction on n.

2.3 Inductive Types, Inductive Definitions, Structural Induction

The natural numbers $\mathbb{N}$ can be viewed as an *inductive (data) type*:

1. $0 \in \mathbb{N}$
2. $succ(n) \in \mathbb{N}$ if $n \in \mathbb{N}$ ($succ(n)$ is commonly written as $n+1$)
3. $\mathbb{N}$ consists only of (finite) things constructable via the preceding two steps.

A recursive definition could be

$$\mathbb{N} = \{0\} \cup \{succ(n) \mid n \in \mathbb{N}\}. \tag{16}$$

Similarly, we define the inductive type $\mathbb{T}(S)$ of *binary trees over S* by[4]

$$\mathbb{T}(S) = \{\perp\} \cup \{tree(u,v,w) \mid u \in \mathbb{T}, v \in S, w \in \mathbb{T}\}. \tag{17}$$

Here, $\perp$ denotes the empty tree, and $tree(u,v,w)$ denotes the tree with left subtree u, root value v, and right subtree w. It is customary to define accompanying *projection functions left*, *value*, and *right* for non-empty trees by

$$left(tree(u,v,w)) = u, \tag{18}$$

$$value(tree(u,v,w)) = v, \tag{19}$$

$$right(tree(u,v,w)) = w. \tag{20}$$

Definitions over inductive types are naturally given recursively. For instance, the height[5] $h(t)$ of a tree t can be defined by

$$h(t) = \begin{cases} 0 & \text{if } t = \perp, \\ 1 + max(h(left(t)), h(right(t))) & \text{if } t \neq \perp. \end{cases} \tag{21}$$

This is usually written more concisely as an *inductive definition*:

$$\begin{cases} h(\perp) & = & 0 \\ h(tree(u,v,w)) & = & 1 + max(h(u), h(w)) \end{cases} \tag{22}$$

Recursive definitions in the form of (21) are sometimes referred to as *top-down* definitions, because they *break down* the given argument by projections. Inductive definitions in the form of (22) are then referred to as *bottom-up* definitions, because they exploit how the argument is *built up* from smaller parts according to the inductive type. Although this distinction is not essential, it is good to be aware of the alternatives.

To prove something about an inductive type, one uses *structural induction.* To illustrate this, let us also define the size $\#(t)$ of binary tree t by

$$\begin{cases} \#(\perp) & = & 0 \\ \#(tree(u,v,w)) & = & \#(u) + 1 + \#(w) \end{cases} . \tag{23}$$

[4] There are many definitions and notations for binary trees. We allow the empty tree.
[5] This is not the most common definition for tree height, but it suits our purpose.

We now prove by structural induction the following property relating h and $\#$:

$$\#(t) + 1 \le 2^{h(t)} \quad \text{for all } t \tag{24}$$

For the base case, where $t = \bot$, we calculate

$$\begin{array}{cl} & 2^{h(\bot)} \\ = & \{ \text{ definition (22) of } h \ \} \\ & 2^0 \\ = & \{ \text{ algebra } \} \\ & 1 \\ = & \{ \text{ definition (23) of } \# \ \} \\ & \#(\bot) + 1. \end{array}$$

And for the inductive step, where $t = \mathit{tree}(u, v, w)$, we calculate

$$\begin{array}{cl} & 2^{h(\mathit{tree}(u,v,w))} \\ = & \{ \text{ definition (22) of } h \ \} \\ & 2^{1+\mathit{max}(h(u),h(w))} \\ = & \{ \text{ algebra, property of } \mathit{max} \ \} \\ & 2 \cdot \mathit{max}\left(2^{h(u)}, 2^{h(w)}\right) \\ \ge & \{ \text{ property of } \mathit{max} \ \} \\ & 2^{h(u)} + 2^{h(w)} \\ \ge & \{ \text{ structural induction hypothesis } \} \\ & \#(u) + 1 + \#(w) + 1 \\ = & \{ \text{ definition (23) of } \# \ \} \\ & \#(\mathit{tree}(u, v, w)) + 1. \end{array}$$

In fact, the bound in (24) is *tight*, in the sense that equality is possible for every height, viz. by the *complete* binary tree for that height.

Exercise. Reformulate the definitions of factorial and the Fibonacci sequence as inductive definitions, viewing the natural numbers as an inductive type, and redo the proof of (15) by structural induction.

3 Recursive Programs: Syntactic and Operational View

In mathematics, definitions (and proofs), including recursive ones, only need to be *effective*. In informatics, we are interested in definitions that allow *efficient* execution. Therefore, programs (implementations of functions) need to be designed with an eye for computational details.

It should be noted that early programming languages (such as FORTRAN) did not allow recursive function definitions. The main reason being that the way these languages were implemented did not support recursion [7]. In particular, the return address for each function invocation was stored in a fixed location, thereby precluding recursion. Only when a *stack* was introduced for function execution, to store actual parameters, local variables, and a return address for each function invocation in a *stack frame*, did it become possible to allow recursion in programs.

Consider a program written in the imperative core of Java. When does such a program involve recursion? That question is less innocent than it may seem. An answer you'll often hear is:

> A program is recursive when it contains a function whose definition contains a call of that function itself.

A classical example is the factorial function:

```java
long fac(int n) {
  return n == 0 ? 1 : n * fac(n-1);
}
```

There are several issues with this answer. The first issue is illustrated by the following example.

```java
long fac_1(int n) {
  return n == 0 ? 1 : n * fac_2(n-1);
}
long fac_2(int n) {
  return n == 0 ? 1 : n * fac_1(n-1);
}
```

The functions `fac_1` and `fac_2` each compute the factorial function, and neither definition contains a call to the function itself. But each function calls the other.

Another issue is illustrated with these two programs:

```java
void bogus() {
  if (false) {
    bogus();
  }
  System.out.println("Done.");
}
void print_abs(int x) { // print absolute value of x
  if (x >= 0)
    System.out.println(x);
  else
    print_abs(-x);
}
```

The function bogus contains a call to itself, but that call is never actually executed. The function print_abs is recursive, but the recursion always stops after one call. You could call these *degenerate* forms of recursion, or *bounded recursion*, as opposed to the 'real thing'.

3.1 Static Call Graph and Dynamic Call Tree

To address these concerns, we distinguish two viewpoints:

- the *static view*, where we consider the program *text* only, and
- the *dynamic view*, where we consider the program's *execution*.

The *static call graph* of a program (text) has

- as nodes: the function definitions in the program, and
- as arrows: $f \to g$ when definition of function f (textually) contains a call to function g.

We say that a program has (static) recursion when its static call graph contains a (directed) cycle. This is a purely syntactic phenomenon.

By this definition, all preceding programs have (static) recursion. We can distinguish the following special situations:

- *Direct recursion.* A cycle of length 1.
- *Indirect recursion.* A cycle of length >1.
- *Mutual recursion.* A cycle of length 2.

The *dynamic call tree* of a program *execution* has

- as nodes: function invocations (active or completed), and
- as arrows: $f \to g$ when invocation f invoked g.

Note that the same function can be invoked multiple times, and these invocations appear as separate nodes, each having their own actual parameter values, etc. Each invocation, except the very first, has exactly one parent that caused it. Thus, there are no (directed or undirected) cycles, and hence it is indeed a tree.

The dynamic call tree evolves over time, and can differ from execution to execution. For a program that uses no function parameters and no object-oriented features, this tree *conforms* to the static call graph, in the sense that there is a homomorphic mapping from the dynamic call tree to the static call graph that respects the arrows: each invocation of f in the tree maps to the definition of f in the graph.

The *depth of recursion* of an invocation in a call tree is the number of preceding invocations of the same function. It is useful to distinguish two types of recursive execution, based on the call tree:

- *Linear recursion.* Each invocation of f leads to ≤ 1 other invocation of f.
- *Branching recursion* (also known as *tree recursion*). Some invocations of f lead to ≥ 2 other invocations of f.

The naive implementation of the Fibonacci function (that returns the n-th Fibonacci number) exhibits branching recursion:

```
long fib(int n) {
  return n <= 1 ? n : fib(n-1) + fib(n-2);
}
```

Note that

- *linear recursion* is possible even when a function definition contains *multiple calls* of the function itself, viz. when those calls are mutually exclusive; for instance, if they occur in different branches of an if-statement.
- *branching recursion* is possible even when a function definition contains only *one call* of the function itself, viz. when that call occurs in a loop.

In linear recursion, the number of invocations of a recursive function equals the maximum depth of recursion. But in branching recursion, that number of invocations can grow *exponentially* in the maximum depth of recursion (recall property (24)). And this is in fact the case for the function `fib` shown above.

3.2 Tail Recursion

One further distinction is relevant for efficiency. A call of function g in the definition of function f is said to be a *tail call*, when execution of f is complete once that call of g returns. Such a tail call can be executed more efficiently by popping f's stack frame *before* invoking g, and passing f's return address as that for g. That saves on memory, especially in deeply nested invocations. This is known as *tail call optimization* (TCO) and compilers can do it automatically.

A function is said to be *tail recursive*, when all its recursive calls are tail calls. The classical factorial function shown above is not tail recursive, because after the recursive call returns, the function must still do a multiplication before returning. But the following generalization `fac_gen` of the factorial function, that computes $a \cdot n!$, is tail recursive:

```
long fac_gen(long a, int n) {
  return n == 0 ? a : fac_gen(a * n, n-1);
}
```

Tail recursion is also important because tail recursive functions can easily be converted into (non-recursive) loops (as we will discuss in more detail in Sect. 5). The loop version corresponding to the tail-recursive `fac_gen` is:

```
long fac_gen_i(long a, int n) {
  while (n != 0) {
    a = a * n;
    n = n - 1;
  }
  return a;
}
```

4 Reasoning About Recursion: Design by Contract

The operational view as captured in the dynamic call tree makes recursion intimidating, and unnecessarily hard to understand. That view is relevant to understand the implementation and *performance* (runtime and memory) of recursive functions. But it is hopelessly inadequate to understand the *correctness* of recursive functions.

There are two aspects to correctness:

– *termination*, and
– *establishing the correct effect*, also known as *partial correctness*.

We deal with termination in Subsect. 4.1. To reason about the correct effect of a recursive function, is (should be) the same as reasoning about any other function. Functions in programs are an *abstraction mechanism* [26]. To reason about them effectively, it is important to separate the details of the call's context from the details of the function's implementation (function body). This can be done through a *contract*, expressed in terms of a *precondition* and a *postcondition* (also see Fig. 1):

– The caller establishes the precondition.
– The body exploits this precondition and establishes the postcondition.
– The caller exploits this postcondition.
– The caller need not 'know' how the function body does its work.
– The body need not 'know' how the caller sets up the parameters and uses the result.

		Two-sided Contract		
		Precondition		**Postcondition**
	Caller	concern		benefit
Party		↓		↑
	Body	benefit	→	concern

Fig. 1. The relationships in two-sided contracts for functions

Adherence to the contract can typically be verified in terms of the program *text*, and need not involve its *execution*. That is, it relates to the static call graph, rather than the dynamic call tree. The verification consists of two parts:

1. verify that prior to *each* call (including recursive calls), the precondition is satisfied;
2. verify that at the end of the function body, the postcondition is satisfied (given that the precondition holds).

For example, the generalized factorial function `fac_gen(a, n)` can be understood in terms of the following contract.

- Precondition: $n \geq 0$
- Postcondition: returned result equals $a \cdot n!$

Note that the *function header* adds some conditions to the contract; in particular, it states the types of the parameters and the result (if any). These are typically checked by the interpreter or compiler. The body of `fac_gen` can be verified by distinguishing two cases, and calculating that the desired result is returned.

Case 1 ($n = 0$):

$$\begin{array}{cl} & a \cdot n! \qquad \text{(the desired result)} \\ = & \{ \text{ assumption } n = 0\text{, definition of } n! \ \} \\ & a \cdot 1 \\ = & \{ \text{ algebra } \} \\ & a \qquad \text{(the result returned by the body for } n = 0\text{)} \end{array}$$

Case 2 ($n \neq 0$):

$$\begin{array}{cl} & a \cdot n! \qquad \text{(the desired result)} \\ = & \{ \ n \geq 1 \text{ by the precondition, definition of } n! \ \} \\ & a \cdot (n \cdot (n-1)!) \\ = & \{ \text{ multiplication is associative } \} \\ & (a \cdot n) \cdot (n-1)! \\ = & \{ \text{ postcondition of } \mathit{fac\_gen}\text{, since its precondition } n - 1 \geq 0 \text{ holds } \} \\ & \mathit{fac\_gen}(a \cdot n, n-1) \qquad \text{(the result returned by the body for } n \neq 0\text{)} \end{array}$$

Note how these two cases correspond to the *base* and *step* of an inductive proof.

In fact, we could have started with these calculations *before* knowing the function's body, and thereby *derive* the implementation. That is, contractual reasoning can be used to design (recursive) programs [8,15,16]. Bertrand Meyer coined the term *design by contract* for this approach in 1986 [18,19].

This approach is so natural that children in primary school, when properly instructed, can successfully apply it [12,23]. It works best when loops are not introduced, and recursion is the only mechanism available for repeated execution. Of course, you would use different terminology: for every function you must formulate a *purpose* (its intended effect) and *assumptions* on when it is supposed to achieve that goal. You need to explain that you can break down problems into problems of the same kind, as long as those new problems are, in some sense, smaller than the original problem.

4.1 Termination of Recursion

That partial correctness is not enough is illustrated by the following function.

```
void miracle() {
  miracle();
}
```

It is easy to verify that function `miracle` satisfies the contract with precondition *True* and postcondition *False*. Just consider the body, and the contract that it satisfies. Consequently, this function solves *all* problems, since its precondition is always satisfied, and *False* implies any other condition. Unfortunately, it does not 'work', since it never terminates. But if it would terminate, it would indeed solve all problems. Termination is a separate concern.

Termination of a direct-recursive function f can be argued by providing a *variant function*, also known as *bound function*, or just *variant*, satisfying these conditions:

1. the variant function is a mapping
   – from f's parameters and relevant global variables that satisfy f's precondition
   – to some *well-founded domain* (such as the natural numbers, or a subset of the integers that is bounded from below);
2. prior to each recursive call of f, the value of the variant function for that call should be *strictly less* than its value upon entry of f's body.

In case of the functions `fac`, `fib`, and `fac_gen`, the well-founded domain of the natural numbers suffices, and n can be used as variant function.

When the variant function maps to the natural numbers, its value can serve as an *upper bound* to the depth of recursion; hence, the name 'bound function'. This upper bound does not have to be tight; it only concerns termination, and not efficiency. Even if the bound is tight, the runtime and memory complexity can be exponentially higher, as happens to be the case for the naive `fib`.

If the recursive function is defined over an *inductive type* using the *bottom-up* style (see Subsect. 3.1), then termination is guaranteed, since the inductive type can serve as well-founded domain, and the function's parameter as variant function.

For functions involving indirect recursion, proving termination is more complex, and beyond the scope of this article.

5 Program Transformations

It can be hard work to design (derive) a program from scratch, even if you ignore efficiency. It would be unfortunate if you had to redesign a program completely to improve its efficiency. This is where program transformations come to the rescue. A *correctness-preserving program transformation* is a technique that you can apply to a program text to obtain a program that is equivalent, as far as correctness is concerned. Usually the aim of such a transformation is to improve the program in some way, for instance, its performance. There is then no need to redesign it from scratch.

Numerous program transformations are available, in particular also in the area of recursion. We cannot treat them here in any depth. Instead, we will show a couple of examples.

5.1 From Loops to Recursion, and Back

We already mentioned a relationship between loops and recursion in Subsect. 3.2. Every loop can be transformed into an unbounded **while**-loop, having the form

```
T var = expr;
while (cond) {
  body;
}
finish;
```

where `var` represents the local variables initialized by expression `expr`, `cond` a boolean condition, and `body` and `finish` a group of statements. Such a loop can be transformed into the call

```
tail(expr);
```

of a *tail-recursive* function `tail` defined by

```
void tail(T var) {
  if (cond) {
    body;
    tail(var);
  }
  else {
    finish;
  }
}
```

This transformation can also be applied in the other direction: from a **`void`** tail-recursive function to while-loop. For non-**`void`** tail-recursive functions there is a similar transformation, as illustrated above with `fac_gen`.

It is well-known that any collection of (mutually) recursive functions can be replaced by non-recursive functions and one **while**-loop, using a *stack*. The stack holds the intermediate state of each active recursive invocation in a *stack frame*. At any time, one such invocation is being executed. A recursive call will temporarily suspend execution of the invoking function, and activate a new invocation that will then be executed. When a recursive invocation terminates, its stack frame is popped from the stack, and control returns to its invoker.

Sometimes, such a stack can be implemented simply by an integer variable that is incremented for each new recursive invocation, and decremented when the invocation terminates. Consider, for instance, a function that recognizes whether parentheses in a string are balanced.

5.2 Accumulation Parameters and Continuations for Tail Recursion

A function that is not tail recursive can be transformed into tail recursive form. We have seen that with `fac_gen`, which introduced a so-called *accumulation parameter* (a). In the case of factorial, this generalization is inspired by the calculation for $n \geq 2$:

$$\begin{array}{cl} & n! \\ = & \{ \text{ apply definition of factorial, using } n \geq 1 \ \} \\ & n \cdot (n-1)! \\ = & \{ \text{ apply definition of factorial, now using } n-1 \geq 1 \ \} \\ & n \cdot ((n-1) \cdot (n-2)!) \\ = & \{ \text{ multiplication is associative } \} \\ & (n \cdot (n-1)) \cdot (n-2)! \end{array}$$

Compare this to the correctness proof of fac_{gen} given in Sect. 4. So, in general, we are apparently interested in a calculation of the form

$$a \cdot n! \tag{25}$$

Thus, we arrive at the generalization $fac_{gen}(a, n) = a \cdot n!$. Here, a is the accumulation parameter which gathers (accumulates) the final result.

The general transformation takes the following form. Assume that our recursive function `f` looks like this:

```java
/** Return f(n) for n >= 0 */
long f(int n) {
  if (n == 0)
    return base_expr;
  else
    return G(f(n-1));
}
```

This function `f` is not tail recursive, because after the recursive call, function `G` still needs to be applied. If your programming language somehow supports function parameters (and preferably also *closures*), then we can employ the *continuation-passing style* (CPS) for generalizing `f` to `f_cps` and `G` to `g`:

```java
/** Return g(f(n)) for n >= 0 */
long f_cps(Function<Long, Long> g, int n) {
  if (n == 0)
    return g.__(base_expr);
  else
    return f_cps(x -> g.__(G(x)), n-1);
}
```

where `Function<D, R>` is a type for functions from domain type `D` to range (result) type `R`, and `x -> g.__(G(x))` is a function that maps `x` to `g(G(x))`.

More in detail, `Function<D, R>` is a *generic interface* in Java defined by

```java
@FunctionalInterface
interface Function<D, R> {
  public R __(D x);
}
```

This is a so-called *functional interface* (introduced in Java 8), which is an interface with just one method.[6] Any object, say `obj`, of any class that implements `Function<D, R>` can serve as a function object, and its function is invoked as `obj.__(...)`.

The expression `x -> E(x)`, where `E(x)` is an expression involving `x`, is a so-called *lambda expression* (also introduced in Java 8), that provides a compact syntax for defining an object of an anonymous inner class implementing a functional interface.

The resulting function `f_cps` is tail recursive. It pushes all the work that still needs to be done *after* the original recursive invocation returns, into the continuation parameter `g`, which accumulates an unevaluated composition of deeper and deeper nested calls of `G`: `g` equals `x -> G(G(...G(x)...))`. When the base case is encountered, this composition `g` is finally evaluated. Here is our standard factorial function transformed into continuation-passing style:

```
/** Return g(n!) */
long fac_cps(Function<Long, Long> g, int n) {
  if (n == 0)
    return g.__(1L);
  else
    return fac_cps(x -> g.__(n * x), n-1);
}
```

Now, the call `fac_cps(x -> x, n)` returns `n` factorial. Fortunately, as in the case of factorials, it is often possible to simplify `g.__(G(x))`, and replace the passing of a function by the passing of an evaluated expression.

5.3 Tupling

When dealing with multiple functions, on the same arguments, and following the same recursion pattern, these functions can be combined into a single function returning a tuple of the values returned by those functions. This technique is known as *tupling* [13,14].

Let's reconsider the Fibonacci function again, which involves a branching recursion. We can redefine it using two mutually recursive functions:

$$fib(0) = 0 \tag{26}$$
$$fib(n+1) = gib(n) \quad \text{for } n \geq 0 \tag{27}$$
$$gib(0) = 1 \tag{28}$$
$$gib(n+1) = gib(n) + fib(n) \quad \text{for } n \geq 0 \tag{29}$$

[6] To be as unobtrusive as possible, we have named that one method `__`. A single underscore is discouraged as identifier in Java, since it is reserved for 'future use'. We have avoided the predefined generic `java.util.functions.Function`, which has one method `apply`, whose name we find too long.

The functions *fib* and *gib* follow the same recursion pattern. Hence, it makes sense to consider the new function *fib_pair* defined by

$$fib\_pair(n) = (fib(n), gib(n)). \tag{30}$$

It can be computed recursively, using a 2-element array as pair:

```
long[] fib_pair(int n) {
  if (n == 0)
    return new long[] {0, 1};
  else {
    long[] p = fib_pair(n-1); // p = {fib(n-1), fib(n)}
    return new long[] {p[1], p[0] + p[1]};
  }
}
Function<Integer, Long> fib_1 = n -> fib_pair(n)[0];
```

This program has a runtime linear in `n`.

Tupling is a transformation technique *dual* to accumulation parameters.

6 Recursion Without Recursion in an OO Language

I have occasionally been guilty of telling the following story in introductory programming classes.

> In the beginning, there is *data*, of various *types*, and accompanying *expressions* to operate on data. *Variables* are used to store data. At each moment in time, a variable has one specific value. *Assignment statements* modify variables by assigning a value computed from an expression. A *selection statement* (also known as **if**-statement) provides conditional execution. All this is relatively easy, but it is still a poor programming formalism: the number of execution steps is bounded by the size of the program.
>
> Then come *loops*, a game changer. First bounded loops, with a precomputed upper bound (a.k.a. **for**-loops), and next unbounded loops (a.k.a. **while**-loops). Now, program size is no longer an upper bound on program execution: loops can repeatedly execute the same program fragments. This also makes reasoning about programs harder.
>
> At this point, I tell them that they know all that is needed to program anything that can be computed. The formalism is now *universal*.
>
> For convenience (or if you don't have unbounded integers), we introduce *arrays* or *lists*. And also *functions*, as possibly *parameterized* abbreviations for program fragments. And when they are ready for it (or not), I introduce them to *recursive functions*.
>
> And then I tell them that when you have recursive functions, you can drop the loops, and you still can program everything that is computable. You either need loops or recursion in a programming language to be universal, and both are hard.

Why 'guilty'? Because the statement 'You either need loops or recursion in a programming language to be universal' is not correct. Of course, I know this from Lambda Calculus [27], but that is not imperative programming. Only recently did it dawn upon me how to connect this to object-oriented programming.

Let's look again at the example of computing factorials. We have already considered generalizations for this problem. Here is another generalization, which we call f. Rather than calling the function itself in the 'inductive step', we let it call a function that was provided as argument. That is, we introduce a function parameter, say g (not to be confused with the continuation parameter of Subsect. 5.2):

$$f(g, 0) = 1 \tag{31}$$

$$f(g, n) = n \times g(n - 1) \quad \text{for } 0 < n \tag{32}$$

This function f is not recursive. It does not compute the factorial of n, but does something more general.[7] If, however, you supply for g any function that computes the factorial (expressible as a pre-post contract for g, which is part of the precondition for f), then f (also) computes the factorial (a postcondition for f):

$$(\forall m :: g(m) = m!) \Rightarrow f(g, n) = n! \tag{33}$$

Thus, the factorial function *fac* is a *fixed point* of f:

$$f(\mathit{fac}, n) = \mathit{fac}(n) \tag{34}$$

Using *currying*,[8] this can be rewritten as

$$f(\mathit{fac})(n) = \mathit{fac}(n). \tag{35}$$

or even more concisely as

$$f(\mathit{fac}) = \mathit{fac}. \tag{36}$$

This might seem to be circular, and therefore pretty useless. But the situation is a bit more subtle. The condition that g should compute the factorial is actually too strong. We can weaken it:

$$(\forall m : m < n : g(m) = m!) \Rightarrow f(g, n) = n! \tag{37}$$

(You could weaken it further, because only $m = n - 1$ when $0 < n$ is needed. But that is a bit less convenient to formulate, and we don't need this even weaker condition.)

Now comes the interesting thing. We could use f itself for its own g, if only g would take an additional function parameter. But that is easy to accommodate by 'upgrading' the specification of f, so that g takes an extra function parameter:

$$f(g, 0) = 1 \tag{38}$$

$$f(g, n) = n \times g(g, n - 1) \quad \text{for } 0 < n \tag{39}$$

[7] The best way to reason about this f is through contracts.

[8] $f(x, y) = (f(x))(y) = f(x)(y)$.

Equation (39) features a *self-application* of g. We now have the property:

$$f(f, n) = n! \tag{40}$$

The proof (omitted) is by induction on n, using (37) for the 'upgraded' f:

$$(\forall m : m < n : f(f, m) = m!) \Rightarrow f(f, n) = n! \tag{41}$$

6.1 Function Parameters and Self-Application in Java

The Java programming language does allow recursive function definitions. But let's pretend that these are forbidden. In Subsect. 5.2, we have seen how we can do function parameters in Java, by using an object as 'carrier' of a function via an instance method. We define the functional interface `PreFunction`:[9]

```
@FunctionalInterface
interface PreFunction {
  public long __(PreFunction g, int n);
}
```

Here is the code for our generalized factorial function f, using a lambda expression:

```
PreFunction f = (g, n) -> n == 0 ? 1 : n * g.__(g, n-1);
```

This is shorthand for

```
PreFunction f = new PreFunction() {
  @Override
  public long __(PreFunction g, int n) {
    if (n == 0)
      return 1;
    else
      return n * g.__(g, n-1);
  }
};
```

It is now possible to invoke $f(g, n)$ as `f.__(g, n)`. We could invoke this f with as argument for g the identity function *id*. First, we define this function *id* by implementing `PreFunction` (anonymously) through a lambda expression:

```
PreFunction id = (g, n) -> n; // independent of g
```

And then test it:[10]

```
  for (int n = 0; n < 5; ++n) {
    System.out.println("id(null, " + n + ") = " + id.__(null, n));
  }
```

It produces as output:

[9] The `Pre` part in the name reminds us that this function is a precursor of another function. By the way, it is true that this is a recursive *type definition*, because `PreFunction` occurs on the right-hand side. That is unavoidable because Java is a strongly typed language. But there is nothing strange about that. A class `Fraction` could have methods that take `Fraction` objects as arguments and return them, without there being any recursion.

[10] It does not matter what its `g` argument is; we chose **`null`**.

```
id(null, 0) = 0
id(null, 1) = 1
id(null, 2) = 2
id(null, 3) = 3
id(null, 4) = 4
```

We now invoke *f* with *id* as parameter:

```
    for (int n = 0; n < 5; ++n) {
      System.out.println("f(id, " + n + ") = " + f.__(id, n));
    }
```

This produces as output:

```
f(id, 0) = 1
f(id, 1) = 0
f(id, 2) = 2
f(id, 3) = 6
f(id, 4) = 12
```

In general, `f(id, n)` will return $n(n-1)$, except for $n = 0$ where it returns 1.
Finally, we invoke *f* with *f* as parameter:

```
    for (int n = 0; n < 5; ++n) {
      System.out.println("f(f, " + n + ") = " + f.__(f, n));
    }
```

obtaining as output:

```
f(f, 0) = 1
f(f, 1) = 1
f(f, 2) = 2
f(f, 3) = 6
f(f, 4) = 24
```

We could even introduce a type for functions of one **int** argument returning a **long**:

```
  @FunctionalInterface
  interface Function {
    public long __(int n);
  }
```

We can then define `Function fac` by

```
  Function fac = n -> f.__(f, n);
```

And test it:

```
    for (int n = 0; n < 5; ++n) {
      System.out.println("fac(" + n + ") = " + fac.__(n));
    }
```

To recap, we have defined a non-iterative non-recursive function `f` which we used to define a non-iterative non-recursive function `fac` to compute factorials. Of course, there is something loopy: `f` is called repeatedly, because `fac` supplies `f` as argument to `f`. This is known as *self-application*. But this is not a classical 'static' recursion, and also not recursive in the data. You could say that `fac` has 'dynamic' recursion, because it is set up at runtime. To emphasize this even further, we could have defined the factorial function as `fac2` in the following manner:

```
Function fac2 =
    i -> ((PreFunction) (g, n) -> n == 0 ? 1 : n * g.__(g, n-1))
         .__((g, n) -> n == 0 ? 1 : n * g.__(g, n-1),
             i
            );
```

Here it is obvious that there is no (static) recursion: since all functions in the definition are anonymous there is no way they can refer to themselves.[11]

It is a bit unfortunate that you have to pass along this g parameter all of the time, especially since it is always the same function. In the object-oriented setting, we can avoid such copying by putting it in an instance variable, and providing a setter. Instead of an interface, we define an abstract class for this, having an instance variable `g` and providing a setter method `set_g()`:

```
abstract class PreFunction2 {
  protected PreFunction2 g;
  public void set_g(PreFunction2 g) {
    this.g = g;
  }
  public abstract long __(int n); // can call g.__()
}
```

We now define `PreFunction2 f2` by

```
PreFunction2 f2 = new PreFunction2() {
  @Override
  public long __(int n) {
    return n == 0 ? 1 : n * g.__(n-1);
  }
};
```

Note how much it resembles the definition of `f` above. We configure and test it as follows:

```
  f2.set_g(f2);
  for (int n = 0; n < 5; ++n) {
    System.out.println("f2(" + n + ") = " + f2.__(n));
  }
```

Now, we do have recursion in the data: the object `f2` has obtained a reference to itself via the setter.

6.2 Fixed-Point Combinator in Java

We can generalize this approach further. Suppose you want to define a recursive function $D \to R$ from domain D to range R without using (static) recursion. We define the generic type `Function<D, R>` for this:

[11] Though a special syntax would be imaginable for recursive anonymous functions.

```
@FunctionalInterface // D -> R
interface Function<D, R> {
  public R __(D n);
}
```

For the recursive calls you introduce a function parameter of the same type $D \to R$. By currying (see above footnote 2), you define a function $(D \to R) \to (D \to R)$, which is isomorphic to a function with two parameters, one of type $D \to R$ and another of type D. For this we define the generic type `PreFunction<D, R>`:

```
@FunctionalInterface // (D -> R) -> (D -> R)
interface PreFunction<D, R> {
  public Function<D, R> __(Function<D, R> g);
}
```

For example, for the factorial function, you define

```
PreFunction<Integer, Long> pre_fac =
  g -> (n -> n == 0 ? 1 : n * g.__(n-1));
```

What we still need is a function Y that, when given such a precursor function f, such as `pre_fac`, returns its *least fixed point*. That is, it returns a 'least' function r such that

$$f(r) = r. \tag{42}$$

Thus, Y must have the property

$$f(Y(f)) = Y(f). \tag{43}$$

It is tempting try the following recursive definition of method `Y_rec_` for Y:

```
<D, R> Function<D, R> Y_rec_(PreFunction<D, R> f) {
  return f.__(Y_rec_(f));
}
```

But it will not work (it leads to a stack overflow), because the call `Y_rec_(f)` starts an infinite recursion. Java follows the so-called *strict* evaluation strategy, where all arguments of a function are evaluated *before* evaluating the function's body. We can fix that by a simple trick: delay evaluation of that argument by surrounding it with a lambda abstraction:

```
<D, R> Function<D, R> Y_rec(PreFunction<D, R> f) {
  return f.__(n -> Y_rec(f).__(n));
}
```

In general, a function F is equivalent to $\lambda x.F(x)$. But in Java, a lambda expression is only evaluated when it is supplied with a concrete argument. Therefore, (the body of) the outer application of `f` is now executed first, and it may invoke its argument (the lambda abstraction) if so desired. We now have an indirect self-application of `f` to `f`.

There is also a non-recursive strict way of defining method `Y_rec` for Y:

```
<D, R> Function<D, R> Y(PreFunction<D, R> f) {
  return ((Selfish<D, R>) x -> x.__(x))
      .__(x -> f.__(n -> x.__(x).__(n)));
}
```

We named it `Y`, and it uses self-application based on the function type:

```
@FunctionalInterface // S = S -> (D -> R)
interface Selfish<D, R> {
  public Function<D, R> __(Selfish<D, R> x);
}
```

The self-application appears *outside* `f`. The factorial function is then defined by

```
Function<Integer, Long> fac_r = Y_rec(pre_fac);
```

In the Lambda Calculus, this fixed-point combinator Y is written concisely as

$$Y = \lambda f.(\lambda x.x(x))(\lambda x.f(\lambda n.x(x)(n))). \tag{44}$$

Here, the term $(\lambda x.x(x))$ captures the self-application. If you look carefully, you recognize this structure in the Java code for `Y` above. But the Java syntax is still obscuring the beauty of this construction.

Remarks.

- It should be noted that in a formalism like Lambda Calculus, the evaluation of functions, even when defined through a fixed-point combinator, is purely based on substitutions and needs *no stack*. Nowadays, we would call that *self-modifying code*.
- It is sobering to realize that the Y-combinator was discovered (invented?) by Haskell Curry long before computing machines existed, possibly as early as 1929 [4]. Alan Turing published his Θ fixed-point combinator in 1937.
- Self-application lies at the heart of Gödel's proof of his famous Incompleteness Theorem.

7 Conclusion

For informatics teachers and anyone who wants to understand more of programming, it is mandatory to know more about recursion. We have presented some key topics for a master class on recursion, thereby highlighting several important aspects of recursion that are often not touched upon in introductory courses.

Important questions both for students and teachers are: What to learn/teach about recursion, and when to do so? The contractual approach explained in Sect. 4 is general, that is, not only applicable to recursive functions, but it is especially helpful in the context of recursion. You cannot go wrong by starting there. For practical programming, the program transformation techniques discussed in Sect. 5 are important. For better theoretical understanding, the preliminaries of Sect. 2, the syntactic and operational view of Sect. 3, and the fixed-point combinator of Sect. 6 are recommended.

The article is not a complete master class on recursion. In particular, more examples and exercises are needed. The following topics were omitted due to space limitations, though they should certainly be included:

– How to nest loops where the nesting depth is controlled by a variable;
– Backtracking, to traverse complex search spaces systematically;
– Dynamic programming, to trade memory and runtime; we recommend [10];
– Branch & bound, to improve performance; see [9] for an abstract approach;
– Recursive descent parsing, to analyze structured texts.

You may have brushed continuation passing (Subsect. 5.2) and fixed-point combinators (Sect. 6) aside as exotic beasts that only appear in the academic world. It is then good to realize that modern programming languages (such as Java) are evolving to incorporate the features underlying these concepts, especially functions as first-class citizens, for a reason. Functional programming is the future. To quote Peyton Jones [21]:

> "If you want to see which features will be in mainstream programming languages tomorrow, then take a look at functional programming languages today. [. . .]
>
> Writing software is all about managing complexity, the only thing that limits how ambitious the software that we write is, is really our ability to manage complexity, and functional languages give you much sharper tools for managing complexity than imperative ones do."

References

1. AlZoubi, O., Fossati, D., Di Eugenio, B., Green, N., Alizadeh, M., Harsley, R.: A hybrid model for teaching recursion. In: Proceedings of the 16th Annual ACM Conference on Information Technology Education (SIGITE), pp. 65–70. ACM (2015)
2. Barron, D.W.: Recursive Techniques in Programming. Macdonald, London (1968)
3. Butgereit, L.: Teaching recursion through games, songs, stories, directories and hacking. In: International Conference on Advances in Computing and Communication Engineering (ICACCE), pp. 401–407 (2016)
4. Cardone, F., Hindley, J.R.: History of lambda-calculus and combinatory logic. In: Gabbay, D.M., Woods, J. (eds.) Handbook of the History of Logic. Logic from Russell to Church, vol. 5. Elsevier, Amsterdam (2009)
5. Computerphile: What on Earth is Recursion? YouTube, May 2014. https://youtu.be/Mv9NEXX1VHc. Follow up: https://youtu.be/0pncNKHj-Sc
6. Computerphile: Programming Loops vs. Recursion. YouTube, September 2017. https://youtu.be/HXNhEYqFo0o. Follow up: https://youtu.be/DVG5G1V8Zx0
7. Dijkstra, E.W.: Recursive programming. Numer. Math. **2**(1), 312–318 (1960)
8. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall, Upper Saddle River (1976)
9. Feijen, W.H.J.: How to Bound in Branch and Bound. WF213 (1995). http://www.mathmeth.com/wf/files/wf2xx/wf213.pdf. Accessed 21 Jan 2018
10. Forišek, M.: Towards a better way to teach dynamic programming. Olymp. Inform. **9**, 45–55 (2015)

11. Ginat, D.: Impasse, conflict, and learning of CS notions. In: Hromkovič, J., Královič, R., Vahrenhold, J. (eds.) ISSEP 2010. LNCS, vol. 5941, pp. 13–21. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11376-5_2
12. Hauswirth, M.: If you have parents, you can learn recursion. The education column by Juraj Hromkovič. Bull. EATCS (123) (2017). http://bulletin.eatcs.org/index.php/beatcs/article/view/516
13. Hoogerwoord, R.: The design of functional programs: a calculational approach. Dissertation. Department Mathematics and Computer Science, Eindhoven University of Technology (1989)
14. Hoogerwoord, R.: 29 Years of (Functional) Programming: what have we learned? rh311. Department of Mathematics and CS, Eindhoven University of Technology (2013)
15. Kaldewaij, A.: Programming: The Derivation of Algorithms. Prentice-Hall, Upper Saddle River (1990)
16. Kourie, D.G., Watson, B.W.: The Correctness-By-Construction Approach to Programming. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27919-5
17. McCauley, R., Grissom, S., Fitzgerald, S., Murphy, L.: Teaching and learning recursive programming: a review of the research literature. Comput. Sci. Educ. **25**(1), 37–66 (2015)
18. Meyer, B.: Design by contract. Technical report TR-EI-12/CO. Interactive Software Engineering Inc., (1986)
19. Meyer, B.: Applying "design by contract". Computer (IEEE) **25**(10), 40–51 (1992)
20. Michaelson, G.: Teaching recursion with counting songs. In: ACM SIGCHI Interaction Design and Children (IDC 2015), Workshop on Every Child a Coder? Research Challenges for a 5–18 Programming Curriculum, Boston, 21 June 2015
21. Peyton Jones, S.: What's the future of programming? The answer lies in functional languages. TechRepublic, October 2017. https://www.techrepublic.com/article/whats-the-future-of-programming-the-answer-lies-in-functional-languages/. Accessed Apr 2018
22. Poundstone, W.: The Recursive Universe: Cosmic Complexity and the Limits of Scientific Knowledge. William Morrow and Company, New York City (1985)/Dover, Mineola (2013)
23. Proulx, V.K.: Introductory computing: the design discipline. In: Kalaš, I., Mittermeir, R.T. (eds.) ISSEP 2011. LNCS, vol. 7013, pp. 177–188. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24722-4_16
24. Rinderknecht, C.: A survey on teaching and learning recursive programming. Inform. Educ. **13**(1), 87–119 (2014)
25. Syslo, M.M., Kwiatkowska, A.B.: Introducing students to recursion: a multi-facet and multi-tool approach. In: Gülbahar, Y., Karataş, E. (eds.) ISSEP 2014. LNCS, vol. 8730, pp. 124–137. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09958-3_12
26. Verhoeff, T.: On abstraction and informatics. In: Proceedings of Selected Papers: 5th International Conference on Informatics in Schools: Situation, Evolution and Perspectives (ISSEP 2011), p. 40 (2011). Full paper on CD-ROM, 12 p. ISBN 978-80-89186-90-7
27. Verhoeff, T.: Informatics everywhere: information and computation in society, science, and technology. Olymp. Inform. **7**, 140–152 (2013)

Epilogue

About Place Cells and Grid Cells

Christos H. Papadimitriou(✉)

Computer Science, Columbia University, New York, NY, USA
christos@columbia.edu

Almost half a century ago, it was discovered that there are special neurons in the hippocampus of mammals that are activated every time the animal is in a particular place within its familiar surroundings [2]. If you let the animal roam freely near this place (say, if its path is the black line in Fig. 1), then this cell will fire at the red dots. These neurons are called *place cells*, and they seem to be part of the mental map animals need to get around in the world: they navigate a familiar terrain by going from one place cell to the next.

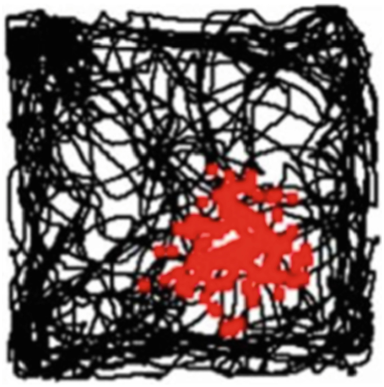


Fig. 1. A place cell fires when the animal is at or near a particular place (image from http://www.ucl.ac.uk/jefferylab/research.) (Color figure online)

Do we have place cells too? All mammals seem to have them, and I doubt that the capability was lost in evolution. Neuroscientists indeed have started spotting human place-like cells. But, for some reason unrelated to scientific evidence or insight, I like to believe that, in our brains, place cells have an elevated mission. That they may also record non-spatial loci particular to our species. Like the downhill pace when the murderer is about to be revealed in the mystery novel we are reading, or the mental labyrinth we inhabit when our algebra is not working. Or the sublime flash when we are starting to understand the gist of a proof that we have been wading through. Or the quiet bliss when—a faint smile in our lip—we are reminiscing about a friend.

My encounters with Juraj have been too few and far across, but I remember them vividly and fondly. A gentle and affable man, cordial and genuine in manner, quietly witty in conversation, a mensch who listens to you with shiny eyes

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 637–640, 2018.
https://doi.org/10.1007/978-3-319-98355-4_36

and understated smile—a smile that tells you "I do have something intelligent to say, but carry on." A meeting with Juraj is a boon to the mood and the intellect—and it almost always includes a hike. Because Juraj wants to share with his friends some of his favorite place cells.

Meanwhile, back to the hippocampus: during the last decade and a half the plot thickened—literally!

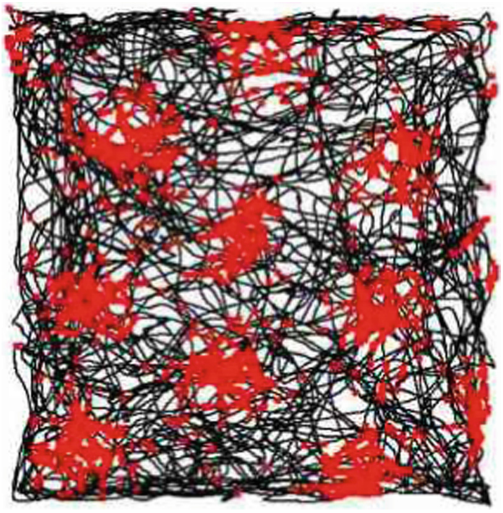


Fig. 2. A grid cell fires when the animal is at or near *any one* of the points in a triangular grid spanning a familiar domain (image from http://www.ucl.ac.uk/news/news-articles/1001/10012001.)

Grid cells are not in the hippocampus, but they live nearby in its gateway, another tiny and ancient piece of our brain called the entorhinal cortex, see [2] for a description. Note that Fig. 2 shows the response of *one cell;* the same neuron fires when the animal is close to any one of the vertices of this network.

At this point one has to stop and admit that this is remarkable. I like to think that the first geometry conceived by humans was discrete, not unlike the one of grid cells. The primitive human probably first experienced mathematics when she found a quiet moment to admire—from her fully erect posture—the starry sky. Much later we settled in places and cultivated the land and had to divide it, and so the Pythagoreans gave geometry its laws and stories. Ultimately, a man in Alexandria codified them all in thirteen books—of which the seventh, incidentally, contains the first piece of code ever written: `‘‘Keep subtracting the smaller from the larger until they’re equal’’`, because for Euclid integers were geometry too, lengths of rope. About twenty centuries went by—some good, most bad—and finally a Frenchman whose Latin name was Cartesius decided to go the opposite way: he found a sensible framework for codifying geometry with numbers. It took two more centuries of exploring the new possibilities, until—the horror!—geometries were stumbled upon that do not adhere by Euclid's axioms. These paradoxical concoctions puzzled and frustrated Immanuel Kant, who believed that geometry is about the only domain in science that cannot be relative, because our brain is embedded in it. And now, another two centuries later, we suddenly discover that brains have their own geometry—and it ain't ours.

But why this particular geometry? In neuroscience, any cell that responds to a stimulus (in this case, the animal's position) is characterized by its *tuning curve,* a mathematical function mapping the space of all stimuli (here, the unit square) to the positive reals. If the stimulus is x, then $f(x)$ is, roughly speaking, the probability that the neuron will fire upon sensing this stimulus. In this particular case, the tuning curve of grid cells is a surface in 3D that brings to mind a terrain with steep bell-shaped hills (capped by red peaks) arranged in a triangular grid, interleaved with concave troughs (the blacker places in the figure), also triangularly arranged. Why this tuning curve? What is the origin and logic of grid cells?

The dominant hypothesis about the function of grid cells is that they help with *path integration.* One idea (and there are many others) is that they help calculate the animal's position by somehow measuring and integrating elementary displacements (the integration is presumably done by another system of neurons, somewhere else). The measurement of the elementary displacements is done by measuring differences in the firing rate of the grid cell. The question is, does this theory justify the bizarre shape? If we were to design a neuron that measures displacement this way, which is the function f that optimizes the accuracy of this measurement?

This is a hard problem, and so we perform a familiar maneuver: we solve an easier, one-dimensional problem, and in fact one whose geometry has no edge effects. Suppose that the domain of the animal is the circumference of a circle; what is the tuning curve that will yield the most accurate measurements? This simpler question has an interesting answer (see Fig. 3).

The optimum tuning curve is a one-dimensional grid cell! Ergo, periodic tuning curves do seem to make sense in the context of path integration. The details of the derivation can be found in [1], but the idea is roughly this: First, you postulate that there is a population of n cells that measure displacement, and that their tuning curves, denoted $f_i(x)$ where x is now an angle, are shifts of one another by multiples of $\frac{2\pi}{n}$—notice that these tuning curves will eventually be proved the same. Measurement accuracy is best captured in this context by the *Fisher information*, a quantity which, through the Cramér-Rao theorem, bounds the variance of measurement. And it is known that in this case the Fisher information is given by the formula $\dot{f}(x)^T C^{-1} \dot{f}(x)$, where $\dot{f}(x)$ is the vector of the derivatives of the tuning curves of the n neurons at the angle x, and C is the matrix of the pairwise correlations of the measurement errors between the n neurons. So, this is the quantity we need to maximize under a "bounded power" constraint, which says that the length of the vector $\dot{f}(x)$ is at most one.

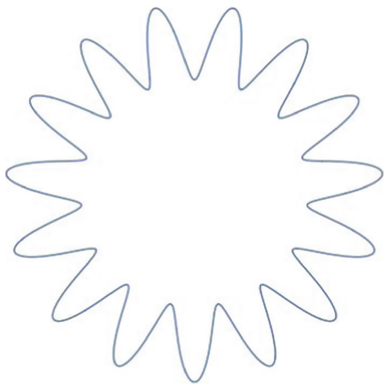


Fig. 3. The optimal one-dimensional tuning curve.

Now, C is both symmetric and circulant, due to the symmetries of the situation, and the Courant-Fischer theorem tells us that in symmetric matrices the unit vector that maximizes $\dot{y}^T C^{-1} y$ is the eigenvector of C corresponding to the smallest eigenvalue (the largest eigenvalue of C^{-1}). This eigenvector is a real-valued sinusoid, because the matrix is circulant, which concludes the sketch.

Does this exercise help us understand the logic behind the tuning curve of the two-dimensional grid cell? As explained in [1], it does, but only very little. The tuning curve of the two-dimensional grid cell is easily seen to be the product of two sinusoids (plus one, of course), and once we assume a product form of two independently optimized factors, the three-dimensional grid cells we know are the solution—at least on the torus. But too many assumptions were used.

There is a little epilogue to the story: how about *three* dimensions? Experiments with bats reveal that, indeed, they do embed their familiar flying space into a three-dimensional grid, whose structure is still a bit of a mystery (for a plausible prediction see the last section of [1]). And bats are not the only mammals who live in three dimensions, there are dolphins in the world, and walruses, and whales.

And of course climbers like my friend Juraj, to whom I am bestowing my warmest feelings and thoughts and my best wishes for a happy birthday.

References

1. Papadimitriou, C.H.: On the optimality of grid cells. Technical report (2018). https://arxiv.org/abs/1606.04876
2. Moser, E.I., Kropff, E., Moser, M.-B.: Place cells, grid cells, and the brain's spatial representation system. Annu. Rev. Neurosci. **31**(1), 69–89 (2008)

Author Index