

11th International Conference on Fun with Algorithms

FUN 2022, May 30–June 3, 2022, Island of Favignana, Sicily, Italy

Edited by

Pierre Fraigniaud

Yushi Uno



Editors

Pierre Fraigniaud 

Université Paris Cité and CNRS, France
pierre.fraigniaud@irif.fr

Yushi Uno

Osaka Metropolitan University, Japan
yushi.uno@omu.ac.jp

ACM Classification 2012

Theory of computation → Design and analysis of algorithms

ISBN 978-3-95977-232-7

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-232-7>.

Publication date

May, 2022

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <https://portal.dnb.de>.

License

This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0):
<https://creativecommons.org/licenses/by/4.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.FUN.2022.0

ISBN 978-3-95977-232-7

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

LIPICs – Leibniz International Proceedings in Informatics

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

Editorial Board

- Luca Aceto (*Chair*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Mikolaj Bojanczyk (University of Warsaw, PL)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University - Brno, CZ)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

■ Contents

Preface	
<i>Pierre Fraigniaud and Yushi Uno</i>	0:vii
List of Authors	
.....	0:ix

Regular Papers

Pushing Blocks by Sweeping Lines	
<i>Hugo A. Akitaya, Maarten Löffler, and Giovanni Viglietta</i>	1:1–1:21
A Practical Algorithm for Chess Unwinnability	
<i>Miguel Ambrona</i>	2:1–2:20
Pushing Blocks via Checkable Gadgets: PSPACE-Completeness of Push-1F and Block/Box Dude	
<i>Joshua Ani, Lily Chung, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, and Jayson Lynch</i>	3:1–3:30
Fun Slot Machines and Transformations of Words Avoiding Factors	
<i>Marcella Anselmo, Manuela Flores, and Maria Madonia</i>	4:1–4:15
Chess Is Hard Even for a Single Player	
<i>N.R. Aravind, Neeldhara Misra, and Harshil Mittal</i>	5:1–5:20
Rolling Polyhedra on Tessellations	
<i>Akira Baes, Erik D. Demaine, Martin L. Demaine, Elizabeth Hartung, Stefan Langerman, Joseph O’Rourke, Ryuhei Uehara, Yushi Uno, and Aaron Williams</i>	6:1–6:16
Beedroids: How Luminous Autonomous Swarms of UAVs Can Save the World?	
<i>Quentin Bramas, Stéphane Devismes, Anaïs Durand, Pascal Lafourcade, and Anissa Lamani</i>	7:1–7:21
Priority Queues with Decreasing Keys	
<i>Gerth Stølting Brodal</i>	8:1–8:19
Zero-Knowledge Proof of Knowledge for Peg Solitaire	
<i>Xavier Bultel</i>	9:1–9:17
Nimber-Preserving Reduction: Game Secrets And Homomorphic Sprague-Grundy Theorem	
<i>Kyle W. Burke, Matthew Ferland, and Shang-Hua Teng</i>	10:1–10:17
Quantum-Inspired Combinatorial Games: Algorithms and Complexity	
<i>Kyle W. Burke, Matthew Ferland, and Shang-Hua Teng</i>	11:1–11:20
Grabbing Olives on Linear Pizzas and Pissaladières	
<i>Hungry Coatis</i>	12:1–12:20
How Fast Can We Play Tetris Greedily with Rectangular Pieces?	
<i>Justin Dallant and John Iacono</i>	13:1–13:19

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno



Leibniz International Proceedings in Informatics

LIPICIS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Synchronization Game on Subclasses of Automata <i>Henning Fernau, Carolina Haase, and Stefan Hoffmann</i>	14:1–14:17
Making Life More Confusing for Firefighters <i>Samuel D. Hand, Jessica Enright, and Kitty Meeks</i>	15:1–15:15
Sorting Balls and Water: Equivalence and Computational Complexity <i>Takehiro Ito, Jun Kawahara, Shin-ichi Minato, Yota Otachi, Toshiki Saitoh, Akira Suzuki, Ryuhei Uehara, Takeaki Uno, Katsuhisa Yamanaka, and Ryo Yoshinaka</i>	16:1–16:17
Skiing Is Easy, Gymnastics Is Hard: Complexity of Routine Construction in Olympic Sports <i>James Koppel and Yun William Yu</i>	17:1–17:20
How Brokers Can Optimally Abuse Traders <i>Manuel Lafond</i>	18:1–18:19
Wordle Is NP-Hard <i>Daniel Lokshтанov and Bernardo Subercaseaux</i>	19:1–19:8
Mirror Games Against an Open Book Player <i>Roey Magen and Moni Naor</i>	20:1–20:12
Fun with FUN <i>Fabien Mathieu and Sébastien Tixeuil</i>	21:1–21:13
All Your bases Are Belong to Us: Listing All Bases of a Matroid by Greedy Exchanges <i>Arturo Merino, Torsten Mütze, and Aaron Williams</i>	22:1–22:28
Playing Guess Who with Your Kids <i>Ami Paz and Liat Peterfreund</i>	23:1–23:10
How to Physically Verify a Rectangle in a Grid: A Physical ZKP for Shikaku <i>Suthee Ruangwises and Toshiya Itoh</i>	24:1–24:12

■ Preface

This LIPICs volume contains the papers presented at the 11th International Conference on Fun with Algorithms (FUN 2022), May 30–June 3, 2022, Island of Favignana, Sicily, Italy. FUN is a series of conferences dedicated to the use, design, and analysis of algorithms and data structures, focusing on results that provide amusing, witty but nonetheless original and scientifically profound contributions to the area. Fun is a notion that can be judged from different perspectives, and be defined in many different manners. The conference defines fun in a broad sense, including aspects such as elegance, simplicity, amusement, surprise, originality, etc. The topics of interest include all aspects of algorithm design and analysis, and of computational complexity, under all types of algorithmic models (sequential, parallel, distributed, streaming, online, etc.).

Twenty-four papers were selected by the members of the program committee out of fifty four regular submissions. The main two criteria for judging the quality of the submitted papers were their scientific quality and their fit to the spirit of the conference. Several selected papers are dedicated to the analysis of the computational complexity of games such as Chess or Wordle, as well as to algorithmic solutions to these games, but many others are studying problems unrelated to games. The latter are either focussing on classical computational problems considered from an unorthodox angle, or are introducing new problems that were judged original and potentially fruitful from an algorithmic perspective.

PC Members FUN 2022

- Oswin Aichholzer (Graz University of Technology, Austria)
- Alkida Balliu (University of Freiburg, Germany)
- Pierluigi Crescenzi (Gran Sasso Science Institute, Italy)
- Miriam Di Ianni (Università di Roma Tor Vergata, Italy)
- David Eppstein (University of California, Irvine, USA)
- Panagiota Fatourou (University of Crete, Greece)
- Fedor Fomin (University of Bergen, Norway)
- Pierre Fraigniaud (Université Paris Cité and CNRS, France) - co-chair
- Magnús M. Halldórsson (Reykjavik University, Iceland)
- Taisuke Izumi (Osaka University, Japan)
- Kei Kimura (Kyushu University, Japan)
- Masashi Kiyomi (Seikei University, Japan)
- Lisa Kohl (CWI, Netherlands)
- Irina Kostitsyna (TU Eindhoven, Netherlands)
- Jayson Lynch (University of Waterloo, Canada)
- Till Miltzow (Utrecht University, Netherlands)
- Neeldhara Misra (IIT Gandhinagar, India)
- Valia Mitsou (Université de Paris, France)
- Takaaki Mizuki (Tohoku University, Japan)
- Lata Narayanan (Concordia University, Canada)
- Harumichi Nishimura (Nagoya University, Japan)
- Yoshio Okamoto (University of Electro-Communications, Japan)
- Boaz Patt-Shamir (Tel-Aviv University, Israel)
- Andrea Pietracaprina (University of Padova, Italy)
- Sergio Rajsbaum (UNAM, Mexico)

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Adele Rescigno (University of Salerno, Italy)
- Ryuhei Uehara (JAIST, Japan)
- Yushi Uno (Osaka Metropolitan University, Japan) - co-chair
- Virginia Vassilevska Williams (MIT, USA)
- Aaron Williams (Williams College, USA)
- Prudence Wong (University of Liverpool, UK)
- Tom van der Zanden (Maastricht University, Netherlands)

We want to thank all authors of the papers submitted to FUN 2022. Selecting a restricted subset of papers from these submissions often led to heartbreaking decisions. We also want to express our profound gratitude to the program committee members (and to their sub-reviewers), who not only produced informative reports on the submitted papers, but also actively participated to the discussions leading to the final decisions.

Last but not least, we want to deeply thank the organization team, Linda Pagli and Giuseppe Prencipe, both from Università di Pisa, Italy, who made possible to have the conference with participants onsite despite the uncertainties caused by the Coronavirus 2019 (COVID-19) pandemic. In fact, the May 2022 meeting on Island of Favignana will gather the participants of both FUN 2020 and FUN 2022, as the former was postponed due to the pandemic.

Pierre Fraigniaud and Yushi Uno
PC Chairs FUN 2022

■ List of Authors


Hugo A. Akitaya (1)
University of Massachusetts Lowell, MA, USA

Miguel Ambrona  (2)
Independent Researcher, Madrid, Spain

Joshua Ani (3)
Massachusetts Institute of Technology,
Cambridge, MA, USA


Marcella Anselmo (4)
Dipartimento di Informatica,
University of Salerno, Fisciano (SA), Italy


N.R. Aravind (5)
Department of Computer Science and
Engineering, Indian Institute of Technology,
Hydresbad, India

Akira Baes  (6)
Département d'Informatique,
Université libre de Bruxelles, Belgium


Jean-Claude Bermond (12)
Université Côte d'Azur, Coati, I3S, CNRS,
INRIA, Sophia Antipolis, France

Quentin Bramas  (7)
University of Strasbourg, ICUBE, CNRS, France

Gerth Stølting Brodal  (8)
Department of Computer Science, Aarhus
University, Denmark


Xavier Bultel  (9)
INSA Centre Val de Loire,
Laboratoire d'informatique fondamentale
d'Orléans, Bourges, France


Kyle W. Burke (10, 11)
Department of Computer Science,
Plymouth State University, NH, USA


Lily Chung  (3)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Michel Cosnard (12)
Université Côte d'Azur, Coati, I3S, CNRS,
INRIA, Sophia Antipolis, France


Justin Dallant  (13)
Université libre de Bruxelles, Belgium

Erik D. Demaine  (3, 6)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Martin L. Demaine  (6)
Computer Science and Artificial Intelligence
Laboratory, Massachusetts Institute of
Technology, Cambridge, MA, USA


Stéphane Devismes  (7)
Université de Picardie Jules Verne,
MIS UR 4290, Amiens, France

Yevhenii Diomidov (3)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Anaïs Durand  (7)
University Clermont Auvergne,
CNRS UMR 6158, LIMOS, Aubière, France


Jessica Enright  (15)
School of Computing Science,
University of Glasgow, UK


Matthew Ferland (10, 11)
Department of Computer Science, University of
Southern California, Los Angeles, CA, USA

Henning Fernau  (14)
Fachbereich IV, Informatikwissenschaften,
Universität Trier, Germany

Manuela Flores (4)
Dipartimento di Informatica,
University of Salerno, Fisciano (SA), Italy


Carolina Haase (14)
Informatik, Hochschule Trier, Germany

Samuel D. Hand  (15)
School of Computing Science,
University of Glasgow, UK

Elizabeth Hartung  (6)
Department of Mathematics, Massachusetts
College of Liberal Arts, North Adams, MA, USA

Frédéric Havet (12)
Université Côte d'Azur, Coati, I3S, CNRS,
INRIA, Sophia Antipolis, France

Dylan Hendrickson  (3)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Stefan Hoffmann  (14)
Fachbereich IV, Informatikwissenschaften,
Universität Trier, Germany

11th International Conference on Fun with Algorithms (FUN 2022).


Editors: Pierre Fraigniaud and Yushi Uno



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


- John Iacono  (13)
Université libre de Bruxelles, Belgium
- Takehiro Ito  (16)
Graduate School of Information Sciences,
Tohoku University, Japan
- Toshiya Itoh  (24)
Department of Mathematical and Computing
Science, Tokyo Institute of Technology, Japan
- Jun Kawahara  (16)
Kyoto University, Japan
- James Koppel (17)
Electrical Engineering and Computer Science,
MIT, Cambridge, MA, USA
- Manuel Lafond  (18)
University of Sherbrooke, Canada
- Pascal Lafourcade  (7)
University Clermont Auvergne,
CNRS UMR 6158, LIMOS, Aubière, France
- Anissa Lamani  (7)
University of Strasbourg, ICUBE, CNRS, France
- Stefan Langerman  (6)
Département d'Informatique,
Université libre de Bruxelles, Belgium
- Daniel Lokshtanov (19)
University of California Santa Barbara,
CA, USA
- Jayson Lynch (3)
Cheriton School of Computer Science,
University of Waterloo, Canada
- Maarten Löffler (1)
Department of Information and Computing
Sciences, Utrecht University, The Netherlands
- Maria Madonia (4)
Dipartimento di Matematica e Informatica,
University of Catania, Italy
- Roey Magen (20)
Weizmann Institute of Science, Rehovot, Israel
- Fabien Mathieu (21)
Swapcard, Paris, France
- Kitty Meeks  (15)
School of Computing Science,
University of Glasgow, UK
- Arturo Merino  (22)
Department of Mathematics,
TU Berlin, Germany
- Shin-ichi Minato  (16)
Kyoto University, Japan
- Neeldhara Misra (5)
Department of Computer Science and
Engineering, Indian Institute of Technology,
Gandhinagar, India
- Harshil Mittal (5)
Department of Computer Science and
Engineering, Indian Institute of Technology,
Gandhinagar, India
- Torsten Mütze  (22)
Department of Computer Science,
University of Warwick, Coventry, UK;
Department of Theoretical Computer Science
and Mathematical Logic, Charles University,
Prague, Czech Republic
- Moni Naor (20)
Weizmann Institute of Science, Rehovot, Israel
- Joseph O'Rourke  (6)
Department of Computer Science, Smith College,
Northampton, MA, USA
- Yota Otachi  (16)
Nagoya University, Japan
- Ami Paz (23)
LISN, CNRS & Paris Saclay University, France
- Liat Peterfreund (23)
LIGM, CNRS & Gustav Eiffel University,
Champs-sur-Marne, France
- Suthee Ruangwises  (24)
Department of Mathematical and Computing
Science, Tokyo Institute of Technology, Japan
- Toshiki Saitoh  (16)
Kyushu Institute of Technology, Japan
- Bernardo Subercaseaux  (19)
Carnegie Mellon University,
Pittsburgh, PA, USA
- Akira Suzuki  (16)
Graduate School of Information Sciences,
Tohoku University, Japan
- Shang-Hua Teng (10, 11)
Department of Computer Science, University of
Southern California, Los Angeles, CA, USA
- Sébastien Tixeuil (21)
Sorbonne Université, CNRS, LIP6, Paris, France

Ryuhei Uehara  (6, 16)
School of Information Science, Japan Advanced
Institute of Science and Technology, Ishikawa,
Japan


Takeaki Uno (16)
National Institute of Informatics, Tokyo, Japan


Yushi Uno (6)
Graduate School of Informatics,
Osaka Metropolitan University, Japan

Giovanni Viglietta (1)
School of Information Science, Japan Advanced
Institute of Science and Technology (JAIST),
Ishikawa, Japan

Aaron Williams  (6, 22)
Department of Computer Science,
Williams College, Williamstown, MA, USA

Katsuhisa Yamanaka  (16)
Iwate University, Japan

Ryo Yoshinaka  (16)
Graduate School of Information Sciences,
Tohoku University, Japan

Yun William Yu  (17)
Computer and Mathematical Sciences,
University of Toronto at Scarborough, Canada;
Department of Mathematics,
University of Toronto, Canada

Pushing Blocks by Sweeping Lines

Hugo A. Akitaya 

University of Massachusetts Lowell, MA, USA

Maarten Löffler 

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Giovanni Viglietta 

School of Information Science, Japan Advanced Institute of Science and Technology (JAIST),
Ishikawa, Japan

Abstract

We investigate the reconfiguration of n blocks, or “tokens”, in the square grid using *line pushes*. A line push is performed from one of the four cardinal directions and pushes all tokens that are maximum in that direction to the opposite direction by one unit. Tokens that are in the way of other tokens are displaced in the same direction, as well.

Similar models of manipulating objects using uniform external forces match the mechanics of existing games and puzzles, such as Mega Maze, 2048 and Labyrinth, and have also been investigated in the context of self-assembly, programmable matter and robotic motion planning. The problem of obtaining a given shape from a starting configuration is known to be NP-complete.

We show that, for every n , there are *sparse* initial configurations of n tokens (i.e., where no two tokens are in the same row or column) that can be compacted into any $a \times b$ box such that $ab = n$. However, only $1 \times k$, $2 \times k$ and 3×3 boxes are obtainable from any arbitrary sparse configuration with a matching number of tokens. We also study the problem of rearranging labeled tokens into a configuration of the same shape, but with permuted tokens. For every initial configuration of the tokens, we provide a complete characterization of what other configurations can be obtained by means of line pushes.

2012 ACM Subject Classification Mathematics of computing \rightarrow Permutations and combinations

Keywords and phrases Reconfiguration, Global Control, Pushing Blocks, Permutation

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.1

Related Version *Full Version*: <http://arXiv.org/abs/2202.12045>

Acknowledgements The authors would like to thank the anonymous reviewers for helpful suggestions that greatly improved the readability of this paper.

1 Introduction

Background. Manipulating a set of objects with uniform external forces is a concept that appears in many game and puzzle mechanics, such as Mega Maze [2], the 2048 puzzle [4, 6], Tilt [14] and dexterity games such as the Labyrinth marble game [1] and Pigs in Clover [3]. It also appears in self-assembly, programmable matter and robotic motion planning, with many applications involving controlling particles in the micro and nanoscale [12, 14, 10, 11]. Having the particles being controlled by a uniform external force is of particular interest since it might be unfeasible to control them individually due to their small scale. The most studied model for self-assembly using uniform external forces is the *tilt* model. Objects called “tokens” are in a 2D board where some locations are marked “blocked”. A tilt move moves all tokens maximally in one of the four cardinal directions, stopping the movement only if there is a collision with a blocked position or with another token [12]. Another model that resembles the dexterity games model uses a *single step* to control tokens via uniform signals to move a single unit in one of the cardinal directions [13, 15].



© Hugo A. Akitaya, Maarten Löffler, and Giovanni Viglietta;
licensed under Creative Commons License CC-BY 4.0

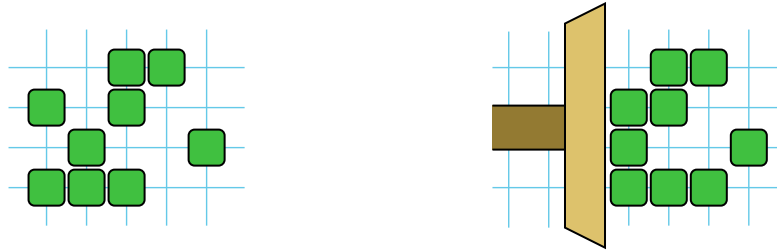
11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 1; pp. 1:1–1:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** A configuration of tokens in the square lattice (left), and the configuration after a \Leftrightarrow push (right). Tokens can be thought of as being pushed by a line coming from the left or, equivalently, as “falling” to the left without exiting their bounding box.

Akitaya et al. [5] introduced the *trash compaction* problem that displaces tokens in a square grid via a *line push*, or simply *push*. A push is also caused by an external force, but unlike the tilt, each token moves by at most one unit in the direction of the force, perhaps better approximating a dexterity game model. Informally, a push is applied in one of the four cardinal directions from which we sweep an axis-aligned line. The first tokens hit by the line are displaced by one unit; in turn, these tokens might displace other tokens, and so on. Figure 1 illustrates an example. If we consider configurations equivalent under translation, the same push operation can be seen as placing a line barrier on one of the sides of the bounding box and applying a uniform force pushing all tokens towards the barrier by at most one unit. Note that pushes are not necessarily reversible.

The trash compaction problem gives an initial configuration of 1×1 tokens in the square grid and asks whether it can be reconfigured via pushes into a rectangular box of given dimensions. It is shown in [5] that the problem is NP-complete in general, but it is polynomial-time solvable for $2 \times k$ rectangles, where k is an arbitrary constant.

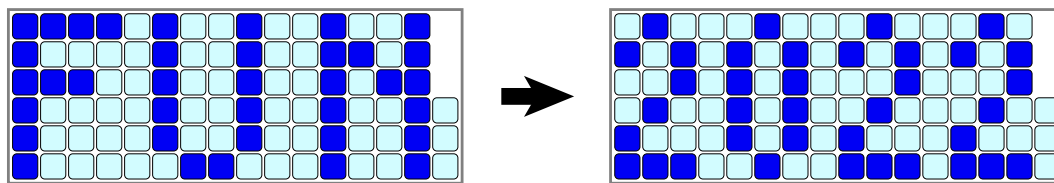
Our contributions. In this paper, we consider reconfiguration problems using pushes in two scenarios: *labeled* and *unlabeled*. Two tokens with the same label are indistinguishable; a configuration is unlabeled if all tokens have the same label, and labeled otherwise.

In Section 2 we make some preliminary observations, where we study certain important configurations called *compact* and the ways they can be reconfigured.

Next, we investigate two types of puzzles. The first is called *Compaction Puzzle*, and is equivalent to the trash compaction problem. In Section 3 we show that, for *sparse* configurations of unlabeled tokens (i.e., where no two tokens are in the same row or column), only rectangles of sizes $1 \times k$, $2 \times k$, and 3×3 can be obtained in general. That is, we give algorithms to push tokens into these rectangles, and we show that there are sparse configurations that cannot be pushed into rectangles of any other size.

The second puzzle is called *Permutation Puzzle*, and the goal is to reconfigure a labeled configuration into another; Figure 2 shows an example. In Section 4 we give a complete characterization of which labeled compact configurations can be obtained from one another. Namely, in Section 4.1 we prove that only even permutations of the tokens are possible, and in Section 4.2 we show exactly which even permutations can be obtained, depending on the initial configuration. Our characterization can be considered a (partial) universality result; that is, after ruling out cases using some easily identifiable necessary conditions, every pair of configurations can be reconfigured into each other (Theorem 20 gives a precise statement).

Section 5 concludes the paper. Due to space constraints, some figures and technical proofs have been omitted from this version of the paper, and can be found in [8].



■ **Figure 2** A Permutation Puzzle, where the goal is to rearrange tokens from the left configuration to the right configuration by means of line pushes. Due to Theorem 20, this puzzle is solvable.

Related work. There is a rich literature regarding reconfiguration through pushing objects, including sliding-block puzzles [20, 18], block-pushing puzzles [19, 21], the 15-puzzle [23], the 2048 puzzle [4, 6], etc. In the block-pushing model, a 1×1 *agent* moves through empty positions in the square lattice and can displace (push) objects that are *free*, while some objects are *blocked* and cannot be moved. The problem of whether the agent can reach a target position, depending on the model of the push move, is known to be PSPACE-complete. Note that this operation, albeit being called “push”, differs from our model since the agent can move specific individual objects.

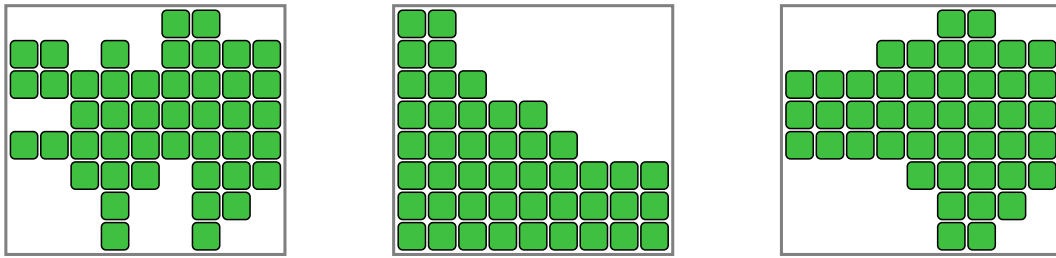
As previously discussed, models that consider movement of objects by uniform external forces have received some attention due to their applications to programmable matter. In the tilt model with 1×1 free and blocked objects, deciding whether a given free object can get to a given position is PSPACE-complete [10]. This implies the same complexity for the reconfiguration problem. Other papers considered some form of universality given a certain placement of blocked objects such as reconfiguration [12, 27], particle computations [14], and building shapes when given edges of free objects stick together when in contact [11, 10]. In the single-step model, for a given configuration of blocked objects, reconfiguration is NP-complete with single steps limited to two directions [16], and universality results also exist for a special configuration of blocked objects [17]. Note that the hardness of the trash compaction problem [5] implies the hardness of the problem of constructing a given shape in the single-step model.

Concerning our line-push model, in addition to the aforementioned paper [5] about trash compaction, a second short paper has appeared [7], introducing some 2-player games based on the same mechanic.

2 Definitions and Preliminaries

Let \mathcal{L} be the 2D square lattice. Let \mathcal{T} be a set of n labeled objects called *tokens*, and let Σ be the set of labels. A *configuration* of \mathcal{T} is an arrangement of \mathcal{T} in \mathcal{L} where no two tokens occupy the same position. Formally, a configuration is a function $C: \mathcal{L} \rightarrow \Sigma \cup \{\text{empty}\}$ where the cardinality of the preimage of an element $\ell \in \Sigma$ is the number of tokens labeled ℓ , and $|C^{-1}(\Sigma)| = |\mathcal{T}| = n$. A lattice position (x, y) is *full* if its image is in Σ , and *empty* otherwise. Notice that we do not distinguish two tokens that have the same label. However we may refer to full positions and tokens interchangeably for ease of reference. The *bounding box* of C is the minimum rectangular subset of \mathcal{L} containing all full positions. In the following, we will identify configurations that are equal under translation.

A *push* is an operation that takes a configuration C and a direction $d \in \{\Rightarrow, \Leftarrow, \Downarrow, \Uparrow\}$ and returns a configuration C' as follows. We describe a \Rightarrow push; the other cases are symmetric. Informally, a \Rightarrow push moves all leftmost tokens one unit to the right, further displacing tokens to the right if there are collisions. Without loss of generality, assume that the lower-left



■ **Figure 3** An incompressible non-compact configuration (left); a canonical configuration (center); a compact configuration (right). The central and right configurations are compatible.

corner of the bounding box is $(0, 0)$, applying the appropriate translation otherwise. For all columns $i \geq 1$ from left to right, and for all rows j , if (i, j) is full and $(i - 1, j)$ is empty, move the token from (i, j) to $(i - 1, j)$. Finally, translate the configuration making the lower-left corner of the bounding box $(1, 0)$.

Note that the last step just translates the configuration, producing an equivalent one. Then, we can also consider an alternative informal interpretation of a push: \Leftrightarrow push places a vertical barrier to the left of $x = 0$ and lets all tokens that can move “fall” towards the left by one unit. We call this interpretation the “gravity” formulation of the puzzle.

Observe that pushes are not necessarily reversible, and every push either does not affect the size of the bounding box, or decreases its area by exactly one row or column. We denote a sequence of pushes by the respective sequence of directions $\langle d_1 \dots d_m \rangle$. We use the notation d^k to express k repetitions of direction d . A configuration is *compressible* if a push can decrease the area of its bounding box, or *incompressible* otherwise. By definition, it is easy to see that a configuration is incompressible if and only if its bounding box contains a full row and a full column. If $|\Sigma| = 1$, i.e., all tokens have the same label, we call the configuration *unlabeled*. When no restrictions are made on $|\Sigma|$, we say that the configuration is *labeled*.

We can now define our Compaction Puzzle.

► **Problem 1 (Compaction Puzzle).** *Given an unlabeled starting configuration, is there a sequence of pushes that produces a configuration whose tokens form an $a \times b$ box?*

If we start from any configuration and we keep pushing in the two directions \Leftrightarrow and \Uparrow alternately, we eventually reach a configuration for which any push in these two directions produces an equivalent configuration. We call such a configuration *canonical*. By definition, a canonical configuration forms an orthogonally convex polyomino with its leftmost column and bottommost row full. A *compact* configuration is a configuration that can be obtained from a canonical configuration by pushes (see Figure 3). Note that all canonical configurations are also compact.

► **Observation 1.** *Configuration C is compact if and only if each row r (resp., column c) has a contiguous interval of tokens and its projection on any other row r' (resp., column c') with the same amount or more tokens is contained in the interval of tokens of r' (resp., c').*

Proof. Let \mathcal{P} denote the conditions after the “if and only if”. Observe that all canonical configurations satisfy \mathcal{P} . Also, \mathcal{P} is easily seen to be preserved by pushes, and therefore all compact configurations satisfy \mathcal{P} , as well.

In order to prove the converse, we first show that if C satisfies \mathcal{P} , then any push performed in C is reversible. The remainder of the proof follows from the fact that we can obtain a canonical configuration by applying \Leftrightarrow and \Uparrow pushes.

Without loss of generality, assume a \Rightarrow push in C and that the lower-left corner of the bounding box is $(0, 0)$. Let H be the set of rows with a token in the leftmost column $(0, \cdot)$. Let V be the set of columns to the right of $(0, \cdot)$ that contain a token in each row in H . Observe that each token in C must be in H or in V , or else there would be a column whose interval of tokens' projection neither contains nor is contained in the interval of tokens of the first column $(0, \cdot)$. Note that applying the \Rightarrow push causes only tokens in H to move. We reverse this operation by applying $\langle \Leftarrow^k \rangle$, with k chosen so that the right side of the bounding box coincides with the rightmost column in V , then applying $\langle \Rightarrow^{k-1} \rangle$. By construction, only tokens in H move, and the sequence brings the leftmost token in each row in H back to its original position in column $(0, \cdot)$. This proof works for the labeled as well as the unlabeled case. \blacktriangleleft

The proof of Observation 1 immediately implies the following corollary.

► **Corollary 2.** *A push applied to a (labeled) compact configuration is reversible.* \blacktriangleright

Note that, by Observation 1, compact configurations are orthogonally convex polyominoes; also, all compact configurations are incompressible. Moreover, a random sequence of pushes results in a compact configuration with high probability (this certainly happens, for instance, if the sequence of pushes has a subsequence of the form $\langle \hat{\uparrow}^k \Rightarrow^k \rangle$ for a large-enough k).

The number of tokens in a row or a column of a compact configuration is said to be the *length* of that row or column. Two compact configurations are *compatible* if they have the same number of rows (resp., columns) of each given length. For example, in Figure 3, the central and right configurations are compatible. The observation below shows that the set of compatible compact configurations is closed under pushes.

► **Observation 3.** *A push applied to a compact configuration C results in a compatible compact configuration C' .*

Proof. Since C can be obtained from a canonical configuration by pushes, then so can C' , and therefore C' is compact, as well. It is enough to show that a \Rightarrow push in C does not change the number of rows (resp., columns) of each given length. Note that labels are irrelevant, and so we will assume C and C' to be unlabeled, without loss of generality. We reuse the notation of the proof of Observation 1. The \Rightarrow push only moves tokens in H , so it is clear that the lengths of rows do not change. We can see the push as removing the leftmost column (which has length $|H|$), moving all columns to the right of V by one unit to the right, and creating a new column of length $|H|$ immediately to the right of V . Thus, the number of columns of each length remains the same, and C' is compatible with C . \blacktriangleleft

Note that compatibility is an equivalence relation on the set of compact configurations. By Observation 3, in any equivalence class of unlabeled compatible compact configurations there is exactly one canonical configuration. Once we reach a compact configuration via a sequence of pushes, no other canonical configuration can be reached, except for the one corresponding to the current compact configuration.

The following observation follows from the fact that we can obtain a canonical configuration by applying $\hat{\uparrow}$ and \Rightarrow pushes and by Corollary 2.

► **Observation 4.** *Any two unlabeled compatible compact configurations can be obtained from each other by pushes.* \blacktriangleright

Two labeled configurations are said to have the *same shape* if they have the same set of full positions (recall that we are identifying configurations that only differ by a translation). We can now define our Permutation Puzzle.

► **Problem 2** (Permutation Puzzle). *Given two same-shaped labeled compact configurations C and C' , is there a sequence of pushes that transforms C into C' ?*

Two same-shaped labeled configurations differ by a permutation of their tokens. If we obtain a same-shaped configuration C' from C , the sequence of pushes results in such a permutation. Note that the set of possible permutations is *finite* and closed under composition, and therefore is a *permutation group*, which we denote as G_C .

The following observation allows us to focus on sequences between canonical configurations. We say that the labeled canonical configuration C obtained from a compact configuration C' by $\langle \hat{\uparrow}^{k_1} \Rightarrow^{k_2} \rangle$, for some k_1, k_2 , is the *canonical form* of C' . Then the following is a direct consequence of Corollary 2.

► **Observation 5.** *A labeled compact configuration C can be obtained from a same-shaped labeled compact configuration C' if and only if the canonical form of C is reachable from the canonical form of C' .* ◻

3 Compaction Puzzles

In this section, we focus on Compaction Puzzles (Problem 1) where the input configuration is *sparse*; that is, where no row or column has more than one token. We assume that the number of tokens is $n = ab$, and we wonder if there is a sequence of pushes that produces an $a \times b$ box. Note that, as long as the configuration is sparse, any push shrinks the bounding box, and therefore is irreversible (unless $n = 1$). Recall that without the sparseness assumption, the problem is NP-complete [5].

One may wonder if the leeway given by sparse configurations is enough to obtain every compact configuration; in this section, we will show that this is not the case.

We begin by observing that, for every n , there exists a “universal” configuration that can be reconfigured into any compact configuration with n tokens (such as an $a \times b$ box).

► **Observation 6.** *There exists a sparse unlabeled configuration with n tokens that can be reconfigured into any compact configuration with n tokens.*

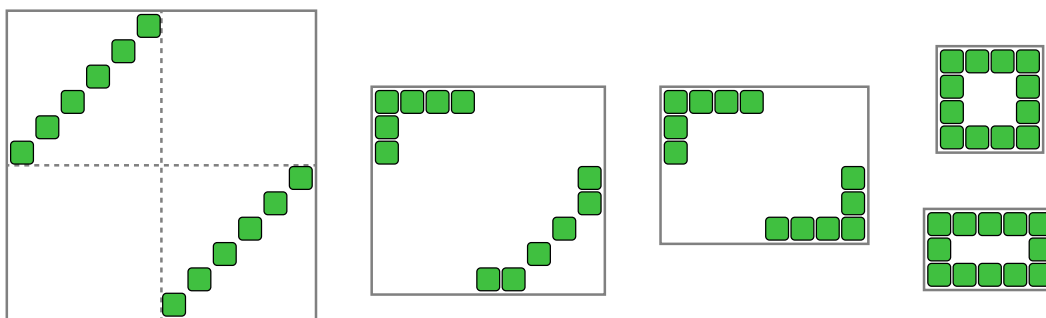
Proof. Such a configuration is the secondary diagonal of an $n \times n$ matrix, i.e., all positions (i, i) are full, for $i \in \{1, \dots, n\}$, and all other positions are empty. Due to Observation 4, it suffices to show that any canonical configuration can be formed.

By definition, a canonical configuration has a monotonic decreasing sequence of column lengths. After $k \Rightarrow$ pushes, the configuration will have a column of length k . We then perform $k \hat{\uparrow}$ pushes bringing the column up aligning its bottom with the next token. Any subsequent $k' \leq k \Rightarrow$ pushes would accumulate tokens in the bottommost contiguous positions of the second leftmost column. We can then repeat this procedure to produce any monotonic decreasing sequence of column lengths. ◀

We will now show that not all configurations can be reconfigured into an $a \times b$ box.

► **Lemma 7.** *For all $a \geq 4$ and $b \geq 3$, there exists a sparse configuration that cannot be reconfigured into an $a \times b$ box.*

Proof. We will describe a sparse configuration $C_{a,b}$ and argue that it cannot be reconfigured into an $a \times b$ box; refer to Figure 4. The configuration $C_{a,b}$ is subdivided into four quadrants; the lower-left and upper-right quadrants are empty, all tokens are in the other two quadrants. We place $n_1 = \lfloor \frac{ab}{2} \rfloor$ tokens in the upper-left quadrant and $n_2 = \lceil \frac{ab}{2} \rceil$ in the lower-right quadrant; each of those quadrants is a square matrix with only its secondary diagonal full.



■ **Figure 4** The configuration $C_{a,b}$ defined in Lemma 7, with $a = 4$ and $b = 3$ (left). Any sequence of pushes produces an “L-shaped” pattern (center), which eventually causes the formation of rows with more than a tokens or columns with more than b tokens (right).

Initially, all the \Rightarrow and \Downarrow pushes (resp., \Leftarrow and \Uparrow pushes) only affect tokens in the upper-left (resp., lower-right) quadrant. Thus, as soon as either $n_1 - 1$ pushes have been made in the upper-left quadrant or $n_2 - 1$ pushes have been made in the lower-right quadrant (whichever comes sooner), all the tokens in that quadrant must be located on the union between a single row and a single column, forming a connected “L-shaped” pattern.

If the L-shaped pattern contains x tokens on the same row and y tokens on the same column, we must have $x + y = n_i + 1$ for some $i \in \{1, 2\}$. Thus, $x + y \geq \lfloor \frac{ab}{2} \rfloor + 1$. It is easy to see (cf. [5, Observation 3]) that it is impossible to form an $a \times b$ box from a configuration where more than a (resp., b) tokens are on the same row (resp., column). Therefore, we must have $x \leq a$ and $y \leq b$, which implies $a + b \geq \lfloor \frac{ab}{2} \rfloor + 1 \geq \frac{ab+1}{2}$. By rearranging terms we have $(a-2)(b-2) \leq 3$; the only solutions with $a \geq 4$ and $b \geq 3$ are $a = 4, b = 3$ and $a = 5, b = 3$.

Let $a = 4$ and $b = 3$ (resp., $a = 5$ and $b = 3$), and let x and y be defined as above. Since $x + y \geq 7$ (resp., $x + y \geq 8$), we must have $x = a$ and $y = b$. Now, further pushes in the same quadrant cause the L-shaped pattern to rigidly move toward the opposite quadrant, eventually creating a row with more than a tokens or a column with more than b tokens. On the other hand, making more pushes in the opposite quadrant ends up forming another, oppositely oriented, L-shaped pattern identical to the first. At this point, any push causes the two L-shaped patterns to rigidly approach each other, eventually creating a row or a column with too many tokens. ◀

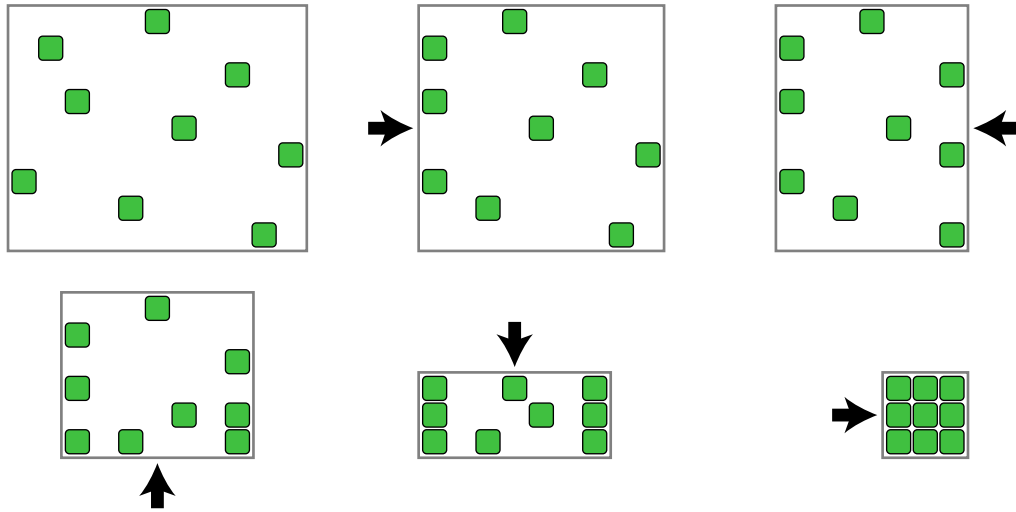
In all other cases, any sparse configuration can be reconfigured into an $a \times b$ box, which yields the following theorem.

► **Theorem 8.** *All sparse configurations of $n = ab$ tokens can be pushed into an $a \times b$ box if and only if $a \leq 2$ or $b \leq 2$ or $a = b = 3$.*

Proof. Lemma 7 provides counterexamples for $a \geq 4$ and $b \geq 3$ (and, symmetrically, for $a \geq 3$ and $b \geq 4$); it remains to show the positive cases.

First, note that the claim for $a = 1$ is trivial (and $b = 1$ is symmetric): Perform \Uparrow pushes until all tokens are in the same row, then perform \Rightarrow pushes until they occupy consecutive positions.

Now assume $a = 2$ ($b = 2$ is symmetric). Perform \Downarrow pushes until half of the tokens are in the same row r . Then, the other half form a sparse subconfiguration below r . Perform \Uparrow pushes until all such tokens are in the row immediately below r . Then, a $2 \times \frac{n}{2}$ box is obtainable by performing \Rightarrow pushes.



■ **Figure 5** Pushing a sparse configuration of $n = 9$ tokens into a 3×3 box (Theorem 8).

Finally, assume that $a = b = 3$; refer to Figure 5. Because the initial configuration is sparse, if we perform \Rightarrow pushes, the number of tokens in the leftmost column increases by at most one with every push. Apply \Rightarrow pushes until there are three tokens in the leftmost column. The subconfiguration obtained by deleting the leftmost column is also sparse. Then, by the same argument we can perform \Leftarrow pushes until there are three tokens in the rightmost column. Since we have not yet performed any vertical push, every row has at most one token. There are exactly three tokens that are not in the leftmost or rightmost columns. Let t_1 , t_2 and t_3 , respectively, be those tokens, from smallest to largest y -coordinate. Perform \Uparrow pushes until the bottommost row of the configuration is one unit below t_2 . Note that t_1 must be in the bottommost row. Symmetrically, apply \Downarrow pushes until the topmost row of the configuration is one unit above t_2 . Now, t_1 , t_2 and t_3 are strictly between the leftmost and rightmost columns, and are one in each row. Then, we obtain a 3×3 box by performing \Rightarrow pushes. ◀

4 Permutation Puzzles

In this section, we will give a complete solution to the Permutation Puzzle (Problem 2): For any given starting compact labeled configuration C , we will determine the set of labeled configurations of the same shape as C that we can reach by means of pushes. Specifically, we will give a description of the permutation group G_C , as defined in Section 2.

By Observation 5, it is not restrictive to limit our attention to canonical configurations. For most of this section, we will assume that all tokens have distinct labels; the general case will be discussed at the end of the section.

We will show that all feasible permutations in this setting are even (Section 4.1), and furthermore that all even permutations are feasible, with some simple restrictions (Section 4.2).

It will be convenient to always consider the lower-left corner of the bounding box to be $(0, 0)$, even after a sequence of pushes. That is, we consider the “gravity” formulation of the puzzle, where the bounding box remains still and the tokens move within it (recall that compact configurations are uncompressible). As a consequence, a push in one direction will cause the tokens to “fall” in the opposite direction within the bounding box.



■ **Figure 6** A configuration with labeled empty cells and the result after the sequence $\langle \leftarrow \leftarrow \leftarrow \downarrow \rangle$. We use yellow for full tokens and light blue for empty cells (and later, dual tokens) in this section.

4.1 All Feasible Permutations Are Even

In this section, we will prove Theorem 11, which states that only *even* permutations are possible in the Permutation Puzzle.

For a labeled canonical configuration C , let C' be an extension of the labeling where also the empty cells inside the bounding box of C get a unique label. Our proof strategy is to first extend permutations of C to permutation of C' , and argue that a permutation of full and empty cells must be even. We then introduce a *dual game* played on the empty cells only, and argue that this dual game has similar properties. Since the dual of any game is always smaller (in terms of bounding box) than the original, our theorem then follows by induction.

Permutations on Full Cells and Empty Cells

Let C be a labeled canonical configuration; that is, a function $C: \mathcal{L} \rightarrow \Sigma \cup \{\text{empty}\}$. We extend C to $C': \mathcal{L} \rightarrow \Sigma \cup \Sigma' \cup \{\text{empty}\}$, where Σ' is a second set of unique labels, $C'(x) \in \Sigma'$ if and only if x is in the bounding box of C , but not in C , and for all other x , $C'(x) = C(x)$.

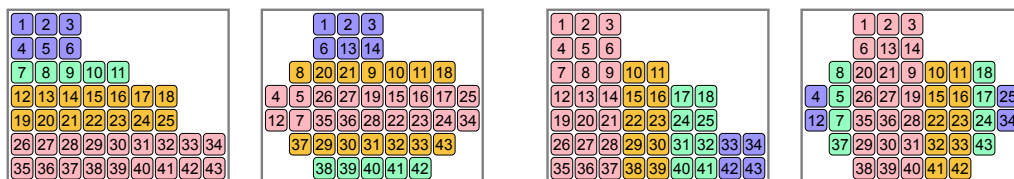
We now define the effect of a push operation on C' (illustrated in Figure 6). We define it for a \leftarrow push; the other directions are symmetric. A \leftarrow push affects each row as follows:

- For each row in which the rightmost cell is *empty*, we shift all tokens and empty cells one position to the right and place the rightmost empty cell at the left; in other words, we perform a single cyclic permutation on the tokens in the row.
- For each row in which the rightmost cell is *full*, nothing changes.

We now argue that for any sequence of pushes which transforms C into another canonical configuration, the effect of these moves on C' must be an even permutation.

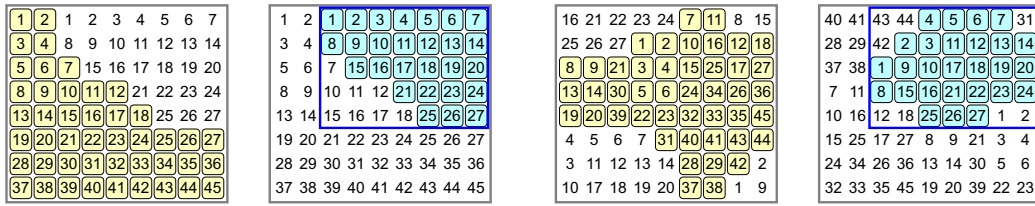
► **Lemma 9.** *Every achievable permutation on both the full and empty cells must be even.*

Proof. By definition, every horizontal (resp., vertical) push causes a cyclic permutation on some rows (resp., on some columns) involving both labeled tokens and labeled empty cells. We will argue that the total number of cyclic permutations on the rows (resp., on the columns) caused by these pushes is even. Since all cyclic permutations on rows (resp., on columns) have the same parity, which depends only on the width (resp., height) of the bounding box, this implies that the overall permutation is even.

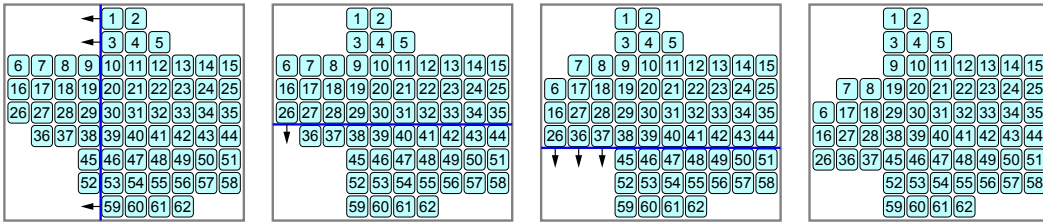


■ **Figure 7** Rows and columns of the same length are always aligned.

1:10 Pushing Blocks by Sweeping Lines



■ **Figure 8** A primal puzzle (yellow) with its corresponding dual puzzle (light blue). The blue rectangle represents the bounding box of the dual puzzle. Note that the tokens in the dual puzzle match the labels on the empty cells of the primal puzzle.



■ **Figure 9** Rules of the dual puzzle. The sequence of pulls $\langle \leftarrow \downarrow \rightarrow \uparrow \rangle$ is shown.

From Observations 1 and 3 it follows that all rows (resp., columns) of the same length must always be aligned throughout the reconfiguration (see Figure 7). In particular, observe that a *vertical* push never influences the *horizontal* placement of the rows of a particular length (note that here we only argue about the shape, not the labels). Now consider the set of rows of length k . Every horizontal push either moves all such rows one cell to the right, or one cell to the left, or not at all. Since both at the start and at the end of the process all rows of length k are aligned with the left border of the bounding box, the total number of pushes that influence the horizontal placement of these rows is even, and thus the number of cyclic permutations performed on these rows is also even.

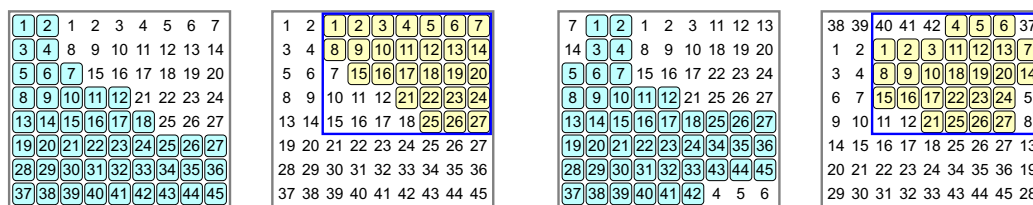
Note that a single push may influence the placement of rows of different lengths; however, for each specific length k , the total number of pushes that influences them is even, and therefore the total number of cyclic permutations on rows is even. The same holds for the cyclic permutations on columns, which concludes the proof. ◀

Now, in order to prove our main theorem, we still need to show that the resulting permutation *restricted to the tokens* is also even.

Dual Puzzles

Given a configuration C , we have extended its labeling to a configuration C' having labels on the empty cells, as well. Now, consider the *restriction* of C' to *only* the empty cells; that is, the function $D: \mathcal{L} \rightarrow \Sigma' \cup \{\text{empty}\}$ which labels exactly the cells that are empty in C but lie inside the bounding box of C . Clearly, if we can prove that a permutation on D is even, then Lemma 9 implies that the corresponding permutation on C must also be even (the product of two permutations is even if and only if they have the same parity).

For convenience, we will consider the bounding box of C as a torus and display it in such a way that all full rows and columns are aligned with the left and bottom of the rectangle (see Figure 8). We call D a *dual configuration*, and we will study the effects of push moves on D , which we will treat as a *dual puzzle* (while the original puzzle on C is the *primal puzzle*).



■ **Figure 10** The dual of a dual puzzle is again a primal puzzle; we see the result after $\langle \Rightarrow \Uparrow \rangle$.

When considering a dual puzzle in isolation, we swap terminology and refer to the empty cells as full, and vice versa. In this context, a *push* move in the primal puzzle either leaves the dual puzzle unchanged or causes a *pull* move in the dual puzzle. Specifically, if we perform a \Leftarrow push move in the primal puzzle from a configuration where only the full rows are touching the right side of the bounding box, nothing happens in the dual puzzle. Otherwise, the \Leftarrow push move in the primal puzzle causes a \Leftarrow pull move in the dual puzzle, where exactly those rows that are farthest from the left boundary are pulled one unit to the left (refer to Figure 9, where the blue line represents a side of the bounding box of the primal puzzle).

Thus, as we play the primal puzzle, we are also playing the dual puzzle. We say that a *dual* configuration is *canonical* when all tokens are aligned with the top and right borders of their bounding box; this way, the empty cells in a canonical (primal) configuration form a canonical (dual) configuration. Most of the results obtained so far for primal puzzles automatically apply to dual puzzles, as well.

In particular, we claim that in the dual game, the equivalent of Lemma 9 still holds. For this, we now consider again an extension of our dual configuration D which labels both the full and empty cells of the bounding box of D . Crucially, this is *not* the same as the original extension C' , because the bounding box of D is smaller than the bounding box of C .

► **Lemma 10.** *Let D be a labeled canonical configuration, and let D' be an extension of D that labels also the empty cells of the bounding box of D' . Every achievable permutation on D' under a sequence of dual moves must be even.*

Proof. Since the number of rows (resp., columns) of any given length in the primal puzzle remains constant, the same is true in the dual puzzle. Also, all of the rows (resp., columns) of the same length must always be aligned in the primal puzzle, and therefore they must be aligned in the dual puzzle, as well. The proof now proceeds exactly as in Lemma 9. ◀

Induction

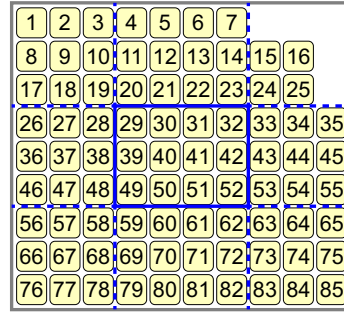
We are now ready to prove that all permutations in G_C must be even.

► **Theorem 11.** *Let C be a labeled canonical configuration, and let $\pi \in G_C$. Then, π is an even permutation.*

Proof. The proof follows from two simple observations. Firstly, if we take the dual of a dual-type puzzle, we obtain another puzzle that again follows the rules of a primal-type puzzle (refer to Figure 10). Secondly, the bounding box of a (primal or dual) puzzle is strictly larger than the bounding box of its dual.

We will prove a stronger statement: that our theorem holds for both primal-type and dual-type puzzles. The proof is by well-founded induction on the size of the bounding box.

If the bounding box of our puzzle is completely full, then moves have no effect, and the only allowed permutation is the identity, which is even.



■ **Figure 11** A configuration with $a = 10$, $b = 9$, $a' = 7$, and $b' = 6$. The blue rectangle in the center surrounds the core tokens.

Now, consider a canonical configuration C in a (primal or dual) puzzle with a bounding box which is not completely full. Such a puzzle has a dual, with a canonical configuration D corresponding to C . The bounding box of D has smaller size, and therefore the induction hypothesis applies to the dual puzzle. After performing some moves and restoring a canonical configuration in both puzzles, the tokens in C have undergone a permutation $\pi \in G_C$, while the tokens in D have undergone a permutation $\sigma \in G_D$. By Lemmas 9 and 10, the overall permutation $\pi\sigma$ is even; by the induction hypothesis, σ is even; hence, π is even, as well. ◀

4.2 Generating All Feasible Permutations

In this section, we will give a complete description of the permutation group G_C , which we already know from Section 4.1 to be a subgroup of the *alternating group* $\text{Alt}(n)$.

Unmovable Central Core

Let the bounding box be an $a \times b$ rectangle. Our first observation is that, if more than half of the rows and more than half of the columns of the bounding box are full, then there is a central box of tokens that cannot be moved. Let a' (resp., b') be the number of full columns (resp., full rows), and let $a'' = a - a'$ and $b'' = b - b'$.

► **Definition 12 (Core).** *If $a' > a''$ and $b' > b''$, the core of C is the set of lattice points in the bounding box of C that are in the central $a' - a''$ columns and in the central $b' - b''$ rows.¹ If $a' \leq a''$ or $b' \leq b''$, the core of C is empty.*

The lattice points in the core are called *core points*, and the tokens in core points are called *core tokens*. Figure 11 shows an example of a non-empty core.

► **Observation 13.** *No permutation in G_C moves any core token.*

Proof. If the core is empty, there is nothing to prove; hence, let us assume that $a' > a''$ and $b' > b''$. From Section 2, we know that there are always exactly a' contiguous full columns and b' contiguous full rows, no matter how the tokens are pushed. Hence, the central $a - 2a'' = a' - a''$ columns (resp., the central $b - 2b'' = b' - b''$ rows) are always full, and are not affected by \Downarrow or \Uparrow pushes (resp., \Rightarrow or \Leftarrow pushes). Therefore, no push can affect the core tokens. ◀

¹ Equivalently, if $a > 2a''$ and $b > 2b''$, the core is obtained by discarding the a'' leftmost columns, the a'' rightmost columns, the b'' topmost rows, and the b'' bottommost rows.

Permutation Groups

In order to understand the structure of G_C , we will review some notions of group theory and prove three technical lemmas.

► **Definition 14.** A permutation group G on $\{1, 2, \dots, n\}$ is 2-transitive if, for every $1 \leq x, y, w, z \leq n$ with $x \neq y$ and $w \neq z$, there is a permutation $\pi \in G$ such that $\pi(x) = w$ and $\pi(y) = z$.

► **Theorem 15** (Jones, [22]). If G is a 2-transitive permutation group on $\{1, 2, \dots, n\}$ and G contains a cycle of length $n - 3$ or less, then G contains all even permutations.² ◻

To express cyclic permutations, we use the standard notation $\sigma = (s_1 s_2 s_3 \dots s_k)$, occasionally adding commas between terms when doing so improves readability. The cycle σ is the permutation that fixes all items except s_1, s_2, \dots, s_k such that $\sigma(s_1) = s_2, \sigma(s_2) = s_3, \dots, \sigma(s_k) = s_1$. Since we are studying permutations puzzles, it is visually more convenient to interpret a permutation as acting on places rather than on items. Thus, for example, $(1\ 2\ 3)$ is understood as the cycle involving the tokens occupying the *locations* labeled 1, 2, and 3 rather than the *tokens* labeled 1, 2, and 3. Also, we will follow the convention to compose chains of permutations from left to right, which is the common one in permutation theory.

The following three lemmas have simple but technical proofs, which can be found in [8].

► **Lemma 16.** Let $\alpha = (1, 2, \dots, a)$ and $\beta = (a - b + 1, a - b + 2, \dots, 2a - b)$ be two cycles spanning $n = 2a - b$ items, with $a \geq 2$ and $1 \leq b < a$. Then, the permutation group generated by α and β acts 2-transitively on $\{1, 2, \dots, n\}$. ◻

► **Lemma 17.** Let $\alpha = (1, \dots, 2a + b + 1, 3a + b + 2, \dots, 3a + 2b + 1)$ and $\beta = (a + 1, \dots, a + b, 2a + b + 2, \dots, 4a + 2b + 2)$ be two cycles spanning $n = 4a + 2b + 2$ items, with $a \geq 0$ and $b \geq 1$. Then, the permutation group generated by α and β acts 2-transitively on $\{1, 2, \dots, n\}$. ◻

► **Lemma 18.** Let A and B be two finite sets such that $A \cap B \neq \emptyset$ and $A \setminus B \neq \emptyset$. Let G be a permutation group on $A \cup B$ whose restriction to A is 2-transitive and whose restriction to $B \setminus A$ is trivial, and let β be a cycle spanning B . Then, the permutation group generated by G and β acts 2-transitively on $A \cup B$. ◻

Generating Cycles

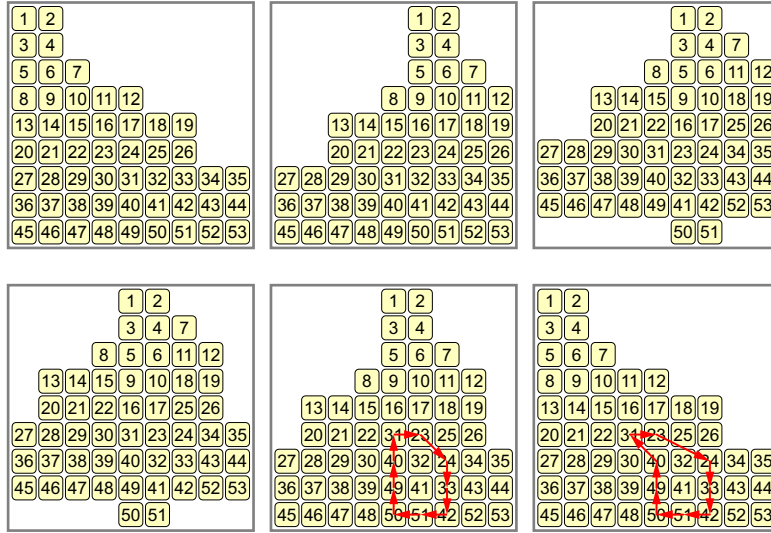
We will now assume that all labels are distinct. Specifically, the n tokens in C are labeled 1 to n from left to right and from top to bottom, as in Figure 11. The general case will be discussed later.

We introduce three types of sequences of moves:

- Type-A k -sequence: $\langle \leftarrow^k \rightleftarrows \downarrow \rightleftarrows \uparrow \rightarrow^k \rangle$ for $0 \leq k < a''$.
- Type-B k -sequence: $\langle \downarrow^k \downarrow \leftleftarrows \uparrow \rightleftarrows \uparrow^k \rangle$ for $0 \leq k < b''$.
- Type-C k -sequence: $\langle \leftarrow^k \downarrow \rightleftarrows \uparrow \rightleftarrows \rightarrow^k \rangle$ for $0 \leq k < a''$.

It is straightforward to check that a type-A k -sequence always produces a cycle of length $2a' + 2b' - 1$ (refer to Figure 12), which involves the lattice points (k, i) and $(a' + k, i)$ for all $0 \leq i < b'$ (plus some other points in the rows $(0, \cdot)$ and (b', \cdot)). Such cycles are called *type-A cycles*.

² Jones' theorem in [22] holds more generally for *primitive* permutation groups. The fact that all 2-transitive permutation groups are primitive is an easy observation.



■ **Figure 12** Performing a type-A k -sequence: $\langle \leftarrow^{k+1} \Downarrow \Rightarrow \Uparrow \Rightarrow^k \rangle$. Recall that a push in a direction causes the tokens to “fall” in the opposite direction within the bounding box.

Symmetrically, a type-B k -sequence produces a *type-B cycle* of length $2a' + 2b' - 1$ involving (among others) the lattice points (i, k) and $(i, b' + k)$ for all $0 \leq i < a'$. Thus, the type-A and the type-B cycles collectively cover all the non-core points that lie in one of the a' full columns or in one of the b' full rows.

A type-C k -sequence produces a cycle, as well, which we call *type-C cycle* (see Figure 13). However, unlike previous cycles, a type-C cycle’s length may vary depending on k . It is easy to see that, if the row (\cdot, i) , with $0 \leq i < b$, has length ℓ_i , then the cycle produced by a type-C k -sequence with $a - \ell_i \leq k < a''$ involves (among others) the lattice point $(\ell_i + k - a'', i)$. Such a point lies at distance $a'' - k - 1$ from the rightmost full token in the row. Thus, the type-C cycles collectively cover all the tokens that are not in a full column. We conclude that type-A, type-B, and type-C tokens collectively cover all non-core points.

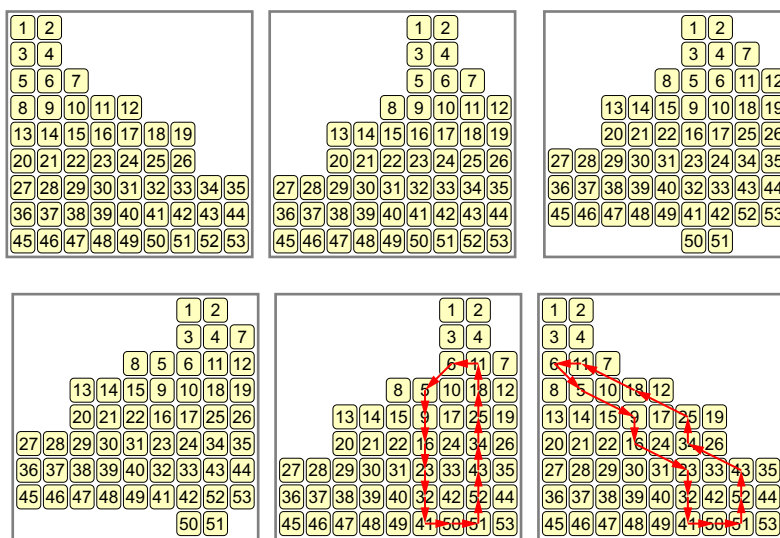
The next theorem states that, in most cases, the group G_C is exactly the alternating group on the non-core tokens, i.e., the group of all even permutation on the $n - (a' - a'')(b' - b'')$ tokens not in the core (or on all n tokens if the core is empty).

► **Theorem 19.** *If $n/2 \geq a' + b' + 1$ and $a''b'' > 1$, then G_C is the alternating group on the non-core tokens.*

Proof. We know from Theorem 11 that G_C only contains even permutations; also, by Observation 13, no permutation in G_C can move any core tokens. Hence, it suffices to show that G_C contains *all* even permutations of the non-core tokens. By applying a reflection to C if necessary, we may assume that $a'' \geq b''$. Also, since $a''b'' > 1$, we have $a'' \geq 2$, and therefore there are at least two distinct type-A cycles.

Let α and β be the type-A cycles for $k = 0$ and $k = 1$, respectively. Observe that, if $a' = 1$, then α and (the inverse of) β satisfy the hypotheses of Lemma 16; if $a' > 1$, then α and β satisfy the hypotheses of Lemma 17. In both cases, the group $G_1 \leq G_C$ generated by α and β acts 2-transitively on the points spanned by α and β and acts trivially on all other points.

The group G_1 , together with the type-A cycle for $k = 2$, satisfies the assumptions of Lemma 18. Therefore, G_C has a subgroup G_2 that acts 2-transitively on the points spanned by the first three type-A cycles and acts trivially on all other points. By repeatedly applying



■ **Figure 13** Performing a type-C k -sequence: $\langle \leftarrow^k \downarrow \leftarrow \uparrow \rightarrow^{k+1} \rangle$.

Lemma 18 to all remaining type-A cycles, we conclude that G_C has a subgroup that acts 2-transitively on a set that includes all the non-core points in the b' full rows of C .

By applying Lemma 18 again to the type-B cycles (the first of which properly intersects all of the full rows), we obtain a subgroup of G_C that acts 2-transitively on a set that includes all the non-core points in the a' full columns and in the b' full rows. Finally, we can apply Lemma 18 to the type-C cycles (all of which properly intersect the full rows) to obtain a subgroup of G_C that acts 2-transitively on all non-core tokens. This implies that G_C itself acts 2-transitively on all non-core tokens, as well.

To conclude the proof, we recall that each type-A cycle has length $2a' + 2b' - 1$. Since $n/2 \geq a' + b' + 1$, these cycles have length at most $n - 3$. Thus, G_C contains all even permutations of the non-core tokens, due to Theorem 15. ◀

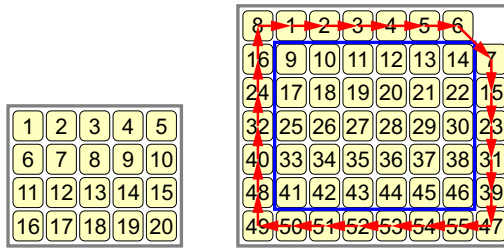
Special Configurations

We will now discuss all configurations of the Permutation Puzzle not covered by Theorem 19.

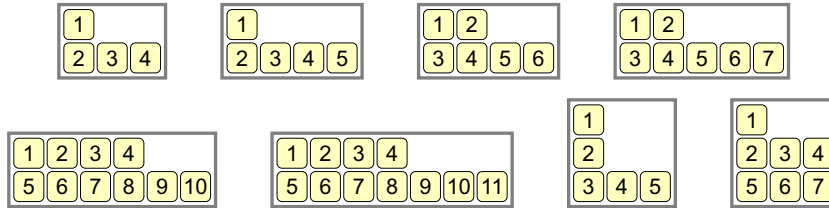
- If $a'' = b'' = 0$, i.e., there are no empty cells in the bounding box, then clearly no token can be moved, and G_C is the trivial permutation group (Figure 14, left).
- Let $a'' = b'' = 1$, i.e., there is exactly one empty cell, located at the top-right corner of the bounding box. The core includes all the tokens, except the ones on the perimeter of the bounding box, which are spanned by the type-A cycle with $k = 0$. It is easy to see that the only possible permutations are iterations of this cycle and its inverse. Therefore, G_C is isomorphic to the cyclic group $C_{2a+2b-5}$ (Figure 14, right).

In the following, we will assume that $a'' \geq b''$ and $a'' \geq 2$, and we discuss all configurations where $n/2 < a' + b' + 1$. We invoke some theorems from [26] about *cyclic shift puzzles*.

- Let $b = 2$, and let the two rows have length 1 and 3, respectively (Figure 15, top row, first image). The two type-A cycles $(1\ 2\ 3)$ and $(1\ 3\ 4)$ form a 2-connected $(3, 3)$ -puzzle involving all tokens, and therefore $G_C = \text{Alt}(n)$, due to [26, Theorem 2].



■ **Figure 14** Two configurations with $a''b'' \leq 1$.



■ **Figure 15** Some special configurations with $b = 2$ and $b = 3$ rows.

- Let $b = 2$, and let the two rows have length 1 and 4, respectively (Figure 15, top row, second image). The first and third type-A cycles $(1\ 2\ 3)$ and $(1\ 4\ 5)$ form a 1-connected $(3, 3)$ -puzzle involving all tokens, and therefore $G_C = \text{Alt}(n)$, due to [26, Theorem 1].
- Let $b = 2$, and let the two rows have length 2 and 4, respectively (Figure 15, top row, third image). We denote the two type-A cycles by $\alpha = (1\ 3\ 4\ 5\ 2)$ and $\beta = (1\ 4\ 5\ 6\ 2)$. It is easy to see that G_C is generated by α and β , because any non-trivial sequence of four pushes necessarily goes through a configuration whose canonical form yields one the permutations α , β , α^{-1} , or β^{-1} .

In order to determine G_C , we transform α and β by a suitable outer automorphism $\psi: \text{Sym}(6) \rightarrow \text{Sym}(6)$. Since ψ is an automorphism, the group G'_C generated by $\psi(\alpha)$ and $\psi(\beta)$ is isomorphic to G_C . The automorphism ψ is defined on a set of generators of $\text{Sym}(6)$ as follows (cf. [25, Corollary 7.13]): $\psi((1\ 2)) = (1\ 5)(2\ 3)(4\ 6)$, $\psi((1\ 3)) = (1\ 4)(2\ 6)(3\ 5)$, $\psi((1\ 4)) = (1\ 3)(2\ 4)(5\ 6)$, $\psi((1\ 5)) = (1\ 2)(3\ 6)(4\ 5)$, $\psi((1\ 6)) = (1\ 6)(2\ 5)(3\ 4)$. We have

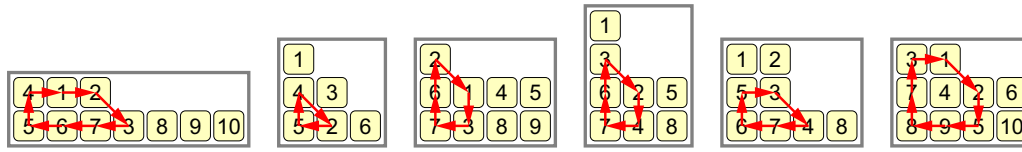
$$\alpha = (1\ 3\ 4\ 5\ 2) = (1\ 2)(1\ 5)(1\ 4)(1\ 3), \quad \beta = (1\ 4\ 5\ 6\ 2) = (1\ 2)(1\ 6)(1\ 5)(1\ 4).$$

Therefore, the two generators of G'_C are

$$\begin{aligned} \psi(\alpha) &= \psi((1\ 2))\psi((1\ 5))\psi((1\ 4))\psi((1\ 3)) = (1\ 6\ 2\ 3\ 5), \\ \psi(\beta) &= \psi((1\ 2))\psi((1\ 6))\psi((1\ 5))\psi((1\ 4)) = (1\ 3\ 2\ 6\ 5). \end{aligned}$$

Both generators $\psi(\alpha)$ and $\psi(\beta)$ leave the number 4 fixed, and thus G'_C is isomorphic to a subgroup of $\text{Sym}(5)$. Moreover, the generators are cycles of odd length, and therefore they produce only even permutations. Hence, G'_C is isomorphic to a subgroup of $\text{Alt}(5)$. Observe that $\psi(\alpha)\psi(\beta) = (1\ 5\ 6)$, which is a 3-cycle involving consecutive elements of the 5-cycle $\psi(\alpha)$. These two cycles generate all even permutations on $\{1, 2, 3, 5, 6\}$ (cf. [26, Proposition 1]).

We conclude that G'_C , and therefore G_C , is isomorphic to $\text{Alt}(5)$, which is a group of order 60 and index 12 in $\text{Sym}(6)$. A permutation $\pi \in \text{Sym}(6)$ is in G_C if and only if $\psi(\pi)$ is even and leaves the number 4 fixed.



■ **Figure 16** Some small configurations where at least three tokens are not covered by the first type-A cycle.

- Let $b = 2$, and let the two rows have length 2 and 5, respectively (Figure 15, top row, fourth image). We denote the three type-A cycles by $\alpha = (1\ 3\ 4\ 5\ 2)$, $\beta = (1\ 4\ 5\ 6\ 2)$, and $\gamma = (1\ 5\ 6\ 7\ 2)$. Consider the permutations $(\alpha\beta^{-1}\gamma)^2 = (2\ 5\ 4)$ and $\beta\gamma\alpha = (1\ 3\ 5\ 4\ 2\ 6\ 7)$. We have a 3-cycle involving consecutive elements of a 7-cycle, which generate all even permutations on $\{1, 2, \dots, 7\}$ (cf. [26, Proposition 1]). We conclude that $G_C = \text{Alt}(n)$.
- Let $b = 2$, and let the two rows have length $\ell \geq 3$ and $\ell + 2$, respectively (Figure 15, bottom row, first image). We denote the two type-A cycles by $\alpha = (1, \ell + 1, \dots, n - 1, \ell, \dots, 2)$ and $\beta = (1, \ell + 2, \dots, n, \ell, \dots, 2)$. The permutation $\beta^{-2}(\alpha^2\beta^{-1}\alpha^{-2}\beta)^2\beta^2 = (n - 3, n - 2, n)$ is a 3-cycle that, together with α , forms a 2-connected $(3, n - 1)$ -puzzle involving all tokens. We conclude that $G_C = \text{Alt}(n)$, due to [26, Theorem 2].
- Let $b = 2$, and let the two rows have length $\ell \geq 3$ and $\ell + 3$, respectively (Figure 15, bottom row, second image). Observe that this configuration is the same as the previous one, except for an extra token, labeled n , in the bottom row. In particular, the first two type-A cycles are the same, and generate all even permutations on $\{1, 2, \dots, n - 1\}$, including the 3-cycle $(1, \ell + 1, \ell + 2)$. This 3-cycle forms a 1-connected $(3, n - 2)$ -puzzle with the third type-A cycle. Since all tokens are involved in this puzzle, we conclude that $G_C = \text{Alt}(n)$, due to [26, Theorem 1].
- Let $b = 3$, and let the three rows have length 1, 1, and 3, respectively (Figure 15, bottom row, third image). The two type-A cycles $(2\ 3\ 4)$ and $(2\ 4\ 5)$ form a 2-connected $(3, 3)$ -puzzle, and therefore they generate all even permutations on $\{2, 3, 4, 5\}$, due to [26, Theorem 2]. In particular, they generate the 3-cycle $(3\ 4\ 5)$, which forms a 1-connected $(3, 3)$ -puzzle with the type-B cycle $(1\ 2\ 4)$, involving all tokens. By [26, Theorem 1], $G_C = \text{Alt}(n)$.
- Let $b = 3$, and let the three rows have length 1, 3, and 3, respectively (Figure 15, bottom row, fourth image). We denote the two type-A cycles by $\alpha = (1\ 2\ 5\ 6\ 3)$ and $\beta = (1\ 3\ 6\ 7\ 4)$. Consider the permutations $(\beta^2\alpha^{-1})^2 = (2\ 5\ 6)$ and $\alpha\beta^{-1} = (1\ 4\ 7\ 3\ 2\ 5\ 6)$. We have a 3-cycle involving consecutive elements of a 7-cycle, which generate all even permutations on $\{1, 2, \dots, 7\}$ (cf. [26, Proposition 1]). We conclude that $G_C = \text{Alt}(n)$.

We will now prove that all configurations not listed above satisfy $n/2 \geq a' + b' + 1$, i.e., they have at least three tokens not lying on the first type-A cycle.

- If $b = 1$, then $a'' = b'' = 0$. This case has already been discussed (Figure 14, left).
- For $b = 2$, we have discussed all cases where $a'' \leq 3$. If $a'' \geq 4$, then $n = 2a' + a''$, while a type-A cycle spans $2a' + 1$ tokens, and leaves out at least three tokens (Figure 16, first image).
- Let $b \geq 3$ and $a' = b' = 1$. Then, a type-A cycle spans three tokens. We are assuming that $a'' \geq 2$, and so $a \geq 3$, implying that $n \geq 5$. The only case where $n = 5$ has already been discussed (it is the case with $b = 3$ rows of length 1, 1, and 3, illustrated in Figure 15, bottom row, third image); in all other cases we have $n \geq 6$, and thus at least three tokens are left out of the type-A cycle (Figure 16, second image).

- Let $b \geq 3$, $a' = 1$, and $b' = 2$. Then, a type-A cycle spans five tokens. We are assuming that $a'' \geq 2$, and so $a \geq 3$, implying that $n \geq 7$. The only case where $n = 7$ has already been discussed (it is the case with $b = 3$ rows of length 1, 3, and 3, illustrated in Figure 15, bottom row, fourth image); in all other cases we have $n \geq 8$, and thus at least three tokens are left out of the type-A cycle (Figure 16, third image).
- Let $b \geq 3$, $a' = 1$, and $b' \geq 3$ (Figure 16, fourth image). Since we are assuming that $a'' \geq 2$, the rightmost column contains at least three tokens, which are not included in the first type-A cycle.
- Let $b \geq 3$, $a' \geq 2$, and $b' = 1$ (Figure 16, fifth image). Since we are assuming that $a'' \geq 2$, the rightmost token is not included in the first type-A cycle. Moreover, the top row contains at least two tokens, which are not in the type-A cycle. In total, there are at least three tokens not spanned by the cycle.
- Let $b \geq 3$, $a' \geq 2$, and $b' \geq 2$ (Figure 16, sixth image). Since we are assuming that $a'' \geq 2$, the rightmost column contains at least two tokens, which are not included in the first type-A cycle. Moreover, the token at $(1, 1)$ is not contained in the first type-A cycle, either. In total, there are at least three tokens not spanned by the cycle.

Arbitrary Labels

We now discuss the case where not all labels are distinct. Clearly, if all the non-core tokens have distinct labels, then our previous analysis carries over verbatim.

Let us now assume that at least two non-core tokens x and y have the same label. In this case, we identify two permutations $\pi_1, \pi_2 \in G_C$ if the configurations $C_1, C_2: \mathcal{L} \rightarrow \Sigma \cup \{\text{empty}\}$ they produce are equal, i.e., $C_1(p) = C_2(p)$ for all $p \in \mathcal{L}$.

Assume that G_C contains all even permutations of the non-core tokens, which we know to be always the case except in some special configurations. Let π be any permutation of the non-core tokens. If π is even, then $\pi \in G_C$, and we can reconfigure the tokens to match π . If π is odd, then let $\pi' = (x y)\pi$. Since π' is even, we have $\pi' \in G_C$. However, π and π' produce equal configurations, because x and y have the same label, and therefore we can reconfigure the tokens to match π , as well.

We now have a complete solution to the Permutation Puzzle.

► **Theorem 20.** *If the configuration C is compact, then the group G_C of possible permutations is as follows.*

- *If no cell in the bounding box is empty, then G_C is the trivial group.*
- *If exactly one cell in the bounding box is empty, then G_C is generated by the cycle of the non-core tokens taken in clockwise order.*
- *If there are exactly 6 tokens and exactly 2 empty cells in the bounding box, then G_C is isomorphic to the alternating group $\text{Alt}(5)$.*
- *In all other cases, G_C is the alternating group on the non-core tokens. Hence, if at least two non-core tokens have the same label, then all permutations of the non-core labels can be obtained; otherwise, only the even permutations of the non-core labels can be obtained.* ┘

5 Conclusions and Open Problems

Concerning the Compaction Puzzle, we showed that all sparse configurations of $n = ab$ tokens can be pushed into an $a \times b$ box if and only if $a \leq 2$ or $b \leq 2$ or $a = b = 3$ (Theorem 8), and that there exist sparse configurations capable of becoming any compact configuration of size n (Observation 6). We leave the following as an open problem.

► **Open Problem 1.** *Is it NP-complete to decide whether a given unlabeled sparse configuration can be pushed into a given compact configuration?*

We do not know the answer to this problem even if we restrict the target configuration to be a rectangle (note that the NP-completeness proof in [5] does not hold for sparse initial configurations).

Concerning the Permutation Puzzle, we have shown that, given two same-shaped compact configurations, all even permutation of the non-core tokens are reachable, with a few minor exceptions (Theorem 20). Notably, if at least two tokens have the same label, then *all* permutations of the non-core labels can be obtained.

In particular, the puzzle in Figure 2 is solvable, because it features a compact configuration with no core tokens, more than two empty cells, and at least two same-labeled (i.e., same-colored) tokens. Thus, according to Theorem 20, any permutation of the labels is feasible in this puzzle, and we only have to verify that the number of tokens with any given label in the initial configuration matches the number of tokens with that label in the goal configuration.

We remark that, even though our proof relies on Theorem 15, which is a deep, non-constructive result, we have nonetheless uncovered a great deal of structure in the Permutation Puzzles. We claim that, when a Permutation Puzzle is solvable, there is a solution within $O(n^4)$ pushes which is computable by a polynomial-time algorithm. However, we conjecture that finding the *shortest* solution is NP-hard (this is known to be the case in other token-shifting puzzles [26, 9, 24]).

► **Open Problem 2.** *Is it NP-hard to find the shortest solution in the Permutation Puzzle?*

When the initial configuration is not compact, the pushes applied in order to obtain a compact one can affect the permutation, making the problem substantially harder. For example, the pushes performed before the configuration becomes incompressible may affect the configuration of core tokens. Furthermore, a solution to the general problem of transforming an arbitrary labeled configuration into a target compact one would imply a solution to Open Problem 1.

► **Open Problem 3.** *Given a (not necessarily compact) labeled configuration, is there a sequence of pushes that transforms it into a (not necessarily compact) target configuration?*

References

- 1 Labyrinth, 1946. BRIO. <https://www.brio.us/products/all-products/games/labyrinth> [Accessed 02/16/2022].
- 2 Mega Maze, 1995. Philips Interactive Media.
- 3 The first mobile game goes viral: Pigs in Clover, 2018. The Strong National Museum of Play. <https://www.museumofplay.org/2018/08/01/the-first-mobile-game-goes-viral-pigs-in-clover/> [Accessed 02/16/2022]. URL: <https://www.museumofplay.org/2018/08/01/the-first-mobile-game-goes-viral-pigs-in-clover/>.
- 4 Ahmed Abdelkader, Aditya Acharya, and Philip Dasler. 2048 without new tiles is still hard. In Erik D. Demaine and Fabrizio Grandoni, editors, *8th International Conference on Fun with Algorithms (FUN 2016)*, volume 49 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FUN.2016.1.
- 5 Hugo A. Akitaya, Greg Aloupis, Maarten Löffler, and Anika Rounds. Trash compaction. In *Proceedings of the 32nd European Workshop on Computational Geometry (EuroCG 2016)*, pages 107–110, 2016.

- 6 Hugo A. Akitaya, Erik D. Demaine, Jason S. Ku, Jayson Lynch, Mike Paterson, and Csaba D. Tóth. 2048 without merging. In *32nd Canadian Conference on Computational Geometry (CCCG 2020)*, pages 285–291, 2020.
- 7 Hugo A. Akitaya, Maarten Löffler, Anika Rounds, and Giovanni Viglietta. Compaction games. In *Workshop on Combinatorial Reconfiguration (CORE), affiliated with ICALP 2021*, page 20, 2021.
- 8 Hugo A. Akitaya, Maarten Löffler, and Giovanni Viglietta. Pushing blocks by sweeping lines. *arXiv:2202.12045 [math.CO]*, 2022.
- 9 Kazuyuki Amano, Yuta Kojima, Toshiya Kurabayashi, Keita Kurihara, Masahiro Nakamura, Ayaka Omi, Toshiyuki Tanaka, and Koichi Yamazaki. How to solve the torus puzzle. *Algorithms*, 5(1):18–29, 2012.
- 10 Jose Balanza-Martinez, Timothy Gomez, David Caballero, Austin Luchsinger, Angel A. Cantu, Rene Reyes, Mauricio Flores, Robert Schweller, and Tim Wylie. Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2625–2641. SIAM, 2020.
- 11 Jose Balanza-Martinez, Austin Luchsinger, David Caballero, Rene Reyes, Angel A. Cantu, Robert Schweller, Luis Angel Garcia, and Tim Wylie. Full tilt: Universal constructors for general shapes with uniform external forces. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2689–2708. SIAM, 2019.
- 12 Aaron Becker, Erik D Demaine, Sándor P Fekete, Golnaz Habibi, and James McLurkin. Reconfiguring massive particle swarms with limited, global control. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 51–66. Springer, 2013.
- 13 Aaron Becker, Golnaz Habibi, Justin Werfel, Michael Rubenstein, and James McLurkin. Massive uniform manipulation: Controlling large populations of simple robots with a common input signal. In *2013 IEEE/RSJ international conference on intelligent robots and systems*, pages 520–527. IEEE, 2013.
- 14 Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Jarrett Lonsford, and Rose Morris-Wright. Particle computation: complexity, algorithms, and logic. *Natural Computing*, 18(1):181–201, 2019.
- 15 David Caballero, Angel A Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller, and Tim Wylie. Building patterned shapes in robot swarms with uniform control signals. In *Canadian Conference on Computational Geometry (CCCG 2020)*, 2020.
- 16 David Caballero, Angel A Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller, and Tim Wylie. Hardness of reconfiguring robot swarms with uniform external control in limited directions. *Journal of Information Processing*, 28:782–790, 2020.
- 17 David Caballero, Angel A Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller, and Tim Wylie. Fast reconfiguration of robot swarms with uniform control signals. *Natural Computing*, 20(4):659–669, 2021.
- 18 Erik D Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In *International Symposium on Mathematical Foundations of Computer Science*, pages 18–33. Springer, 2001.
- 19 Erik D Demaine, Martin L Demaine, Michael Hoffmann, and Joseph O’Rourke. Pushing blocks is hard. *Computational Geometry*, 26(1):21–36, 2003.
- 20 Robert A Hearn. The complexity of sliding block puzzles and plank puzzles. *Tribute to a Mathematician*, pages 173–183, 2005.
- 21 Robert A Hearn and Erik D Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005.
- 22 Gareth A. Jones. Primitive permutation groups containing a cycle. *Bulletin of the Australian Mathematical Society*, 89(1):159–165, 2014.

- 23 D Kornhauser, G Miller, and P Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium on Foundations of Computer Science, 1984.*, pages 241–250. IEEE, 1984.
- 24 Daniel Ratner and Manfred Warmuth. The $(n^2 - 1)$ -puzzle and related relocation problems. *Journal of Symbolic Computation*, 10:111–137, 1990.
- 25 Joseph J. Rotman. *An introduction to the theory of groups*. Springer-Verlag, fourth edition, 1995.
- 26 Kwon Kham Sai, Giovanni Viglietta, and Ryuhei Uehara. Cyclic shift problems on graphs. *IEICE Transactions on Information and Systems*, E105-D(3), 2022.
- 27 Yinan Zhang, Xiaolei Chen, Hang Qi, and Devin Balkcom. Rearranging agents in a small space using global controls. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3576–3582. IEEE, 2017.

A Practical Algorithm for Chess Unwinnability

Miguel Ambrona  

Independent Researcher, Madrid, Spain

Abstract

The FIDE Laws of Chess establish that if a player runs out of time during a game, they lose unless there exists no sequence of legal moves that ends in a checkmate by their opponent, in which case the game is drawn. The problem of determining whether or not a given chess position is *unwinnable* for a certain player has been considered intractable by the community and, consequently, chess servers do not apply the above rule rigorously, thus unfairly classifying many games.

We propose, to the best of our knowledge, the first algorithm for *chess unwinnability* that is sound, complete and efficient for practical use. We also develop a prototype implementation and evaluate it over the entire Lichess Database (containing more than 3 billion games), successfully identifying *all* unfairly classified games in the database.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Software and its engineering → Software libraries and repositories

Keywords and phrases chess, helpmate, unwinnability, timeout, dead position

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.2

Related Version *Full Version*: <https://chasolver.org/FUN22-full.pdf>

Supplementary Material <https://github.com/miguel-ambrona/D3-Chess>

Acknowledgements Special thanks to Pooya Farshim, for very fruitful discussions and all his feedback; Elena Gutiérrez, for all her help and comments; Antonio Nappa, for providing the hardware; the Lichess Team; the Stockfish Team; and many others: <https://chasolver.org/acks.html>. I would also like to express my sincere gratitude to Andrew Buchanan and Andrey Frokin, for sharing with me and letting me include two original compositions, and for all the feedback. Finally, I would like to thank Maarten Loeffler, for having found a gap in an earlier version of the proof of Lemma 8, and for all his comments. I am also very thankful to the other anonymous reviewers of FUN 2022, for their valuable time and careful reading of this manuscript.

1 Introduction

Chess clocks have been used since 1883 [14] and are an essential tool in (tournament) chess to enforce game termination. They introduce a reliable upper-bound on the duration of games, ensuring that players will not excessively delay the match. This is crucial for designing and respecting tournament schedules.

A chess clock consists of two adjacent and entangled (countdown) timers that can never run simultaneously. Each player is responsible for one of the timers and must complete the game before their timer gets down to zero. Otherwise the player would “flag”, i.e., lose on time. During the game, the player with the turn must press the clock’s button after making a move (this is not necessary in online chess). This action, which concludes the player’s turn, will stop their timer and resume their opponent’s timer, who now has the turn and must proceed analogously.

There exists a wide variety of *time controls* that specify the initial allotted time and (optionally) a time bonus after every prescribed number of moves, ranging from several hours (or even days) to just 15 seconds to complete the entire game [1]. What is common to all time controls is that running out of time leads to a defeat. But not always! The clock is



© Miguel Ambrona;

licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 2; pp. 2:1–2:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

just a tool to guarantee that the game will finish, but actual game position is given a higher priority than the clock state. For example, if your last move has checkmated your opponent, you win, even if your timer got down to zero while executing it [7, Article 5.1.1]. Or if you just have the king (because all your other pieces were captured), you cannot win anymore, not even on time! This folklore rule is a particular case of the following more general rule described in Article 6.9 of the FIDE Laws of Chess [7].

... if a player does not complete the prescribed number of moves in the allotted time, the game is lost by that player. However, the game is drawn if the position is such that the opponent cannot checkmate the player's king by any possible series of legal moves.

1.1 The problem

Thus, in order to rigorously apply Article 6.9, one must be able to tell whether or not a given position can be won by the player who still has time on their clock.

► Remark 1. A position being *winnable* does not mean that a certain player can force a victory. Rather, it refers to the existence of a sequence of legal moves that ends in a checkmate by the player. Such sequence, which typically contains a poor choice of moves, is sometimes referred to as a *helpmate* [22].

Deciding whether or not a position is unwinnable, i.e. whether a helpmate does not exist, is usually relatively simple for a human. For example, it is not very hard to realize that no player can deliver checkmate in Position 1 (it is a so-called *dead position*), since the pawn wall is blocked and the bishops are not useful to make any progress. According to the FIDE Laws of Chess [7, Article 5.2.2] the game is finished as soon as the position becomes dead. No further moves are permitted and would be considered illegal.¹

However, other such positions can be more involved. For example, it is not so easy to understand/prove why Position 2 is also dead (White to move).² Interestingly, if in Position 2 the pawn on a4 were on a5, the position would be winnable for White. Indeed, the following is a possible helpmate sequence in that case:

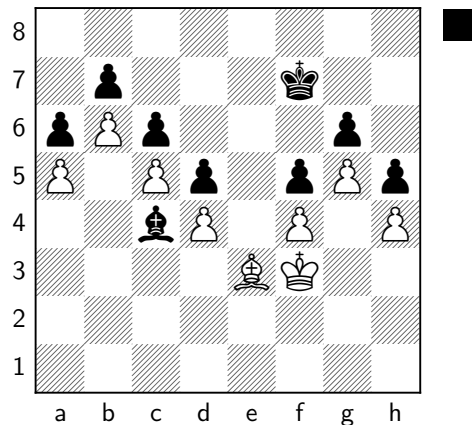
1 ♖a6 ♜e8 2 ♙b7 ♜d8 3 ♙c8 ♜e8 4 ♙d7+ ♜d8 5 ♙e8 ♜c8 6 ♙f7 ♜d8 7
 ♙g8 ♜e8 8 ♙b7 ♜d8 9 g5 ♜e8 10 ♜c8 a4 11 ♙f7#

Note how the pawn being on a5 gave Black an additional tempo on move 10; without it, Black would have been in stalemate. This position, devised by the prominent chess composer Andrew Buchanan³, evidences the hardness of deciding unwinnability, as very subtle changes in the position can alter the result.

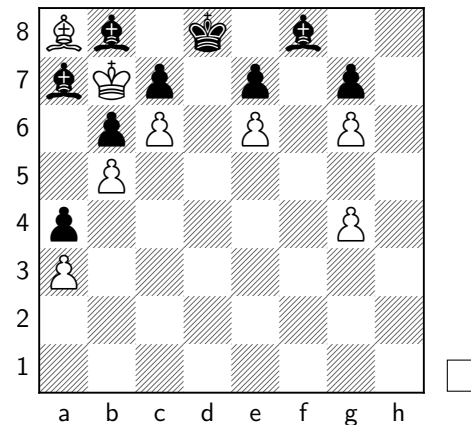
¹ This rule was introduced on July 1, 1997, and gave birth to a completely new genre of chess compositions called *dead reckoning* [3, 4, 20]. See Appendix C for two original compositions of this kind.

² The player to move is indicated by either a white or a black box adjacent to the diagram.

³ The original composition asks: *What was the last move?* (In Position 2). Since it is White to move, the last move must have been either ... ♜e8-d8 or ... a5-a4, but only one of these moves comes from a position that is not dead. It's instructive to guess which one is the case here!



■ **Position 1** Lichess game tLUsoyti.



■ **Position 2** A. Buchanan, *StrateGems* 2002.

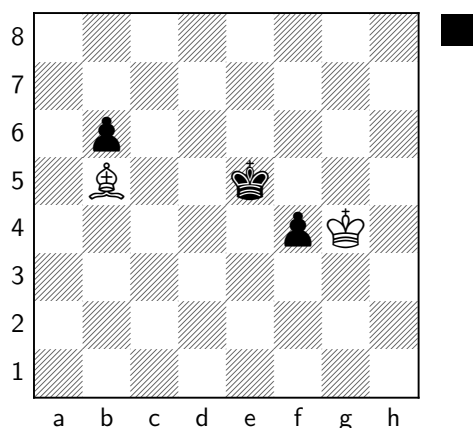
1.2 Related work

Online chess. Chess servers are an important point of reference to understand what the current state-of-the-art with respect to chess unwinnability is. It turns out that given the apparent complexity of deciding unwinnability, chess servers only analyze whether the intended winner has sufficient material to checkmate. Indeed, the three most popular chess servers adjudicate timeouts to date as follows.

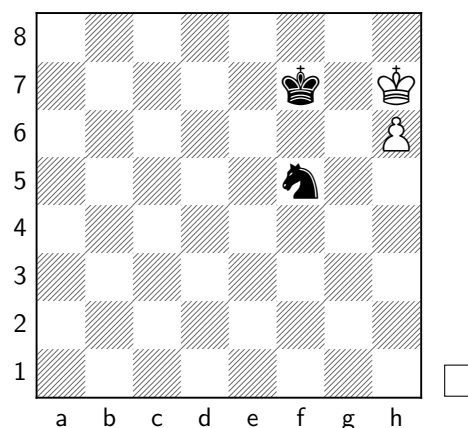
- Chess.com, currently the Internet's biggest online chess server, declares a position as drawn if, after a timeout, the player with time on the clock has *insufficient material* [5, 16, 17]. Namely, if they have (i) a lonely king, (ii) a king and a bishop, (iii) a king and a knight, (iv) a king and two knights. This decision is supported by the claim that they do not follow the FIDE Laws of Chess for adjudicating timeouts and instead follow the USCF rules, which specify that the game is drawn (in case of insufficient material) if there is no *forced mate* by the intended winner.
- Lichess.org, one of the most popular chess websites in the world while remaining 100% free/libre and open-source, focuses on positions without pawns. Unlike other servers, *Lichess never declares a position as unwinnable when it is indeed winnable*. (Although it fails to identify all unwinnable positions.) In particular, Lichess correctly classifies all insufficient material positions that do not contain pawns. For example, KQ vs KB (king and queen vs king and bishop) positions are (correctly) declared as unwinnable for the player holding the bishop (see Lemma 6).
- Chess24 seems to proceed as Chess.com does. As an example, see Position 3. It corresponds to a Chess24 game that, according to FIDE rules, was unfairly classified as a draw after Black ran out of time. White has only a bishop, but there is still a chance for White to checkmate Black (of course not by force) if Black promoted to a knight and trapped themselves in a corner!

We do not advocate following the FIDE Laws of Chess over other choices. Every chess server should have the right to choose their own rules, especially for situations that occur infrequently and do not significantly impact the experience of chess players. However, we believe that the approach followed by Lichess is more satisfactory: *in case unwinnability cannot be determined by the system's logic, declare the position as winnable*. Otherwise, very unfair situations may arise. For example, using only the above-mentioned material rules

2:4 A Practical Algorithm for Chess Unwinnability



■ **Position 3** Black ran out of time. White can still *helpmate* with under promotion to a knight. (Chess24 game V01NB3MGSYqXv3BhVsCBkA [8].)



■ **Position 4** White ran out of time. Black can *force a victory* with just a knight. (Lichess game HaTT3dsU.)

in Position 4, after White ran out of time, would lead to classifying the game as drawn. However, not only can Black *helpmate* in that position, but Black can also force a victory, for example, with the following forced sequence.⁴

1 ♖h8 ♜h4 2 ♖h7 ♜g2 3 ♖h8 ♜f4 4 ♖h7 ♜e6 5 ♖h8 ♜f8 6 h7 ♜g6#

An absurd situation arises: In Position 4, White is completely lost. However, White can still draw the game by letting their time run out (if they are playing on a server that adjudicates timeouts by following simple rules based on the amount of material) [9].

Existing tools for unwinnability. Labelle [12] performed a computer search over a database of 2M+ *over the board* games (from 1998 to 2011), searching for dead drawn positions by forced insufficient material or forced stalemate where players continued making moves (illegal moves, according to [7, Article 5.2.2]). This demonstrates that dead positions occur in professional chess and can be easily missed by chess arbiters.

Varmose [21] implemented an algorithm that identifies blocked positions that only involve bishops and pawns, which only misses some corner cases. This is a non-trivial step towards solving unwinnability, but the tool cannot identify all unwinnable positions.

Other tools for solving *helpmate* problems, such as the analyzer by Paliulionis [18] or the solver by Dugovic [6] can potentially identify any unwinnable position as they perform an exhaustive search over the tree of moves. Nevertheless, such tools can hardly be utilized to decide unwinnability, they would incur a prohibitive cost for most positions [15].

This state of affairs leaves the problem of automatically checking whether or not a position can be won by a given player inadequately addressed. Given the intractability of a simple brute force approach, we ask:

How can we rule out all sequences of legal moves without actually exploring them all?

⁴ The trick is to always watch the g6 square when the white king is on h8, preventing White from stalemating themselves by pushing the pawn. This has to be done while maneuvering the knight to f8 from where it controls g6 as before as well as h7, thus forcing White to push the pawn.

On the complexity of chess unwinnability. Chess unwinnability for an appropriate generalization of chess over an $n \times n$ board has been studied by Brunner, Demaine, Hendrickson, and Wellman [2]. The authors prove, via a reduction from a one-player game called Subway Shuffle [10], that chess unwinnability is PSPACE-complete.

Their generalization of chess does not impose any restriction on the length of games. A natural alternative generalization, motivated by the 75-moves rule⁵, would be to impose a polynomial bound on the round complexity of the game. In that case, it would follow from the results of Brunner et al. that chess unwinnability is coNP-complete under such generalization. These intractability results, however, do not apply to the (constant) case $n = 8$, our goal in this paper.

1.3 Our contributions

We pursue the study of chess unwinnability and establish several results that, together, form an algorithm which is sound, complete and computationally practical.

Static analysis. Our main contribution is a mechanism for statically determining that a position is unwinnable without explicitly exploring game variations (Section 3), which we believe, can be of independent interest and applicable to other board games. This algorithm is particularly effective on *blocked* positions, e.g. Position 1, where players have access to limited and disjoint regions of the board and can make no progress. Our static analysis is performed in two steps: (i) identifying what pieces can move and all the squares that each can potentially reach; (ii) based on the previous information and based on the number of pieces that can check and constrain the movement of the intended loser’s king, our analysis may conclude that checkmate is impossible.

The problem associated to (i), that we coin the *mobility problem*, is arguably the most challenging part of the analysis. In Section 3.1, we provide an algorithm (Figure 7) that over-approximates the true solution to the mobility problem on the given position (Corollary 9). (Informally, the solution provided by our algorithm is always greater than or equal to the actual solution.) We then show (Lemma 11) that our routine for addressing our second step (ii), described in Figure 8, is sound when given as input the true solution to the mobility problem. (It is never wrong when its output is “unwinnable”.) Furthermore, we argue that this routine is monotone (Lemma 10), which allows us to conclude that it is also sound when given an over-approximation to the actual mobility solution (Theorem 12). Consequently, the composition of our two routines leads to a (static) unwinnability algorithm which is **sound** (but not complete).

Search of variations. We propose an algorithm for exploring variations, enhanced with a transposition table and selected heuristics for deciding what moves to explore further (Section 2.2, Figure 5). This building block is combined with our static analysis to build an algorithm for chess unwinnability (Section 4, Figure 9) that is sound and complete.

Implementation. We implement our algorithm and evaluate it over the entire database of Lichess standard rated games [13], successfully identifying all the games that were unfairly classified after a timeout (Section 5). These results evidence that the algorithms developed in this work are suitable for practical use and could be adopted by real-world chess servers.

⁵ Establishing that a game is drawn if 75 full moves are completed without captures or pawn movements.

2 Preliminaries

We assume the reader to be familiar with the basic rules of chess, including piece movement and game completion. Nevertheless, we establish a formal notation, which facilitates a rigorous description of our main algorithms and results.

Let $\mathcal{S} := \{\text{a1, b1, } \dots, \text{h8}\}$ be the set of all 64 *squares* on a chessboard. Let the set of *piece types* be $\mathcal{T} := \{\text{♔, ♚, ♖, ♗, ♘, ♙, ♜, ♝}\}$. A *position* pos is a collection of pieces, i.e., triples $(t, c, s) \in \mathcal{T} \times \{\text{w, b}\} \times \mathcal{S}$ where t is a piece type, c is the piece color and s is a square. Given a piece $P := (t, c, s)$ we define $P.\text{type} = t$, $P.\text{side} = c$ and $P.\text{sq} = s$.

A position is *valid* if for every distinct $P, P' \in \text{pos}$, it holds that $P.\text{sq} \neq P'.\text{sq}$. A position is said to be *legal* if it can be reached from the initial position of a chess game by a sequence of legal moves. We define the *king-distance* between two squares as the number of moves that it takes for a king to go from one to the other over an empty board. We define *knight-distance* analogously. We say that two squares are *adjacent* if they are at king-distance 1. A rank is a row of 8 squares, whereas a file is a column of 8 squares. Unless specified otherwise, we measure the depth of a variation (a sequence of moves) in halfmoves or *plies*.

For every square $s \in \mathcal{S}$, we denote by $\oplus(s)$ the set of squares that share a border with s (that is, adjacent squares of a different color) and by $\boxplus(s)$ the squares that are diagonally adjacent to s (i.e., adjacent squares of the same color). We define $\boxtimes(s) := \oplus(s) \cup \boxplus(s)$. We also denote by $\ominus(s)$ the set of squares that are at knight-distance 1 from s . The empty or singleton set containing the adjacent square to s from which a white (respectively, black) pawn could move to s in one non-capturing move is denoted by $\sqcap(s)$ (respectively, $\sqcup(s)$). Finally, we denote by $\blacktriangleleft(s)$ (respectively, $\blacktriangleright(s)$) the set of squares from which a white (respectively, black) pawn attacks s .

► **Definition 2** (Unwinnability). *We say that a position is unwinnable for a given player if there does not exist a sequence of legal moves that ends in a checkmate by the player.*

Our goal is to build an algorithm for solving chess unwinnability, which given a position and an intended winner, after a finite number of steps always outputs a binary value in $\{\text{Unwinnable, Winnable}\}$, indicating unwinnability. Ideally, when the position is declared as Winnable, we would like our algorithm to also provide a helpmate sequence, i.e., a witness of non-unwinnability. We require our algorithm to be sound.

► **Definition 3** (Soundness). *We say an algorithm for solving chess unwinnability is sound if it is never wrong when its output is Unwinnable.*

Another desirable property is completeness, meaning that the algorithm will be able to identify all unwinnable positions.

► **Definition 4** (Completeness). *We say an algorithm for solving chess unwinnability is complete if it is never wrong when its output is Winnable.*

Since chess is finite, it is not hard to implement a chess unwinnability algorithm that is both sound and complete, by performing an exhaustive search over the tree of variations. The challenge here, however, is to achieve efficiency while preserving soundness and completeness. Note that an exhaustive search would terminate relatively quickly in most positions, since it just needs to find a helpmate sequence for the intended winner. However, in positions where no checkmate is possible, the whole tree of variations would need to be exhausted before unwinnability could be concluded.

Our starting point will be a routine that performs an exhaustive search, enhanced with a transposition table and with heuristics for selecting what moves to explore first (see Section 2.2). We will first establish some preliminary results that ensure unwinnability in positions that contain no pawns.

<p><u>Find-Helpmate_c(pos, depth, maxDepth):</u> Global variables: table, cnt, nodesBound</p> <p>Inputs: position, depth (int), maxDepth (int) Output: bool (<i>true</i> if a checkmate sequence was found, <i>false</i> otherwise)</p> <ol style="list-style-type: none"> 1: if the intended winner is checkmating their opponent in pos then return true 2: if the intended winner has just the king or the position is unwinnable according to Lemma 5 or Lemma 6 or the position is stalemate or the intended winner is receiving checkmate in the position then return false 3: increase cnt and set $d := \text{maxDepth} - \text{depth}$ 4: if $\text{cnt} > \text{nodesBound} \vee d < 0$ then return false ▷ The search limits are exceeded 5: if $(\text{pos}, D) \in \text{table}$ with $D \geq d$ then return false ▷ pos was already analyzed 6: store (pos, d) in table 7: for every legal move m in pos do: 8: let $\text{inc} = \text{match Score}(\text{pos}, m)$ with Normal $\rightarrow 0$ Reward $\rightarrow 1$ Punish $\rightarrow -2$ 9: if Find-Helpmate_c(pos.move(m), depth + 1, maxDepth + inc) then return true 10: return false ▷ No mate was found after exploring every legal move

■ **Figure 5** Find-Helpmate_c routine, returns *true* if a checkmate sequence for player $c \in \{w, b\}$, the intended winner, is found or *false* otherwise. The base call should be done on $\text{depth} = 0$, $\text{cnt} = 0$, and an empty table. The value of maxDepth and nodesBound can be chosen to set the limits of the search. See the full version of this paper for details about the **Score** routine.

2.1 Preliminary results

We say a position is *pawn-free* if it does not contain pawns of any color. We refer to the full version of this paper for a formal a proof of the following lemmas.

▶ **Lemma 5.** *A pawn-free position is unwinnable for a player with just a knight if their opponent does not have knights, bishops or rooks.*

▶ **Lemma 6.** *A pawn-free position is unwinnable for a player with just bishops of one square color if their opponent does not have knights or bishops of the opposite square color.*

2.2 Search of variations

We propose a dedicated search of variations, enhanced with a transposition table and heuristics that reward some variations, which will be explored further before others (described in Figure 5). This routine, called Find-Helpmate_c, constitutes an important building block of our main algorithm for chess unwinnability, described in Section 4, where it is combined with our static analysis (see Section 3).

Find-Helpmate_c is a recursive algorithm that outputs *true* only when it has found a checkmate position for the intended winner. Otherwise, the algorithm will output *false* based on several criteria:

- The game is over, but the intended winner did not checkmate their opponent.
- The conditions of Lemma 5 or Lemma 6 apply.
- The position was found in the transposition table (a table storing all positions that have been explored so far), so it is not necessary to repeat the search that starts from it.
- All legal moves have been explored without having found a checkmate.
- The search limits were reached.

If the final output of Find-Helpmate_c on the given position is *false*, and the search limits were not reached in any of its recursive calls, the position is truly unwinnable.

The search limits include a maximum depth for the variations being explored and a limit on the total number of explored positions. Before exploring the position after a legal move, we determine with our *Score* heuristic (described in the full version of this paper) whether the maximum depth limit will be increased (rewarded), decreased (punished) or remain the same in the analysis of the variation associated with the move.

We refer to Section 4 for details of how Find-Helpmate_c is integrated into our chess unwinnability algorithm via iterative deepening [11].

3 Static algorithm

The search of variations provided by Find-Helpmate_c (Figure 5), which is enhanced with a transposition table and our heuristics for selecting what moves to explore first, will potentially terminate on any position, correctly classifying it with respect to unwinnability if the search limits were sufficiently large. However, in blocked positions, the search space can become prohibitively large. For example, in Position 1 the search would need to iterate over and store the (more than) 80K positions that can arise from that board configuration before deciding that the position is unwinnable. This would greatly exceed the maximum computation time that we should dedicate to a single position if we want our algorithm to be competitive and suitable for its integration in real-world chess servers, which usually handle tens of games terminating every second.

In order to offload unnecessary computations from our main routine in blocked positions, we design a mechanism that allows us to conclude that certain positions are unwinnable without explicitly exploring all variations. Our algorithm is divided into two phases:

- (i) Identifying what pieces can move and all the squares that each can potentially reach.
- (ii) Identifying the *king's region*, defined as the set of all squares that can be reached by the intended loser's king, as well as identifying all the intended winner's pieces that can move inside the king's region, the so-called *intruders*. Based on the number of intruders and their piece type, our algorithm may conclude that checkmate is impossible.

► **Remark 7.** This algorithm does not need to be complete, since it is backed by our main search routine. In fact, as evidenced by [2] or by the example in Position 2, deciding unwinnability without exploring variations may be an impossible task. Nevertheless, we require that our static algorithm be **sound** in the sense that it can be fully trusted when it classifies a position as *unwinnable*.

Performing step (i) is arguably the most challenging part of the analysis. For every $P \in \text{pos}$ and every $s \in \mathcal{S}$, we need to decide whether or not piece P , currently on square $P.\text{sq}$, can potentially go to square s after a sequence of legal moves. Define $M_{P \rightarrow s}^*$ as 1 if the above displacement is possible and 0 otherwise. Our *mobility algorithm* will try to approximate the correct value of $M_{P \rightarrow s}^*$ in the given position for every piece and every target square.

3.1 Mobility algorithm

We consider binary variables $M_{P \rightarrow s} \in \{0, 1\}$ for every $P \in \text{pos}$ and every $s \in \mathcal{S}$, encoding the output of our mobility algorithm. Since the function associated to solving step (ii) is monotone (see Lemma 10), any over-approximation of the actual solution is acceptable for the static algorithm to be sound. Namely, we allow for solutions $M_{P \rightarrow s}$ that satisfy $M_{P \rightarrow s} \geq M_{P \rightarrow s}^*$, coined *admissible* solutions. Intuitively, this is possible because wrongly

concluding that a piece can move more than what it really can is not harmful (in the sense that it may lead to the conclusion that the position is winnable when it is actually unwinnable, but not vice versa). However, we hope for an approximation that is as close as possible to the actual solution. (Observe that a degenerate output of $M_{P \rightarrow s} = 1$ for all P and s is admissible, but not useful, because step (ii) would simply return “possibly winnable”.)

We also define additional variables representing square reachability and clearance. This is useful to model pawn captures and king movements more accurately. More concretely, we consider the following binary variables: $M_{P \rightarrow s}$ for every $P \in \text{pos}$ and every $s \in \mathcal{S}$; C_P for every $P \in \text{pos}$; and R_s^c for every $s \in \mathcal{S}$ and every $c \in \{\text{w}, \text{b}\}$, defined as follows:

- $M_{P \rightarrow s}$ indicates if piece P , currently on $P.\text{sq}$, can eventually *move* to square s .
- R_s^c indicates if square s can eventually be *reached* by a *non-king* piece of color c (or if it is currently occupied by such a piece).
- C_P indicates if piece P can be *cleared* from its current square (by moving or being captured).

Given a piece P and a square s , we define the P -predecessors of s , denoted by $\text{pred}_P(s)$, as the squares that are at king-distance 1 from s (except for knight predecessors, which are at king-distance 2), from which a piece of type $P.\text{type}$ can reach s in one *non-capture* move over an empty board. More concretely,

$$\text{pred}_P(s) = \begin{cases} \blacksquare(s) & \text{if } P = (\hat{\Delta}, \text{w}, _) \\ \blacktriangleleft(s) & \text{if } P = (\hat{\Delta}, \text{b}, _) \\ \text{♞}(s) & \text{if } P.\text{type} = \text{♞} \\ \text{♟}(s) & \text{if } P.\text{type} = \text{♟} \\ \text{♚}(s) & \text{if } P.\text{type} = \text{♚} \\ \text{♛}(s) & \text{if } P.\text{type} \in \{\text{♖}, \text{♗}\} \end{cases} \quad \text{pred-capt}_P(s) = \begin{cases} \blacksquare(s) & \text{if } P = (\hat{\Delta}, \text{w}, _) \\ \blacktriangleleft(s) & \text{if } P = (\hat{\Delta}, \text{b}, _) \\ \text{pred}_P(s) & \text{otherwise} \end{cases}$$

$$\text{prom}(P) = \begin{cases} \{\text{a8}, \dots, \text{h8}\} & \text{if } P = (\hat{\Delta}, \text{w}, _) \\ \{\text{a1}, \dots, \text{h1}\} & \text{if } P = (\hat{\Delta}, \text{b}, _) \\ \emptyset & \text{otherwise} \end{cases}$$

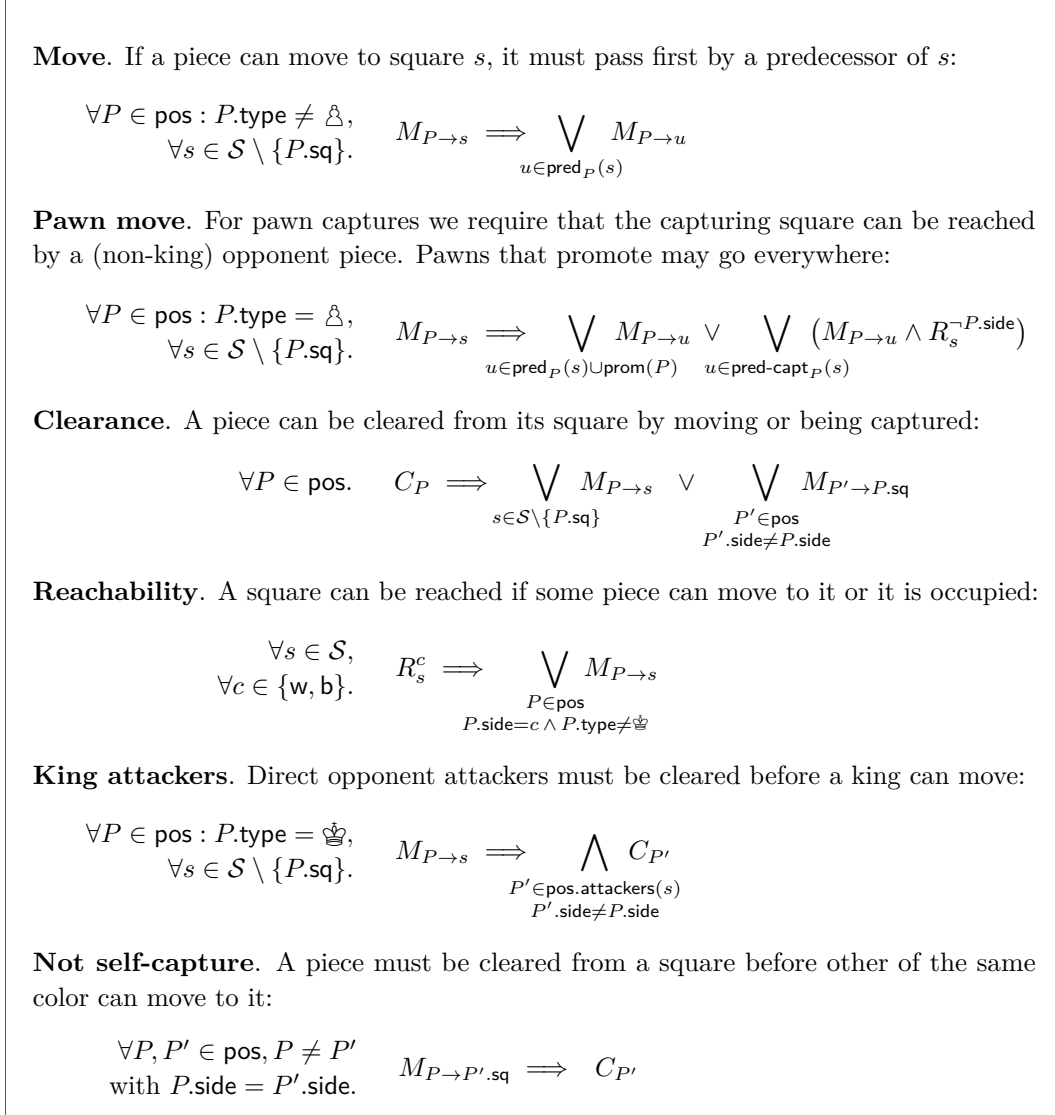
We define $\text{pos.attackers}(s)$ as the set $\{P \in \text{pos} : P.\text{sq} \in \text{pred-capt}_P(s)\}$. The mobility algorithm from Figure 7 greedily activates the mobility (reachability and clearance) variables as soon as it is possible (i.e. not forbidden by the logic of the implications from Figure 6). For example, if P is a knight and there is some piece P' of the same color as P , currently on h8 , $M_{P \rightarrow \text{h8}}$ can be set to true as soon as $(M_{P \rightarrow \text{g6}} = 1$ or $M_{P \rightarrow \text{f7}} = 1)$ and $C_{P'} = 1$. But it cannot be set to true otherwise: for the knight to reach h8 , it must first reach a direct predecessor of h8 and the ally piece on h8 must first be cleared.

► **Lemma 8 (Mobility Soundness).** *Let pos be a position where no player has castling rights and en passant is not possible. Let $M \leftarrow \text{Mobility}(\text{pos})$ (see Figure 7). If $M_{P \rightarrow s} = 0$ for some piece $P \in \text{pos}$ and some square s , then there is no sequence of legal moves after which piece P , starting from pos , can reach square s .*

Proof. The result follows from the fact that all implications from Figure 6 are sound, in the sense that they are all satisfied by the true solution to the mobility problem.

We will prove a more general result, involving the extra variables for reachability and clearance. Let $(\{M_{P \rightarrow s}\}_{P,s}, \{C_P\}_P, \{R_s^c\}_{s,c})$, for $P \in \text{pos}$, $s \in \mathcal{S}$, $c \in \{\text{w}, \text{b}\}$, be the final state of the execution of $\text{Mobility}(\text{pos})$. We will argue that, for all $n \in \mathbb{N}$:

- (a) If there exists a sequence of n legal halfmoves (plies) after which piece $P \in \text{pos}$ ends at square s , then $M_{P \rightarrow s} = 1$.
- (b) If there exists a sequence of n legal halfmoves after which a square s is occupied by a non-king piece of color c , then $R_s^c = 1$.
- (c) If there exists a sequence of n legal halfmoves after which a piece $P \in \text{pos}$ is cleared from its current square $P.\text{sq}$, then $C_P = 1$.



■ **Figure 6** Logical implications for the static algorithm.

We proceed by induction on n . Assume $n = 0$, it is easy to see that (a) will hold, because the Mobility algorithm sets $M_{P \rightarrow P.\text{sq}}$ to 1 in step (2), for every piece P . (Note that the algorithm never changes the value of a variable to 0 after it has been set to 1.) To see that (b) holds, note that the *reachability* rule from Figure 6 will set $R_{P.\text{sq}}^{P.\text{side}}$ to 1 for all $P \in \text{pos}$, but those are exactly the squares that can be reached in no halfmoves by pieces of the corresponding color. Finally, (c) holds trivially, because there is no sequence of 0 halfmoves that allows a piece to be cleared from their current square.

Now, assume that the result is true for sequences of halfmoves of length n . We will argue that it must also be true for sequences of $n+1$ halfmoves. We start with (a). Consider a piece $P \in \text{pos}$ and a square s such that $M_{P \rightarrow s} = 0$. We distinguish four cases depending on the piece type of P :

- If $P.\text{type} = \hat{\Delta}$, then there are at most two rules in Figure 6 of the form $M_{P \rightarrow s} \implies f$, the *move* rule, with $f_1 = \bigvee_{u \in \text{pred}_P(s)} M_{P \rightarrow u}$; and possibly the *not self-capture* rule, with $f_2 = C_{P'}$ (if there was a piece P' of the same color as P originally at s). Because variable

Mobility(pos):

Inputs: a position

Output: mobility solution $\{M_{P \rightarrow s}\}_{P \in \text{pos}, s \in \mathcal{S}}$

- 1: set $M_{P \rightarrow s} := 0$, $C_P := 0$, $R_s^c := 0$ for all $P \in \text{pos}$, $s \in \mathcal{S}$ and $c \in \{\text{w}, \text{b}\}$ and let \vec{X} be the state containing all these variables
- 2: set $M_{P \rightarrow P.\text{sq}} := 1$, for all $P \in \text{pos}$ ▷ Every piece can “move” to its current square
- 3: **for every** variable V in \vec{X} that is still set to 0 **do**
- 4: **if** for every rule from Figure 6 of the form “ $V \Rightarrow f$ ”, formula f evaluates to *true* on the current state of variables \vec{X} **then** set V to 1 in \vec{X}
- 5: repeat steps 3 and 4 until no new variables are set to 1
- 6: **return** $\{M_{P \rightarrow s}\}_{P \in \text{pos}, s \in \mathcal{S}}$

■ **Figure 7** Mobility algorithm.

$M_{P \rightarrow s}$ was not set to 1 in any iteration of steps 3-4 from Figure 7, clause f_1 (or clause f_2 when applicable) must evaluate to false on the final state. Applying the induction hypothesis, this means that in n halfmoves P did not have time to reach any of the P -predecessors of s , or that there was a piece P' of the same color as P initially on square s that did not have time to be cleared from s . Therefore, it is impossible for piece P to reach s in one more halfmove, as desired.

- If $P.\text{type} \in \{\text{♙}, \text{♚}, \text{♜}, \text{♝}\}$, the sliding pieces, we will argue that $M_{P \rightarrow s} = 0$ implies that for every sliding direction starting at s in which P can potentially move, it must hold that every square t in the direction (counting from s) satisfies $M_{P \rightarrow t} = 0$, until there is (possibly) a square t^* such that $C_{P'} = 0$ with $P'.\text{sq} = t^*$ and $P'.\text{side} = P.\text{side}$ for some $P' \neq P$. If we can show that, the induction hypothesis gives us that piece P cannot reach in n halfmoves any square in the relevant sliding directions of s , unless there is a piece of the same color as P between the square and s , which cannot be cleared in n halfmoves. In that case we can safely conclude that P cannot reach s in $n+1$ halfmoves, as desired. The above claim can be proved by induction on the sliding direction. Let t be the predecessor of s in a certain direction. As before, if $M_{P \rightarrow t}$ was not set to 1 in any iteration of steps 3-4 from Figure 7, we know that either $M_{P \rightarrow u} = 0$ for all P -predecessors of t (and in particular for the next square in the direction!) or $C_{P'} = 0$ for some piece of the same color as P with $P'.\text{sq} = t$. If the first case is true, we can continue the induction on the direction from the next square. If the second case is true, we can stop the induction since we have already found an ally blocker in the direction, as desired.
- If $P.\text{type} = \text{♔}$, we can proceed as before but this time there is one extra rule that comes into play, the *king attackers* rule. Again, by applying the induction hypothesis, we can conclude that in n halfmoves it was impossible for P to reach a P -predecessor of s , or that there was an ally piece on s that did not have time to be cleared, or that there was at least an enemy directly attacking s that did not have time to be cleared.⁶ Consequently, we conclude that it is impossible for king P to reach s in one extra move.
- Finally, if $P.\text{type} = \text{♟}$ a similar reasoning, now involving rules *pawn move* and *not self-capture*, applies. By the induction hypothesis we can conclude that in just n halfmoves, either (i) it was impossible for P to reach a P -predecessor of s ⁷ and it was impossible for

⁶ Direct enemies cannot be blocked, so the only way they stop attacking s is by moving or being captured.

⁷ For double pawn pushes we need to argue as for sliding pieces over the jumped square, but a similar technique applies.

pawn P to reach a promoting square and it was impossible for P to reach a P -capture-predecessor of s while at the same time having a (non-king) enemy piece reaching s ; or (ii) it was impossible for an ally initially at s to be cleared from s in n halfmoves. This makes it impossible for pawn P to reach s in one extra move.

To see (b), let s be an arbitrary square, let $c \in \{w, b\}$ and assume that $R_s^c = 0$. Note that variable R_s^c only appears in one rule from Figure 7, the *reachability* rule. Since R_s^c has not been activated, we conclude that $M_{P \rightarrow s} = 0$ for all $P \in \text{pos}$ such that $P.\text{side} = c$ and $P.\text{type} \neq \text{♔}$. As we have shown above, this means none of the non-king pieces of color c could have reached s in a sequence of $n+1$ halfmoves, as desired.

Finally, to see (c), let P be any piece in the position and assume that $C_P = 0$. Note that C_P only appears in the *clearance* rule. Because C_P has not been activated, we can conclude that $M_{P \rightarrow s} = 0$ for all $s \in \mathcal{S} \setminus \{P.\text{sq}\}$ and that $M_{P' \rightarrow P.\text{sq}} = 0$ for all $P' \in \text{pos}$ such that $P'.\text{side} \neq P.\text{side}$. As we have shown above, this means that there is no sequence of $n+1$ legal halfmoves after which piece P could have left square $P.\text{sq}$ and it is also impossible (in $n+1$ halfmoves) that any enemy piece could have reached $P.\text{sq}$, capturing P . We can conclude that piece P must still be at its initial square in pos after $n+1$ halfmoves as desired. ◀

The following is an immediate consequence of Lemma 8. Observe that, whenever $M_{P \rightarrow s} = 0$ for some $P \in \text{pos}$, $s \in \mathcal{S}$, the lemma guarantees that there is no sequence of legal moves that allows P to reach s , so the true solution to the mobility problem will also satisfy $M_{P \rightarrow s}^* = 0$.

► **Corollary 9.** *Let pos be a position where no player has castling rights and en passant is not possible. Let M^* be the true solution to the mobility problem of pos and let $M \leftarrow \text{Mobility}(\text{pos})$.*

$$M_{P \rightarrow s}^* \leq M_{P \rightarrow s} \quad \forall P \in \text{pos}, s \in \mathcal{S} .$$

3.2 Declaring unwinnability from a mobility problem solution

The second step of our static algorithm is described in Figure 8. It is based on the idea that a position is unwinnable if there is no good candidate mating square. Namely, if for every square in the board, (i) either the square cannot be reached by the intended loser's king, or (ii) the square cannot be attacked by the intended winner, or (iii) its adjacent squares (escaping squares for the intended loser's king) cannot all be blocked by defender pieces or covered/attacked by the intended winner pieces at the same time.

► **Lemma 10.** *The function induced by algorithm $\text{Unwinnable}^{\text{SS}}$ from Figure 8 is monotone in the following sense. For every pos , $c \in \{w, b\}$ and any two mobility solutions M, M' :*

$$\forall P \in \text{pos}, s \in \mathcal{S}. M_{P \rightarrow s} \leq M'_{P \rightarrow s} \implies \text{Unwinnable}^{\text{SS}}(\text{pos}, c, M) \geq^8 \text{Unwinnable}^{\text{SS}}(\text{pos}, c, M').$$

We refer to the full version of this paper for a formal proof.

► **Lemma 11.** *Let pos be a position and let $c \in \{w, b\}$. Let M^* be the true solution to the mobility problem on pos . If $\text{Unwinnable}^{\text{SS}}(\text{pos}, c, M^*)$ returns true, the position is indeed unwinnable for player c .*

⁸ By convention, *true* > *false*.

$\text{Unwinnable}^{\text{SS}}(\text{pos}, c, \{M_{P \rightarrow s}\}_{P \in \text{pos}, s \in \mathcal{S}})$:

Inputs: position, intended winner, solution to the mobility problem

Output: bool (*true* if position is declared unwinnable, *false* otherwise)

- 1: **if** *en passant* is possible **or** a player has *castling rights* in **pos** **then return false**
- 2: for every piece $P \in \text{pos}$, define $\text{region}(P) := \{s \in \mathcal{S} \mid M_{P \rightarrow s} = 1\}$
- 3: let K_c (resp. K_{-c}) be the intended winner's king (resp. intended loser's king)
- 4: set $\text{intruders} := \{P \in \text{pos} \mid P.\text{side} = c \wedge \text{region}(P) \cap \text{region}(K_{-c}) \neq \emptyset\}$
- 5: **if** $\exists P \in \text{intruders}$ with $P.\text{type} \neq \text{king}$ **then return false**
- 6: **if** $\exists P, P' \in \text{intruders}$ with $\text{color}(P.\text{sq}) \neq \text{color}(P'.\text{sq})$ **then return false**
- 7: for $P \in \text{pos}$, define $\text{att-region}(P) := \{s \in \mathcal{S} \mid \text{pred-capt}_P(s) \cap \text{region}(P) \neq \emptyset\}$
- 8: for $s \in \mathcal{S}$, let $\text{blockers}(s) := \{P \in \text{pos} \mid P.\text{side} \neq c \wedge \text{region}(P) \cap \clubsuit(s) \neq \emptyset\}^a$
- 9: for $s \in \mathcal{S}$, define $\text{assistants}(s) := \{P \in \text{pos} \mid P.\text{side} = c \wedge \text{att-region}(P) \cap \clubsuit(s) \neq \emptyset\}$
- 10: **if** $\exists s \in \text{region}(K_{-c})$ such that $|\text{blockers}(s)| + |\text{assistants}(s)| \geq |\clubsuit(s)|$ **and** $\exists P \in \text{pos}$ satisfying $s \in \text{att-region}(P) \wedge P.\text{side} = c$ **then return false**
- 11: **return true** ▷ The position must be unwinnable

^a We could design a more complete check that looks at all neighbours of s , but the condition on step 10 would be significantly more involved (to ensure monotonicity).

■ **Figure 8** Statically unwinnable algorithm, which may conclude that a position is unwinnable for an intended winner based on an admissible solution to the mobility problem.

Proof. Since $\text{Unwinnable}^{\text{SS}}$ did not return any value in step 5 or step 6, the set of pieces that can check the intended loser's king is empty or formed entirely by same-colored bishops. Furthermore, every square s in the board is such that: (i) the intended loser's king cannot reach it, or (ii) the square cannot be attacked by the intended winner, or (iii) its adjacent squares cannot not all be blocked by defenders or covered by attackers at the same time. Therefore, the intended loser will always have at least a legal move when they are in check. ◀

Since the mobility algorithm (Figure 7) always provides admissible solutions, since the static check (Figure 8) is sound on the true solution of the mobility problem, and because the static check is a (decreasing) monotone function, the composition of the mobility algorithm with the static check constitutes an algorithm for chess unwinnability that is **sound**.

► **Theorem 12.** *Let pos be a position and let $c \in \{w, b\}$. If $\text{Unwinnable}^{\text{SS}}(\text{pos}, c, \text{Mobility}(\text{pos}))$ outputs true, then the position is unwinnable for player c .*

Proof. Let $M \leftarrow \text{Mobility}(\text{pos})$ and let M^* be the true solution to the mobility problem on pos . If $\text{Unwinnable}^{\text{SS}}(\text{pos}, c, M) = \text{true}$, then *en passant* is not possible and no player has *castling rights* in pos (see step 1 of Figure 8), so we can apply Corollary 9 and conclude that $M^* \leq M$. We can now apply Lemma 10 and get:

$$\text{true} = \text{Unwinnable}^{\text{SS}}(\text{pos}, c, M) \leq \text{Unwinnable}^{\text{SS}}(\text{pos}, c, M^*) .$$

Hence we must have $\text{Unwinnable}^{\text{SS}}(\text{pos}, c, M^*) = \text{true}$. By virtue of Lemma 11, pos must be unwinnable for player c , as desired. ◀

Unwinnable^{full}(pos, c):

Inputs: position, intended winner

Output: Unwinnable or Winnable (definite solution to the chess unwinnability problem)

```

1: if true  $\leftarrow$  UnwinnableSS(pos, c, Mobility(pos)) then return Unwinnable
2: for every  $d \in \mathbb{N}$  do ▷ Iterative deepening
3:   set  $b_d \leftarrow$  Find-Helpmatec(pos, 0, maxDepth = d) (global nodesBound = bound(d))
4:   if  $b_d = \text{true}$  then return Winnable
5:   else if the search was not interrupted (in step 4 of Figure 5) then
6:     return Unwinnable

```

■ **Figure 9** Main routine for deciding chess unwinnability. It is based on our static algorithm (Figure 8) and our search routine (Figure 5) integrated via iterative deepening. Function `bound` must be increasing on d for the algorithm to be complete. The transposition table used by `Find-Helpmatec` should be initialized to empty at the beginning, but it can be shared between different calls to `Find-Helpmatec` in step 3. On the other hand, the global counter `cnt` should be initialized to 0 on every base call to `Find-Helpmatec` in step 3.

4 Unwinnability algorithms

We present our main routine for solving chess unwinnability in Figure 9. Our algorithm consists of a search of variations (Figure 5), preceded by a static analysis (Figure 8) on the given position. Such analysis will prevent the search routine from exploring large trees of variations exhaustively, whenever it concludes that the given position is unwinnable via our alternative and much lighter mechanism described in Section 3. Our main routine achieves:

- *Soundness*: Given that it combines a search of variations with our static algorithm for identifying blocked positions, it is sound by virtue of Theorem 12.
- *Completeness*: This is due to the fact that the search over variations is exhaustive for a sufficiently large `maxDepth` limit, and this in turn will be eventually reached during the iterative deepening loop (step 2 of Figure 9).
- *Efficiency*: As evidenced by the experimental results from Section 5, our algorithm is practical. We identified all the unfairly classified games from the Lichess Database [13].

4.1 An alternative quicker version of our algorithm

We propose a significantly more efficient chess unwinnability algorithm, inspired by the fact that most (if not all) unwinnable positions can be classified in the following two categories:

- *Imminently terminating* positions, where the tree of variations is very small. This is usually due to the existence of forced lines, which never end on a checkmate by the intended winner (all variations end in either stalemate, checkmate by the intended loser, or insufficient mating material for the intended winner.) An example is Position 15.
- *Blocked* positions, where players can maneuver over limited and disjoint regions of the board, what prevents their interaction (and thus any possible checkmate). (See Position 1.)

The quick version of our algorithm is described in Figure 10. It performs a depth-first search over the tree of variations and stops if a certain (small) depth D is reached, with the hope that it will be sufficient to exhaustively explore the tree of variations of imminently

Unwinnable^{quick}(pos, c):

Inputs: position, intended winner

Output: Unwinnable, Winnable, or PossiblyWinnable

- 1: advance the position as long as there is only one legal move
- 2: perform a depth-first search over the tree of variations of `pos` and interrupt the search if (i) checkmate is found for player `c` or (ii) depth `D` is reached
- 3: **if** checkmate was found on the previous search **then return** Winnable
- 4: **else if** the search was not interrupted **then return** Unwinnable
- 5: **else if** the position only contains pieces of type ♔, ♕, ♖ **and** there are no *semi-open files* in the position **then**
- 6: **if** `true` ← Unwinnable^{SS}(pos, c, Mobility(pos)) **then return** Unwinnable
- 7: **return** PossiblyWinnable ▷ Unwinnability could not be determined

■ **Figure 10** Quick routine for analyzing unwinnability.

terminating positions. Note that this search will be almost instantaneous in most positions, because it is interrupted as soon as depth D can be reached.

After that, and if the previous search did not conclude unwinnability, our quick algorithm simply performs a call to our static routine (Figure 8). But this is done only if the position is such that there exists no *semi-open files* (files with pawns of only one color) and the only existing pieces are kings, pawns and/or bishops. This heuristic is supported by the fact that positions that do not satisfy these properties will very likely be non-blocked.

Our quick algorithm is extremely light, requiring only a few microseconds on average per position. It is also sound, but not complete. However, as we detail in Section 5, with an (empirically chosen) depth bound of $D = 9$, all unfairly classified games from the Lichess Database except three were correctly identified by Unwinnable^{quick}.

5 Experimental results

We have implemented all the algorithms described in this work and evaluated their performance in real-world games from the Lichess Database [13]. Our source code is written in C++ and leverages the code of the open-source chess engine Stockfish [19] for move generation and chess-related functions. Our implementation is publicly available as open-source and can be found on this link: <https://github.com/miguel-ambrona/D3-Chess>.

The Lichess Database of standard rated games includes 3,099,534,127 games up to date. We have applied our algorithm from Figure 9 to the final position of all games that ended in a victory by timeout. In total, 981,467,875 games (31% of all games) were analyzed in about 88 hours of CPU time (323 μ s per position on average). All experiments were performed on a 3.5GHz Intel-Core i9-9900X CPU with 32GB of RAM, running Ubuntu 20.04 LTS.

Our analysis led to identifying a total of 84,100 games that were unfairly classified. Namely, games that were lost by the player who ran out of time, but their opponent could not have checkmated them by any possible sequence of legal moves. We refer to Appendix A for some remarkable positions of unfairly classified games; the remaining can be found on the following link (where our tool can also be tried interactively without installation): <https://chasolver.org>.

■ **Table 1** Performance of the Full and Quick versions of our algorithm when applied to all games from January 2022 that ended in a victory by timeout. (A total of 32,599,280 games.)

Full Algorithm (Figure 9)	vs	Quick Algorithm (Figure 10)
2700	average # positions per second	200,000+
370 μ s	average time per position	4.96 μ s
1270 μ s	standard deviation	9.06 μ s
141 ms	maximum time per position	586 μ s
2462 (100%)	unwinnable positions identified	2462 (100%)
3 h 21 min	total execution time	2 min 42 s

Our analysis identified *all* the unfairly classified games in the database. In all other games, the tool provided a checkmate sequence for the player who did not run out of time.

5.1 Comparison between the full and quick routines

Here we perform a comparison between our full algorithm, described in Figure 9 (we use $\text{bound}(d) = 10,000$)⁹, and our quicker version, described in Figure 10 (we use $D = 9$).

The latter is designed to be significantly faster, but it is not complete. Note that the quick version may terminate without having found a help mating sequence, declaring the position as “probably winnable”. Consequently, the quick version may fail to find all unwinnable positions. In fact, out of the exactly 84,100 games that were unfairly classified (identified with the full version of our tool), the quick version can identify 84,097 of them, missing only 3 positions in the entire database. These three positions are: `FKr42ZRT` (Position 12), `bKHPqNEw` (Position 13) and `f6c1lu7R` (Position 14).

In Table 1, we present a comparison of the performance of the two versions of our tool when analyzing all Lichess games from January 2022 that ended in a victory after a timeout. We also present in Figure 11 the execution times of this analysis (for the quick algorithm).

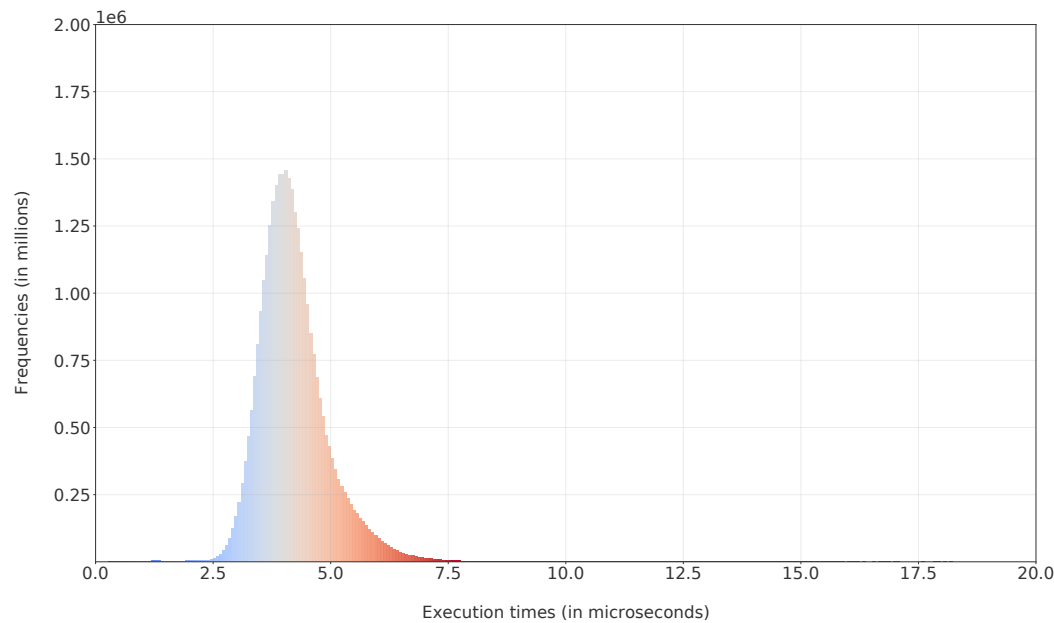
5.2 Conclusions and future work

We believe that our algorithms are suitable for practical use and in particular chess servers (and chess software) can leverage them to accurately classify games after a timeout, following Article 6.9 of the FIDE Laws of Chess [7]. Furthermore, given the results of Figure 11, chess servers could also apply our tools after every single move during games, to terminate games as soon as a dead position is achieved, correctly applying [7, Article 5.2.2].

Although our tool successfully solved all positions from the Lichess Database, we note that there exist artificial positions that are not efficiently captured by our logic and the tool could take a long time to analyze. Indeed, Position 12 with several additional black dark-squared bishops is an example.

A very interesting direction for future work would be to equip our mobility algorithm with extra rules (see Figure 6), that increase its scope. That way, the quick algorithm could potentially identify all unfairly classified games from the Lichess Database. Out of the three positions that our quick version cannot currently handle, Position 13 could be solved by setting $D = 14$ (instead of $D = 9$ as in the experiments above; see Figure 10), but that would

⁹ Note that `bound` should technically be an increasing function on d for the algorithm to be complete. In practice this is not necessary and a constant amount of 10K nodes per iteration seems empirically good.



■ **Figure 11** Quick analysis of 32,599,280 Lichess positions from January 2022.

significantly affect its performance. Positions 12 and 14, which look surprisingly similar, are harder to address. Our mobility algorithm would need to “understand” that although the pawn on g2 (of both positions) can be captured, that would leave White with no legal moves.

Finally, it would be very interesting to explore whether the ideas presented in this paper can be applied to other games that require similar analyses.

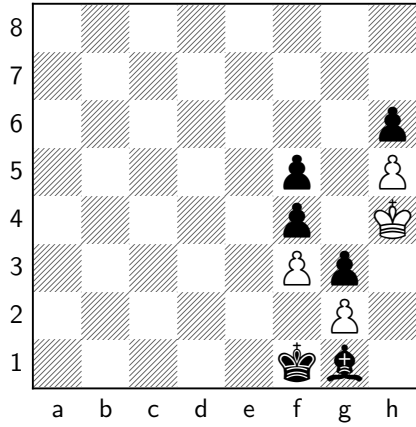
References

- 1 Lichess Blog. Announcing instant chess!, 2017. URL: <https://lichess.org/blog/WN7V-jAAAdH8ITR/announcing-instant-chess>.
- 2 Josh Brunner, Erik D. Demaine, Dylan Hendrickson, and Julian Wellman. Complexity of retrograde and helpmate chess problems: Even cooperative chess is hard, 2020. *arXiv: 2010.09271*.
- 3 Andrew Buchanan. Dead reckoning, 2001. URL: <http://anselan.com/tutorial.html>.
- 4 Andrew Buchanan. Dead reckoning: Castling & en passant. *StrateGems. U.S. Chess Problem Magazine*, 2001. URL: <http://anselan.com/DRSGtext.html>.
- 5 Chess.com. Game 13251713497. URL: <https://www.chess.com/game/live/13251713497>.
- 6 Daniel Dugovic. Helpmate solver, 2020. URL: <https://github.com/ddugovic/Stockfish/blob/master/src/types.h#L159>.
- 7 International Chess Federation. FIDE Laws of Chess Handbook, 2018. URL: <https://handbook.fide.com/chapter/E012018>.
- 8 Chess24. Community Feedback. Time ran out but insufficient material to mate - sufficient material, actually!, 2020. URL: <https://chess24.com/en/community/feedback/time-ran-out-but-insufficient-material-to-mate---sufficient-material--actually>.
- 9 Chess.com. Forums. Is the best move to let your time run out?, 2021. URL: <https://www.chess.com/forum/view/endgames/is-the-best-move-to-let-your-time-run-out>.
- 10 Robert A. Hearn. Games, puzzles, and computation. PhD thesis, Massachusetts Institute of Technology, 2006.
- 11 Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985. doi:10.1016/0004-3702(85)90084-0.

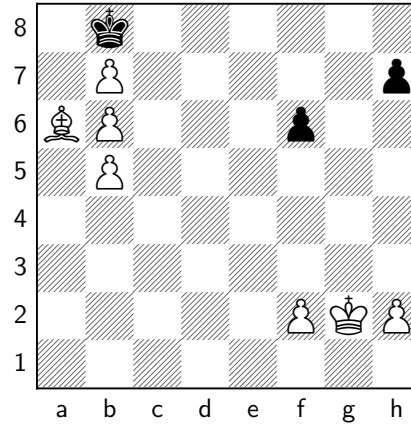
2:18 A Practical Algorithm for Chess Unwinnability

- 12 François Labelle. Illegal moves by grandmasters, 2011. URL: <https://wismuth.com/chess/illegal-moves.html>.
- 13 Lichess. Open database, 2022. URL: <https://database.lichess.org/>.
- 14 J.I. Minchin, J. Zukertort, and W. Steinitz. *London International Chess Tournament 1883*. ISHI Press, 2012. URL: <https://books.google.es/books?id=QEqyNAEACAAJ>.
- 15 Ornicar/lila. Issue number 6804. Detect all positions without a legal sequence of moves to checkmate, 2020. URL: <https://github.com/ornicar/lila/issues/6804#issuecomment-724002709>.
- 16 Chess.com. The Rules of Chess. My opponent ran out of time. Why was it a draw? URL: <https://support.chess.com/article/268-my-opponent-ran-out-of-time-why-was-it-a-draw>.
- 17 Chess.com. The Rules of Chess. What does ‘insufficient mating material’ mean? URL: <https://support.chess.com/article/128-what-does-insufficient-mating-material-mean>.
- 18 Viktoras Paliulionis. Helpmate analyzer. URL: <http://helpman.komtera.lt>.
- 19 Stockfish. Open source chess engine, 2022. URL: <https://stockfishchess.org/>.
- 20 Ronald Turnbull. Dead reckoning: a new discovery in problem chess. *The Problemist*, pages 140–141, July 2001. URL: <http://anselan.com/DRtPturnbull.html>.
- 21 Jakob Varmose. Deadposition2. URL: <https://github.com/jakobvarmose/deadposition2>.
- 22 Wikipedia. Helpmate, 2021. URL: <https://en.wikipedia.org/wiki/Helpmate>.

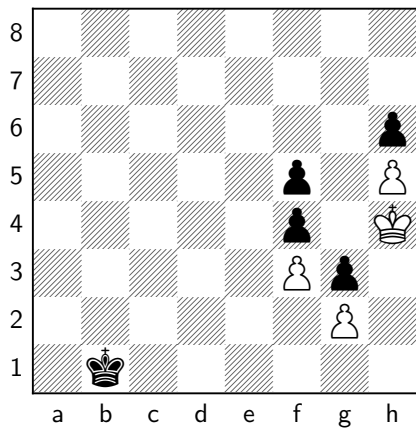
A Positions from unfairly classified Lichess games



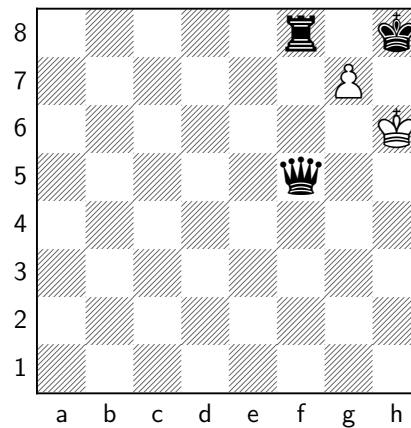
Position 12 Lichess game FKr42ZRT.



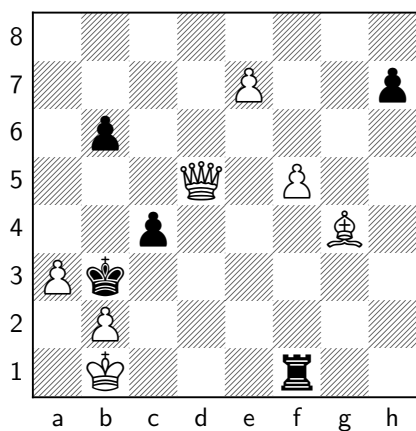
Position 13 Lichess game bKHPqNEw.



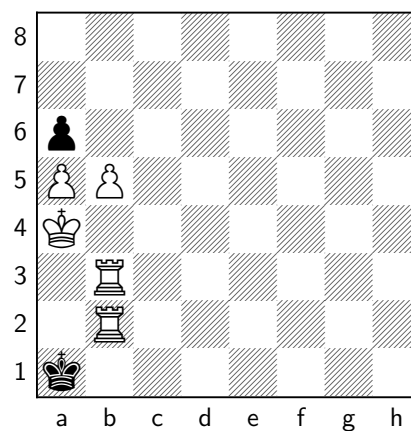
Position 14 Lichess game f6c1lu7R.



Position 15 Lichess game 0awUhnkq.

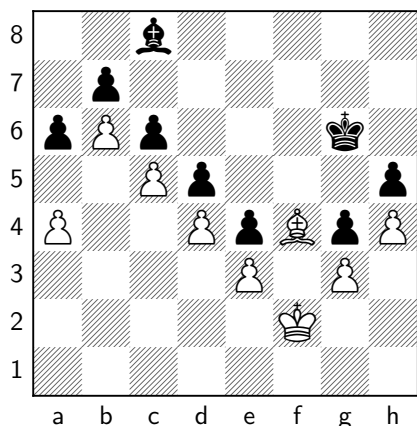


Position 16 Lichess game ZNBhS4pz.

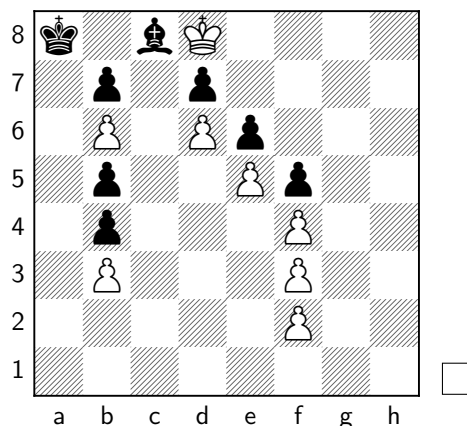


Position 17 Lichess game 3y8e8sCm.

B Puzzles: Are the following positions dead?



Position 18 *Is this a dead position?*
Lichess game QRvIMh3z.

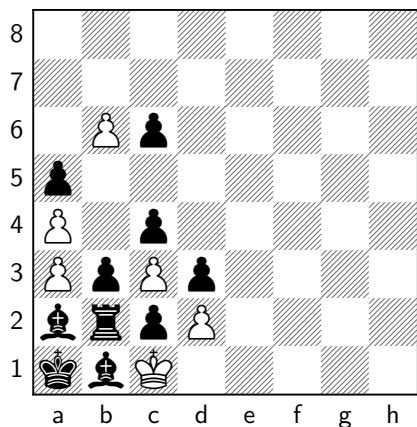


Position 19 *Is this a dead position?*
Miguel Ambrona (Spain).

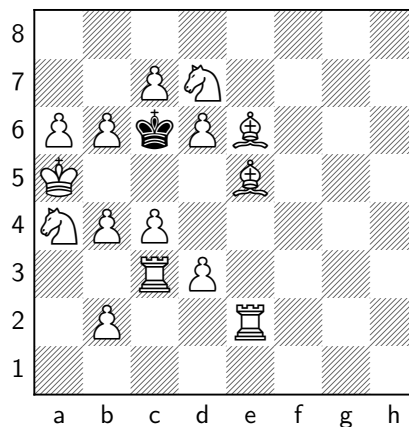
C Original compositions based on dead reckoning

We present two original compositions, by Andrew Buchanan and Andrey Frolkin, that the authors kindly offered to be included in this article. (Solutions available in the full version.)

In both problems the objective is to determine what the last move was. Solving them requires a clever retrograde analysis based on the fact that, by virtue of the FIDE Laws of Chess [7, Article 5.2.2], a game is finished as soon as a dead position is reached and no more moves are permitted. This genre of chess compositions is known as *dead reckoning* and can lead to unique motifs that cannot be enforced otherwise.



Position 20 *It is Black's turn. Last move?*
A. Buchanan (Singapore).
Original.



Position 21 *Whose turn is it? Last move?*
A. Frolkin (Ukraine) & A. Buchanan (Singapore).
Original.

Pushing Blocks via Checkable Gadgets: PSPACE-Completeness of Push-1F and Block/Box Dude

Joshua Ani ✉

Massachusetts Institute of Technology, Cambridge, MA, USA

Lily Chung ✉ 

Massachusetts Institute of Technology, Cambridge, MA, USA

Erik D. Demaine ✉ 

Massachusetts Institute of Technology, Cambridge, MA, USA

Yevhenii Diomidov ✉

Massachusetts Institute of Technology, Cambridge, MA, USA

Dylan Hendrickson ✉ 

Massachusetts Institute of Technology, Cambridge, MA, USA

Jayson Lynch ✉

Cheriton School of Computer Science, University of Waterloo, Canada

Abstract

We prove PSPACE-completeness of the well-studied pushing-block puzzle Push-1F, a theoretical abstraction of many video games (first posed in 1999). We also prove PSPACE-completeness of two versions of the recently studied block-moving puzzle game with gravity, Block Dude – a video game dating back to 1994 – featuring either liftable blocks or pushable blocks. Two of our reductions are built on a new framework for “checkable” gadgets, extending the motion-planning-through-gadgets framework to support gadgets that can be misused, provided those misuses can be detected later.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases gadgets, motion planning, hardness of games

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.3

Acknowledgements This work was initiated during extended problem solving sessions with the participants of the MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.892) taught by Erik Demaine in Spring 2019. We thank the other participants for their insights and contributions.

We would like to thank our reviewers for their detailed and useful feedback.

We would like to thank Aaron Williams for useful discussion including how to restructure the paper and how to better present the results and checkable gadget framework.

Figures produced using SVG Tiler (<https://github.com/edemaine/svgtiler>), diagrams.net, and Inkscape.

1 Introduction

In the *Push* family of pushing-block puzzles, introduced by O’Rourke in 1999 [14], a 1×1 agent must traverse a unit-square grid, some cells of which have a “block”, from a given start location to a given target location. Refer to Figure 1. In *Push-k* [7, 8]), the agent’s move (horizontal or vertical by one square) can *push* up to k consecutive blocks by one square, provided that there is an empty square on the other side. In the *-F* variation (described in [8, 14] but first given notation in [10]), some of the blocks are *fixed* in the grid, meaning they cannot be traversed or pushed by the agent or other blocks. Push-1F has the same allowed moves as the famous *Sokoban* puzzle video game, invented in 1982 and analyzed at



© Joshua Ani, Lily Chung, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, and Jayson Lynch;

licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 3; pp. 3:1–3:30

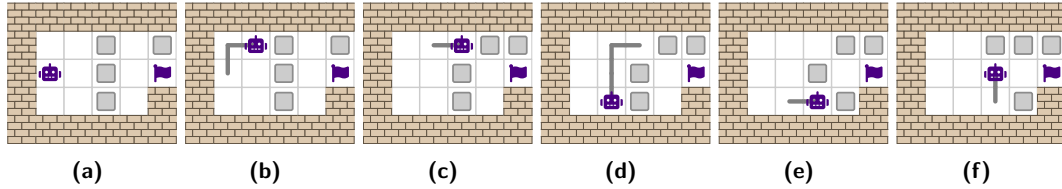
Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

3:2 Pushing Blocks via Checkable Gadgets

FUN 1998 [6], but crucially Push-1F’s goal is for the agent to reach a target location, which is much simpler than Sokoban’s “storage” goal where the blocks must be pushed to certain locations.



■ **Figure 1** Sample Push-1F puzzle and solution sequence. In steps (c) and (e), for example, the agent cannot push right again. The agent is drawn as a robot head; the traversed path between steps is drawn as a gray line; pushable blocks are drawn as boxes; fixed blocks are drawn as brick walls; and the goal location is drawn as a flag. *Robot and flag icons from Font Awesome under CC BY 4.0 License.*

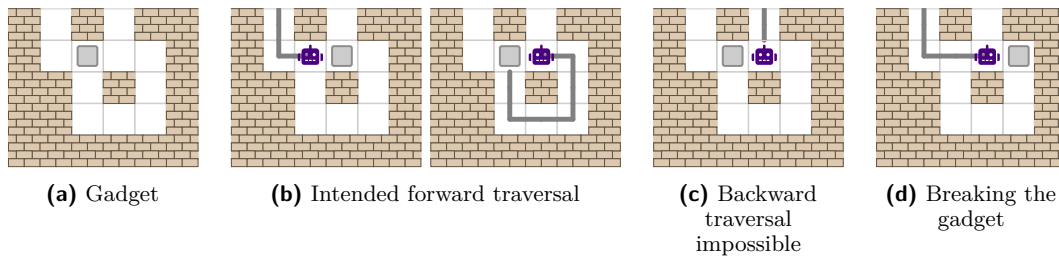
In this paper, we prove that Push-1F is PSPACE-complete, settling an open problem from [8, 10], and complementing previous PSPACE-hardness for Push- k F for $k \geq 2$ from 20 years ago [10].

To gain some intuition about why Push-1F is so difficult to prove PSPACE-hard, and how we surmount that difficulty, consider the attempt at a “diode” gadget in Figure 2. The goal of this gadget is to allow repeated traversals from the left entrance to the right (as in Figure 2b), while always preventing “backward” traversal from the right to the left (as in Figure 2c). But given the opportunity for forward traversal, the agent can instead “break” the gadget to allow future forward and backward traversal (as in Figure 2d).

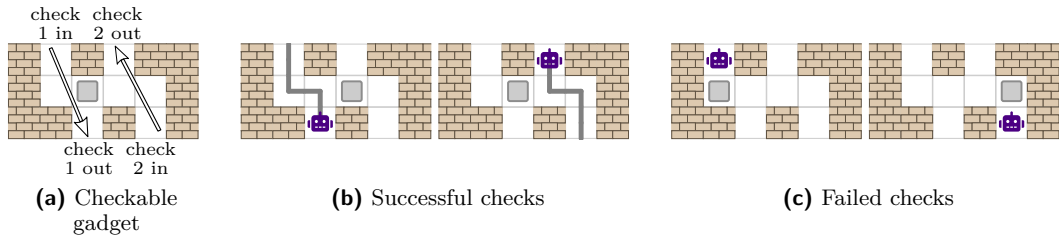
To solve this problem, we introduce the idea of a *checkable gadget* where, after the agent completes the “main” gadget traversal puzzle, the agent is forced (in order to solve the overall puzzle) to do a specified sequence of *checking* traversals of every gadget, all of which must succeed in order to solve the overall puzzle. If designed well, these checking traversals can detect whether a gadget was previously “broken”, and allow traversal only if not. In the case of Figure 2, one can think of the gadget as a four-location gadget (the top three rows) which has its bottom two locations connected. This four-location gadget is “checkable”: we will demand that, after completing the main puzzle, the agent follows the two checking traversals shown in Figure 3. In order for these checking traversals to both be possible, the agent cannot push the block into either corner, preventing the agent from breaking the gadget during the main gadget traversal puzzle. We call this process of removing broken states from a gadget by demanding that the checking traversals remain legal *postselection*.¹

We develop a general framework of checkable gadgets that enable a reduction to focus on the main gadget traversal puzzle, assuming all gadgets remain unbroken (i.e., the checking traversals remain possible at the end), while the framework ensures that the agent makes these checking traversals at the end (without other unintended traversals). This framework builds upon the motion-planning-through-gadgets framework introduced at FUN 2018 [9] and developed further in [2, 3, 11–13] to handle checkable gadgets.

¹ In quantum computing, for example, “postselection is the power of discarding all runs of a computation in which a given event does not occur” [1]. In probability theory, postselection is equivalent to conditioning on a particular event.



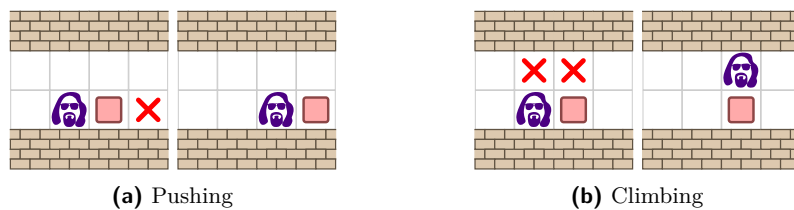
■ **Figure 2** A broken Push-1F diode gadget.



■ **Figure 3** The top three rows of the Push-1F diode gadget of Figure 2, as a checkable gadget. The checking traversals are “check 1 in \rightarrow check 1 out” and “check 2 in \rightarrow check 2 out”, denoted by the hollow arrows.

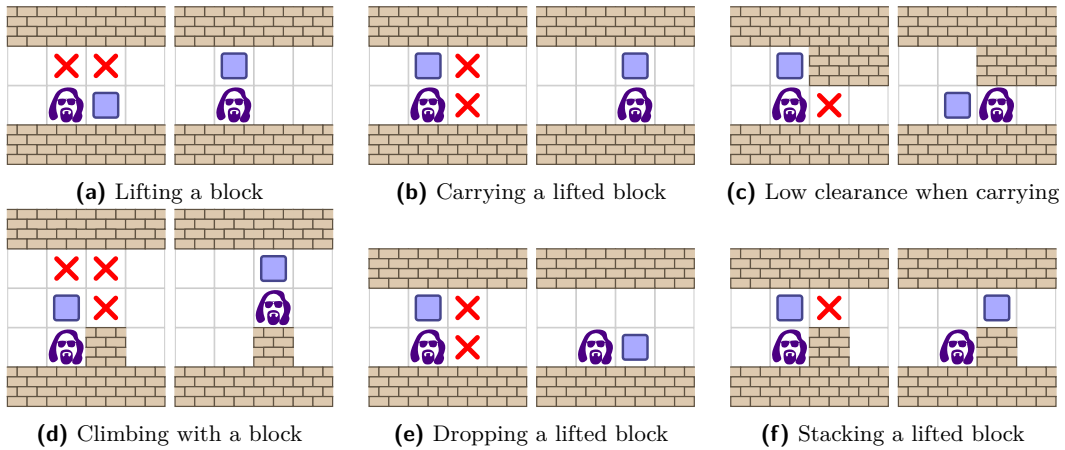
We also apply our framework to resolve the complexity of *Block Dude*, a puzzle video game made over a dozen times on many platforms, originally under the name “Block-Man 1” (Soleau Software, 1994); see [5] for details. Barr, Chung, and Williams [5] recently formalized this game’s mechanics, along with several variations, and proved them all NP-hard. In this paper, we prove PSPACE-completeness of three of these variations, including the original video game mechanics:

1. *BoxDude* is like Push-1 but where all pushable blocks and the agent experience gravity, falling straight down whenever they have blank spaces below them. In addition to moving horizontally left or right, the agent can “climb” on top of horizontally adjacent blocks (be they pushable or fixed), provided the square above the agent is empty. See Figure 4.



■ **Figure 4** Mechanics for BoxDude, with pushable boxes shown in red. Squares marked with a red \times must be empty for the move to be possible.

2. In *BlockDude* (as in the Block Dude video games), blocks cannot be pushed; instead, nonfixed blocks can be “picked up” by the agent from a horizontally adjacent position to the position immediately above the agent, provided that that position and the intermediate diagonal position are empty. See Figure 5. The agent can then carry one such block to another location (provided the ceiling offers height-2 clearance), and then drop the block



■ **Figure 5** Mechanics for BlockDude, with liftable blocks shown in blue. Squares marked with a red \times must be empty for the following move to be possible.

in front of them, again provided that that position and the intermediate diagonal position are empty.² They can also stack the block on top of another block. If the agent tries to move past a low ceiling while carrying a block, the block will be dropped behind them.

3. In *BloxDude*, nonfixed blocks can be pushed (as in *BoxDude*) and/or picked up (as in *BlockDude*).

The other variations described in [5], called \dots Duderino instead of \dots Dude, change the goal of a puzzle to place the k nonfixed blocks into k specified storage locations, as in Sokoban. We leave open the complexity of *BoxDuderino*, *BlockDuderino*, and *BloxDuderino*.

All of the games we consider can easily be simulated in polynomial space, and thus are in $\text{NPSpace} = \text{PSPACE}$ by Savitch’s Theorem. Proving PSPACE -hardness is much more complicated, and is the goal of this paper.

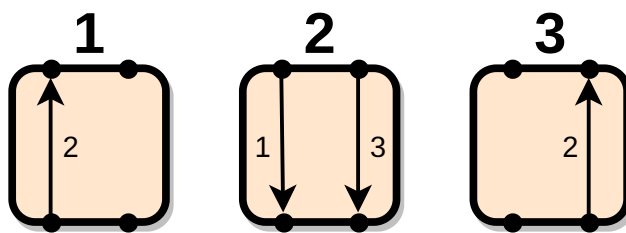
The rest of this paper is organized as follows. In Section 2, we review the motion-planning-through-gadgets framework. In Section 3, we prove that *BlockDude* and *BloxDude* are PSPACE -complete using standard reductions from motion-planning-through-gadgets. In Section 4, we develop our checkable gadget framework. In Section 5, we prove that *BoxDude* is PSPACE -complete using our checkable gadget framework. In Section 6, we prove that *Push-1F* is PSPACE -complete via a much more involved application of our checkable gadget framework.

2 Gadgets Framework

The *motion-planning-through-gadgets framework* is an abstract motion planning model used for proving computational hardness results. Here we give the definitions and results we need for this paper; see [11–13] for more details.

A *gadget* G consists of a finite set $Q(G)$ of *states*, a finite set $L(G)$ of *locations* (entrances/exits), and a finite set $T(G)$ of *transitions* of the form $(q, a) \rightarrow (r, b)$ where $q, r \in Q(G)$ are states and $a, b \in L(G)$ are locations. The transition $(q, a) \rightarrow (r, b) \in T(G)$

² A complication in some implementations of the game is that the agent can only pick up or drop the block in front of them, with the agent’s orientation determined by their previous move. (Some implementations allow turning around in place.) This detail will not affect our results.



■ **Figure 6** State diagram for the locking 2-toggle gadget. Each box represents the gadget in a different state, in this case labeled with the numbers 1, 2, 3. Dots represent the four locations of the gadget. Arrows represent transitions in the gadget and are labeled with the states to which those transitions take the gadget. In state 2, the agent can traverse either tunnel going down, which blocks off both downward traversals until the agent reverses that traversal.

means that an agent can *traverse* the gadget when it is in state q by entering at location a and exiting at location b which changes the state of the gadget from q to r . We use the notation $a \rightarrow b$ for a traversal by the agent that does not specify the state of the gadget before or after the traversal. A *traversal sequence* $[a_1 \rightarrow b_1, \dots, a_k \rightarrow b_k]$ on the locations $L(G)$ is *legal* from state s_0 if there is a corresponding sequence of transitions $[(a_1, s_0) \rightarrow (b_1, s_1), \dots, (a_k, s_{k-1}) \rightarrow (b_k, s_k)]$, where each start state of each transition matches the end state of the previous transition (s_0 for the first transition). We define gadgets in figures using a *state diagram* which gives, for each state $q \in Q$, a labeled directed multigraph $G_q = (L(G), E_q)$ on the locations, where a directed edge (a, b) with label r represents the transition $(q, a) \rightarrow (r, b) \in T(G)$.

Figure 6 shows the state diagram of a key gadget called the *locking 2-toggle* [11]. This gadget has four locations (drawn as dots) and three states 1, 2, 3. The central state, 2, allows for two different transitions. Each of those transitions takes the gadget to a different state, from which the only transition returns the agent to the prior location and returns the gadget to state 3.

A *system of gadgets* S consists of a set of gadgets, an initial state for each gadget, and a *connection graph* on the gadgets’ locations. If two locations a, b of two gadgets (possibly the same gadget) are connected by a path in the connection graph, then an agent can traverse freely between a and b (outside the gadgets).³ We call edges of the connection graph *hallways*, and for clarity in figures, we add extra vertices to the connection graph called *branching hallways*, which we can equivalently think of as a one-state gadget that has transitions between all pairs of locations. A *system traversal* is a sequence of traversals $a_1 \rightarrow b_1, \dots, a_k \rightarrow b_k$, each on a potentially different gadget in S , where the connection graph has a path from b_i to a_{i+1} for each i . We write such a traversal as $a_1 \rightarrow^* b_k$, ignoring the intermediate locations. A system traversal is *legal* if the restriction to traversals on a single gadget G is a legal traversal sequence from the initial state of G assigned by S , for every G in S . Note that gadgets are “local” in the sense that traversing a gadget does not change the state (and thus traversability) of any other gadgets.

The *reachability* or *1-player motion planning* problem with a finite set of gadgets \mathcal{G} asks whether there is a legal system traversal $s \rightarrow^* t$ from a given start location s to a given goal location t (by a single agent) in a given system of gadgets S , which contains only gadgets from \mathcal{G} .

³ Equivalently, we can think of identifying locations a and b topologically, thereby contracting the connected components of the connection graph. Alternatively, if we think of the gadgets as individual “levels”, then the connection graph is like an “overworld” map connecting the levels together.

Because we are working with 2D games, we also consider *planar motion planning*, where every gadget additionally has a specified cyclic ordering of its vertices and the system of gadgets is embedded in the plane without intersections. More precisely, a system of gadgets is *planar* if the following construction produces a planar graph: (1) replace each gadget with a wheel graph, which has a cycle of vertices corresponding to the locations on the gadget in the appropriate order, and a central vertex connected to each location; and (2) connect locations on these wheels with edges according to the connection graph. In *planar reachability*, we restrict to planar systems of gadgets. Note that this definition allows rotations and reflections of gadgets, but no other permutation of their locations.

2.1 Simulation

To define a notion of gadget simulation, we can think of a system of gadgets as being characterized by its set of possible traversal sequences (as formalized by the related *gizmo* framework of [12]).

► **Definition 1.** A *(local) simulation* of a gadget G in state q consists of a system S of gadgets, together with an injective function m mapping every location of G to a distinct location in S , such that a traversal sequence $[a_1 \rightarrow b_1, \dots, a_k \rightarrow b_k]$ on the locations in G is legal from state q if and only if there exists a sequence of system traversals $m(a_1) \rightarrow^* m(b_1), \dots, m(a_k) \rightarrow^* m(b_k)$ that is legal in the sense that the concatenation of the restrictions of the system traversals $m(a_i) \rightarrow^* m(b_i)$ to traversals on a single gadget G is a legal traversal sequence for G from the initial state of G assigned by S , for every G in S .

A *planar simulation* of a gadget G in state q is a simulation (S, m) where S is furthermore a planar system of gadgets, and the cyclic order of locations of G must map via m to locations in cyclic order around the outside face of S .

A [planar] simulation of an entire gadget G consists of a [planar] simulation of G in state q , for all states $q \in Q(G)$, that differ only in their assignments of initial states. A finite set \mathcal{G} of gadgets [planarly] *simulates* a gadget G if there is a [planar] simulation of G using only gadgets in \mathcal{G} .

These definitions of simulation imply that, if we take a larger system of gadgets and replace each instance of gadget G with the system S using the appropriate initial states (matching up locations that correspond via m), then the entire system behaves equivalently. In particular, this substitution preserves reachability of locations from one another. Furthermore, if the larger system and the simulation are both planar, then the full resulting system is planar. More formally:

► **Lemma 2.** Let H be a gadget, and let \mathcal{G} and \mathcal{G}' be finite sets of gadgets. If \mathcal{G} [planarly] simulates H , then there is a polynomial-time reduction⁴ from [planar] reachability with $\{H\} \cup \mathcal{G}'$ to [planar] reachability with $\mathcal{G} \cup \mathcal{G}'$.

2.2 Known Hardness Results

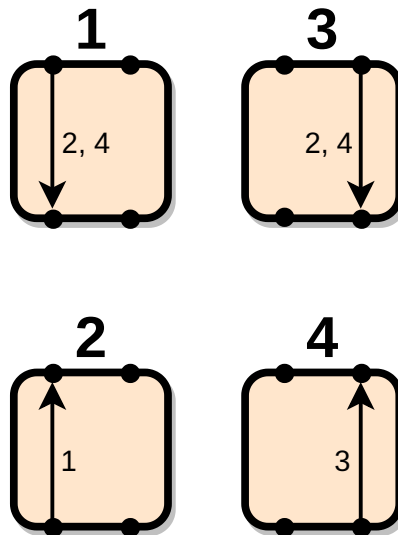
We can now formally state the problems we will reduce from in this paper.

In Section 3, we use the locking 2-toggle to show PSPACE-completeness of BlockDude puzzles.

⁴ Throughout this paper, reductions are *many-one/Karp*: a reduction from A to B maps an instance of A to an equivalent (in terms of decision outcome) instance of B .

► **Theorem 3** ([11, Theorem 10]). *Planar reachability with any interacting- k -tunnel reversible deterministic gadget is PSPACE-complete.*

The locking 2-toggle is an example of an interacting- k -tunnel reversible deterministic gadget [11] and thus we obtain PSPACE-completeness of planar reachability with the locking 2-toggle. We recommend readers interested in this more general dichotomy to refer to [11].



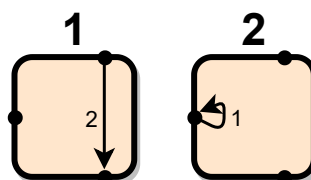
■ **Figure 7** State diagram for a nondeterministic locking 2-toggle. From state 1, the left tunnel can be traversed so as to leave the gadget in either state 2 or state 4. Formally, in the multigraph for state 1 there are two different edges, one labeled 2 and the other labeled 4.

We also use the nondeterministic locking 2-toggle shown in Figure 7. This is used in Section 5 to show PSPACE-completeness of BoxDude puzzles. Its behavior resembles that of the locking 2-toggle, but because it is not deterministic it is not covered by the prior theorem.

► **Theorem 4** ([2, Theorem 3.1]). *Planar reachability with the nondeterministic locking 2-toggle is PSPACE-complete.*

The final main gadget we will make use of is a type of self-closing door shown in Figure 8. This gadget will be used in our result on Push-1F in Section 6.

► **Theorem 5** ([3, Theorem 4.2]). *Planar reachability with any normal or symmetric self-closing door is PSPACE-hard.*

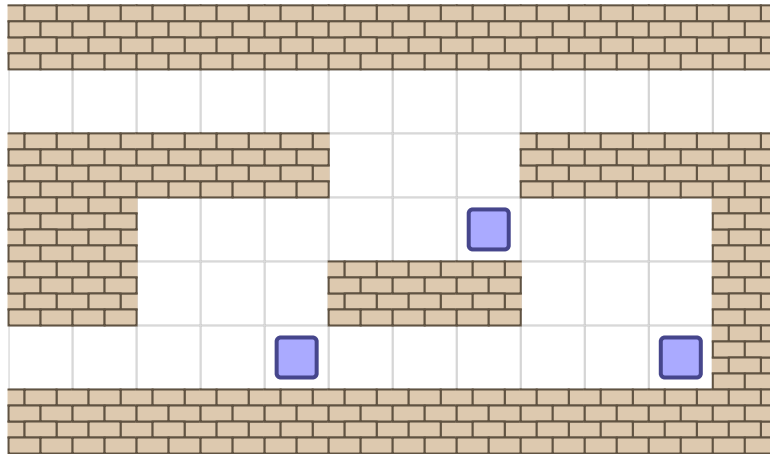


■ **Figure 8** State diagram for the directed open-optional self-closing door. The door must be opened by visiting its opening location before every traversal.

3 BlockDude and BloxDude are PSPACE-complete

In this section, we show that BlockDude and BloxDude are PSPACE-complete using a reduction from planar reachability with locking 2-toggles, shown in Figure 6, which is PSPACE-complete by Theorem 3. Recall from Section 1 in this model blocks can be picked up by BlockDude from an adjacent square. BloxDude allows both picking up and pushing blox, and the reduction will be a small modification to the BlockDude proof.

We will build hallways allowing the player to move between connected locations on gadgets. To connect more than two locations, we need a branching hallway, which is shown in Figure 9. This allows the player to freely move between any of the three entrances.

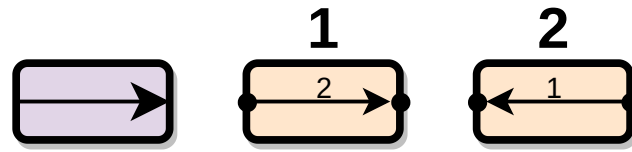


■ **Figure 9** A branching hallway for BlockDude. Blue squares represent blocks (which can be picked up).

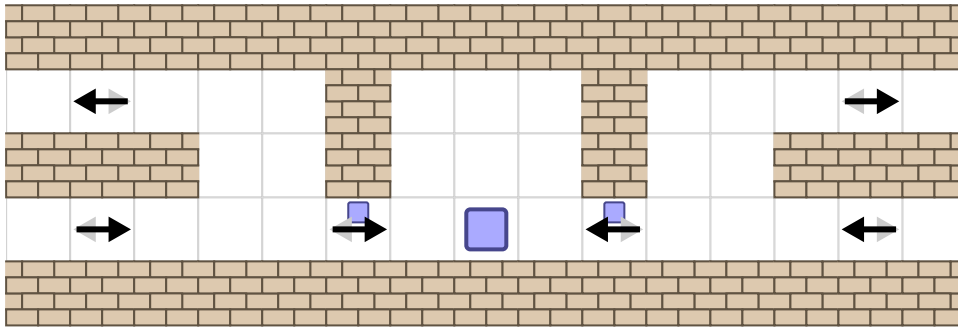
We now describe how the player can use the branching hallway in a way that always lets them move between any of its entrances. Whenever the player is outside the branching hallway, both bottom blocks will be in their original positions, and the top block will be somewhere on the middle platform, depending on the most recently taken exit. When the player arrives at the branching hallway, they will first move the top block to the right side of the middle platform (the position in Figure 9). The only case where this is nontrivial is when the player enters at the bottom with the top block on the left. In this case, the player can go under the middle platform and climb up from the right by moving both bottom blocks. Then they can pick up the top block and step back down on the right, causing the carried block to fall onto the right end of the middle platform. Finally, they can reset the bottom blocks and return to the bottom entrance. Once the top block is on the right, the player can take whichever exit they need. If they take the top left exit, they will move the top block to the left first.

To embed an arbitrary planar graph in BlockDude, we also need to be able to turn hallways and in particular to make vertical hallways despite gravity. Fortunately, the branching hallway in Figure 9 can achieve both goals. If we ignore the top-right entrance, the agent can turn around and make some vertical progress. By chaining these switchbacks in alternating orientation, we can build an arbitrarily tall vertical hallway.

To complete the proof of PSPACE-hardness, we only need to build a locking 2-toggle. We will construct the locking 2-toggle out of simpler pieces, as shown in Figure 11. The simpler pieces are two kinds of 1-toggle: one just for the player, and one that the player can carry



■ **Figure 10** Icon and state diagram for the 1-toggle. Leftwards and rightwards traversals must alternate.



■ **Figure 11** The schematic for our locking 2-toggle for BlockDude. Arrows with a faded backward arrowhead are 1-toggles. Only the player can go through the 1-toggle unless it has a block icon above the arrow, in which case the player can carry a block through.

a block through. The state diagram for a 1-toggle is given in Figure 10. When the player arrives at (say) the bottom left entrance, they can grab the block in the middle and bring it to the left side, and use it to reach the top left entrance. With the block stuck on the left, the right side cannot be traversed until the player returns to the top left, puts the block back, and exits the bottom left. The player cannot move through this gadget in any way not allowed by a locking 2-toggle. They may leave the block on the left side when they exit the bottom left, but this does not achieve anything; it only prevents them from traversing the right side.

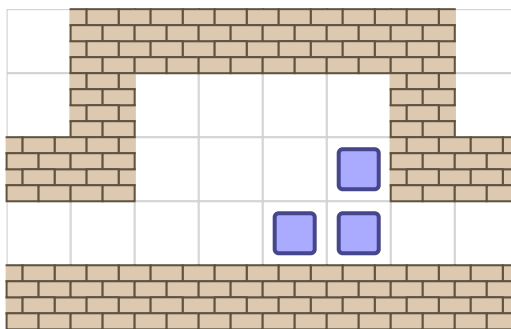
Our 1-toggle for just the player is shown in Figure 12. In the state shown, the player can not enter on the right. If they enter on the left, they can move the blocks to exit on the right, but in doing so must block the left entrance. Because of the 1-high hallways, the player can not bring a block through this gadget.

The 1-toggle that lets the player carry a block through is more complicated, and is shown in Figure 13. If the player enters on the left with or without a block, they can get to the right as follows:

- Move the top staircase to the right, so they can climb all the way down.
- Move the top staircase and then the bottom staircase to a single pile in the bottom left corner.
- Move the single pile to the bottom right corner.
- Use three blocks to build a staircase to the middle platform on the right, and move the rest of the blocks up to that platform.
- Use another three blocks to build a staircase to the right exit.

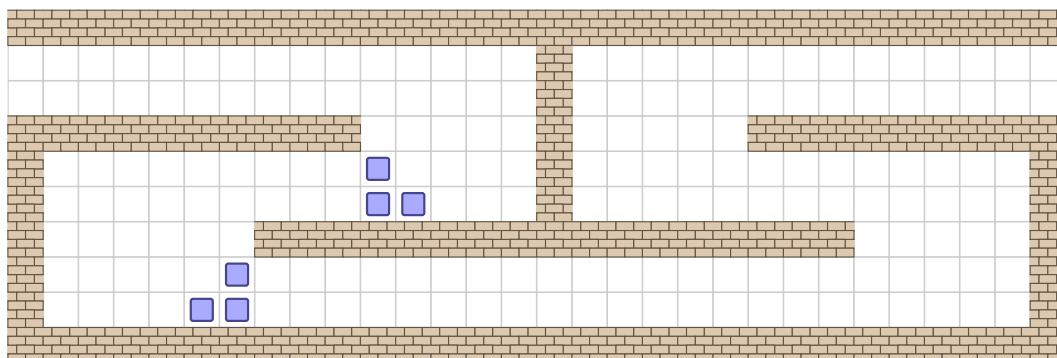
To reach either exit, there must be at least three blocks on the bottom level to form a staircase to the middle platform, and three blocks on the middle platform to form a staircase to the exit. In particular, six blocks must stay inside the gadget, so the player can leave with

3:10 Pushing Blocks via Checkable Gadgets



■ **Figure 12** A 1-toggle for BlockDude, currently traversable from left to right.

a block only if they brought one with them. If the player tries to enter the side opposite the one they most recently exited, they will be blocked by both staircases and unable to get across the gadget.



■ **Figure 13** A 1-toggle for BlockDude that lets the player carry a block through it, currently traversable from left to right.

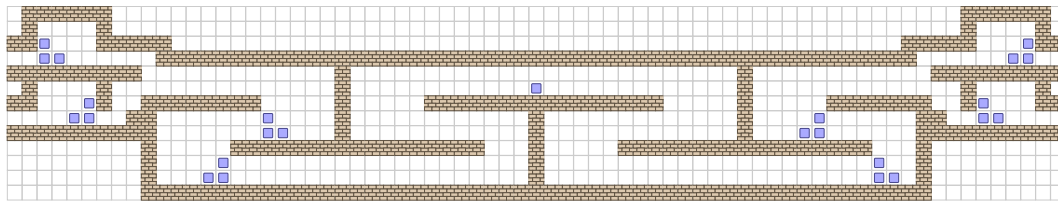
This 1-toggle might break if the player brings several additional blocks to it, but it will never be possible to bring more than one additional block because of the structure of our locking 2-toggles.

With these components, we can fill in our schematic for a locking 2-toggle (Figure 11), which we show in full in Figure 14. To summarize: the player can enter on either side, at the lower entrance. They can get to the block in the center, but must return to the side they came from. Then they can use this block to reach the top exit on the same side. This makes the center block inaccessible from the other side, so the other side cannot be traversed until the player comes back in the opposite direction and returns the center block.

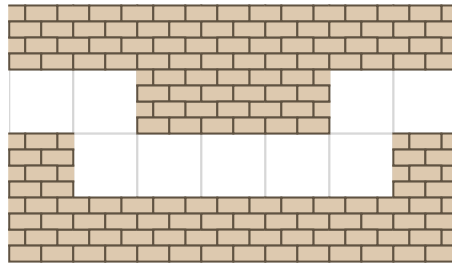
3.1 BloxDude is PSPACE-complete

In this section we discuss how to adapt the prior proof for BlockDude puzzles to work for blox which can both be picked up and pushed. All the valid traversals from our BlockDude constructions remain and we only need to prevent unwanted movement of the blox due to pushing.

First, whenever there is a hallway in which a blox should not be able to be moved, such as all three hallways from the branching hallway, we add a step in the hallway, as shown in Figure 15. Thus the blox cannot be carried and if it is pushed to the step it will become stuck.

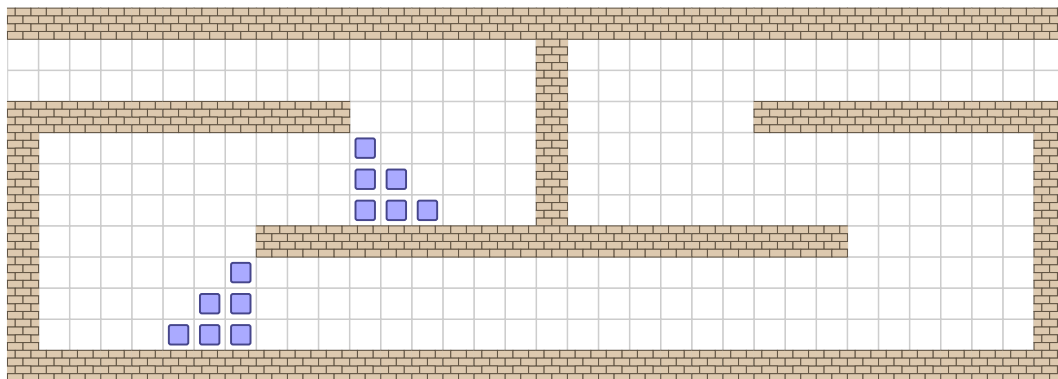


■ **Figure 14** The full locking 2-toggle for BlockDude, combining Figures 11, 12, and 13.



■ **Figure 15** A blox cannot be moved through this hallway.

Next we show how to adapt the 1-toggle with block traversal so it works in this setting. This is given in Figure 16. The three-block-tall staircases ensure that bringing a single blox from the wrong direction does not allow deconstructing a staircase from behind. In particular, the middle layer has two blox in a row which cannot be pushed and thus one extra blox will not enable the Dude to deconstruct the staircase from that side.



■ **Figure 16** A 1-toggle for BloxDude that lets the player carry a block through it, currently traversable from left to right.

We also need a regular 1-toggle, and the construction in Figure 12 can be broken in the blox model. Luckily we have a hallway that prevents blox from being carried or pushed through it, so we can add such a hallway to each end of the gadget in Figure 16 preventing extra blox from entering or leaving. This yields a regular 1-toggle which does not permit blox to pass through.

Once we have the prior two gadgets, it is clear the locking 2-toggle in Figure 11 will still work in the blox model, giving the desired PSPACE-hardness result.

4 Checkable Gadget Framework

In this section, we introduce a new extension to the gadgets framework which will be used in the rest of the paper. This extension allows us to indirectly construct a gadget G by first constructing a “checkable” version of G , and then using “postselection” to obtain G . The checkable G behaves identically to G except that the agent can make undesired traversals into “broken” states which prevent later “checking” traversals. The postselection operation removes these possibilities by guaranteeing that the agent will perform the checking traversals at the end, so to solve reachability, the agent could never perform the undesired traversals. The price we pay for this ability to constrain the behavior of gadgets is that the resulting simulations are no longer drop-in replacements as in the local simulations of Definition 1; instead we obtain “nonlocal simulations” which require altering the entire surrounding system of gadgets:

► **Definition 6.** *A finite set of gadgets \mathcal{G} [planarly] nonlocally simulates a gadget H if, for every finite set of gadgets \mathcal{G}' , there is a polynomial-time (many-one/Karp) reduction from [planar] reachability with $\{H\} \cup \mathcal{G}'$ to [planar] reachability with $\mathcal{G} \cup \mathcal{G}'$.*

Lemma 2 says that simulations are nonlocal simulations, so this notion is a generalization of Definition 1.

Next we define “checkable” gadgets via “postselection”, which transforms a gadget with broken states (where a checking traversal sequence is impossible) into an idealized gadget where those broken states are prevented. At this stage, the prevention is by a magical force, but we will later implement this force with a nonlocal simulation.

► **Definition 7.** *Let G be a gadget, C be a traversal sequence on $L(G)$, and $L' \subset L(G)$. Call a state q of G **broken** if C is not legal from q . Assume that broken states are preserved by transitions on L' in the sense that, if q is broken and there is a transition $(q, a) \rightarrow (q', b)$ where $a, b \in L'$, then q' is also broken.*

Define **Postselect**(G, C, L') to be the gadget G' where $L(G') = L'$, $Q(G')$ contains the nonbroken states of G , and $T(G')$ contains the transitions of G restricted to L' and $Q(G')$.⁵ When there exist C and L' such that **Postselect**(G, C, L') is equivalent to G' , we say that G is a **checkable** G' , and we call C the **checking traversal sequence**.

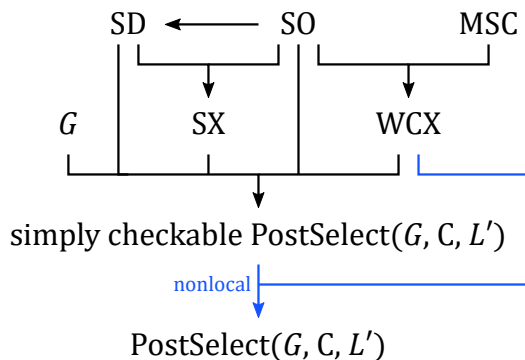
A traversal sequence X is legal for **Postselect**(G, C, L') from state q if and only if XC is legal for G from q , because both are equivalent to there being a nonbroken state reachable by traversing X . Intuitively, **Postselect**(G, C, L') is the gadget that results from forcing the agent to traverse C after solving reachability, to ensure that the gadget was left in a nonbroken state, and hiding locations in $L \setminus L'$. **Postselect**(G, C, L') behaves like G on the locations L' except that transitions into broken states are prohibited.

We now state the main result of the checkable gadget framework, which is in terms of two simple (and often easy-to-implement) gadgets SO (single-use opening) and MSC (merged single-use closing gadgets) defined in Section 4.1.

► **Theorem 8.** *For any G, C , and L' satisfying the assumptions of Definition 7, $\{G, SO, MSC\}$ planarly nonlocally simulates **Postselect**(G, C, L').*

⁵ If every state of G is broken, then **Postselect**(G, C, L') has no states. In this case, it is impossible to use **Postselect**(G, C, L') in a system of gadgets because that requires specifying an initial state, so all of our theorems hold vacuously.

The goal of this section is to prove Theorem 8. Figure 17 provides a schematic overview of the gadget simulations throughout this section that culminate in this result. In Section 4.1, we describe the base gadgets needed for our construction. In Section 4.2, we prove that nonlocal simulations compose in the natural way. In Section 4.3, we introduce a particularly simple kind of checkable gadget, and show that they nonlocally simulate the gadget they are based on. Finally, in Section 4.4 we use all of these tools to prove Theorem 8.



■ **Figure 17** Overview of gadget simulations used for postselection. Black arrows show local simulations and blue arrows show nonlocal simulations.

4.1 Base Gadgets

We now define two base gadgets and three additional derived gadgets, shown in Figure 18, that we use to implement the machinery of checkable gadgets. All five of these gadgets can change state only a bounded number of times; they are “LDAG” in the language of [13].

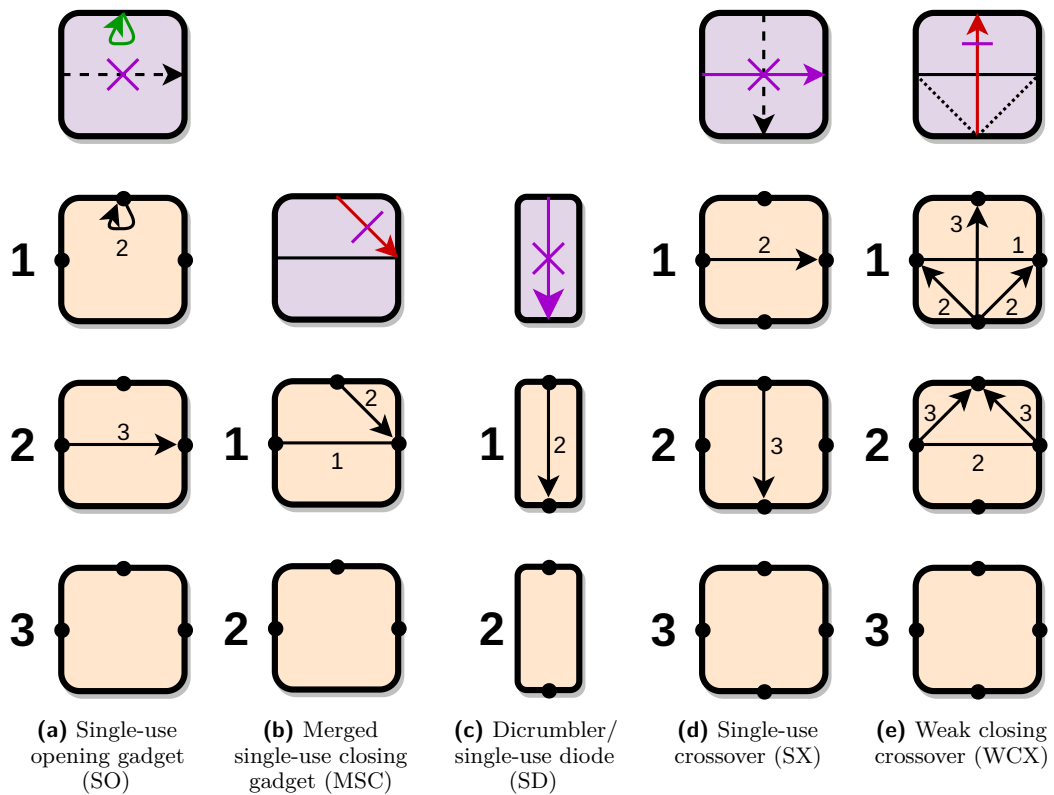
The two base gadgets required for our construction are shown in Figure 18a–18b:

- (a) The *single-use opening (SO)* gadget, shown in Figure 18a, is a three-state three-location gadget. In state 1, the “opening” location has a self-loop traversal (also called a button, or a port in [3]), which transitions to state 2. State 2 allows a single traversal between the other two locations, after which (in state 3) no traversals are possible.
- (b) The *merged single-use closing (MSC)* gadget, shown in Figure 18b, is a two-state three-location gadget. In the “open” state 1, horizontal traversals in both directions are freely available. After a traversal from top to right, the gadget transitions to the “closed” state 2, where no traversals are possible.

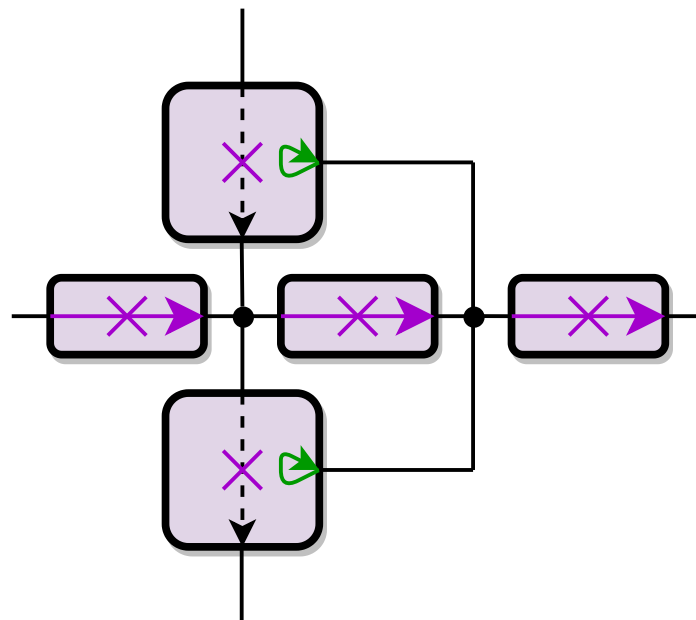
Next we describe three useful gadgets for our construction which can be built from these base gadgets.

The *dicumbler/single-use diode (SD)* gadget, shown in Figure 18c, is a two-state two-location gadget. In state 1, there is a single directed traversal between the two locations, which permanently closes the gadget in state 2 where no traversals are possible. The SD gadget can be simulated by either of the two base gadgets: it is equivalent to state 2 of SO, and to MSC restricted to the two locations incident to the closing traversal.

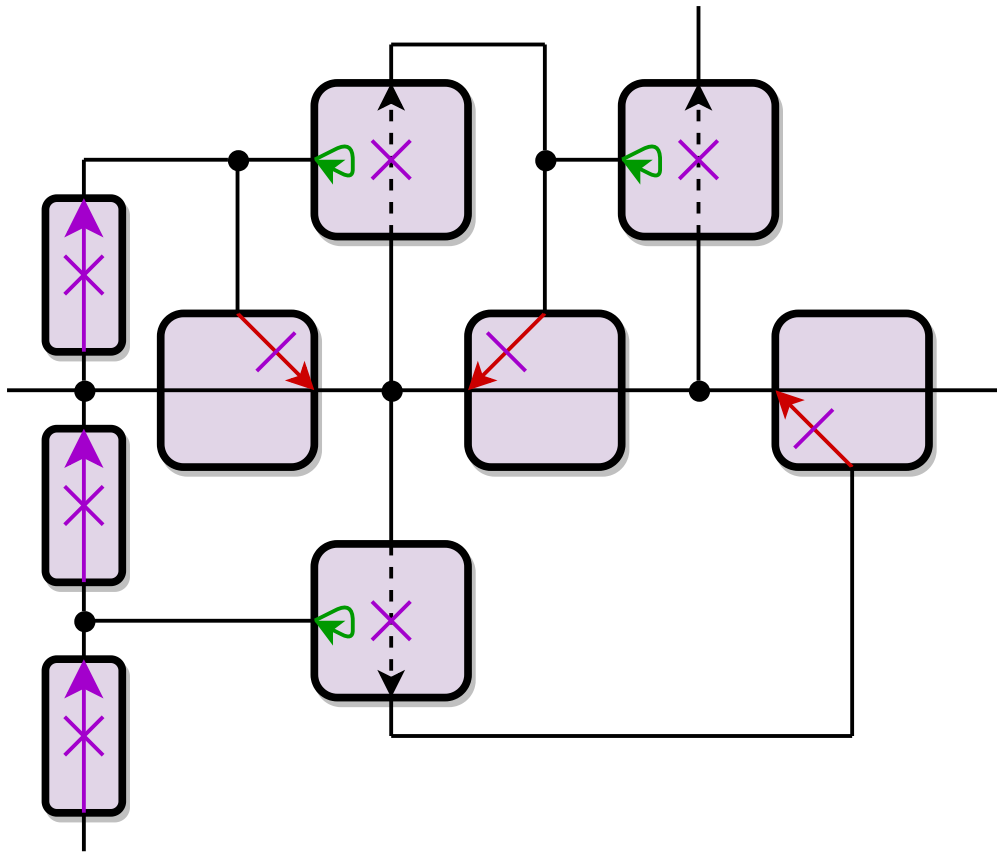
The *single-use crossover (SX)* gadget, shown in Figure 18d, allows one traversal from left to right and then one from top to bottom. It can be simulated using SO and SD gadgets as shown in Figure 19. The top location in the simulation cannot be entered until the top SO is opened. This opening is possible only after traversing the first two SDs, which prevents any further traversals coming from the left or going to the right. The bottom SO prevents premature traversals going to the bottom.



■ **Figure 18** Icons (top) and state diagrams (bottom) for two base gadgets (a–b) and three derived gadgets (c–e). Green arrows show opening traversals, red arrows show closing traversals, and purple crosses indicate traversals that close themselves.



■ **Figure 19** Construction of the single-use crossover from SO and SD gadgets.



■ **Figure 20** Construction of the weak closing crossover from SD, SO, and MSC gadgets.

The *weak closing crossover (WCX)*, shown in Figure 18e, initially allows traversals freely between the left and right. If a bottom-to-top traversal is performed, no more traversals are possible. However, a bottom-to-left or bottom-to-right traversal is also possible (which also opens up left-to-top or right-to-top traversals), making the crossover “leaky”. The weak closing crossover can be simulated using SO, MSC, and SD gadgets, as shown in Figure 20. To open the upper-right SO, the agent needs to traverse the upper-left SO and then close the middle MSC. To open the upper-left SO, the agent will need to close the leftmost MSC. Having closed both the left and the middle MSCs, the agent is forced to traverse the bottom SO and close the rightmost MSC. The bottom SO can only be opened by the agent traversing entering the bottom and traversing bottom two SDs, preventing any future traversals from the bottom. In summary, in order to exit the top, the agent must have entered the bottom in the past, and have closed all three MSCs. Entering the bottom changes to state 2, and exiting the top changes to state 3.

4.2 Nonlocal Simulation Composition

A crucial fact about nonlocal simulation is that nonlocal simulations can be composed:

► **Lemma 9.** *Let \mathcal{G} and \mathcal{H} be finite sets of gadgets. Suppose \mathcal{G} [planarly] nonlocally simulates every gadget in \mathcal{H} , and \mathcal{H} [planarly] nonlocally simulates another gadget H . Then \mathcal{G} [planarly] nonlocally simulates H .*

Proof. For a finite set of gadgets \mathcal{G}' , we must find a polynomial-time reduction from reachability with $\{H\} \cup \mathcal{G}'$ to reachability with $\mathcal{G} \cup \mathcal{G}'$. Let $\mathcal{H} = \{H_1, \dots, H_n\}$, where $n = |\mathcal{H}|$, and let \mathcal{H}_i be the prefix $\{H_1, \dots, H_i\}$, so $\mathcal{H}_n = \mathcal{H}$. Then we construct a chain of reductions between reachability with different sets of gadgets:

$$\{H\} \cup \mathcal{G}' \rightarrow \mathcal{G} \cup \mathcal{H}_n \cup \mathcal{G}' \rightarrow \mathcal{G} \cup \mathcal{H}_{n-1} \cup \mathcal{G}' \rightarrow \dots \rightarrow \mathcal{G} \cup \mathcal{H}_1 \cup \mathcal{G}' \rightarrow \mathcal{G} \cup \mathcal{G}'.$$

The first reduction is because $\mathcal{H} = \mathcal{H}_n$ nonlocally simulates H . The remaining reductions come from the assumption that \mathcal{G} nonlocally simulates each $H_i \in \mathcal{H}$, which implies that there is a polynomial-time reduction from reachability with $\{H_i\} \cup \mathcal{G} \cup \mathcal{H}_{i-1} \cup \mathcal{G}' = \mathcal{G} \cup \mathcal{H}_i \cup \mathcal{G}'$ to reachability with $\mathcal{G} \cup \mathcal{G} \cup \mathcal{H}_{i-1} \cup \mathcal{G}' = \mathcal{G} \cup \mathcal{H}_{i-1} \cup \mathcal{G}'$. \blacktriangleleft

4.3 Simply Checkable Gadgets

Next, we define a special kind of checkable gadgets, called “simply checkable” gadgets. A simply checkable G is essentially a checkable G where the checking sequence consists of a single traversal between two locations not in $L(G)$, called c_{in} and c_{out} . Simply checkable gadgets will be a useful as an intermediate step in our proof of Theorem 8.

► **Definition 10.** For a gadget G , a **simply checkable** G is a gadget G' satisfying the following properties:

1. $L(G') = L(G) \sqcup \{c_{in}, c_{out}\}$ has two new locations c_{in}, c_{out} . For planar gadgets, the cyclic orderings of the shared locations $L(G)$ are the same. (Locations c_{in} and c_{out} can be added to the cyclic order anywhere.)
2. There is a function $f : Q(G) \rightarrow Q(G')$ assigning a state of G' to each state of G .
3. For any traversal sequence X that is legal for G from state q , the concatenated traversal sequence $X \cdot [c_{in} \rightarrow c_{out}]$ is legal for G' from $f(q)$.
4. Every traversal sequence that ends at c_{out} and is legal for G' from state $f(q)$ has the form

$$X \cdot [c_{in} \rightarrow \bullet, \bullet \rightarrow \bullet, \dots, \bullet \rightarrow c_{out}]$$

where X is legal for G from state q and the omitted \bullet locations (if any) belong to $L(G)$.

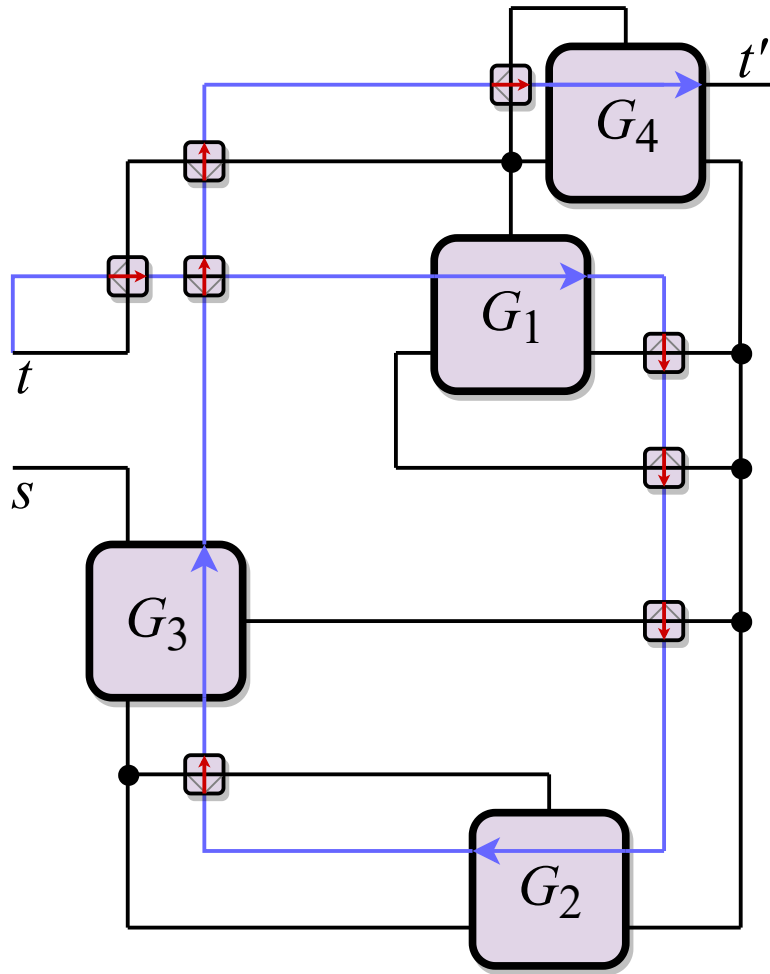
Intuitively, a simply checkable G in state $f(q)$ behaves the same as G does in state q , provided that afterward the agent performs a traversal sequence from c_{in} to c_{out} (which may involve the agent exiting and re-entering the gadget, but only via nonchecking locations). The gadget can do essentially anything in a traversal sequence not ending in c_{out} .

Any simply checkable G is also a checkable G : if G' is a simply checkable G , then $\text{Postselect}(G', [c_{in} \rightarrow c_{out}], L(G))$ is equivalent to G .

We show that a simply checkable G can nonlocally simulate G while preserving planarity, using an auxiliary gadget. First, define the hallway gadget to be the one-state two-location gadget with transitions in both directions between the locations (i.e., a “branching hallway” with only two locations). A **checkable hallway crossover** is a simply checkable hallway where the added locations c_{in} and c_{out} are not adjacent in the cyclic order, i.e., they interleave with the two hallway locations. For example, the weak closing crossover from Figure 18e is a checkable hallway crossover, where the horizontal traversal corresponds to the hallway, the bottom location is c_{in} , and the top location is c_{out} .

► **Lemma 11.** Let G' be a simply checkable G and let CHX be a checkable hallway crossover. Then

1. $\{G'\}$ nonlocally simulates G ; and
2. $\{G', CHX\}$ planarly nonlocally simulates G .



■ **Figure 21** Our nonlocal simulation for the proof of Lemma 11. The system is modified by replacing each copy of G with a copy of G' and adding the blue path from t through $c_{in} \rightarrow c_{out}$ on each one.

Proof. For any gadget set \mathcal{G}' , we construct a polynomial-time reduction from reachability with $\{G\} \cup \mathcal{G}'$ to reachability with $\{G'\} \cup \mathcal{G}'$, or from planar reachability with $\{G\} \cup \mathcal{G}'$ to planar reachability with $\{G', CHX\} \cup \mathcal{G}'$. Suppose we have a [planar] system S of gadgets from $\{G\} \cup \mathcal{G}'$, along with a designated starting location s and target location t . Let G_1, \dots, G_n denote the copies of G in S , and let q_1, \dots, q_n be their respective initial states in S . We build a new system S' of gadgets from $\{G'\} \cup \mathcal{G}'$ as follows; refer to Figure 21.

1. Replace each copy G_i of gadget G with initial state q_i in S by a corresponding copy G'_i of G' with initial state $f(q_i)$, whose copies of c_{in} and c_{out} are named $c_{in,i}$ and $c_{out,i}$.
2. Connect t to $c_{in,1}$. In the planar case, we place a copy of CHX on each crossing this creates, with the check line on the way from t to $c_{in,1}$.
3. Connect $c_{out,i}$ to $c_{in,i+1}$ for each i . In the planar case, we place a copy of CHX on each crossing this creates, with the check line on the way from $c_{out,i}$ to $c_{in,i+1}$.

Our reduction outputs this new system S' along with the same start location s and the new target location $t' = c_{out,n}$.

3:18 Pushing Blocks via Checkable Gadgets

This construction clearly takes polynomial time. To prove that the reduction is valid, we must show that there is a legal system traversal $s \rightarrow^* c_{\text{out},n}$ in S' if and only if there is a legal system traversal $s \rightarrow^* t$ in S .

First suppose there is a legal system traversal $s \rightarrow^* t$ in S . Then this solution can be extended to a legal system traversal $s \rightarrow^* c_{\text{out},n}$ in S' by appending the traversal $c_{\text{in},i} \rightarrow c_{\text{out},i}$ on G'_i for each i in increasing order, and in the planar case, adding the needed traversals of the inserted copies of CHX (including the check traversals needed to get from t to $c_{\text{in},1}$ and from each $c_{\text{out},i}$ to $c_{\text{in},i+1}$). The appended $c_{\text{in},i} \rightarrow c_{\text{out},i}$ traversals are all valid because Property 3 of Definition 10 requires that any legal traversal sequence for G can be extended by $c_{\text{in}} \rightarrow c_{\text{out}}$ to yield a legal traversal sequence for G' . For the same reason, the appended $c_{\text{in}} \rightarrow c_{\text{out}}$ traversals in copies of CHX are valid. Also, the inserted hallway traversals of the copies of CHX are all valid from the definition of checkable hallway crossover, because they occur before all appended $c_{\text{in}} \rightarrow c_{\text{out}}$ traversals.

Now suppose that there is a legal system traversal $s \rightarrow^* c_{\text{out},n}$ in S' . Define $c'_{\text{in},i}, c'_{\text{out},i}$ to be the check in and out locations for all checkable gadgets (copies of both G' and CHX), in the order that these check traversals occur in the intended solution described above. By Property 4 of Definition 10, the agent can only exit the i th checkable gadget (G' or CHX) at $c'_{\text{out},i}$ if it previously entered at the corresponding $c'_{\text{in},i}$. In S' , the only location connected to $c'_{\text{in},i+1}$ is $c'_{\text{out},i}$ (ignoring hallway traversals of CHX gadgets), so this property implies that $c_{\text{out},i}$ was previously visited as well. By induction, the solution must have reached $c'_{\text{in},1}$ via t , and then traversed all of the $c'_{\text{in},i}$ and $c'_{\text{out},i}$ locations (possibly with some detours). Consider the prefix X' of the solution up to the first time t is visited, and let X be the modification to remove any hallway traversals of the copies of CHX. We claim X is a solution for S . Clearly X is a system traversal $s \rightarrow^* t$ and satisfies all unmodified gadgets (from \mathcal{G}'). By Property 4 of Definition 10, $c'_{\text{in},i}$ and $c'_{\text{out},i}$ are visited at most once in the full solution, and the prefix of the solution prior to visiting $c'_{\text{in},i}$ is legal for the i th checked gadget. Because each $c'_{\text{in},i}$ is visited after t , it is not visited in X , and thus X is legal for G_i . Similarly, X makes only hallway traversals of CHX, so removing those traversals is valid in S where there were direct connections before the crossings were introduced. Therefore X is a valid system traversal $s \rightarrow^* t$ in S . ◀

4.4 Postselected Gadgets

We now finally prove our main result, Theorem 8: postselection can be achieved using only the two base gadgets from Section 4.1, while preserving planarity.

It will be convenient to assume all of our gadgets are *transitive*: if there are two transitions $(q_1, \ell_1) \rightarrow (q_2, \ell_2) \rightarrow (q_3, \ell_3)$, then there is also a transition $(q_1, \ell_1) \rightarrow (q_3, \ell_3)$. For reachability, this makes no difference: we can replace any gadget with its transitive closure without affecting the answers to any reachability problems, since we can always think of the transition $(q_1, \ell_1) \rightarrow (q_3, \ell_3)$ as a sequence of two transitions. That is, every gadget is equivalent for reachability to some transitive gadget, and in particular there are nonlocal simulations in both directions.

Proof of Theorem 8. Assume without loss of generality that G is transitive, by replacing G with its transitive closure.

We will show that $\{G, \text{SO}, \text{MSC}, \text{SD}, \text{SX}, \text{WCX}\}$ planarly *locally* simulates some gadget G' which is a simply checkable $\text{Postselect}(G, C, L')$. As shown in Section 4.1 (Figures 19 and 20 in particular), $\{\text{SO}, \text{MSC}\}$ planarly locally simulates WCX, SX, and SD. By combining these local simulations, we obtain that $\{G, \text{SO}, \text{MSC}\}$ planarly locally simulates the same G' . By

Lemma 2, this is also a nonlocal simulation. By Lemma 11, for any checkable hallway crossover gadget CHX, $\{G', \text{CHX}\}$ planarly nonlocally simulates G' . Because $\{\text{SO}, \text{MSC}\}$ planarly simulates the weak closing crossover (Figure 20), which is a checkable hallway crossover, it follows from Lemma 9 that $\{G, \text{SO}, \text{MSC}\}$ planarly nonlocally simulates $\text{Postselect}(G, C, L')$, proving the theorem.

Now we show that $\{G, \text{SO}, \text{MSC}, \text{SD}, \text{SX}, \text{WCX}\}$ planarly locally simulates some gadget G' which is a simply checkable $\text{Postselect}(G, C, L')$. Unpacking the definitions of “simply checkable” and Postselect , we must simulate a gadget G' that satisfies the following properties:

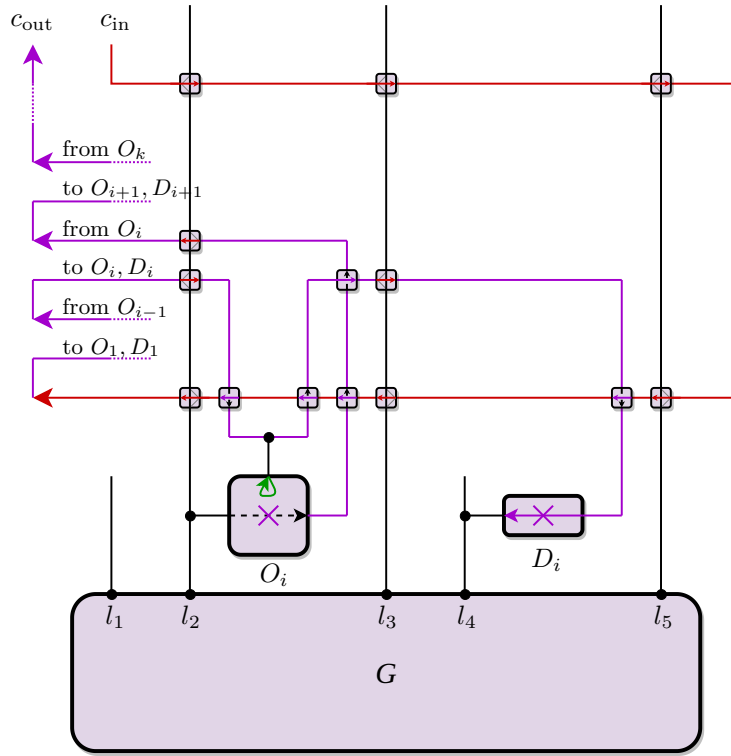
1. $L(G') = L' \sqcup \{c_{\text{in}}, c_{\text{out}}\}$.
2. There is a function f from unbroken states of G to states of G' .
3. For any traversal sequence X on L' , if XC is legal for G from state q , then $X \cdot [c_{\text{in}} \rightarrow c_{\text{out}}]$ is legal for G' from state $f(q)$.
4. Any traversal sequence that ends with c_{out} and is legal for G' from state $f(q)$ has the form $X \cdot [c_{\text{in}} \rightarrow \bullet, \bullet \rightarrow \bullet, \dots, \bullet \rightarrow c_{\text{out}}]$, where X is a traversal sequence on L' , XC is legal for G from state q , and all the omitted \bullet locations are in L' .

We construct our simulation of the gadget G' starting from G as follows; refer to Figure 22.

1. For purposes of description, orient so that G has all of its locations on the top of its bounding box. We will place the locations for the simulated gadget on a horizontal line L above G (so they will lie on the outside face).
2. For each location $l \in L'$, add a long upward edge e_l connecting l in G to a new location l' on L . Because the edges are all vertical, they do not cross each other, and the l' locations appear in the same cyclic (left-to-right) order as $l \in L'$.
3. Place c_{in} on L left of all e_l edges. Starting from c_{in} , draw a non-self-crossing path that crosses each of the e_l in one rightward pass, then turn down, then cross each e_l a second time in one leftward pass in between the first pass and G . We ensure any further crossings with the edges e_l take place between these two delimiter passes, which we call the top and bottom delimiters, by routing paths across the bottom delimiter before crossing any e_l . These delimiters serve to “cut off” the rest of the construction, preventing leakage.
4. For each traversal $a_i \rightarrow b_i$ in the sequence $C = [a_1 \rightarrow b_1, \dots, a_k \rightarrow b_k]$, add a single-use opening gadget O_i and a dicumbler D_i , near locations b_i and a_i respectively. Connect the opening location of O_i to the entrance of D_i (routing up across the bottom delimiter, then horizontally, then down). Connect the exit of D_i to a_i , and connect b_i to the entrance of O_i .
5. Connect the exit of each O_i to the opening location of O_{i+1} , routing up across the bottom delimiter, then all the way left, then up, then right, then down.
6. Finally, connect c_{in} to the opening location of O_1 after the two delimiter passes; and connect the exit of O_k to c_{out} , routing up across the bottom delimiter, then all the way left, then up.

We call the path we have constructed from c_{in} to c_{out} the *checking path*. For an unbroken state q of G , the corresponding state $f(q)$ of G' is simulated by placing G in state q and all other gadgets in their usual initial states.

This construction is nonplanar in two ways: our new checking path crosses the edges e_l and also crosses itself. In the former case we replace the crossing with a weak closing crossover, oriented so that the checking path closes e_l . In the latter case we replace the crossing with a single-use crossover, oriented correctly so that the agent can traverse the two directions in the expected order detailed below. We must prove this construction has the properties stated above. By construction, its locations are $L' \sqcup \{c_{\text{in}}, c_{\text{out}}\}$.



■ **Figure 22** The simulation of a simply checkable, postselected version of the gadget G . The two initial crossings of the edges e_l connecting locations in L' to the outside are shown in red. The rest of the checking path is shown in purple. All further crossings of the checking path with edges e_l occur between the two initial crossings. In this example, $L = \{l_1, l_2, l_3, l_4, l_5\}$ and $L' = \{l_2, l_3, l_5\}$. The i th checking traversal $[l_4 \rightarrow l_2]$ is enforced by O_i and D_i .

Suppose XC is legal for G from state q . We can perform $X \cdot [c_{\text{in}} \rightarrow c_{\text{out}}]$ in the simulation where G starts in q by first performing X in the natural way (using the edges e_l) and then following the checking path: starting at c_{in} , for each i we visit the opening location of O_i , then go through D_i , then traverse $a_i \rightarrow b_i$ via G , then traverse O_i . This path brings us to c_{out} at the end, and its restriction to G is exactly XC .

Now suppose that there is a legal traversal sequence for G' from state $f(q)$ ending in c_{out} . Putting ourselves in the position of a forgetful agent, we find ourselves at c_{out} and must determine how we got there. We can induct backwards along the checking path (as in the proof of Lemma 11) to show that we must have visited c_{in} , using the facts that in order to exit the closing side of a weak closing crossover we must have entered it on the opposite side, and that in order to exit from O_i we must have visited its opening location.

Thus at some point in the path we entered G' through c_{in} , crossed all the e_l twice, and then for every $a_i \rightarrow b_i$ of C in order we opened O_i , traversed D_i , and later traversed O_i . Crossing each e_l twice closes the weak closing crossovers, making e_l no longer traversable. Between traversing D_i and O_i , we somehow must have gotten from a_i to b_i . We cannot have used the edges e_l because they were already closed during the initial crossings. So we must have made transitions only in G , of the form $(q_1, \ell_1 = a_i) \rightarrow (q_2, \ell_2) \rightarrow \dots \rightarrow (q_k, \ell_m = b_i)$. Since G is transitive, we could equivalently have made the single transition $(q_1, a_i) \rightarrow (q_k, b_i)$, and in particular have traversed $a_i \rightarrow b_i$.

Similarly, after the initial two crossings of the e_l , we can't have left this simulated gadget or entered G except for the traversals of C . Finally, we take advantage of the fact that before entering c_{in} , the simulation behaves exactly like G except that only locations in L' are accessible. So the full path through the simulation G' ending at c_{out} must have the following form:

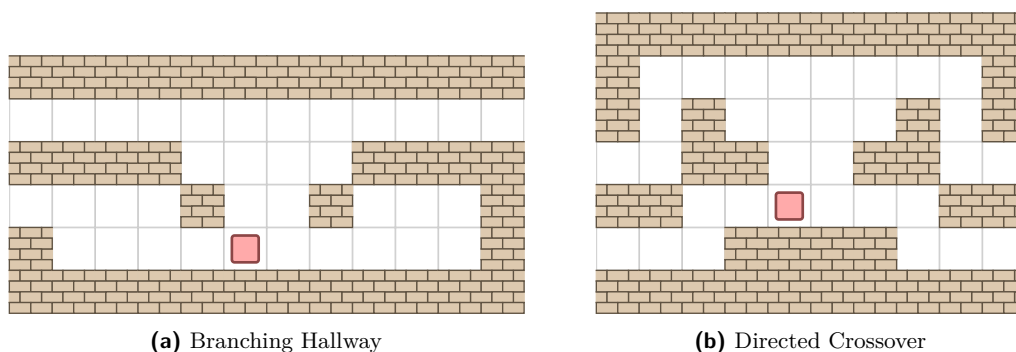
1. We use G' as if it were G (restricted to the locations of L') with initial state q , performing some traversal sequence X .
2. We enter G' through c_{in} .
3. We possibly leak out of G' or into G via locations in L' , through the weak closing crossovers at the initial two crossings with each e_l . Call the sequence of traversals made during this phase Y .
4. Eventually, we finish all of initial crossings with e_l , and moved to the O_i s and D_i s.
5. We perform the traversal sequence C in G without any additional traversals in G in between and without leaving G' .
6. Finally, we leave G' through c_{out} .

Therefore the sequence of traversals on G' has the form $X \cdot [c_{in} \rightarrow \bullet, \bullet \rightarrow \bullet, \dots, \bullet \rightarrow c_{out}]$ and the sequence of traversals just on G is XYC , where X and Y are traversal sequences on L' and the omitted \bullet locations are in L' . In particular, XYC is legal for G from state q , so by the assumption that broken states are preserved by transitions on L' , XC is legal for G from q . This is the final condition we needed, so G' is a simply checkable $\text{Postselect}(G, C, L')$. ◀

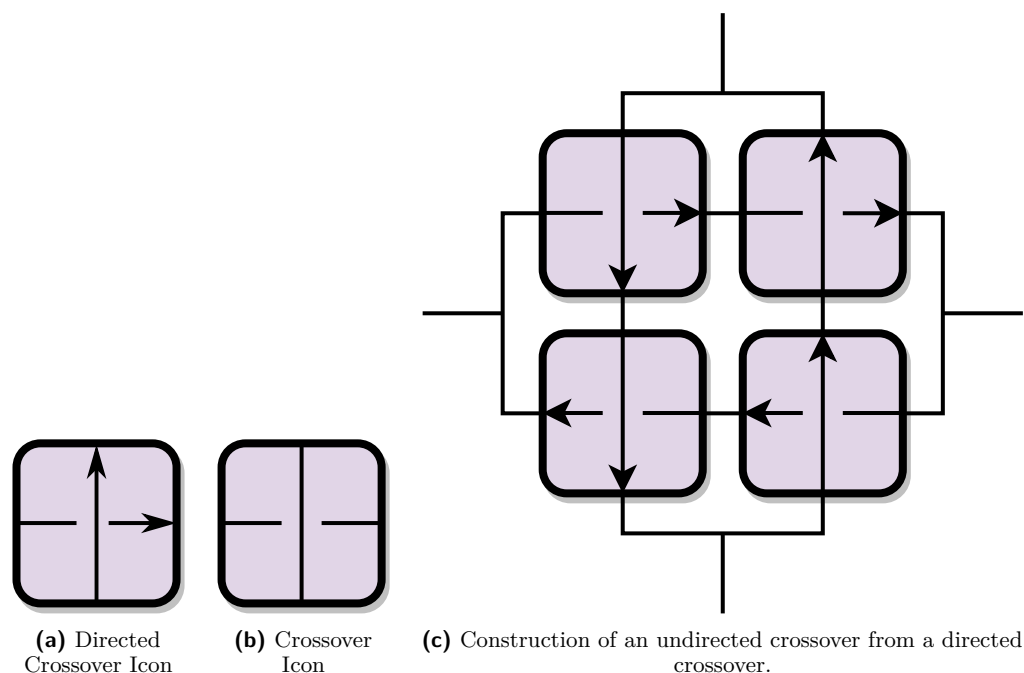
5 BoxDude is PSPACE-complete

We now show that BoxDude is PSPACE-complete via a reduction from reachability with nondeterministic locking 2-toggles. In this model, boxes can be pushed horizontally by the Dude but cannot be picked up. We will make use of the postselection construction from Section 4 in order to nonlocally simulate nondeterministic locking 2-toggles.

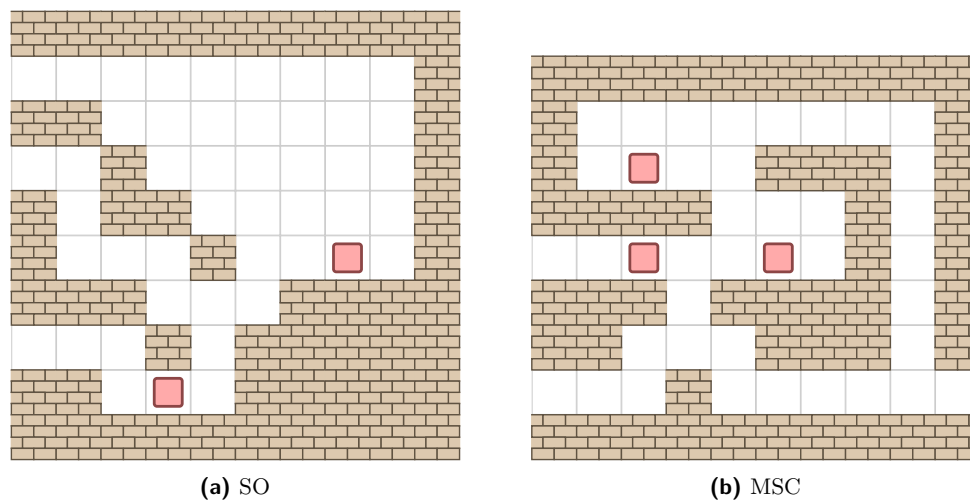
Similarly to BlockDude we must build a branching hallway in order to connect the locations of our gadgets. This time, we also build a directed crossover gadget. These gadgets are shown in Figure 23. Directed crossovers can be used to construct undirected crossovers as in Figure 24. This allows us to connect locations in nonplanar ways, and reduce from reachability instead of planar reachability. We note a diode gadget is easy to build by simply having a height 2 drop.



■ **Figure 23** Hallway connection gadgets for BoxDude. Pushable boxes are in red. The branching hallway gadget is fully traversable from any of its three locations to the others. The directed crossover can be traversed only from bottom-left to top-right or from bottom-right to top-left.



■ **Figure 24** Icons for directed and undirected crossovers. The undirected crossover can be constructed from four directed crossovers as shown in [10].

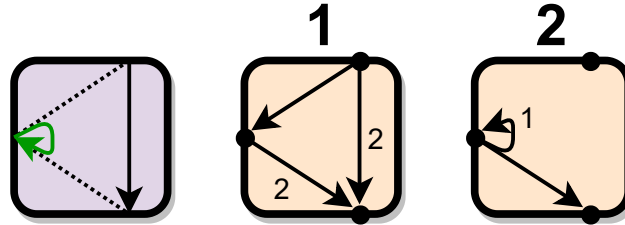


■ **Figure 25** SO and MSC gadgets for BoxDude.

Postselection requires us to additionally simulate the gadgets SO and MSC. These gadgets are shown in Figure 25.

Next we build a checkable *leaky door* gadget. A leaky door has two states (“open” and “closed”), and three locations, called “opening”, “entrance”, and “exit”. Similar to a self-closing door [3], the gadget can be traversed in the open state from entrance to exit, but doing so transitions the door to the closed state. In the closed state, it is not possible to enter the gadget through the entrance at all, but visiting the opening location allows the gadget to transition back to the open state. Unlike a self-closing door, it is possible to go

from the entrance to the opening location when the gadget is in the open state. It is also always possible to go from the opening location to the exit, but doing so transitions the door to the closed state. The full state diagram for the leaky door is shown in Figure 26.



■ **Figure 26** Icon and state diagram for the leaky door gadget.

The checkable leaky door is shown in Figure 27. We apply postselection to this gadget with the checking traversal sequence [opening \rightarrow opening, entrance \rightarrow opening].⁶ We now analyze which states are *broken* in the sense that this traversal sequence is impossible from those states.

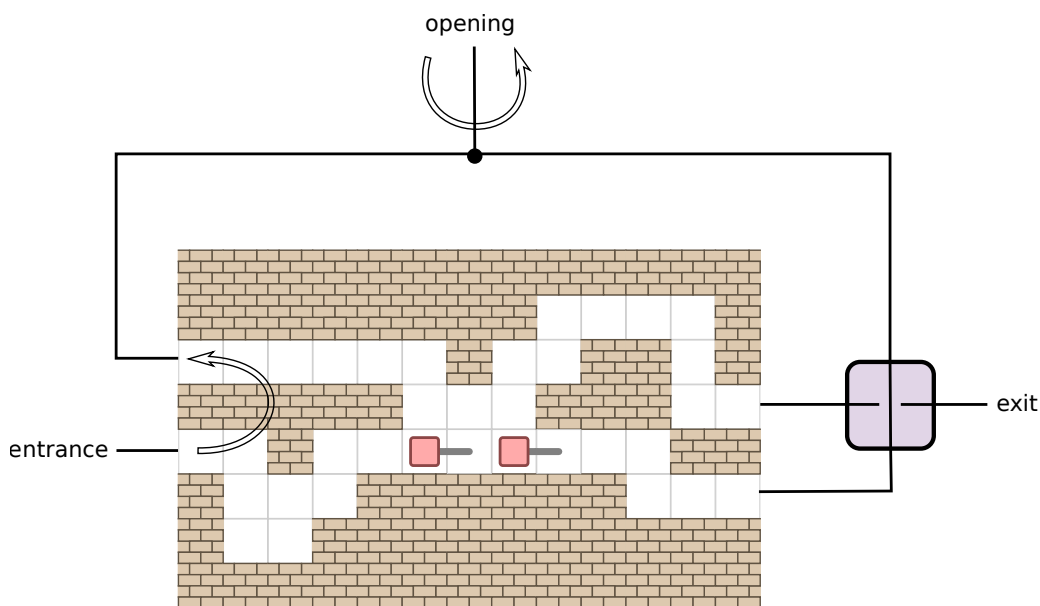
- If the left box is further to the left than its current location, the gadget state is broken since the entrance is unusable.
- If the left box is more than one square to the right of its current location, the gadget state is broken because the opening location is unreachable from the entrance.
- If the two boxes are adjacent, the gadget state is broken for the same reason.

Moving the right box more than one square to the right is never advantageous for the player, so we assume it does not occur.

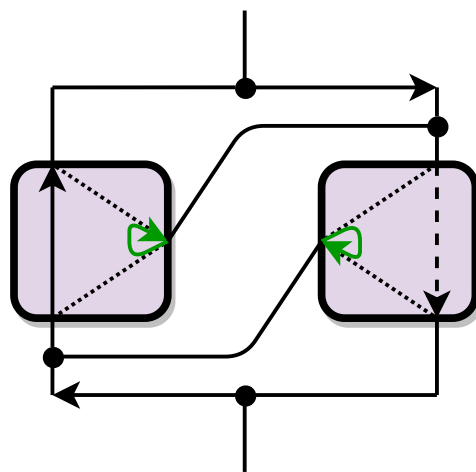
We will show that the postselection of this gadget is exactly the leaky door gadget. When the right box is in its current location, we say that the gadget is in the closed state; when it is one square to the right the gadget is in the open state. Because the left box cannot move more than one square to the right, it follows that any traversal to the exit location must leave the gadget in the closed state. In the closed state, no traversals are possible from the entrance without breaking the gadget by putting two boxes adjacent. Visiting the opening allows transitioning to the open state. In the open state, additional traversals are available from the entrance. The agent may go from entrance to exit by using the connected opening locations to reset the gadget to the closed state and then using the right block to reach the exit. It is also possible to leak from the entrance to the opening location, and from the opening location to the exit (transitioning to the closed state). Thus the traversals within unbroken states are exactly those allowed by the leaky door gadget. By Theorem 8 the checkable leaky door, along with the SO and MSC gadgets built earlier, nonlocally simulate the leaky door.

We now build a 1-toggle gadget, shown in Figure 10, using a pair of leaky doors. This construction is shown in Figure 28. It can be seen that none of the leaks are useful to an agent traversing the gadget, since the most they accomplish is bringing the agent back to its starting location without changing any state.

⁶ The first check from opening to opening does not enforce anything but merely allows access to the location in case the gadget was last left in the closed state. The check from entrance to opening cannot be done if the gadget is in the closed state.

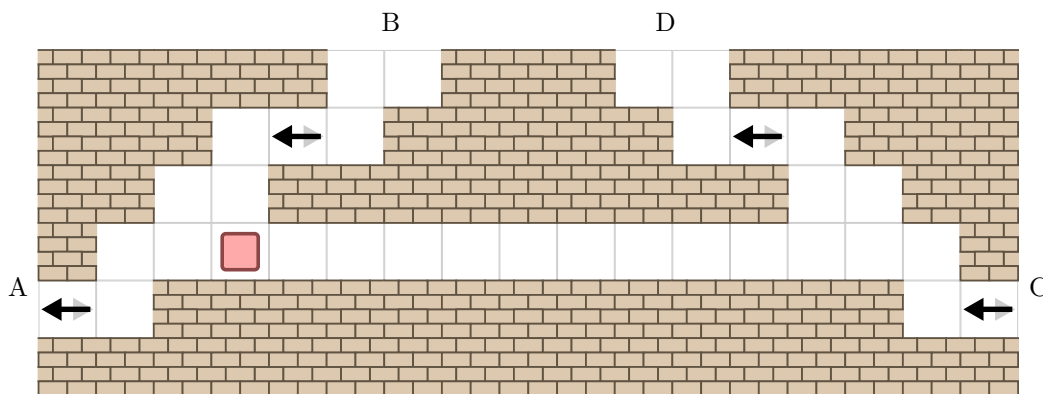


■ **Figure 27** A checkable leaky door, shown in the closed state. The crossover and branching hallway needed to connect the top left and bottom right hallways have been abstracted. Horizontal “tracks” display the range of locations for each box in unbroken states. (The right box can move farther right but it is never advantageous to do this.) The two boxes may not be adjacent in unbroken states.



■ **Figure 28** A 1-toggle built from leaky doors. Solid or dashed arrows inside gadgets show the traversal from entrance to exit in an open or closed leaky door, respectively. Green self-loops are opening locations of leaky doors. Arrows outside gadgets are diodes.

We are now in a position to build a nondeterministic locking 2-toggle. By Theorem 4, reachability with this gadget is PSPACE-complete. The final construction, shown in Figure 29, is quite simple in appearance; the complexity is hidden in the 1-toggles used to protect the locking 2-toggle’s locations. Traversing from A to B is only possible when the box is on the left side of the gadget, and conversely for C to D. Since the box’s position can only be changed when exiting the gadget through A or C (corresponding to which side the gadget is



■ **Figure 29** A nondeterministic locking 2-toggle, currently locked to the left side. Locations B and C are protected with inwards-directed 1-toggles; locations A and D with outwards-directed 1-toggles. (Note: the middle portion of the gadget would actually need to be wider than shown in this diagram in order to make enough space to route locations B and D away from each other.)

locked to), the gadget simulates a locking 2-toggle. Note that this gadget cannot be broken by moving the box further to the left than its current position, since doing so renders the gadget fully untraversable. This is because in this state location A is permanently unusable and B and D cannot be reached from inside the gadget. The agent can only exit out of C, so that C’s 1-toggle points inwards. Since C’s and D’s 1-toggles always point in different directions, D is also permanently unusable. The only remaining traversal is $B \rightarrow C$, but this is impossible also because C’s 1-toggle points inwards.

Using Theorem 8 and Lemma 9, our simulations imply that the BoxDude gadgets we have explicitly built nonlocally simulate a nondeterministic locking 2-toggle. In particular, there is a polynomial-time reduction from planar reachability with nondeterministic locking 2-toggles, which is PSPACE-complete by Theorem 4, to BoxDude. Hence BoxDude is PSPACE-complete.

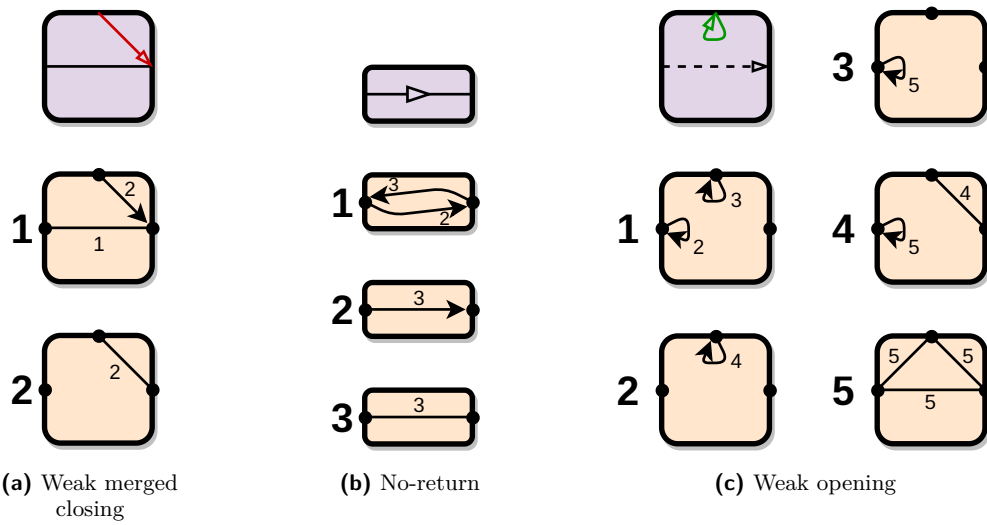
6 Push-1F is PSPACE-complete

In this section, we show that Push-1F is PSPACE-complete using a reduction from planar reachability with self-closing doors, shown in Figure 8, which is PSPACE-complete by Theorem 5. Recall that in this model there is no gravity, and the agent can push one block at a time in any direction. We will make several uses of postselection from Section 4 in order to nonlocally simulate various gadgets along the way.

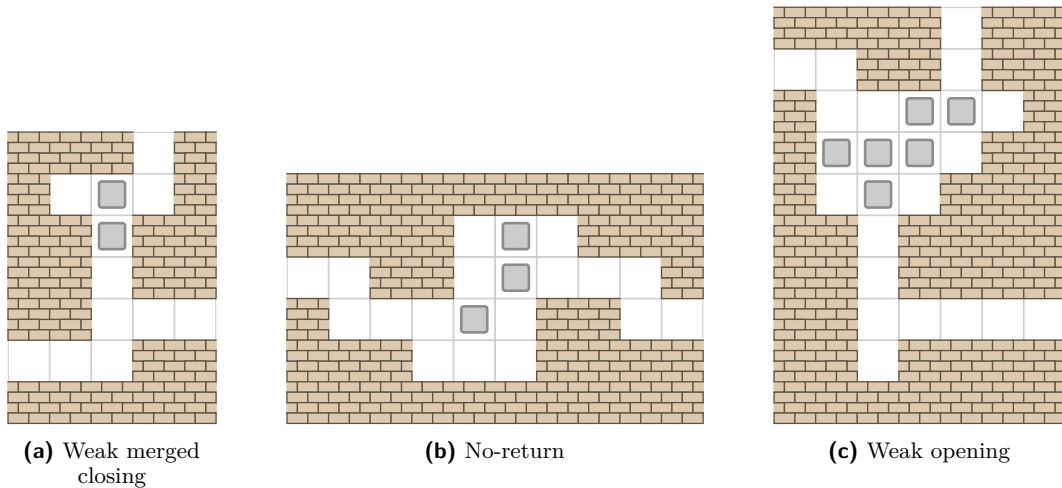
In order to use postselection, we must build single-use opening (SO) and merged single-use closing (MSC) gadgets. We start by building a *weak merged closing* gadget, based on the Lock gadget from [8]. The weak merged closing gadget acts like the MSC except that the closing traversal can be performed multiple times. We also use a gadget introduced in [8] called a *no-return* gadget. After a no-return gadget is traversed from left to right, it cannot immediately be traversed from right to left. However, initially traversing it from the right or traversing left to right twice breaks the gadget, making it fully traversable. Finally, we build a *weak opening* gadget. A weak opening gadget’s exit cannot be used in traversals until both of its input locations are visited separately. Figure 30 shows the state diagrams for these gadgets, and Figure 31 shows how to implement them in Push-1F.

We combine the weak merged closing, no-return, and weak opening gadgets to make a dicumbler; this allows us to simulate ordinary SO and MSC gadgets using the gadgets we have built so far. These simulations are shown in Figure 32. Having built these gadgets, we can now take advantage of the machinery of checkable gadgets. The structure of the remaining gadget simulations used in this section is outlined in Figure 33.

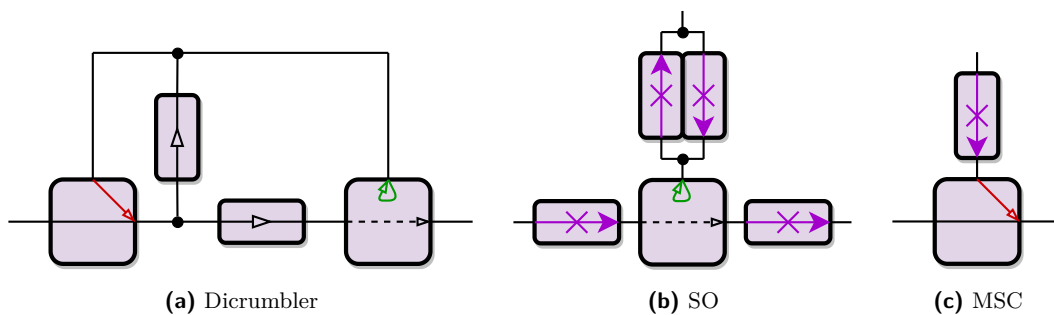
3:26 Pushing Blocks via Checkable Gadgets



■ **Figure 30** Icons and state diagrams for Push-1F base gadgets.

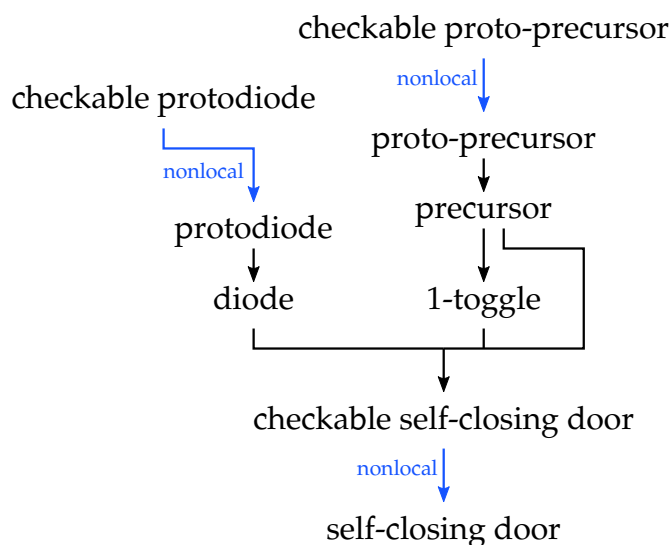


■ **Figure 31** Constructions of base gadgets for Push-1F.



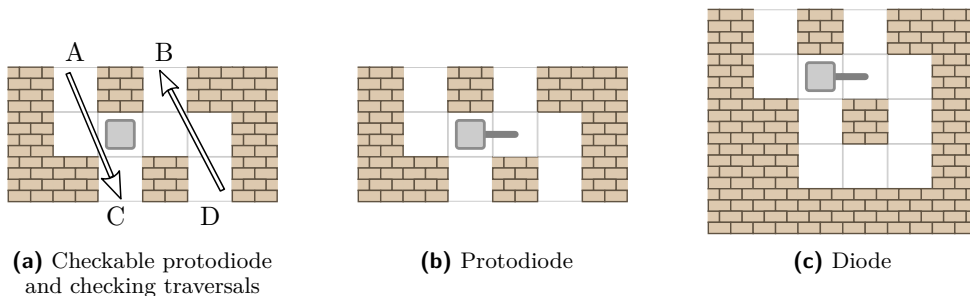
■ **Figure 32** Constructions of gadgets required for postselection in Push-1F.

We first nonlocally simulate a diode, which allows traversal in only one direction. We accomplish this by building a checkable *protodiode*, where the protodiode is a certain four-location gadget which easily simulates a diode. Refer to Figure 34. We apply postselection to



■ **Figure 33** Overview of gadget simulations used for Push-1F. Black arrows show local simulations and blue arrows show nonlocal simulations.

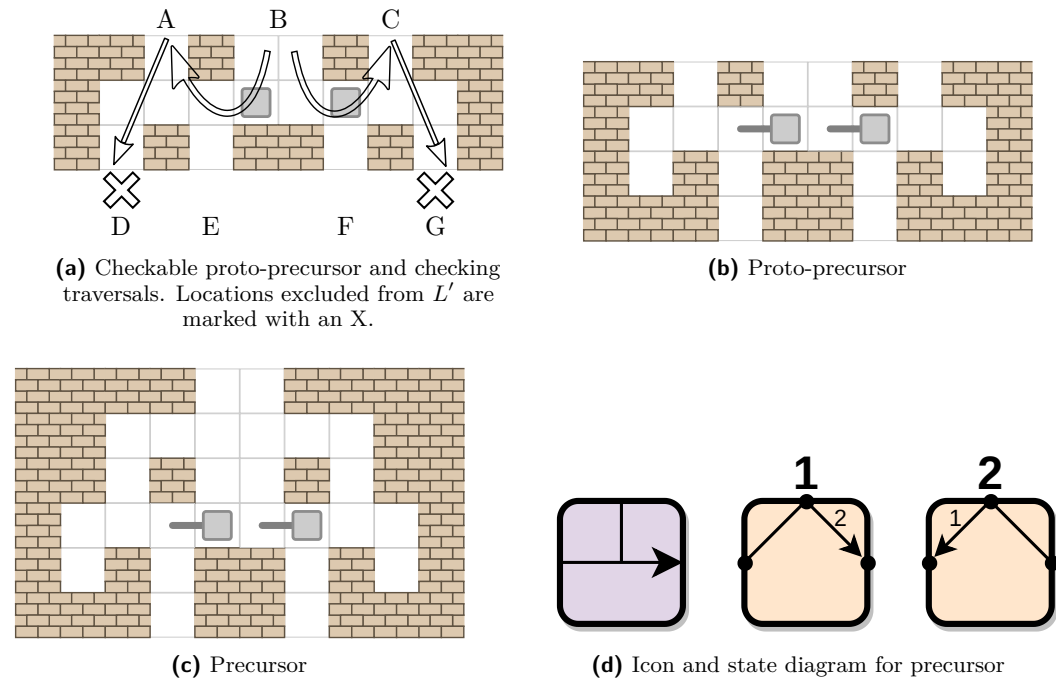
the checkable protodiode with the checking traversals $[A \rightarrow C, D \rightarrow B]$ to nonlocally simulate the protodiode. The nonbroken states are exactly those in which the block is confined to the middle two squares. Connecting the bottom two locations of the protodiode yields a diode.



■ **Figure 34** Nonlocal diode simulation for Push-1F. Horizontal tracks show where the block is allowed to move in the protodiode and diode, as if it is confined by a magical force.

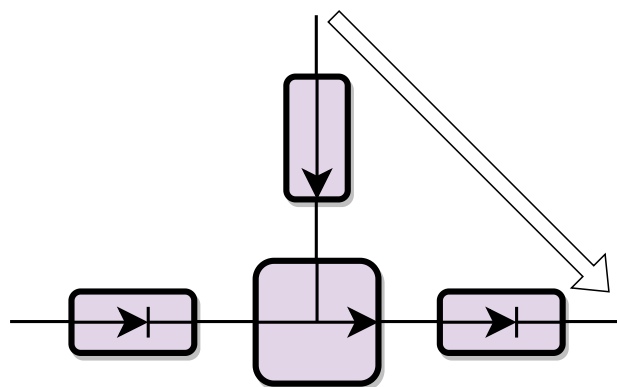
We now nonlocally simulate a *precursor* gadget, which will be used to build a 1-toggle and a checkable self-closing door. The precursor’s state diagram is shown in Figure 35d. We begin by building a checkable *proto-precursor*, where again the proto-precursor is a certain gadget which easily simulates the precursor. Refer to Fig 35. We apply postselection to the checkable proto-precursor with the checking traversals $[A \rightarrow D, C \rightarrow G, B \rightarrow A, B \rightarrow C]$. We close off locations D and G during postselection by not including them in the set $L' = \{A, B, C, E, F\}$ of locations on the proto-precursor. The nonbroken states are exactly those in which the blocks are confined to the four center-most spaces, and the two blocks are not adjacent. Entering a broken state is irreversible with respect to transitions on the locations in L' because D and G were excluded in L' . (If D or G were included then it would be possible to un-break the gadget from some broken states by pushing a block back into the center.) Thus we can use postselection to nondeterministically simulate the proto-precursor; joining its upper three locations together yields the precursor gadget. Additionally, closing

the top location of the precursor gadget produces a 1-toggle.



■ **Figure 35** Nonlocal precursor simulation for Push-1F. As before, horizontal tracks in the proto-precursor and precursor show spaces to which blocks are magically confined. The magical force also prevents the pair of blocks in the proto-precursor and precursor from being adjacent.

Finally, we nonlocally simulate a self-closing door. Our construction of a checkable self-closing door is shown in Figure 36. This gadget is almost identical to a self-closing door, except that it permits a traversal from the opening location to the exit location exactly once, after which the gadget is fully untraversable. We eliminate this problem by applying postselection with the checking traversal sequence [opening \rightarrow opening, entrance \rightarrow exit]. The sole broken state is the fully untraversable one arising from the aforementioned undesired traversal. If we imagine that a magical force prevents the gadget from being left in such a state, then we obtain exactly a self-closing door.



■ **Figure 36** Checkable self-closing door for Push-1F using the precursor gadget, two diodes, and a 1-toggle.

We have demonstrated a series of planar, nonlocal gadget simulations culminating in the planar nonlocal simulation of a self-closing door. Because planar reachability through systems of self-closing doors is PSPACE-complete by Theorem 5, so is Push-1F.

7 Open Problems

The primary remaining question is the complexity of Push-1 block puzzles where there are no fixed blocks allowed in the puzzle. Push-1 can easily simulate fixed blocks using 2×2 arrangements of movable blocks, so we only need to make all fixed areas two blocks thick. Our constructions of the gadgets SO and MSC needed to apply postselection all use two-block thick spacing, so we have shown that postselection is available for Push-1 gadgets. Unfortunately, our postselected constructions for Push-1F critically use one-block-thick spacing.

Another question we do not address is the related block storage question for \dots Dude puzzles, named \dots Duderino in [5], in which the blocks have target locations to occupy. This is comparable to the difference between Push-1F and Sokoban. It is generally expected that the storage version of block-pushing puzzles is at least as hard as reaching a single goal location; however, this result does not directly follow. We believe using the reconfiguration version of the gadgets framework from [4] may help build a gadget-based proof.

We have another open question related to the technique of postselected gadgets. When defining a postselected gadget, we only specified a single traversal sequence to be checked. It seems likely that one could enforce the choice of one of several possible sequences using more complex constructions like those found in the SAT reduction for DAG gadgets in [11]. Are there cases where this sort of flexibility is useful?

References

- 1 Scott Aaronson. Quantum computing, postselection, and probabilistic polynomial-time. *Proceedings of the Royal Society A*, 461:3473–3482, 2005. doi:<http://doi.org/10.1098/rspa.2005.1546>.
- 2 Joshua Ani, Sualeh Asif, Erik D Demaine, Yevhenii Diomidov, Dylan Hendrickson, Jayson Lynch, Sarah Scheffler, and Adam Suhl. PSPACE-completeness of pulling blocks to reach a goal. *Journal of Information Processing*, 28:929–941, 2020.
- 3 Joshua Ani, Jeffrey Bosboom, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, and Jayson Lynch. Walking through doors is hard, even without staircases: Proving PSPACE-hardness via planar assemblies of door gadgets. In *Proceedings of the 10th International Conference on Fun with Algorithms (FUN 2020)*, Favignana, Italy, September 2020.
- 4 Joshua Ani, Erik D. Demaine, Yevhenii Diomidov, Dylan H. Hendrickson, and Jayson Lynch. Traversability, reconfiguration, and reachability in the gadget framework. In Petra Mutzel, Md. Saidur Rahman, and Slamun, editors, *Proceedings of the 16th International Conference and Workshops on Algorithms and Computation (WALCOM 2022)*, volume 13174 of *Lecture Notes in Computer Science*, pages 47–58, Jember, Indonesia, March 2022. doi:10.1007/978-3-030-96731-4_5.
- 5 Austin Barr, Calvin Chang, and Aaron Williams. Block dude puzzles are NP-hard (and the rugs really tie the reductions together). In *Proceedings of the 33rd Canadian Conference on Computational Geometry*, Halifax, Canada, 2021.
- 6 Joseph Culberson. Sokoban is PSPACE-complete. In *Proceedings of the International Conference on Fun with Algorithms*, pages 65–76, Elba, Italy, June 1998.
- 7 Erik D. Demaine, Martin L. Demaine, Michael Hoffmann, and Joseph O’Rourke. Pushing blocks is hard. *Computational Geometry: Theory and Applications*, 26(1):21–36, August 2003.

3:30 Pushing Blocks via Checkable Gadgets

- 8 Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *Proceedings of the 12th Annual Canadian Conference on Computational Geometry (CCCG 2000)*, pages 211–219, Fredericton, Canada, August 2000.
- 9 Erik D. Demaine, Isaac Grosf, Jayson Lynch, and Mikhail Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, pages 18:1–18:21, La Maddalena, Italy, June 2018.
- 10 Erik D. Demaine, Robert A. Hearn, and Michael Hoffmann. Push-2-F is PSPACE-complete. In *Proceedings of the 14th Canadian Conference on Computational Geometry*, pages 31–35, Lethbridge, Canada, August 2002.
- 11 Erik D. Demaine, Dylan Hendrickson, and Jayson Lynch. Toward a general theory of motion planning complexity: Characterizing which gadgets make games hard. In *Proceedings of the 11th Conference on Innovations in Theoretical Computer Science*, Seattle, Washington, January 2020. arXiv:1812.03592.
- 12 Dylan Hendrickson. Gadgets and gizmos: A formal model of simulation in the gadget framework for motion planning. Master's thesis, Massachusetts Institute of Technology, 2021.
- 13 Jayson Lynch. *A framework for proving the computational intractability of motion planning problems*. PhD thesis, Massachusetts Institute of Technology, 2020.
- 14 Joseph O'Rourke and The Smith Problem Solving Group. PushPush is NP-hard in 3D. arXiv:cs/9911013, November 1999. URL: <https://arXiv.org/abs/cs/9911013>.

Fun Slot Machines and Transformations of Words Avoiding Factors

Marcella Anselmo ✉

Dipartimento di Informatica, University of Salerno, Fisciano (SA), Italy

Manuela Flores ✉

Dipartimento di Informatica, University of Salerno, Fisciano (SA), Italy

Maria Madonia ✉

Dipartimento di Matematica e Informatica, University of Catania, Italy

Abstract

Fun Slot Machines are a variant of the classical ones. Pulling a lever, the player generates a sequence of symbols which are placed on the reels. The machine pays when a given pattern appears in the sequence. The variant consists in trying to transform a losing sequence of symbols in another one, in such a way that the winning pattern does not appear in any intermediate step. The choice of the winning pattern can be crucial; there are “good” and “bad” sequences. The game results in a combinatorial problem on transformations of words avoiding a given pattern as a factor. We investigate “good” and “bad” sequences on a k -ary alphabet and the pairs of words that witness that a word is “bad”. A main result is an algorithm to decide whether a word is “bad” or not and to provide a pair of witnesses of minimal length when the word is “bad”. It runs in $O(n)$ time with a preprocessing of $O(n)$ time and space to construct an enhanced suffix tree of the word.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words; Theory of computation → Design and analysis of algorithms; Theory of computation → Formal languages and automata theory

Keywords and phrases Isometric words, Words avoiding factors, Index of a word, Overlap, Lee distance

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.4

Funding Partially Supported by INdAM-GNCS Project 2021, FARB Project ORSA218107 of University of Salerno and TEAMS Project of University of Catania.

1 Introduction

Everybody knows what a slot machine is, some more, some less. In a simple model, there is a screen where a certain number of symbols appears in a line. Every symbol is placed on some reel that “spins” when the game is activated. The machine pays out according to the pattern of symbols displayed when the reels stop spinning, e.g., whether or not it contains a given factor. One of the first model was composed of three spinning reels containing a total of five symbols each: horseshoes, diamonds, spades, hearts and a Liberty Bell; the bell gave the machine its name. Three consecutive bells in a row produced the biggest payoff, ten nickels. Later on, fruit symbols were placed on the reels besides the original bell to refer to the fruit-flavoured gums offered.

Recently, the designers of the *MMM company*, leader in the field of fun machines, have designed a new machine, called *Fun Slot Machine*, and they are evaluating the product. The machine offers a variant to the usual game. After the player has pulled the lever twice without having found the winning pattern, she/he can take the last two sequences of symbols that have appeared - both of which do not contain the winning pattern - and try to transform the first into the second, so that the pattern does not appear this time either in any intermediate step. When the player succeeds, the slot machine pays a consolation prize. The game has



© Marcella Anselmo, Manuela Flores, and Maria Madonia;
licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 4; pp. 4:1–4:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

rules to follow. Only the symbols where the two sequences differ can be exchanged and the exchange must be done step by step, following the sequence of symbols on the reels. The MMM's designers claim that this game is not just a gamble. There are "good" and "bad" patterns, "good" and "bad" numbers of reels. They propose the following examples to support their claim.

Suppose that the fun slot machine has 6 reels, each with 4 symbols on, A, C, T, G , which stand for the favourite fruit flavours, *Apricot*, *Cherry*, *Tamarind*, and *Grape*. The symbols are placed on each reel in this cyclical order A, C, T, G . Consider the case that the winning pattern is $AGAC$ and the displayed sequence is $AGAAAC$ the first time and $AGATAC$ the second time. The player has lost both times, because $AGAC$ has not appeared. Hence, the player can try the variant to the game, that is, to transform $AGAAAC$ in $AGATAC$ so that $AGAC$ does not appear either. The sequences differ only in their fourth position, where the first sequence contains A , while the second one T . In the first step, symbol A can be swapped either in C or in G , its neighbourhoods on the reel. Unfortunately, for the player (but not for the owner), in both cases $AGAC$ will be displayed. There is no way to win in this case.

The situation would have been different if the number of reels was 5, and not 6. If the displayed sequences were $AGAAA$ and $AGATA$, the exchange of A in the fourth position to C , would have yield $AGAC$, but there is a winning transformation, $AGAAA$ in $AGAGA$ and then in $AGATA$.

Consider now a different scenario. The symbols on the reels are as before, but the winning pattern is AAA . Suppose there are 5 reels and the losing sequences are $ACAGA$ and $AGATA$. The player wants to transform $ACAGA$ in $AGATA$, which differ in their positions 2 and 4. The swap of G into T in the fourth position is safe, no AAA is displayed. Then, there are two possibilities to swap C into G in the second position; either through A or through T . While the first choice would display AAA , the second one would be winning; the transformation of $ACAGA$ in $ACATA$, then in $ATATA$ and finally in $AGATA$, never let AAA appear, and the player will gain the consolation prize. One can show (and we will prove it in the sequel) that when the pattern is AAA , for any number of reels and any pairs of losing sequences, the player has always a possibility to gain.

In some sense, the pattern AAA is "good", because it always leaves a chance of winning, while $AGAC$ is "bad", since, in some situations, it leaves no chance of winning. Now, the question of the MMM's designers is: how should the number of reels, the number and the order of symbols on, and the winning pattern be chosen, whether I am a player or the owner of the machine?

Bad and good sequences of symbols have been investigated in the literature. They were introduced in the binary case, that is when only two symbols are available. A binary word (i.e., a sequence of symbols in a finite set of two symbols) f is called d -good if for any pair of words u and v of length d which do not contain the factor f , u can be transformed in v by exchanging one by one the bits on which they differ and generating only words which do not contain f . It is *good* if it is d -good for all d . A binary word f is *bad* if it is not good. The *index* of a binary bad word is the threshold d from which the word is no longer d -good, and a pair of words (u, v) showing that the word is not good is called a pair of *witnesses* for the bad word. Recently, good and bad words have been considered in the case of larger sets of symbols where they are referred to as isometric and non-isometric words; see [1], and [2] on quaternary words. Also, binary bad words have been considered in the two-dimensional setting, and bad pictures have been investigated [3].

The fun slot machine problem thus concerns transformations of words that avoid a given factor. Actually, it is not only a problem in combinatorics of words. It can be stated as a problem on some graphs, called k -ary n -cubes, introduced in [12], and their isometric sub-graphs. More specifically, it concerns a problem on k -ary n -cube avoiding a k -ary word f [2].

Let us mention such framework in this introduction, while it will be no more considered in the rest of the paper. The k -ary n -cube, Q_n^k , is a graph with k^n vertices, each associated to a word of length n over a k -ary alphabet identified with $\mathbb{Z}_k = \{0, 1, \dots, k-1\}$. Two vertices in Q_n^k are adjacent whenever their associated words differ in exactly one position, and the mismatch is given by two symbols x and y , with $x = (y \pm 1) \bmod k$. Special cases include rings (when $n = 1$), hypercubes Q_n (when $k = 2$) and tori. They have been introduced in [12], in the context of interconnection networks. The binary case ($k = 2$) has been extensively investigated [6]. In order to obtain some variants of hypercubes such that the number of vertices increases slower than in a hypercube, Hsu introduced Fibonacci cubes [7]. They received a lot of attention afterwards (see [9] for a survey). These notions have been then extended to define the generalized Fibonacci cube $Q_n(f)$ [8]. It is the subgraph of the hypercube Q_n obtained by considering only vertices associated to binary words that do not contain a given word f as a factor. In this framework, a binary word f is good when, for any $n \geq 1$, $Q_n(f)$ can be isometrically embedded into Q_n , and bad, otherwise [10]. More generally, given a k -ary word f , the k -ary n -cube avoiding f , $Q_n^k(f)$, is obtained from Q_n^k by elimination of the vertices containing f as a factor. Good and bad words are investigated in this more general setting in [2], where they are referred to as isometric and non-isometric words.

Coming back to the fun slot machines, f represents the winning pattern, u and v the displayed losing sequences, and their length is the number of reels. The goal is to find a transformation of u in v that changes only the positions where u and v differ and f is never displayed. If f is good then there is always a chance of winning. If it is bad, in the situation where the number of reels is greater than or equal to the index, and u and v are witnesses for f , then there is no chance of winning!

The main result in this paper is an algorithm to test whether a word is good or not. Further, in case the word is bad, the algorithm provides its index and a pair of witnesses of length equal to the index. Note that, when $k > 4$, no good word exists and any bad word has index equal to its length [1]. Therefore, the computation of the index is given for $k = 2, 3, 4$. It is based on the construction of the pairs of witnesses for f given in [1, 2] to show Theorem 4. The construction generalizes to k -ary words the one given for binary words in [13]. We will revisit it, while highlighting some valuable features and adding some more results. We will show that the index is the minimal length of the above constructed witnesses. Moreover, the index of a quaternary word can be directly computed from the word, without going through its binary representation, as it was in [2].

The algorithm presented in this paper runs in linear time and space, for k -ary words (with $k = 2, 3, 4$). This complexity can be achieved thanks to a preprocessing of linear time and space for computing the suffix tree of the word and enhancing it in order to answer Lowest Common Ancestor (LCA) queries in constant time. An example of execution of the algorithm is provided. The first part of the algorithm follows the one provided in a very recent paper [4] to efficiently check whether a word is isometric. This algorithm is based on the characterization in [2] and applies some methods of the pattern matching with mismatches. Note that the algorithm in this paper not only checks whether a word is isometric, but it also provides its index and a pair of witnesses of minimal length, while keeping the same linear

complexity. A cubic time algorithm for the computation of the index and some related words was given in [13] for binary words. The algorithm presented here works for k -ary words, with $k = 2, 3, 4$, while improving the time complexity.

To conclude, observe that in the previously mentioned examples, *AGAC* is a bad word, 6 is its index, and *AGAAAC* and *AGATAC* are witnesses for *AGAC*; that's why there is no possibility of winning the game. On the other hand, *AAA*, or more generally a sequence of (three) equal symbols, is good. Could it have been for this reason that, more than a century ago, the Liberty Bell machine paid the maximum when three bells in a row were displayed?

2 Fun Slot Machines and Isometric Words

A *Fun Slot Machine* is composed of d reels that can spin in both directions. Each reel carries k symbols, s_1, s_2, \dots, s_k . Let Σ be the set of all symbols, and s_1, s_2, \dots, s_k be the order in which symbols follow in each reel. This means that when a reel is spinning in a clockwise direction, s_i appears after s_{i-1} , for $i = 2, \dots, k$, and s_1 after s_k ; vice versa in the opposite direction. The unique winning pattern is f of length $n < d$. The player inserts the coin and then pulls the lever. The displayed sequence u of d symbols is called a word or string over Σ of length d . The problem is whether the shorter word f is a factor of u or not. If f is not a factor of u , we say that u is f -free or that u avoids the factor f .

Let us formalize the problem with the terminology of the combinatorics of words. First, let us recall some preliminary notions.

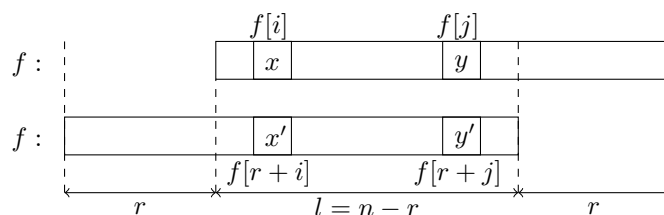
Let Σ be an alphabet and $|\Sigma| = k$. Throughout the paper, Σ will be identified with $\mathbb{Z}_k = \{0, 1, \dots, k - 1\}$, the ring of integers modulo k . A word (or string) $f \in \Sigma^*$ of length n is $f = x_1x_2 \cdots x_n$, where x_1, x_2, \dots, x_n are symbols in Σ . The set of words over Σ of length n is denoted Σ^n . Let $f[i]$ denote the symbol of f in position i , i.e. $f[i] = x_i$. Then, $f[i..j] = x_i \cdots x_j$, for $1 \leq i \leq j \leq n$, is a factor of f . A word $s \in \Sigma^*$ is said f -free if it does not contain f as a factor. The prefix of f of length l is $pre_l(f) = f[1..l]$; while the suffix of f of length l is $sufl_l(f) = f[n - l + 1..n]$. When $pre_l(f) = sufl_l(f)$ then $pre_l(f)$ is referred to as an overlap of f of length l .

Let $u, v \in \Sigma^*$ be two words of the same length. Then, the *Hamming distance* $dist_H(u, v)$ between u and v is the number of positions at which u and v differ. The *Lee distance* between two words $u, v \in \mathbb{Z}_k^n$, $u = x_1 \cdots x_n$ and $v = y_1 \cdots y_n$ is

$$dist_L(u, v) = \sum_{i=1}^n \min(|x_i - y_i|, k - |x_i - y_i|).$$

In the sequel, Σ will denote a generic alphabet of cardinality k , while Δ to denote the quaternary alphabet $\Delta = \{A, C, T, G\}$, referred to as the *genetic alphabet*. Symbols A and T (C and G , resp.) will be called *complementary symbols*, in analogy to the Watson-Crick complementary bases they represent. The alphabet Δ will be identified with \mathbb{Z}_4 , in such a way that A, C, T , and G will be identified with 0, 1, 2, and 3, respectively. Therefore, pairs of complementary symbols have Lee distance 2, whereas pairs of distinct non-complementary symbols have Lee distance 1.

Let us now define what we have called “good” and “bad” word in the Introduction. To be more precise and to avoid ambiguity with similar definitions in other papers, from now on, “good” and “bad” words will be referred to as “Lee-isometric” and “Lee-non-isometric” words, respectively. The definitions are based on the process of transforming a word in another one of equal length, changing one symbol at a time.



■ **Figure 1** The word f and its 2-error overlap of length l .

Let f be a word in Σ^n . A *Lee-transformation* of length h from u to v is a sequence of words w_0, w_1, \dots, w_h such that $w_0 = u$, $w_h = v$, and for any $i = 0, 1, \dots, h-1$, $\text{dist}_L(w_i, w_{i+1}) = 1$. The Lee-transformation is *f-free* if for any $i = 0, 1, \dots, h$, the word w_i is *f-free*. The word $f \in \Sigma^n$ is *Lee-isometric* if for all $d \geq n$, and *f-free* words $u, v \in \Sigma^d$, there is an *f-free* Lee-transformation from u to v of length equal to $\text{dist}_L(u, v)$. A word is *Lee-non-isometric* if it is not Lee-isometric. Note that there exists an *f-free* Lee-transformation from u to v if and only if there exists an *f-free* Lee-transformation from v to u .

A pair (u, v) of words $u, v \in \Sigma^d$ is referred to as a pair of *Lee-witnesses* for f , if u and v are *f-free* words and there does not exist an *f-free* Lee-transformation from u to v of length equal to $\text{dist}_L(u, v)$. If f is Lee-non-isometric then its *Lee-index*, denoted by $I_L(f)$, is the shortest length d of u, v such that (u, v) is a pair of Lee-witnesses for f . The *Lee-index* of a Lee-isometric word is defined ∞ .

► **Example 1.** Let Δ be the quaternary genetic alphabet, $f = ACT$, $u = ACCCT$, and $v = ACGCT$. Observe that $\text{dist}_L(u, v) = 2$, since they differ in their third position only and $\text{dist}_L(C, G) = 2$. The sequences $ACCCT, ACGCT$ and $ACCCT, ACTCT, ACGCT$ are two Lee-transformations from u to v of length equal to $\text{dist}_L(u, v) = 2$; they are not *f-free*. Actually, no *f-free* Lee-transformation exists from u to v . This shows that ACT is Lee-non-isometric and that (u, v) is a pair of Lee-witnesses for ACT .

Let us state the following definition (see Figure 1).

► **Definition 2.** Let Σ be a k -ary alphabet, $f \in \Sigma^n$, and q be an integer $1 \leq q \leq n$. The word f has a q -Lee-error overlap of length l , if $\text{dist}_L(\text{pre}_l(f), \text{suf}_l(f)) = q$. Its shift is $r = n - l$; its error positions are the m positions where $\text{pre}_l(f)$ differs from $\text{suf}_l(f)$.

► **Remark 3.** Using the notations in the previous definition, if f has a q -Lee-error overlap of length l , then $m \leq l, q$. In particular, when $k = 4$ and $q = 2$, then $m = 1$ or $m = 2$. The case $m = 1$ holds if $\text{pre}_l(f)$ and $\text{suf}_l(f)$ differ in exactly one position and the error is given by a pair of complementary symbols. For example, $f = AGAC \in \Delta^4$ has a 2-Lee-error overlap of length $l = 2$. Indeed, $m = 1$ and $\text{dist}_L(AG, AC) = 2$.

If $m = 2$ then $\text{pre}_l(f)$ and $\text{suf}_l(f)$ differ in two different positions i and j and the errors are given by pairs of non-complementary symbols.

Next theorem, proved in [1, 2], provides a characterization of Lee-isometric words, which is fundamental to test whether a word is Lee-isometric or not. Furthermore, it allows us to restrict the investigation to k -ary alphabets with $k = 2, 3, 4$.

► **Theorem 4** ([1, 2]). Let Σ be a k -ary alphabet and $f \in \Sigma^*$.

- f is Lee-isometric if and only if it has no 2-Lee-error overlap, when $k = 2, 3, 4$
- f is never Lee-isometric, when $k > 4$.

The proof of Theorem 4 is constructive. In the case that $k = 2, 3, 4$ and f has a 2-Lee-error overlap, the proof provides a pair of Lee-witnesses showing that f is Lee-non-isometric. Applying Theorem 4, one can show that, when $k = 2, 3, 4$, the Lee-index of a Lee-non-isometric word f satisfies $n + 1 \leq I_L(f) \leq 2n - 1$ [2].

3 Computing the Lee-index of a Lee-non-isometric Word

The Lee-index of a “bad” word has been introduced as a threshold on the length of words that witness the “badness” of the word. In this section, we are going to show how to compute the Lee-index of a “bad” - actually, Lee-non-isometric - word. The results prove the correctness of the algorithm in the next section.

Let $\Sigma = \{0, 1, \dots, k - 1\}$, $f = f_1 f_2 \dots f_n$ be a word over Σ , $u, v \in \Sigma^d$ be f -free words, and $h, j \in \Sigma$. The reverse of f is $f^R = f_n \dots f_2 f_1$. The h -shift of j is $j^{S(h)} = (j + h) \bmod k$, while the h -shift of f is $f^{S(h)} = f_1^{S(h)} f_2^{S(h)} \dots f_n^{S(h)}$. When $k = 2$, the 1-shift of f is its complement. Next result allows us to restrict the domain of strings to be studied.

► **Lemma 5.** *Let Σ be a k -ary alphabet and $f \in \Sigma^*$. Then*

- *f is Lee-isometric if and only if f^R is Lee-isometric*
- *for any $h \in \Sigma$, f is Lee-isometric if and only if $f^{S(h)}$ is Lee-isometric.*

Proof. Suppose that f is Lee-isometric and let $u, v \in \Sigma^d$ be f^R -free words, for some $d \geq n$. Clearly, $u^R, v^R \in \Sigma^d$ are f -free words and $\text{dist}_L(u^R, v^R) = \text{dist}_L(u, v)$. Since f is Lee-isometric, then there is an f -free Lee-transformation from u^R to v^R of length equal to $\text{dist}_L(u^R, v^R)$, say $w_0 = u^R, w_1, \dots, w_h = v^R$. Since w_i is f -free, for $1 \leq i \leq h$, then w_i^R is f^R -free, for $1 \leq i \leq h$ and $w_0^R, w_1^R, \dots, w_h^R$ is an f^R -free Lee-transformation from u to v of length equal to $\text{dist}_L(u, v)$. The same reasoning applied to f^R shows that the converse is true, since $(f^R)^R = f$. The second claim can be proved by a similar reasoning, noting that, for any $u, v \in \Sigma^*$, $\text{dist}_L(u, v) = \text{dist}_L(u^{S(h)}, v^{S(h)})$ and that u is f -free if and only if $u^{S(h)}$ is $f^{S(h)}$ -free. ◀

Let us show how to compute the Lee-index of a Lee-non-isometric k -ary word f . Recall that, when $k > 4$, any word is Lee-non-isometric and its Lee-index is equal to its length [1]. Therefore, from now on, all considerations are done only for alphabets of cardinality $k = 2, 3, 4$. The computation of the Lee-index is based on the construction of the pairs of witnesses for f given in [1, 2] to show Theorem 4. The construction generalizes to k -ary words the one given for binary words in [13]. It will turn out that the Lee-index is the minimal length of such witnesses. Let us revisit this construction of pairs of witnesses, while highlighting some valuable features and adding some more results. First, let us state the following notation. Refer to Figures 2 and 3 for the construction of $\alpha_r, \beta_r, \eta_r$, and γ_r .

Notation. Let Σ be a k -ary alphabet and $f \in \Sigma^n$ have a 2-Lee-error overlap of length l and shift $r = n - l$. Let i, j , with $1 \leq i \leq j \leq l$, be error positions in f (possibly, $i = j$). Then,

- $f^{(i)}$ is the word obtained from f replacing $f[i]$ by $f[r + i]$
- if r is even, $t = j + r/2$, and $i \neq j$ then $f^{(j,t)}$ is the word obtained from f replacing $f[j]$ with $f[r + j]$, and $f[t]$ by $f[i]$
- $f^{(i,-)}$ is the word obtained from f replacing $f[i]$ with $(f[i] - 1) \bmod k$
- $f^{(i,+)}$ is the word obtained from f replacing $f[i]$ with $(f[i] + 1) \bmod k$
- $\alpha_r(f) = \text{pre}_r(f) f^{(i)}$ and $\beta_r(f) = \text{pre}_r(f) f^{(j)}$
- $\alpha'_r(f) = \text{pre}_r(f) f^{(i,-)}$ and $\beta'_r(f) = \text{pre}_r(f) f^{(i,+)}$
- if r is even, $\eta_r(f) = \text{pre}_r(f) f^{(i)} \text{su}_{f_{r/2}}(f)$
- if r is even, $\gamma_r(f) = \text{pre}_r(f) f^{(j,t)} \text{su}_{f_{r/2}}(f)$.

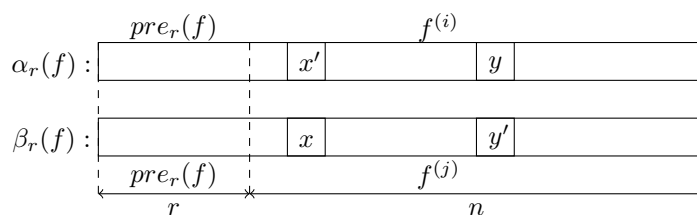


Figure 2 The words $\alpha_r(f)$ and $\beta_r(f)$.

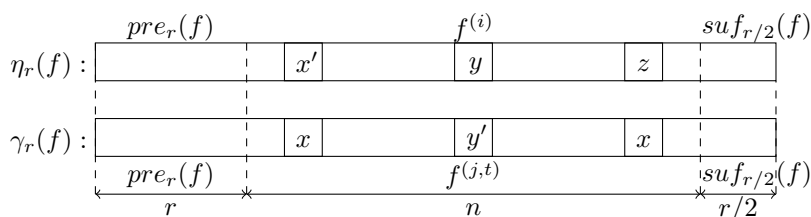


Figure 3 The words $\eta_r(f)$ and $\gamma_r(f)$.

The words introduced in the previous notation will constitute the pairs of witnesses for f . Note that such words are constructed in such a way to be f -free, unless for $\beta_r(f)$. When $\beta_r(f)$ is f -free then $(\alpha_r(f), \beta_r(f))$ is a pair of witnesses for f . Otherwise, it becomes necessary to consider the other words as above introduced. It turns out that $\beta_r(f)$ contains f as a factor if and only if the following *Condition*⁺ holds [2].

► **Definition 6.** Let Σ be a k -ary alphabet and $f \in \Sigma^n$ have a 2-Lee-error overlap of length l , shift $r = n - l$ and error positions i, j , with $1 \leq i < j \leq l$. We say that the 2-Lee-error overlap satisfies *Condition*⁺ if r is even, $j - i = r/2$, $f[r + i] = f[r + j]$, and $f[i..i + r/2 - 1] = f[j..j + r/2 - 1]$.

Let us start considering the case $k = 2, 3$. In this case, the Lee distance of two words coincides with the number of positions where they differ. Then, a 2-Lee-error overlap is given by two distinct error positions. This is no more true when $k = 4$; the quaternary case will be treated later.

Consider a Lee-non-isometric word $f \in \Sigma^n$, with $|\Sigma| = k$ and $k = 2, 3$. Then, f has a 2-Lee-error overlap. Actually, it may have more than one 2-Lee-error overlap. For any 2-Lee-error overlap, it is possible to construct a pair of witnesses for f as follows.

► **Proposition 7** ([2]). Let f be a Lee-non-isometric k -ary word, $k = 2, 3$. Consider a 2-Lee-error overlap of f , of length l , shift $r = n - l$, and error positions i, j , with $1 \leq i < j \leq l$.

1. If the 2-Lee-error overlap does not satisfy *Condition*⁺ then $(\alpha_r(f), \beta_r(f))$ is a pair of witnesses for f
2. If the 2-Lee-error overlap satisfies *Condition*⁺ and $1 \leq i \leq r/2$ then $(\eta_r(f), \gamma_r(f))$ is a pair of witnesses for f
3. If the 2-Lee-error overlap satisfies *Condition*⁺ and $i > r/2$ then f has a 2-Lee-error overlap of shift $r' > r$, which does not satisfy *Condition*⁺ and $(\alpha_{r'}(f), \beta_{r'}(f))$ is a pair of witnesses for f .

Let us show some examples of the construction in Proposition 7 for a ternary and a binary alphabet, respectively.

► **Example 8.** Consider the ternary alphabet $\Sigma = \{0, 1, 2\}$. Let $f = 021 \in \Sigma^3$. The word f has a 2-Lee-error overlap of length 2; here $n = 3$, $r = 1$, and $n - r = 2$. Hence, $f = 021$ is Lee-non-isometric from Theorem 4. Following the proof of the same theorem, let us exhibit a pair (α, β) of Lee-witnesses for f . We have that $pre_2(f)$ disagrees from $suf_2(f)$ in positions $i = 1$ and $j = 2$, and $f[i] = 0$, $f[j] = 2$, $f[r + i] = 2$ and $f[r + j] = 1$. Then, $\alpha_r(f) = pre_r(f)f^{(i)} = 0221$ and $\beta_r(f) = pre_r(f)f^{(j)} = 0011$. The words $\alpha_r(f)$ and $\beta_r(f)$ are f -free words and $dist_L(\alpha_r(f), \beta_r(f)) = 2$. Moreover, there is no f -free Lee-transformation from $\alpha_r(f)$ to $\beta_r(f)$ of length $dist_L(\alpha_r(f), \beta_r(f)) = 2$. Indeed, if we replace $\alpha_r(f)[2]$ with $\beta_r(f)[2] = 0$ then f occurs at position 2 of $\alpha_r(f)$; if we replace $\alpha_r(f)[3]$ with $\beta_r(f)[3] = 1$ then f occurs at position 1.

► **Example 9.** Consider the binary alphabet $B = \{0, 1\}$. Let $f = 0011 \in B^4$. The word f has a 2-Lee-error overlap of length 2. Hence, $f = 0011$ is Lee-non-isometric from Theorem 4. Note that the 2-Lee-error overlap of f satisfies *Condition*⁺. Indeed, we have $i = 1$, $j = 2$, $r = 2$, $f[r + i] = f[3] = 1 = f[4] = f[r + j]$ and $f[i] = 0 = f[j]$. Therefore, following the proof of Theorem 4 in the case that the 2-Lee-error overlap of f satisfies *Condition*⁺ and $1 \leq i \leq r/2$, let us set $t = (r/2) + j = 3$ and let us consider the two words $\eta_r(f) = pre_r(f)f^{(i)}suf_{r/2}(f) = 0010111$ and $\gamma_r(f) = pre_r(f)f^{(j,t)}suf_{r/2}(f) = 0001011$. The words $\eta_r(f)$ and $\gamma_r(f)$ are f -free words and $dist_L(\eta_r(f), \gamma_r(f)) = 3$. Moreover, there is no f -free Lee-transformation from $\eta_r(f)$ to $\gamma_r(f)$ of length $dist_L(\eta_r(f), \gamma_r(f)) = 3$. Indeed, if we replace $\eta_r(f)[3]$ with $\gamma_r(f)[3] = 0$ then f occurs at position 3 of $\eta_r(f)$; if we replace $\eta_r(f)[4]$ with $\gamma_r(f)[4] = 1$ then f occurs at position 1 and if we replace $\eta_r(f)[5]$ with $\gamma_r(f)[5] = 0$ then f occurs at position 4.

Consider now the case of the quaternary alphabet $\Delta = \{A, C, T, G\}$.

The construction of a pair of witnesses for a Lee-non-isometric quaternary word f is obtained in [2] referring to the binary representation of f . Actually, there is an isomorphism between quaternary words and binary words of even length. It is given by the map which associates to A, C, T, G the binary words 00, 01, 11, 10, respectively. Hence, a word $f \in \Delta^n$ will be possibly denoted as $(f)_4$ to stress its belonging to the quaternary alphabet, while its binary representation in $\{0, 1\}^{2n}$ will be denoted as $(f)_2$. The correspondence preserves the Lee distance.

Let $(f)_4 \in \Delta^n$ be a Lee-non-isometric word, r be the shift of a 2-Lee-error overlap of $(f)_4$ and m the number of its error positions. Recall that $m = 1$ or $m = 2$; see Remark 3. Note that $(f)_2$ has 2-Lee-error overlap of shift $2r$. A pair of witnesses for $(f)_4$ is obtained in [1, 2] considering the pair of witnesses for $(f)_2$, constructed as in Proposition 7, and representing it in quaternary. Recall that $(A)_2 = 00$, $(C)_2 = 01$, $(T)_2 = 11$, and $(G)_2 = 10$. For example, if the 2-Lee-error overlap of shift $2r$ of $(f)_2$ fills case 1 of Proposition 7 then $(\alpha_{2r}((f)_2), \beta_{2r}((f)_2))$ is a pair of witnesses for $(f)_2$. Thus, the quaternary representation of this pair of witnesses for $(f)_2$, that is $((\alpha_{2r}((f)_2))_4, (\beta_{2r}((f)_2))_4)$, is a pair of witnesses for $(f)_4$.

Next proposition shows that, indeed, the pair of witnesses for $(f)_4$ as just obtained from a 2-Lee-error overlap can be directly constructed from $(f)_4$, without going through its binary representation. Therefore, let us simply denote by f the quaternary word. Example 11 shows both constructions.

► **Proposition 10.** *Let f be a Lee-non-isometric k -ary word, $k = 4$. Consider a 2-Lee-error overlap of f , of length l , shift $r = n - l$, and error positions i, j , with $1 \leq i \leq j \leq l$ ($i = j$ if $m = 1$).*

1. *If $m = 1$, then $(\alpha'_r(f), \beta'_r(f))$ is a pair of Lee-witnesses for f .*

2. If $m = 2$ then

- a. if the 2-Lee-error overlap does not satisfy *Condition*⁺, then $(\alpha_r(f), \beta_r(f))$ is a pair of Lee-witnesses for f .
- b. If the 2-Lee-error overlap satisfies *Condition*⁺ and $i \leq r/2$, then $(\eta_r(f), \gamma_r(f))$ is a pair of Lee-witnesses for f .
- c. If the 2-Lee-error overlap satisfies *Condition*⁺ and $i > r/2$, then f has a 2-Lee-error overlap of shift $r' > r$, which does not satisfy *Condition*⁺ and $(\alpha_{r'}(f), \beta_{r'}(f))$ is a pair of Lee-witnesses for f .

Proof. Consider $(f)_2$, the binary representation of f . Since f has a 2-Lee-error overlap of shift r , then $(f)_2$ has a 2-Lee-error overlap of shift $2r$; let i, j , with $i < j$, be its error positions.

In the case $m = 1$, we have that $i = j$ and that $f[i]$ and $f[r + i]$ are two complementary symbols. Moreover, we have $s = 2i - 1$ and $z = s + 1 = 2i$. Suppose $f[i] = A$ and $f[r + i] = T$; the other cases can be treated analogously. We have $(f)_2[s] = 0$, $(f)_2[2r + s] = 1$, $(f)_2[z] = 0$, $(f)_2[2r + z] = 1$. Therefore, the quaternary representation of $(f)_2^{(s)}$ ($(f)_2^{(z)}$, resp.) coincides with the word obtained from f by replacing the symbol A in position i with the symbol G (C , resp.), that is $((f)_2^{(s)})_4 = f^{(i,-)}$ ($((f)_2^{(z)})_4 = f^{(i,+)}$, resp.). Moreover, $pre_{2r}((f)_2)$, represented in the quaternary alphabet, coincides with $pre_r(f)$. Hence, $(\alpha_{2r}((f)_2))_4 = pre_r(f)f^{(i,-)}$ and $(\beta_{2r}((f)_2))_4 = pre_r(f)f^{(i,+)}$. In [2], it is proved that the quaternary representation of $(\alpha_{2r}((f)_2), \beta_{2r}((f)_2))$ is a pair of Lee-witnesses for f . Hence the thesis follows.

In the case $m = 2$, we have that $i < j$, $f[i]$ and $f[r + i]$ ($f[j]$ and $f[r + j]$, resp.) differ because of two non-complementary symbols.

- a. If the 2-Lee-error overlap does not satisfy *Condition*⁺ then the quaternary representation of $(\alpha_{2r}((f)_2), \beta_{2r}((f)_2))$ is a pair of Lee-witnesses for f ; see [1]. Suppose $f[i] = T$, $f[r + i] = C$, $f[j] = G$ and $f[r + j] = A$; the other cases can be treated analogously. We have $s = 2i - 1$ and $z = 2j - 1$, $(f)_2[s] = 1$, $(f)_2[2r + s] = 0$, $(f)_2[z] = 1$, $(f)_2[2r + z] = 0$. Therefore, $(f)_2^{(s)}$ ($(f)_2^{(z)}$, resp.), represented in the quaternary alphabet, coincides with the word obtained from f by replacing T (G , resp.) in position i (j , resp.) with C (A , resp.), that is $((f)_2^{(s)})_4 = f^{(i)}$ ($((f)_2^{(z)})_4 = f^{(j)}$, resp.). Moreover, $pre_{2r}((f)_2)$, represented in the quaternary alphabet, coincides with $pre_r(f)$ and the thesis follows.
- b. If the 2-Lee-error overlap of f of shift r satisfies *Condition*⁺ and $i \leq r/2$ then the 2-Lee-error overlap of $(f)_2$ of shift $2r$ satisfies *Condition*⁺. The proof is given for $f[i] = f[j] = A$ and $f[r + i] = f[r + j] = C$; the other cases can be treated analogously. Then, $s = 2i$ and $z = 2j$. Since the 2-Lee-error overlap of f of shift r satisfies *Condition*⁺, then r is even, $j - i = r/2$. Therefore, $z - s = 2(j - i) = r$ is even and it is equal to the half of $2r$, the shift of the 2-Lee-error overlap of $(f)_2$. Moreover, from $f[r + i] = f[r + j]$ and $f[i..i + r/2 - 1] = f[j..j + r/2 - 1]$, it follows $f[2r + s] = f[2r + z]$ and $f[s..s + r - 1] = f[z..z + r - 1]$. At last, $i \leq r/2$, implies $s \leq r$. In this case, from [1], $\eta_{2r}((f)_2)$, $\gamma_{2r}((f)_2)$, represented in the quaternary alphabet, is a pair of Lee-witnesses for f . Recall that $\eta_{2r}((f)_2) = pre_{2r}((f)_2)(f)_2^{(s)} su_{f_{r/2}}((f)_2)$ and $\gamma_{2r}((f)_2) = pre_{2r}((f)_2)(f)_2^{(z,t)} su_{f_{r/2}}((f)_2)$. Then, $(f)_2^{(s)}$ will be obtained from $(f)_2$ by replacing $(f)_2[s] = 0$ with $(f)_2[2r + s] = 1$. Therefore, its quaternary representation coincides with the word obtained from f by replacing A in position i with C , that is $((f)_2^{(s)})_4 = f^{(i)}$. Analogous considerations show that $(\eta_{2r}((f)_2))_4 = pre_r(f)f^{(i)} su_{f_{r/2}}(f)$ and $(\gamma_{2r}((f)_2))_4 = pre_r(f)f^{(j,t)} su_{f_{r/2}}(f)$ and the thesis follows.
- c. This case can be treated as case a. ◀

► **Example 11.** Let $(f)_4 = AGCT \in \Delta^*$ and, hence, $(f)_2 = 00100111 \in (B^2)^*$. Then, $(f)_4$ has a 2-Lee-error overlap of length $l = 1$ and shift $r = 3$. In this case $m = 1$ and $i = 1$ is the unique error position with $(f)_4[1] = A$ and $(f)_4[4] = T$, that is the error is caused by two complementary symbols. A pair of witnesses for $(f)_4$ can be constructed from a pair of witnesses for $(f)_2$, as follows. Consider the 2-Lee-error overlap of $(f)_2$ of even length $2l = 2$ and shift $2r = 6$. It is caused by two different error positions, $s = 1$ and $z = 2$, since $(f)_2[1] = (f)_2[2] = 0$, while $(f)_2[7] = (f)_2[8] = 1$. Starting from this 2-Lee-error overlap of $(f)_2$, the pair $(\alpha_6((f)_2), \beta_6((f)_2))$ of Lee-witnesses for $(f)_2$ can be obtained. The pair is composed of $\alpha_6((f)_2) = 00100110100111$ and $\beta_6((f)_2) = 00100101100111$. In this case, the 2-Lee-error overlap of $(f)_2$ does not satisfy *Condition*⁺, since $z - s = 1$ is different from $r = 3$. Therefore, coming back to the quaternary representation, $(\alpha_3(AGCT), \beta_3(AGCT)) = (AGCGGCT, AGCCGCT)$ is a pair of Lee-witnesses for $(f)_4$.

On the other hand, this pair of witnesses for $AGCT$ can be directly constructed from $AGCT$, without going through the binary representation. Following Proposition 10 in the case $m = 1$, $\alpha_3(f) = pre_3(f)f^{(1,-)}$, $\beta_3(f) = pre_3(f)f^{(1,+)}$. Observe that $f^{(1,-)} = GGCT$ and $f^{(1,+)} = CGCT$. Hence, $\alpha_3(f) = pre_3(f)f^{(1,-)} = AGCGGCT$ and $\beta_3(f) = pre_3(f)f^{(1,+)} = AGCCGCT$. Finally, $(\alpha_3(AGCT), \beta_3(AGCT)) = (AGCGGCT, AGCCGCT)$ is the same pair of Lee-witnesses for $(f)_4$, as previously computed using the binary representation.

Let us come back to the computation of the Lee-index in the general case of a k -ary word with $k = 2, 3, 4$. Proposition 14 proves that the Lee-index is the minimal length of the Lee-witnesses as constructed in Propositions 7 and 10.

► **Remark 12.** Let $f \in \Sigma^*$ and let $u, v \in \Sigma^*$ be two f -free words. Consider any f -free Lee-transformation from u to v of length equal to $dist_L(u, v)$. Then, only symbols in the positions where u and v differ are modified in this transformation. Moreover, at each step of the Lee-transformation, a symbol x can be replaced by y only if $dist_L(x, y) = 1$. Hence, each position i such that $dist_L(u[i], v[i]) = d$ is replaced exactly d times.

► **Remark 13.** Let $f \in \Sigma^*$ be a Lee-non-isometric word and (u, v) be a pair of Lee-witnesses for f with $u, v \in \Sigma^d$ be f -free words. Let $h = dist_L(u, v)$ and suppose h to be minimal. Let $V = \{i_1, i_2, \dots, i_m\}$, with $1 \leq i_1 < i_2 < \dots < i_m \leq d$, be the set of all the positions where u and v differ; $m \leq h$. Consider a Lee-transformation of length h , $u = w_0, w_1, \dots, w_h = v$. Then, for any $j = 1, 2, \dots, h-1$, w_j has f as a factor. Moreover, for any error position $i \in V$, and any Lee-transformation of u in v of length h which starts changing $u[i]$ (cfr. Remark 12), the word obtained after this first replacement contains an occurrence of f including position i .

► **Proposition 14.** Let f be a Lee-non-isometric k -ary word, $k = 2, 3, 4$. Then, the Lee-index of f , $I_L(f)$, is the minimum length of words $\alpha_r(f)$, $\eta_r(f)$, or $\alpha'_r(f)$, as appropriate, where r is taken over the shifts of all 2-Lee-error overlaps of f .

Proof. Let f be a Lee-non-isometric k -ary word and $|f| = n$. Let (u, v) be the pair of witnesses of minimal distance $d_L(u, v)$ among all witnesses of minimal length. Note that (u, v) is also of minimal distance. Then, $I_L(f) = |u|$. Let V be the set of all error positions. The minimality of the distance of u and v implies that, when changing in u the symbol in any position $i \in V$, as in a Lee-transformation from u to v , then an occurrence f_i of f appears as a factor; see Remark 13. This f_i covers i and another error position (the same, if $|V| = 1$). The minimality of the length of u implies that the occurrences of f_i for all $i \in V$ completely cover u . Hence, let f_{i_1} be the occurrence that covers position 1 and f_{i_2} be the occurrence that covers position n .

If f_{i_1} and f_{i_2} contain both positions i_1 and i_2 , then f has a 2-Lee-error overlap of shift $r = |u| - n$ and error positions i_1 and i_2 . Then, u will eventually be $\alpha_r(f)$, $\alpha'_r(f)$, $\beta_r(f)$, or $\beta'_r(f)$, and the claim is proved.

Otherwise, there is another error position $i_3 \in V$ and a corresponding intermediate occurrence f_{i_3} of f . Each occurrence of f must contain two of these error positions. Observe that if an occurrence of f contains the first and the last error position (in non-decreasing order) then it also contains the second one. Then, there are two occurrences of f , say f_i and f_j which contains both error positions i and j ; note that i and j cannot be the first and the last error position in this case. Then f has a 2-Lee-error overlap with error positions i and j ; let r be its shift. Consider the pair $(\alpha_r(f), \beta_r(f))$, if $i \neq j$, or $(\alpha'_r(f), \beta'_r(f))$, if $i = j$. The length of such words is strictly less than $|u|$. Then, this pair cannot be a pair of witnesses for f , because of the minimality of the length of u . This means that $\beta_r(f)$ ($\beta'_r(f)$, resp.) is not f -free. Hence, *Condition*⁺ holds for the 2-Lee-error overlap with shift r and f occurs at position $r/2$ in $\beta_r(f)$ ($\beta'_r(f)$, resp.). Further, the shortest pair of witnesses that can be constructed in such situation with three occurrences of f covering u is $(\eta_r(f), \gamma_r(f))$ and finally $u = \eta_r(f)$ or $u = \gamma_r(f)$. ◀

Proposition 14 suggests to compute the Lee-index of a word considering all its 2-Lee-error overlaps, obtaining for each 2-Lee-error overlap the corresponding pair of witnesses as in Proposition 7 (if $k = 2, 3$) or in Proposition 10 (if $k = 4$), and then computing the minimal length of a so obtained witness. The algorithm in next section will analyse the possible 2-Lee-error overlaps in increasing order of their shift. Nevertheless, note that it is not possible to stop at the first found 2-Lee-error overlap. The 2-Lee-error overlap that corresponds to the witnesses of minimal length can be a subsequent one, as shown in Example 15.

► **Example 15.** Consider the ternary alphabet $\Sigma = \{0, 1, 2\}$. For any $h \geq 0$, let $w = 2^h$ and $f = 0w0w1w1$, with $|f| = n = 3h + 4$. It can be observed that, for any $h \geq 0$, f has two 2-Lee-error overlaps, one of length $h + 2$, for which $d_L(0w0, 1w1) = 2$ and one of length $h + 1$, for which $d_L(w0, 1w) = 2$. Then, for any $h \geq 0$, f is Lee-non-isometric applying Theorem 4. The first pair of Lee-witnesses for f can be constructed starting from the 2-Lee-error overlap of length $l = h + 2$ and shift $r = n - l = 2h + 2$, following the proof of Theorem 4. This overlap satisfies *Condition*⁺ and the corresponding pair of Lee-witnesses for f is $(\eta_r(f), \gamma_r(f))$, with $\eta_r(f) = 0w0w(1w0w1w1)w1$ and $\gamma_r(f) = 0w0w(0w1w0w1)w1$. The length of this pair of Lee-witnesses is $|\eta_r(f)| = |\gamma_r(f)| = 6h + 7$. The second pair of Lee-witnesses for f can be constructed starting from the 2-Lee-error overlap of length $h + 1$, following the proof of Theorem 4 again. This overlap does not satisfies *Condition*⁺ and the corresponding pair of Lee-witnesses for f is $(\alpha_r(f), \beta_r(f))$, with $\alpha_r(f) = 0w0w1(2w0w1w1)$ and $\beta_r(f) = 0w0w1(02^{h-1}10w1w1)$. The length of this pair of Lee-witnesses is $|\alpha_r(f)| = |\beta_r(f)| = 5h + 7$. Thus, the 2-Lee-error overlap that corresponds to the witnesses of minimal length is the second one and then it provides the Lee-index $I_L(f) = 5h + 7$.

4 The Algorithm

In this section, the results provided in Section 3 are applied to design an algorithm that computes the Lee-index of a word and yields a pair of Lee-witnesses of minimal length. It is assumed that Σ is a k -ary alphabet, with $k \leq 4$ and $f \in \Sigma^*$ is a finite sequence $f[0]f[1] \cdots f[n-1]$ of symbols in Σ , where n is the length of f and $f[i]$'s are its symbols. Note that now the indices of f start from 0, and not 1, in view of an implementation of the algorithm in the main programming languages. Recall that if f is Lee-isometric then its

Lee-index is ∞ , else it is the minimal length of two words u, v , such that (u, v) is a pair of Lee-witnesses for f . Observe that an algorithm to compute the Lee-index and its witnesses is given for binary alphabet in [13]; it runs in $O(n^3)$ time. The algorithm designed in this section runs in $O(n)$ time and space. Finally, note that this algorithm can be easily modified, without changing its complexity, to compute the index and related witnesses of a word, referring to Hamming distance, as considered, for example, in [13], or to other distances, instead of Lee one.

Let us sketch an algorithm which inputs a k -ary non-empty word f of length n , with $k = 2, 3, 4$, and outputs an integer I , that is the Lee-index of f , and a pair (u, v) of Lee-witnesses for f of length I . Its pseudo-code is given by Algorithm 1 and an example follows. The algorithm starts looking for all 2-Lee-error overlaps of f and saving them into a list. This is done by function `TwoErrorOverlaps` which can be computed in time and space $O(n)$, thanks to a preprocessing step which uses an enhanced suffix tree to answer Lowest Common Ancestor (LCA) queries in constant time. If there are not 2-Lee-error overlaps, then the algorithm sets the Lee-index I to ∞ . Otherwise, for each 2-Lee-error overlap, it constructs a pair of Lee-witnesses calling the function `WitnessesConstructor`. According to Propositions 7 and 10, the construction depends on whether the *Condition*⁺ is satisfied; function `CondPlus` checks this. Then, it outputs the Lee-index I as the minimal length of all these pairs of Lee-witnesses, following Proposition 14. It also outputs a pair (u, v) of Lee-witnesses of length I . Note that the Lee-index of f is upper bounded by $I_L(f) \leq 2n - 1$ in [2]; then I can be initialized as $I = 2n$ (in Line 6). Since the 2-Lee-error overlaps are at most $n - 1$, there are $O(n)$ calls to `WitnessesConstructor`, each running in time $O(1)$. The overall time and space complexity of Algorithm 1 is thus $O(n)$.

► **Proposition 16.** *Let Σ be a k -ary alphabet, with $k \leq 4$ and $f \in \Sigma^n$ be a non-empty word of length n . The Lee-index of f and a pair of Lee-witnesses of minimal length for f can be computed in time $O(n)$ with additional $O(n)$ space.*

Proof. Let us analyse the main functions in Algorithm 1.

■ `TwoErrorOverlaps` inputs the word f and outputs all lengths of its 2-Lee-error overlaps in the list `2eolens` and all corresponding error positions in the list `allerrpos`. It is similar to Algorithm 3 in [4], with the difference that Algorithm 3 in [4] checks only if a word f has at least one 2-Lee error overlap, while this function finds all 2-Lee error overlaps of f and stores their lengths in the list `2eolens`. Further, it stores all corresponding error positions in the list `allerrpos`, which is, therefore, a list of lists. It is based on a technique called the Kangaroo method [5, 11], used in designing efficient pattern matching with mismatches algorithms. It computes the number of mistakes in a given alignment by “jumping” from one error to the next. It allows to check, for a given position i in f , whether f has a 2-Lee-error overlap of length $n - i$ in time $O(1)$. To do this, it first computes in time and space $O(n)$ the suffix tree of f enhanced to answer to Lowest Common Ancestor (LCA) queries in time $O(1)$. A call to `LCA(i, j)` returns the length of the longest common prefix between the suffix of f starting from position i and the one starting from position j . The function `TwoErrorOverlaps` checks whether the suffix starting at i has two mismatches with its prefix of the same length. It uses a variable l which gives the length of the current overlap and a variable d which contains the current Lee distance. They are increased when a mismatch has been found. Since there are at most two LCA queries for a given i , this can be done in $O(1)$ time. Thus, the time and space complexity of `TwoErrorOverlaps` is $O(n)$.

■ **Algorithm 1** Computing the Lee-index and Lee-witnesses for f .

Input: a k -ary non-empty word f of length n , with $k \leq 4$

Output: an integer I , Lee-index of f , and a pair of words (u, v) , Lee-witnesses for f of length I

```

1 (2eolens, allerrpos) ← TwoErrorOverlaps(f);
2 (u, v) ← (empty, empty);
3 if 2eolens is empty then
4   I ← ∞;
5 else
6   I ← 2(len(f));
7   for i ← 0 to len(2eolens) - 1 do
8     (utmp, vtmp) ← WitnessesConstructor(f, 2eolens[i], allerrpos[i]);
9     if len(u) < I then
10      I ← len(u);
11      (u, v) ← (utmp, vtmp);
12 return I, (u, v);

13 function TwoErrorOverlaps(f):
14   (2eolens, allerrpos, n) ← ([], [], len(f));
15   for i ← 1 to n - 1 do
16     (l, d, allerrpostmp) ← (0, 0, []);
17     while d ≤ 2 do
18       l ← l + LCA(l, i + l);
19       if l < n - i then
20         allerrpostmp.append(l + 1);
21       if d = 2 and l = n - i then
22         2eolens.append(l);
23         allerrpos.append(allerrpostmp);
24       if d < 2 and l < n - i then
25         l ← l + 1;
26         d ← d + dL(f[l], f[i + l]);
27       else
28         BREAK
29   return (2eolens, allerrpos);
30 end function

31 function WitnessesConstructor(f, l, errpos):
32   (n, r, i) ← (len(f), n - l, errpos[0]);
33   if len(errpos) = 1 then
34     (falfa1, fbeta1) ← (f, f);
35     alfa1[i] = (falfa1[i] - 1) mod 4;
36     beta1[i] = (fbeta1[i] + 1) mod 4;
37     u ← prer(f) + falfa1;
38     v ← prer(f) + fbeta1;
39   else
40     j ← errpos[1];
41     cplus ← CondPlus(f, r, errpos[0], errpos[1]);
42     if cplus = False then
43       (falfa, fbeta) ← (f, f);
44       falfa[i] = f[r + i];
45       fbeta[j] = f[r + j];
46       u ← prer(f) + falfa;
47       v ← prer(f) + fbeta;
48     else
49       if i ≤ r/2 then
50         (feta, fgamma) ← (f, f);
51         feta[i] = f[r + i];
52         fgamma[j] = f[r + j];
53         fgamma[r/2 + j] = f[i];
54         u ← prer(f) + feta;
55         v ← prer(f) + fgamma;
56   return (u, v);
57 end function

58 function CondPlus(f, r, i, j):
59   (cond1, cond2, cond3) ← (False, False, False);
60   if r mod 2 = 0 then
61     if j - i = r/2 then
62       cond1 ← True;
63       if f[r + i] = f[r + j] then
64         cond2 ← True;
65         if LCA(i, j) ≥ r/2 then
66           cond3 ← True;
67   return (cond1 and cond2 and cond3);
68 end function

```

- `WitnessesConstructor` inputs the word f , the length l of a 2-Lee error overlap of f , its corresponding error positions in the list $errpos$ and outputs a pair (u, v) of Lee-witnesses for f . This function constructs the pair (u, v) , according to Proposition 10, in two different ways, following that the 2-Lee-error overlap is caused by two complementary symbols (i.e., $m = 1$ and the list $errpos$ has only one element) or by two non-complementary symbols (i.e., $m = 2$ and the list $errpos$ has two elements). Note that the first case may occur only if the alphabet cardinality is $k = 4$. In this case, the function constructs u by appending to the prefix of f of length $r = n - l$ the word $f^{(i,-)}$ as defined in the list of Notation given in Section 3. Similarly, it constructs v , this time appending $f^{(i,+)}$. The second case follows case $m = 2$ in Proposition 10. It has other two subcases following that the function `CondPlus` returns `False` or `True`. In the first subcase, the pair (u, v) may be constructed according to case 2.a of Proposition 10. In the second case, if $i \leq r/2$, it may be constructed as case 2.b. Otherwise, it is case 2.c, and there is nothing else to do because it is proved that f always has another 2-Lee-error overlap of (even) length smaller than l , and thus the pair (u, v) will be constructed as in case 2.a in a subsequent call of the function. Since `CondPlus` runs in $O(1)$ time, all these instructions can be executed in constant time. Thus, the time complexity of `WitnessesConstructor` is $O(1)$.
- `CondPlus` inputs the word f , the integers r, i and j , where i and j are the error positions in the 2-Lee-error overlap of shift r . This function outputs `True` if $Condition^+$ is verified, `False`, otherwise. Recall that $Condition^+$ is defined in the list of Notation in Section 3. Note that $Condition^+$ may be `True` only if r is even. If r is even, then the function checks the other conditions in $cond1$, $cond2$, and $cond3$. In particular, $cond3$ is `True` iff $f[i..i + r/2 - 1] = f[j..j + r/2 - 1]$. This check is done in $O(1)$ time testing if $LCA(i, j) \geq r/2$, rather than in $O(r)$ time comparing all the symbols. Thus, all the instructions of `CondPlus` can be done in $O(1)$ time.

In summary, the overall time complexity of Algorithm 1 can be obtained as the sum of the cost of `TwoErrorOverlaps` and at most $n - 1$ times the cost of `WitnessesConstructor`. Thus, it is $O(n) + O(n) = O(n)$. The space complexity of Algorithm 1 is due to `TwoErrorOverlaps`, thus it is $O(n)$. ◀

► **Example 17.** Let us run Algorithm 1 to compute the Lee-index and a pair of Lee-witnesses for $f = f[0]f[1] \cdots f[5] = AGATAC$. It starts calling the function `TwoErrorOverlaps` with input $f = AGATAC$. This function finds two 2-Lee-error overlaps. The first one is of length 4, where the errors are in positions 1, 3 and are caused by non-complementary symbols; in fact, $f[1] = G \neq f[3] = T$ and $f[3] = T \neq f[5] = C$. The second one is of length 2, and has a unique error position 1; in fact, $f[1] = G \neq f[5] = C$ and further $dist_L(G, C) = 2$, since G and C are complementary symbols. Thus, `TwoErrorOverlaps` outputs $2eolens = [4, 2]$ and $allerrpos = [[1, 3], [1]]$.

Then, coming back to Line 3 of the main algorithm, because $2eolens$ is not empty, the algorithm initializes the output variable $I = 12$. For $i = 0$ to 1 the algorithm calls twice the function `WitnessesConstructor`.

The first call takes as input $(AGATAC, 4, [1, 3])$ and sets $n = 6, r = 2, i = 1$. Because $len(errpos) = 2$, then $j = 3$ and $cplus = False$ after calling `CondPlus(AGATAC, 2, 1, 3)`. Thus, the function `WitnessesConstructor` computes and outputs $u = AGATATAC$ and $v = AGAGACAC$, obtained as $\alpha_r(f)$ and $\beta_r(f)$; they have two error positions containing non-complementary symbols. Since $len(u) = 8 < I = 12$, the algorithm updates $I = 8$ and $(u, v) = (AGATATAC, AGAGACAC)$.



The second call to `WitnessesConstructor` takes as input $(AGATAC, 2, [1])$ and sets $n = 6, r = 4, i = 1$. Because $len(errpos) = 1$, then the function outputs $u = AGATAAATAC$ and $v = AGATATATAC$, obtained as $\alpha_r(f)$ and $\beta_r(f)$; they have one error position containing complementary symbols. Since $len(u) = 10$ is greater than $I = 8$ the algorithm does not update neither I nor (u, v) .

Therefore, the main algorithm outputs the Lee-index $I = 8$ and the pair of Lee-witnesses $(u, v) = (AGATATAC, AGAGACAC)$.



References

- 1 Marcella Anselmo, Manuela Flores, and Maria Madonia. On k-ary n-cubes and isometric words. *Preprint*, 2021. URL: <https://docenti.unisa.it/uploads/rescue/385/8179/afm-k-aryisometricwords.pdf>.
- 2 Marcella Anselmo, Manuela Flores, and Maria Madonia. Quaternary n-cubes and isometric words. In Thierry Lecroq and Svetlana Puzynina, editors, *Combinatorics on Words*, pages 27–39, Cham, 2021. Springer International Publishing.
- 3 Marcella Anselmo, Dora Giammarresi, Maria Madonia, and Carla Selmi. Bad pictures: Some structural properties related to overlaps. In Galina Jirásková and Giovanni Pighizzini, editors, *DCFS 2020*, volume 12442 of *Lect. Notes Comput. Sci.*, pages 13–25. Springer, 2020. doi:10.1007/978-3-030-62536-8_2.
- 4 Marie-Pierre Béal and Maxime Crochemore. Checking whether a word is hamming-isometric in linear time. *arXiv preprint arXiv:2106.10541*, 2021.
- 5 Zvi Galil and Raffaele Giancarlo. Improved string matching with k mismatches. *ACM SIGACT News*, 17(4):52–54, 1986.
- 6 Frank Harary, John P. Hayes, and Horng-Jyh Wu. A survey of the theory of hypercube graphs. *Computers & Mathematics with Applications*, 15(4):277–289, 1988. doi:10.1016/0898-1221(88)90213-1.
- 7 W. . Hsu. Fibonacci cubes-a new interconnection topology. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):3–12, 1993. doi:10.1109/71.205649.
- 8 Aleksandar Ilić, Sandi Klavžar, and Yoomi Rho. Generalized fibonacci cubes. *Discrete Mathematics*, 312(1):2–11, 2012. doi:10.1016/j.disc.2011.02.015.
- 9 Sandi Klavžar. Structure of fibonacci cubes: A survey. *Journal of Combinatorial Optimization*, 25, May 2013. doi:10.1007/s10878-011-9433-z.
- 10 Sandi Klavžar and Sergey V. Shpectorov. Asymptotic number of isometric generalized Fibonacci cubes. *Eur. J. Comb.*, 33(2):220–226, 2012.
- 11 Gad M. Landau and Uzi Vishkin. Efficient string matching in the presence of errors. In *26th Annual Symposium on Foundations of Computer Science*, pages 126–136. IEEE, 1985.
- 12 Weizhen Mao and David M. Nicol. On k-ary n-cubes: theory and applications. *Discrete Applied Mathematics*, 129(1):171–193, 2003. doi:10.1016/S0166-218X(02)00238-X.
- 13 Jianxin Wei. The structures of bad words. *Eur. J. Comb.*, 59:204–214, 2017.

Chess Is Hard Even for a Single Player

N.R. Aravind  

Department of Computer Science and Engineering, Indian Institute of Technology, Hyderabad, India

Neeldhara Misra  

Department of Computer Science and Engineering, Indian Institute of Technology, Gandhinagar, India

Harshil Mittal 

Department of Computer Science and Engineering, Indian Institute of Technology, Gandhinagar, India

Abstract

We introduce a generalization of “Solo Chess”, a single-player variant of the game that can be played on chess.com. The standard version of the game is played on a regular 8×8 chessboard by a single player, with only white pieces, using the following rules: every move must capture a piece, no piece may capture more than 2 times, and if there is a King on the board, it must be the final piece. The goal is to clear the board, i.e. make a sequence of captures after which only one piece is left.

We generalize this game to unbounded boards with n pieces, each of which have a given number of captures that they are permitted to make. We show that GENERALIZED SOLO CHESS is NP-complete, even when it is played by only rooks that have at most two captures remaining. It also turns out to be NP-complete even when every piece is a queen with exactly two captures remaining in the initial configuration. In contrast, we show that solvable instances of GENERALIZED SOLO CHESS can be completely characterized when the game is: a) played by rooks on a one-dimensional board, and b) played by pawns with two captures left on a 2D board.

Inspired by GENERALIZED SOLO CHESS, we also introduce the GRAPH CAPTURE GAME, which involves clearing a graph of tokens via captures along edges. This game subsumes GENERALIZED SOLO CHESS played by knights. We show that the GRAPH CAPTURE GAME is NP-complete for undirected graphs and DAGs.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → Design and analysis of algorithms

Keywords and phrases chess, strategy, board games, NP-complete

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.5

Related Version *Full Version:* <https://arxiv.org/abs/2203.14864> [1]

Funding The second author acknowledges support from IIT Gandhinagar and the SERB-MATRICES grant MTR/2017/001033. The second and third authors also acknowledge support from the SERB-ECR grant ECR/2018/002967.

Acknowledgements The authors would like to thank the anonymous reviewers of FUN 2022 for their useful feedback.

1 Introduction

Chess, the perfect-information two-player board game, needs to introduction. With origins dating back to as early as the 7th century, organized chess arose in the 19th century to become one of the world’s most popular games in current times. At the time of this writing, the recent pandemic years witnessed a phenomenal growth of the already popular game, among spectators and amateur players alike. One of the most active computer chess sites, chess.com, is reported to have more than 75 million members, and about four million people sign in everyday.



© N.R. Aravind, Neeldhara Misra, and Harshil Mittal;
licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

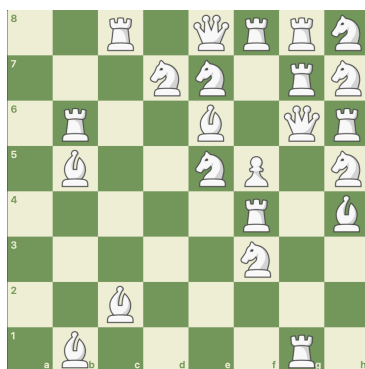
Editors: Pierre Fraigniaud and Yushi Uno; Article No. 5; pp. 5:1–5:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

5:2 Chess Is Hard Even for a Single Player



■ **Figure 1** An example of a Solo Chess configuration.

As the reader likely knows already, chess is played on a square chessboard with 64 squares arranged in an eight-by-eight grid. At the start, each player (one controlling the white pieces, the other controlling the black pieces) controls sixteen pieces: one king, one queen, two rooks, two bishops, two knights, and eight pawns. The object of the game is to checkmate the opponent’s king, whereby the king is under immediate attack (in “check”) and there is no way for it to escape. There are also several ways a game can end in a draw. The movements of the individual pieces are subject to different constraints. While several chess engines exist for this classical version of the game, it is also known that the generalized version of chess, played on a $n \times n$ board by two players with $2n$ pieces is complete for the class EXPTIME [5]. Cooperative versions of chess are also known to be hard [3].

The game of SOLO CHESS is an arguably natural single-player variant of the game. We consider here a version that can be found among the chess puzzles on chess.com. The game is played on a regular 8×8 chessboard by a single player, with only white pieces, using the following rules: every move must capture a piece, no piece may capture more than 2 times, and if there is a King on the board, it must be the final piece. Given a board with, say, n pieces in some configuration, the goal is to play a sequence of captures that “clear” the board. To the best of our knowledge, chess.com presents its players only with solvable configurations, even if this may not always be obvious¹. The solutions, however, need not be unique.

While the focus of our contribution here is on computational aspects of determining if a solo chess instance is solvable, we refer the reader to [2] for a comprehensive and entertaining introduction to combinatorial game theory at large.

Our Contributions

We introduce a natural generalization of SOLO CHESS that we call GENERALIZED SOLO CHESS(P, d), where $P \subseteq \{\text{♔, ♑, ♖, ♗, ♘, ♙}\}$ is a collection of piece types and $d \in \mathbb{N}$. This version of the game is played on the infinite integer lattice where we are given, initially, the positions of n pieces, each of which is one of the types given in P . We are also given, for each piece, the number of captures it can make – and further, this bound is at most d . The goal is to figure out if there is a sequence of $(n - 1)$ valid captures such that: a) no piece captures more than the number of times it is allowed to capture; and b) the sequence of captures, when played out, “clears the board”, i.e. only one piece remains at the end.

¹ c.f. “Crazy Mode”.

We focus on settings where $|P| = 1$, i.e., when all pieces are of the same type. When the game is played only with rooks, we show that the problem is NP-complete even when $d = 2$, but is tractable when the game is restricted to a one-dimensional board for arbitrary d .

► **Theorem 1** (A characterization for rooks on 1D boards). *GENERALIZED SOLO CHESS (\mathbb{R}, d) with n rooks can be decided in $O(n)$ time for any $d \in \mathbb{N}$.*

► **Theorem 2** (Intractability for rooks). *GENERALIZED SOLO CHESS ($\mathbb{R}, 2$) is NP-complete.*

When all pieces are queens, note that GENERALIZED SOLO CHESS played on a one-dimensional board is equivalent to the game played by rooks. On the other hand, on a two-dimensional board, the game turns out to be hard even when all pieces can capture twice in the initial configuration, which is in the spirit of the regular game and is a strengthening of the hardness that we have for rooks.

► **Theorem 3** (Intractability for queens). *GENERALIZED SOLO CHESS ($\mathbb{Q}, 2$) is NP-complete even when all queens are allowed to capture at most twice.*

When all pieces are bishops, no piece has a valid move if the game is restricted to a one-dimensional board. On the other hand, it is easy to check that GENERALIZED SOLO CHESS played on a two-dimensional board with bishops only can be reduced to GENERALIZED SOLO CHESS played on a two-dimensional board with rooks only, by simply “rotating” the board 45 degrees. Therefore, we do not discuss the case of bishops explicitly.

We now turn to the case when the game is played only with pawns: as with bishops, the game is not interesting on a one-dimensional board. However, when played on a two-dimensional board with pawns that have two captures left, it turns out that we can efficiently characterize the solvable instances.

► **Theorem 4** (A characterization for pawns). *GENERALIZED SOLO CHESS ($\mathbb{P}, 2$) with n white pawns, each of which can capture at most twice, can be decided in $O(n)$ time.*

When the game is played by knights only, again the game is trivial on a one-dimensional board. On a two-dimensional board, consider the following graph that is naturally associated with any configuration of knights: we introduce a vertex for every occupied position, and a pair of vertices are adjacent if and only if the corresponding positions are mutually attacking. Note that for all other pieces considered so far, an attacking pair of positions need not imply that a capture is feasible, since there may be blocking pieces in some intermediate locations. Knights are unique in that the obstacles are immaterial. This motivates the GRAPH CAPTURE game: here we are given a graph with tokens on vertices, and the goal is to clear the tokens by a sequence of captures. The tokens can capture along edges and the number of captures that the tokens can make is given as a part of the input.

Note that GENERALIZED SOLO CHESS(\mathbb{K}, d) is a special case of of GRAPH CAPTURE(d). We show that solvable instances of the latter on undirected graphs are characterized by the presence of a rooted spanning tree with the property that every internal node has at least one leaf neighbor. However, we also show that finding such spanning trees is intractable. We also show that GRAPH CAPTURE(d) is NP-complete on DAGs.

► **Theorem 5** (Intractability of the graph capture game). *GRAPH CAPTURE(2) is NP-complete on undirected graphs and DAGs even when every token can capture at most twice.*

We remark that Theorem 5 has no immediate implications for Generalized Solo Chess (\mathbb{K}, d).

5:4 Chess Is Hard Even for a Single Player

The rest of the paper is organized as follows. We establish the notation that we will use in Section 2. The proof of Theorems 1 and 2 is given in Section 3.1.1 and Section 3.1.2, respectively. The proof of Theorem 3 is discussed in Section 3.2 and the proof of Theorem 4 is given in Section 3.3. Finally, the proof of Theorem 5 is shown separately for undirected graphs and DAGs in Sections 4.1 and 4.2.

2 Preliminaries

We use $[n]$ to denote the set $\{1, 2, \dots, n\}$. We consider the following generalization² of Solo Chess. We fix a subset P of $\{\text{♔}, \text{♚}, \text{♙}, \text{♘}, \text{♖}, \text{♗}\}$ and a positive integer d . The generalized game is played on an infinite two-dimensional board with n pieces. For each piece, we are given an initial location and the maximum number of captures the piece is permitted to make. Such an instance is solvable if there exists a sequence σ of $(n - 1)$ valid captures with each piece making at most as many captures as it is allowed to make. We note that a capture is valid if it respects the usual rules of movements in chess. The formal definition of the problem is the following.

GENERALIZED SOLO CHESS(P, d):

Input: A configuration C , which is specified by a list of n triplets (p, z, m) , where $p \in P$, $z \in \mathbb{N} \times \mathbb{N}$, and $0 \leq m \leq d$. We use C_i to refer to the i^{th} triplet in C .

Output: Decide if there exists a sequence of $(n - 1)$ captures starting from the board position described by C , such that the piece corresponding to $C[i][0]$ moves at most $C[i][2]$ times for all $1 \leq i \leq n$.

We note that GENERALIZED SOLO CHESS(P, d) is interesting when $d \geq 2$. Indeed, when $d = 1$, it can be efficiently determined if an instance of GENERALIZED SOLO CHESS($P, 1$) is solvable:

► **Observation 6.** *When $d = 1$, a configuration C is winning if and only if there's a square z containing a piece, such that z is reachable in one move from every other piece.*

Proof. The sufficiency of this condition is clear; to see the necessity, for each square y on which a capture was made, let $p(y)$ be the last piece to capture on y . Then $p(y)$ must be the last piece standing (as it can neither move again nor be captured), and further, y is the occupied square at the end of the game. Since there is exactly one occupied square at the end, this shows that all captures were made to the same square. ◀

Most of our results rely only on elementary graph-theoretic terminology and the notions of polynomial time reductions and NP-completeness. We refer the reader to the texts [6, 7] for the relevant background. The well-known [6, 4] NP-complete problems that we use in our reductions are the following:

1. RED-BLUE DOMINATING SET. Given a bipartite graph $G = (R \uplus B, E)$ and a positive integer k , determine if there is a subset $S \subseteq R$, $|S| \leq k$ such that $N[v] \cap S \neq \emptyset$ for all $v \in B$.

² Since our focus is on the case when the game is played by pieces of one type only, we do not involve the ♔ in our set of pieces. Note that because of the convention that kings are never captured, any such involving only kings is trivial.

2. COLORFUL RED-BLUE DOMINATING SET. Given a bipartite graph $G = (R \uplus B, E)$ where the red vertices are partitioned into k disjoint parts, determine if there is a choice of exactly one vertex from each part such that every blue vertex has at least one neighbor among the chosen vertices.
3. 3-SAT. Given a CNF formula with at most three literals per clause, determine if there is a truth assignment to the variables that satisfies the formula.

3 Solo Chess with a single piece

3.1 Rooks

3.1.1 1-Dimensional boards

In this section we consider GENERALIZED SOLO CHESS restricted to one-dimensional board played by rooks. It will be convenient to reason about such instances by using strings to represent game configurations; to this end we introduce some terminology.

► **Definition 7** (Configuration). *A configuration is string s over $\{0, 1, 2, \dots, d, \square\}$. It denotes a board of size $1 \times N$, where N is the length of the string. The cell $(1, j)$ is empty if $s[j] = \square$, and is otherwise occupied by a rook with b moves left where $b := s[j]$.*

We refer the reader to Figure 2 for an example and how a given board position translates to a configuration as defined above. Informally, a configuration is *solvable* if there is a valid sequence of moves that clears the board.

► **Definition 8** (ℓ -solvable configuration). *Let s be a configuration of length N and let $1 \leq \ell \leq N$. We say that s is ℓ -solvable if there exists a sequence of moves that clears the corresponding board with the final rook at the cell $(1, \ell)$.*

Our main goal in this section is to establish the following:

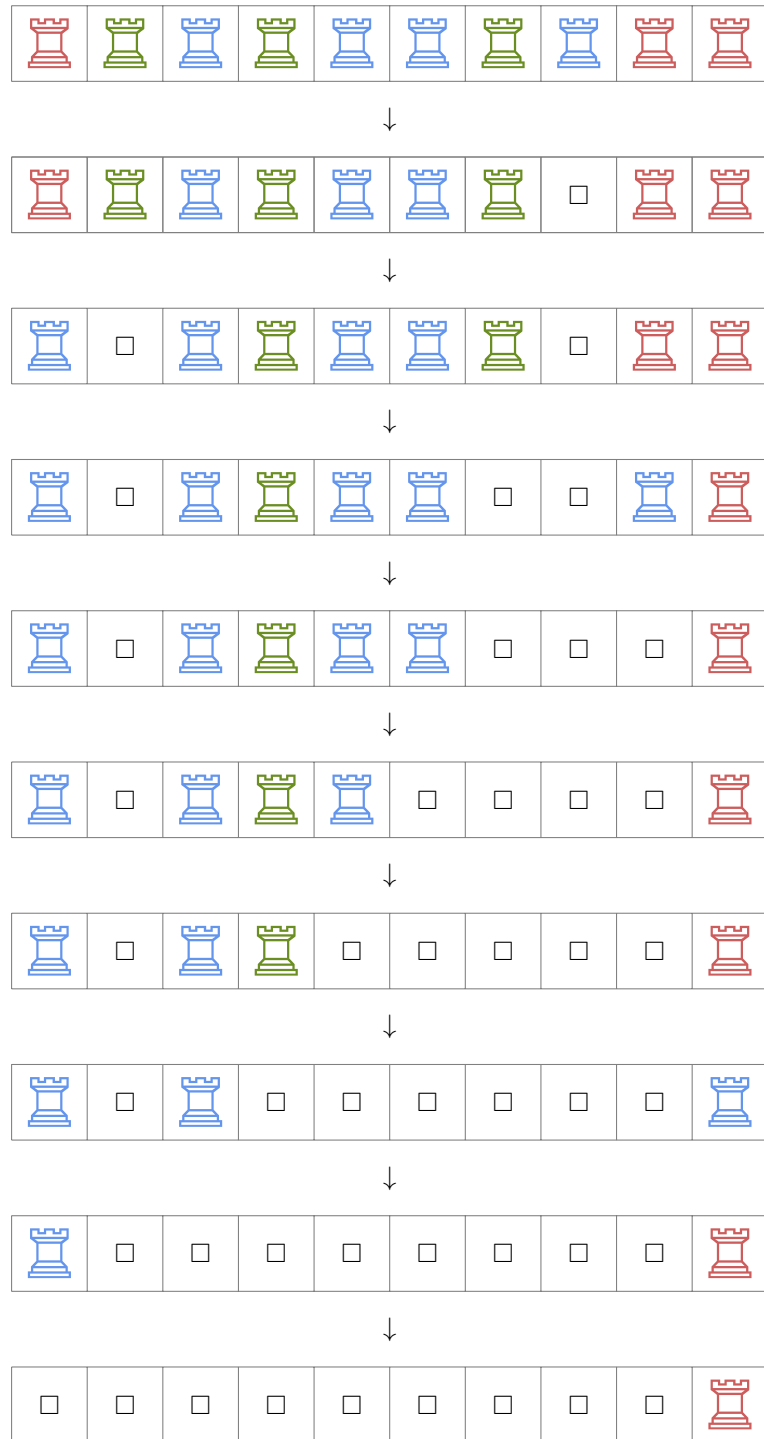
► **Theorem 1** (A characterization for rooks on 1D boards). *GENERALIZED SOLO CHESS (\mathbb{N}, d) with n rooks can be decided in $O(n)$ time for any $d \in \mathbb{N}$.*

Note that for any sequence (say σ) of moves that clears the board with final rook at the cell $(1, \ell)$, no move of σ empties the cell $(1, \ell)$ and thus, there's no move of σ wherein the cells containing the captured piece and the capturing piece are at different sides, i.e., one at left and the other at right, of the cell $(1, \ell)$. So, note that s is ℓ -solvable if and only if there is a position such that the sub-configurations to the left and right of location ℓ are independently solvable. We now develop a criteria for solving 1D configurations where the target piece is one of the extreme locations on the board. Note that when $d = 2$, the first criteria below amounts to saying that s is N -solvable iff $s[N] \neq \square$ and $s[1, \dots, N-1]$ has at least as many 2's as 0's; and the second criteria states that s is 1-solvable iff $s[1] \neq \square$ and $s[2, \dots, N]$ has at least as many 2's as 0's. A direct proof of this simpler statement is given in the Appendix [1].

► **Lemma 9.** *For every configuration s of length N ,*

1. s is N -solvable iff $s[N] \neq \square$ and $\sum_{\substack{1 \leq i \leq N-1: \\ s[i] \notin \{0, \square\}}} (s[i] - 1) \geq \text{number of 0's in } s[1, \dots, N-1]$
2. s is 1-solvable iff $s[1] \neq \square$ and $\sum_{\substack{2 \leq i \leq N: \\ s[i] \notin \{0, \square\}}} (s[i] - 1) \geq \text{number of 0's in } s[2, \dots, N]$

5:6 Chess Is Hard Even for a Single Player



■ **Figure 2** An example of a valid sequence of captures that clears the board. The initial configuration corresponds to the string 0212112100. In other words, the red, blue, and green rooks denote rooks with zero, one, and two moves left, respectively. Notice that this is not a unique solution – there are several other valid sequences that also successfully clear this board.

Proof. We argue the first claim since the proof of the second is symmetric. For the forward implication, we show (using induction on m) that the following statement is true for all integers $m \geq 0$: For every configuration s such that $\sum_{\substack{1 \leq i \leq N-1: \\ s[i] \notin \{0, \square\}}} (s[i] - 1) = m$, if s is N -solvable,

then $s[N] \neq \square$ and $m \geq$ number of 0's in $s[1, \dots, N-1]$. For the base case, consider $m = 0$. The only configurations s with $\sum_{\substack{1 \leq i \leq N-1: \\ s[i] \notin \{0, \square\}}} (s[i] - 1) = 0$ are the ones for which $s[1, \dots, N-1]$ is a string over $\{0, 1, \square\}$. Among these, the only N -solvable configurations s are the ones for which $s[1, \dots, N-1]$ is a string over $\{1, \square\}$ and $s[N] \neq \square$. Thus, the statement is true for $m = 0$.

As induction hypothesis, assume that the statement is true for all integers $0 \leq m \leq p$, for some integer $p \geq 0$. Let's argue that the statement is true for $m = p + 1$. Let s be a configuration such that $\sum_{\substack{1 \leq i \leq N-1: \\ s[i] \notin \{0, \square\}}} (s[i] - 1) = p + 1$ and s is N -solvable. As s is N -solvable,

there exists a sequence of moves (say σ) that clears the corresponding $1 \times N$ board with the final rook at the cell $(1, N)$. Clearly, $s[N] \neq \square$. Let $t \geq 1$ be the least integer such that the capturing piece in t^{th} move of σ is not a 1-rook. Let \tilde{s} denote the configuration corresponding to the board obtained after t^{th} move of σ . Note that \tilde{s} is N -solvable and $\sum_{\substack{1 \leq i \leq N-1: \\ \tilde{s}[i] \notin \{0, \square\}}} (\tilde{s}[i] - 1) \leq p$. Using induction hypothesis, $p \geq$ number of 0's in $\tilde{s}[1, \dots, N-1]$.

Also, number of 0's in $\tilde{s}[1, \dots, N-1] \geq$ number of 0's in $s[1, \dots, N-1] - 1$; this is because the number of 0-rooks at the cells $(1, 1), \dots, (1, N-1)$ doesn't decrease in the first $t-1$ moves of σ , and decreases by at most 1 in the t^{th} move of σ . Therefore, we have $p + 1 \geq$ number of 0's in $s[1, \dots, N-1]$, as desired.

For the converse, we show (using induction on m) that the following statement is true for all integers $m \geq 0$: For every configuration s such that $s[N] \neq \square$, if $s[1, \dots, N-1]$ has exactly m 0's and $\sum_{\substack{1 \leq i \leq N-1: \\ s[i] \notin \{0, \square\}}} (s[i] - 1) \geq m$, then s is N -solvable.

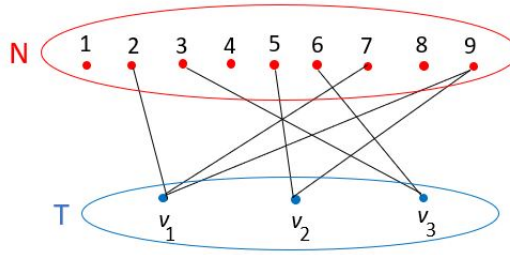
For the base case, consider $m = 0$. Let s be a configuration such that $s[N] \neq \square$ and $s[1, \dots, N-1]$ has no 0's. For each $1 \leq i < N$, the cell $(1, i)$ in the corresponding board is either empty or has a $1/2/\dots/d$ -rook. The board can be cleared with the final piece at $(1, N)$ by making the $1/2/\dots/d$ -rooks (if any) to capture rook at the cell $(1, N)$. Thus, s is N -solvable. So, the statement is true for $m = 0$.

As induction hypothesis, assume that the statement is true for all integers $0 \leq m \leq p$, for some integer $p \geq 0$. Let's argue that the statement is true for $m = p + 1$. Let s be a configuration such that $s[N] \neq \square$, $s[1, \dots, N-1]$ has exactly $(p + 1)$ 0's and $\sum_{\substack{1 \leq i \leq N-1: \\ s[i] \notin \{0, \square\}}} (s[i] - 1) \geq p + 1$.

While there is a 1-rook in the cells $(1, 1), \dots, (1, N-1)$ that can capture a 0-rook in the cells $(1, 1), \dots, (1, N-1)$, make such a move. Once no such move can be made, there exist integers $1 \leq u < v < N$ such that $s[u, \dots, v] = 0\square^\lambda x$ or $s[u, \dots, v] = x\square^\lambda 0$, for some $\lambda \geq 0$ and some $2 \leq x \leq d$. In the former (resp. latter) case, the x -rook at the cell $(1, v)$ (resp. $(1, u)$) can be made to capture the 0-rook at the cell $(1, u)$ (resp. $(1, v)$), and the configuration corresponding to the resulting board is N -solvable by induction hypothesis. \blacktriangleleft

We conclude that a configuration s of length N is solvable iff there exists $1 \leq \ell \leq N$ such that

- $s[\ell] \neq \square$,



■ **Figure 3** An instance of Red-Blue Dominating Set.

- $\sum_{\substack{1 \leq i \leq \ell-1: \\ s[i] \notin \{0, \square\}}} (s[i] - 1) \geq \text{number of } 0\text{'s in } s[1, \dots, \ell - 1], \text{ and}$
- $\sum_{\substack{\ell+1 \leq i \leq N: \\ s[i] \notin \{0, \square\}}} (s[i] - 1) \geq \text{number of } 0\text{'s in } s[\ell + 1, \dots, N].$

3.1.2 2-Dimensional boards

► **Theorem 2** (Intractability for rooks). *GENERALIZED SOLO CHESS $(\mathbb{N}, 2)$ is NP-complete.*

Proof. We reduce from the RED-BLUE DOMINATING SET problem. Let $J := \langle G = (N \cup T, E); k \rangle$ be an instance of RED-BLUE DOMINATING SET. Recall that G is a bipartite graph with bipartition N and T ; and J is a YES-instance if and only if there exists a subset $S \subseteq N$ of size at most k such that every vertex v in T has a neighbor in S . We let the vertices in N be denoted by $[n]$ and let $T := \{v_1, \dots, v_m\}$. We refer to the vertices of N and T as non-terminals and terminals, respectively.

We first describe the construction of the reduced instance of GENERALIZED SOLO CHESS $(\mathbb{N}, 2)$ based on J . The game takes place on a $(2m + 1) \times (n + m + k + 1)$ board. The initial position of the rooks is as follows:

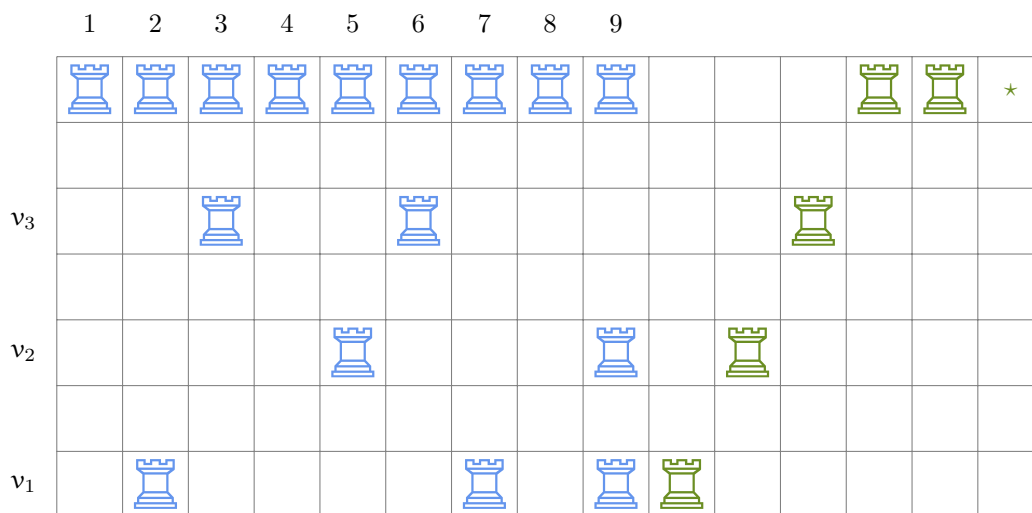
- *Non-terminal* rooks. For all $i \in [n]$, we place a 1-rook in the cell $(2m + 1, i)$.
- *Terminal* rooks. For all $j \in [m]$, we place a 1-rook in the cell $(2j - 1, \ell)$ for each ℓ such that $\ell \in N(v_j)$.
- *Collector* rooks. For all $j \in [m]$, we place a 2-rook in the cell $(2j - 1, n + j)$.
- *Cleaner* rooks. For all $\ell \in [k]$, we place a 2-rook in the cell $(2m + 1, n + m + \ell)$.
- *Target location*. Finally, we place on 1-rook at the location $(2m + 1, n + m + k + 1)$.

The non-terminal and terminal rooks correspond to the non-terminal and terminal vertices in the graph, and their relative positioning as described above captures the graph structure. The rooks on every row are expected to “clear to one of the columns corresponding to a vertex they are dominated by”, and the other auxiliary rooks added to the board above help with clearing the board after this phase, as explained further below.

Forward Direction

Assume that J is a YES-instance. That is, there exists $S \subseteq [n]$ of size at most k such that every vertex in T has a neighbour in S . For each $j \in [m]$, let $f(j)$ denote an arbitrary but fixed $i \in S$ such that $i \in N(v_j)$. Consider the following sequence of moves (c.f. Figure 5):

- For each $j \in [m]$ and each $\ell \in N(v_j) \setminus \{f(j)\}$, the terminal 1-rook at the cell $(2j - 1, \ell)$ captures rook at the cell $(2j - 1, f(j))$.



■ **Figure 4** The reduced instance of GENERALIZED SOLO CHESS played by rooks corresponding to the instance shown in Figure 3.

- For each $j \in [m]$, the collector 2-rook at the cell $(2j - 1, n + j)$ captures the rook at the cell $(2j - 1, f(j))$, and the 1-rook at the cell $(2j - 1, f(j))$ so obtained then captures rook at the cell $(2m + 1, f(j))$.
- For each $i \in [n]$, if there's a non-terminal 1-rook at the cell $(2m + 1, i)$, then it captures one of the 0-rooks at the cells $(2m + 1, f(1)), \dots, (2m + 1, f(m))$.

Now, the board is empty except for the top row which has one 1-rook at the target location, k cleaner 2-rooks and at most k 0-rooks, i.e., the 0-rooks at the cells $(2m + 1, f(1)), \dots, (2m + 1, f(m))$. Using Lemma 9, this corresponds to a $(n + m + k + 1)$ -solvable configuration.

Reverse Direction

Suppose the reduced instance is solvable. We first make some claims about any valid sequence of s moves, denoted by σ , that clears the board. Let $\rho_\sigma((x, y), \ell)$ denote the type of the piece at the location (x, y) after ℓ moves of σ have been played. If (x, y) is an empty location after ℓ moves of σ have been played, then we let $\rho_\sigma((x, y), \ell) = \square$.

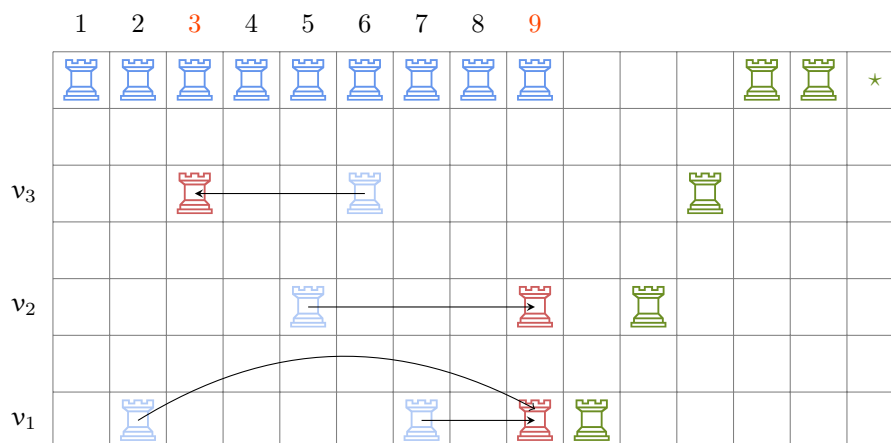
Let $\zeta(t)$ denote the set of locations $(2m + 1, \cdot)$ occupied by red rooks on the top row of the board after t moves of σ have been made, in other words: $\zeta(t) = \{i \mid \rho_\sigma((2m + 1, i), t) = \text{♖}\}$.

▷ **Claim 10.** $|\cup_{1 \leq t \leq s} \zeta(t)| \leq k$.

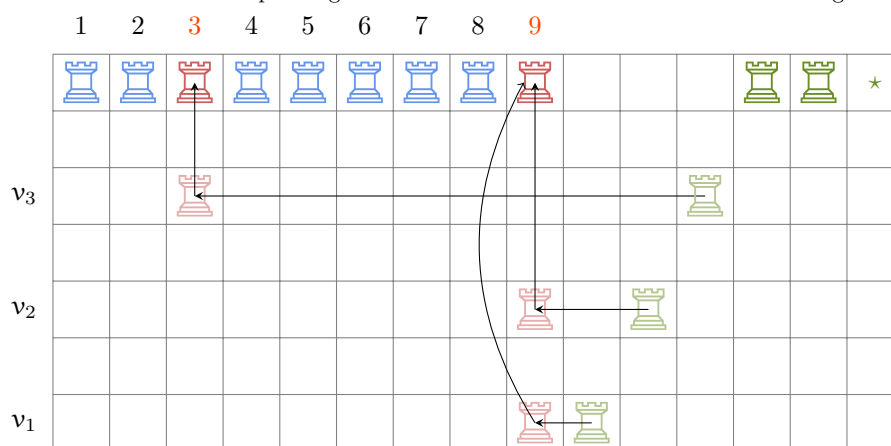
Proof. Let $i \in \cup_{1 \leq t \leq s} \zeta(t)$. That is, there exists $1 \leq t \leq s$ such that the cell $(2m + 1, i)$ has a 0-rook after t moves of σ . Let $p > t$ denote the first move of σ that empties the cell $(2m + 1, i)$. Note that the cell $(2m + 1, i)$ has a 1-rook before the p^{th} move of σ . So, there exists $t < q < p$ such that a 2-rook captures 0-rook at cell $(2m + 1, i)$ in q^{th} move of σ . Also, such a 2-rook is one among the k cleaner rooks. Thus, $|\cup_{1 \leq t \leq s} \zeta(t)| \leq k$. ◁

Let $j \in [m]$ and $i \in [n]$. We say that i is an j -*affected index* if there is some $t \in [s]$ such that the rook at position $(2j - 1, i)$ was captured by the green rook originally at position $(2j - 1, n + j)$ in the t^{th} move of σ .

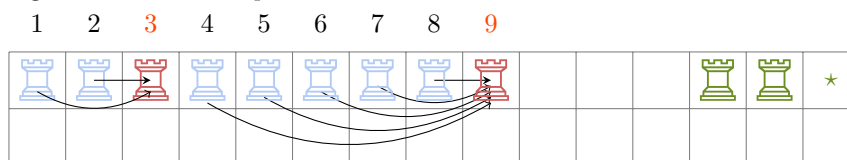
5:10 Chess Is Hard Even for a Single Player



(a) All blue rooks on rows corresponding to blue vertices clear row-wise to the dominating set vertices.



(b) The green rooks on the rows corresponding to blue vertices “pick up” the red rooks and capture along the column to get the rook on the top row.



(c) All blue rooks on the top row capture one of the red rooks leaving us in a solvable state with the two green rooks making the final captures.

■ **Figure 5** All illustration of the forward direction.

▷ **Claim 11.** For a fixed $j \in [m]$, there is exactly one $i \in [n]$ such that i is a j -affected index.

Proof. Let $j \in [m]$. Let $t \in [s]$ denote the first move of σ wherein the collector 2-rook (say g) at the cell $(2j - 1, n + j)$ either gets captured (Case 1) or captures (Case 2).

In Case 1, a terminal 1-rook in the row $2j - 1$ captures g in the t^{th} move of σ . After the t^{th} move of σ , the cell $(2j - 1, n + j)$ has a 0-rook. Let $p > t$ denote the first move of σ that empties the cell $(2j - 1, n + j)$. Note that the cell $(2j - 1, n + j)$ has a 1-rook before the p -th move of σ . So, there exists $t < q < p$ such that a 2-rook captures 0-rook at the cell $(2j - 1, n + j)$ in q^{th} move of σ . However, no such 2-rook exists on the board. Thus, Case 1 does not arise.

In Case 2, there exists $i \in [n]$ such that g captures the rook at the cell $(2j - 1, i)$ in the t^{th} move of σ . Note that i is the unique j -affected index. \triangleleft

We call $i \in [n]$ an *affected index* if there is some $t \in [s]$ such that the position $(2m + 1, i)$ was occupied by a red rook after t moves of σ .

\triangleright **Claim 12.** If $i \in [n]$ is a j -affected index for some $j \in [m]$, then i is also an affected index.

Proof. Assume that $i \in [n]$ is a j -affected index for some $j \in [m]$. That is, there exists $t \in [s]$ such that in the t^{th} move of σ , the collector 2-rook at the cell $(2j - 1, n + j)$ captures the rook at the cell $(2j - 1, i)$. After the t^{th} move of σ , the cell $(2j - 1, i)$ has a 1-rook. Let $t' > t$ denote the first move of σ wherein the 1-rook at the cell $(2j - 1, i)$ either gets captured (Case 1) or captures (Case 2).

In Case 1, a 1-rook captures the 1-rook at the cell $(2j - 1, i)$ in the t'^{th} move of σ . After the t'^{th} move of σ , the cell $(2j - 1, i)$ has a 0-rook. Let $p > t'$ denote the first move of σ that empties the cell $(2j - 1, i)$. Note that the cell $(2j - 1, i)$ has a 1-rook before the p^{th} move of σ . So, there exists $t' < q < p$ such that a 2-rook captures 0-rook at the cell $(2j - 1, i)$ in the q^{th} move of σ . However, no such 2-rook exists on the board. Thus, Case 1 does not arise.

In Case 2, in the t'^{th} move of σ , the 1-rook at the cell $(2j - 1, i)$ captures either rook at the cell $(2m + 1, i)$ (Subcase 1), or rook at the cell $(2j' - 1, i)$ for some $j' \in [m] \setminus \{j\}$ (Subcase 2).

In Subcase 1, the cell $(2m + 1, i)$ has a 0-rook after t' moves of σ . So, i is an affected index.

In Subcase 2, the cell $(2j' - 1, i)$ has a 0-rook after t' moves of σ . Let $p' > t'$ denote the first move of σ that empties the cell $(2j' - 1, i)$. Note that the cell $(2j' - 1, i)$ has a 1-rook before the p'^{th} move of σ . So, there exists $t' < q' < p'$ such that a 2-rook captures 0-rook at the cell $(2j' - 1, i)$ in the q'^{th} move of σ . Note that this 2-rook is the collector rook at the cell $(2j' - 1, n + j')$. After the q'^{th} move of σ , the cell $(2j' - 1, i)$ has a 1-rook. Let $t'' > q'$ denote the first move of σ wherein the 1-rook at the cell $(2j' - 1, i)$ either gets captured or captures. As before, it can be argued that in the t''^{th} move, the 1-rook at the cell $(2j' - 1, i)$ is not captured, and it either captures rook at the cell $(2m + 1, i)$ (in which case we are done), or rook at the cell $(2j'' - 1, i)$ for some $j'' \in [m] \setminus \{j, j'\}$ (in which case the collector 2-rook at the cell $(2j'' - 1, n + j'')$ captures the 0-rook at the cell $(2j'' - 1, i)$ in some subsequent move). Repeatedly using the same argument proves the claim. \triangleleft

Consider $S := \{\ell \mid \ell \in [n] \text{ and } \ell \text{ is an affected index}\}$. We claim that S is a dominating set in G . Indeed, consider any non-terminal vertex $v_j \in B$. If i is the unique j -affected index, then i is also an affected index, and therefore belongs to the dominating set. Note that $i \in N(v_j)$ by construction, therefore we are done. Also, by Claim 6, we have that $|S| \leq k$. This concludes the proof in the reverse direction. \blacktriangleleft

3.2 Queens

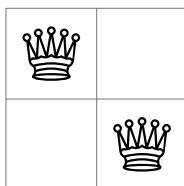
Recall the reduction described for the proof of Theorem 2. It is straightforward to check that if we introduce a large number – say $O(n^2)$ many – empty columns between every pair of consecutive columns of the original board, then we can also replace the rooks by queens and the reduction will remain valid. This is because the vast empty spaces essentially “nullify” the additional diagonal moves of the queens, thereby reducing their behavior to being equivalent to rooks. Also note that the operation of adding empty columns does not affect the forward direction: all pairs of mutually attacking locations remain mutually attacking even after this modification.

5:12 Chess Is Hard Even for a Single Player

We now present the following strengthening of this hardness result. Recall that in the previous reduction, we had pieces that were allowed to capture twice and others that were allowed to capture once. With queens, however, we can adapt the reduction so that every piece is allowed to capture twice, bringing this closer to the spirit of traditional solo chess:

► **Theorem 3** (Intractability for queens). *GENERALIZED SOLO CHESS ($\text{♚}, 2$) is NP-complete even when all queens are allowed to capture at most twice.*

We note that this result can be achieved by replacing every queen that is allowed to capture once with the following pair of queens that are both allowed to capture twice, with the queen on the bottom right replacing the “original” 1-queen:



■ **Figure 6** The reduced instance of GENERALIZED SOLO CHESS played by rooks corresponding to the instance shown in Figure 3.

We call the queen on the top-left corner the supporting queen, and refer to the other queen as its partner. Once all the 1-queens of the reduced instance are replaced in this way, we ensure that all supporting queens have the property that they do not attack any queen other than their partner. To achieve this, we shift them north-west along their diagonals appropriately if required. Note that the fact that the supporting queens attack only their partners forces that they are never captured by another piece, and that they capture their partner queen, which replaces the partner with a 1-queen, as desired. We omit the details here.

3.3 Pawns

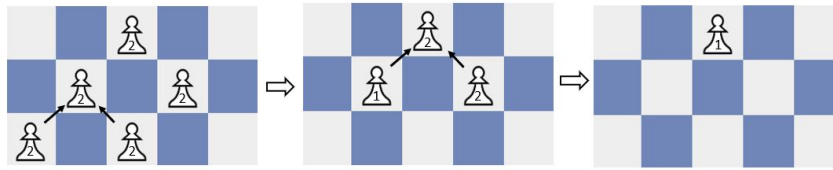
In contrast to the cases of Rooks, Queens and Bishops, we show that GENERALIZED SOLO CHESS($\text{♟}, 2$) can be decided by an algorithm whose running time is linear in the number of pawns when all pawns are allowed to capture at most twice in the initial configuration.

► **Theorem 4** (A characterization for pawns). *GENERALIZED SOLO CHESS ($\text{♟}, 2$) with n white pawns, each of which can capture at most twice, can be decided in $O(n)$ time.*

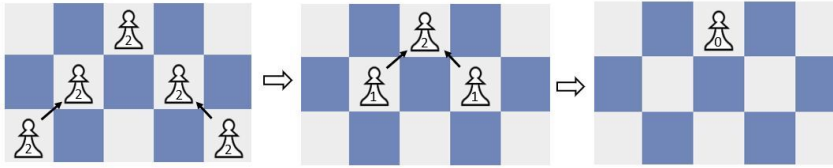
We denote by V the set of squares that initially contain a pawn, and by t the location of the target pawn. For $u, v \in V$, we say that u is a *parent* of v (and that v is a *child* of u) if u is diagonally one capture move away from v . We denote by $C(v)$ the children of v . Further if two vertices share a common parent, then we call them *siblings* of each other. We say that an initial configuration of pawns is *super-solvable* if the final capturing pawn has one move remaining after the final capture.

► **Definition 13.** *We say that a configuration of GENERALIZED SOLO CHESS($\text{♟}, 2$) with position set V is a skewed binary tree rooted at square v if the following are true:*

- (a) *All pawns are on squares of the same color.*
- (b) *All squares in $V \setminus \{v\}$ are below v .*
- (c) *Every square in $V \setminus \{v\}$ has a parent in V .*
- (d) *Every non-empty row below v contains exactly two squares of V with a common parent, except possibly the last (bottom-most) row which may contain one square of V .*



■ **Figure 7** The initial configuration in this example is a skewed binary tree. Note that it is super-solvable because the shown sequence of moves clears the board such that the final pawn has one move left - here, the 2-pawn at the cell (2, 4) does the final capture and becomes a 1-pawn.



■ **Figure 8** The initial configuration in this example is not a skewed binary tree. Note that it is not super-solvable because any sequence of moves that clears the board (one such sequence is shown) is such that the final pawn has no moves left.

The following result is the key to the characterization of solvable instances.

► **Lemma 14.** *An instance of GENERALIZED SOLO CHESS($\Delta, 2$) is super-solvable if and only if the initial configuration is a skewed binary tree.*

In particular, we can verify in linear time whether a given configuration of GENERALIZED SOLO CHESS($\Delta, 2$) is super-solvable, as each of the properties (a)-(d) can be checked in linear time.

We first observe that pawns can capture only in the forward direction (upward for W pawns) and only pawns on squares of the same color. Thus, we shall henceforth assume that all pawns are on squares of the same color and also that there is exactly one pawn whose initial square has the largest y co-ordinate; we shall call this the target pawn. If our assumption is false, we report the instance as a NO instance, and do not proceed further. We now describe the proof of Lemma 14.

Proof. We prove the claim by induction on $|V|$. If $|V| = 1$, the instance is trivially super-solvable and also satisfies the definition of a skewed binary tree. If $|V| = 2$, then the instance is super-solvable if and only if the unique vertex $v \in V \setminus \{t\}$ is a child of t , and this configuration is a skewed binary tree.

Thus, we suppose that $|V| \geq 3$. The necessity of conditions (a), (b), (c) has already been noted so that in the rest of this section we consider only configurations that satisfy (a), (b) and (c). We shall now establish the necessity of condition (d).

Firstly, we claim that t has two children. Suppose not, and let v be the only child of t . Then the last capture must be from v to t , and the last but one capture must be at v , so that the token at v has only one move remaining. When this token captures at t , it has zero moves left after the capture.

Thus, we can assume that t has two children u, v . Consider a valid super-solvable sequence σ ; let u be the vertex from which the final capture was made at t . Then the token at u must have had two moves left before this capture and therefore no capture in σ was ever made at u . Also, the last but one capture in σ must have been made from v to t . This implies

5:14 Chess Is Hard Even for a Single Player

that the sequence obtained from σ by excluding the final capture is a valid super-solvable sequence for $V \setminus \{t, u\}$. Since $V \setminus \{t, u\}$ is super-solvable, by the induction hypothesis, property (d) holds; i.e. there are exactly two squares in every row below v , except possibly for the bottom-most non-empty row. This shows that property (d) holds for V as well.

For the other direction, suppose that a given configuration with V as the set of squares is a skewed binary tree, and that $|V| \geq 3$. Then by definition t has two children u, v and it must be the case that one of u, v , say u has no child outside $C(v)$. Then $V \setminus \{u, v\}$ must induce a skewed binary tree; let σ be a super-solvable sequence for $V \setminus \{u, v\}$. Appending the captures $u \rightarrow t, v \rightarrow t$ yields a super-solvable sequence for the original configuration.

This completes the proof of Lemma 14. ◀

We now proceed to the proof of Theorem 4.

Proof. Let V be the initial position set and t be the target square.

Case 1: t has a single child x . In this case, we note that the instance is solvable if and only if the configuration restricted to $V \setminus \{t\}$ with x as target is super-solvable, which by Lemma 14 in linear time.

Case 2: t has two children x, y , and one of them, say y , has no child other than the common child of x, y . In this case, the instance is solvable if and only if the configuration restricted to $V \setminus \{t, y\}$ with target x is super-solvable, which we can verify in linear time.

Case 3: t has two children x, y , and $|C(x) \cup C(y)| = 3$; let $C(x) \cup C(y) = \{a, b, c\}$. Then the instance is solvable if and only if there's a re-labeling u, v, w of $\{a, b, c\}$ such that $C(u) \cup C(v) \subseteq C(w)$ and the configuration restricted to $V \setminus \{t, x, y, u, v\}$ with target w is super-solvable. This can again be verified in linear time.

This completes the proof of Theorem 4. ◀

4 Graph Capture Game

We introduce a game on graphs, which generalizes GENERALIZED SOLO CHESS(\mathcal{D}, d) when played on undirected graphs and GENERALIZED SOLO CHESS(\mathcal{D}, d) when played on directed graphs.

GRAPH CAPTURE(G, d):

Input: A graph $G = (V, E)$.

Output: Decide if there exists a sequence of token captures (along the edges of G) such that only a single token remains, with the constraint that each token may capture at most d times.

Our main result in this section is the following:

► **Theorem 5** (Intractability of the graph capture game). *GRAPH CAPTURE(2) is NP-complete on undirected graphs and DAGs even when every token can capture at most twice.*

In the rest of this section, we say that G is solvable if GRAPH CAPTURE($G, 2$) is a YES-instance.

4.1 Undirected Graphs

We prove Theorem 5 for undirected graphs.

A rooted tree is a pair (T, v) , with v denoting the root vertex; given a rooted tree (T, v) and a vertex w of T , we denote by $C(w)$ the children of w , and by $T(w)$ the subtree rooted at w .

► **Lemma 15.** *A graph $G = (V, E)$ is solvable if and only if G contains a vertex v and a spanning tree T such that every internal node of the rooted tree (T, v) has a leaf neighbor.*

We prove Lemma 15 in the Appendix [1] and now turn to a proof of part (a) in Theorem 5.

Proof. We proceed by a reduction from COLORFUL RED-BLUE DOMINATING SET. Let $\langle G = (R \uplus B, E); k \rangle$ be an instance of COLORFUL RED-BLUE DOMINATING SET with color classes $V_1 \cup \dots \cup V_k$. We assume, without loss of generality, that $|V_1| = \dots = |V_k| = n$ and let $V_j := \{v_1^{(j)}, \dots, v_n^{(j)}\}$. We begin by describing the construction of the reduced instance. We begin with the graph G and make the following additions:

1. For all $i \in [n]$ and $j \in [k]$, introduce a vertex $u_i^{(j)}$ and make it adjacent to $v_i^{(j)}$. We call these the *red partner vertices*.
2. For each $u_i^{(j)}$, introduce two neighbors $p_i^{(j)}$ and $q_i^{(j)}$, and finally, introduce two vertices $r_i^{(j)}$ and $s_i^{(j)}$ that are adjacent only to $p_i^{(j)}$ and $q_i^{(j)}$ respectively. In other words, each $u_i^{(j)}$ has two degree two neighbors, which in turn have a leaf neighbor each. Combined, we refer to the collection of vertices $S_j := \{u_i^{(j)}, p_i^{(j)}, q_i^{(j)}, r_i^{(j)}, s_i^{(j)} \mid i \in [n]\}$ as the *selection gadget* for V_j .
3. For all $j \in [k]$, introduce a vertex w_j and make it adjacent to $u_i^{(j)}$ for all $i \in [n]$. We call these vertices the *guards*.
4. We finally add a vertex \star that is adjacent to all the red partner vertices. We also add the vertices p and q and the edges (\star, p) and (p, q) .

We let H denote the graph thus constructed based on G and ask if H has a rooted spanning tree for which every internal node has a leaf neighbor. This completes a description of the construction. We briefly describe the intuition for the equivalence of the two instances. Because of the vertices selection gadgets, the partner vertices are forced to find their leaf neighbors in any spanning tree among the red vertices that they partner – except for at most one, which can use the guard vertex as the leaf neighbor. This leads to one red vertex being left “free” of being a leaf neighbor to a partner vertex in each color class, hence we have a selection of one blue vertex per color class. Since these are the only possible entry points for the blue vertices into the spanning tree, the vertices “chosen” by the selection gadget must correspond to a dominating set. We now formalize this intuition.

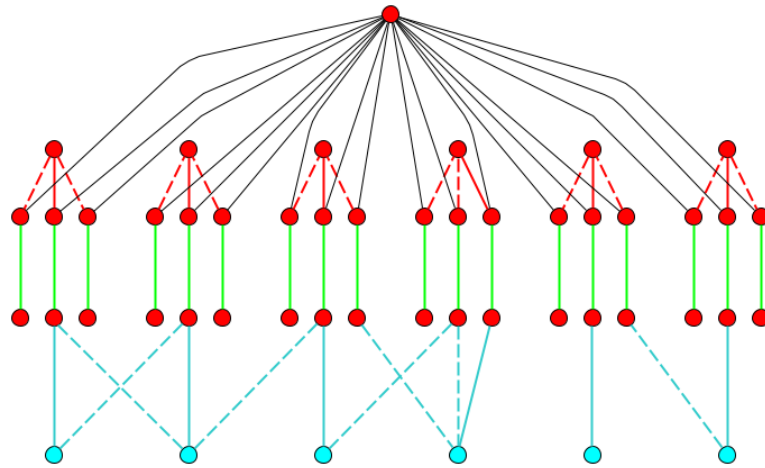
Forward Direction

Assume that $\langle G = (R \uplus B, E); k \rangle$ is a YES instance. That is, there exist $1 \leq j_1, \dots, j_k \leq n$ such that every vertex in B has a neighbour in $\{v_{j_1}^{(1)}, \dots, v_{j_k}^{(k)}\}$. For each $b \in B$, let $f(b)$ denote an arbitrary but fixed $\ell \in [k]$ such that $v_{j_\ell}^{(\ell)} \in N(b)$.

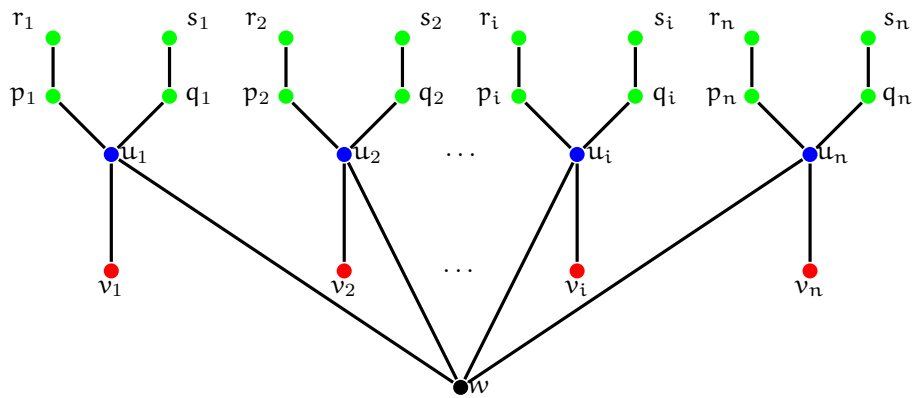
Let T denote the spanning tree of H with the following edge set:

- T contains all the edges of $H \left[\{\star, p, q\} \cup R \cup \bigcup_{1 \leq \ell \leq k} S_\ell \right]$
- For every $b \in B$, T contains the edge $\{v_{j_{f(b)}}^{(f(b))}, b\}$
- For every $1 \leq \ell \leq k$, T contains the edge $\{u_{j_\ell}^{(\ell)}, w_\ell\}$

Note that every internal node of the rooted tree (T, \star) has a leaf neighbour.



■ **Figure 9** An illustration of the reduction from Red-Blue dominating set. The solid lines belong to the spanning tree. The “spikes” from the selection gadget are omitted for clarity.



■ **Figure 10** A schematic showing the selection gadget for one color class.

Reverse Direction

Assume that there exist $r \in V(H)$ and a spanning tree (say T) of H such that every internal node of the rooted tree (T, r) has a leaf neighbour. In T , \star is adjacent to p and at least one red partner vertex. So, \star has at least two neighbours in T . Thus, \star is not a leaf node of (T, r) .

The only neighbours of \star in T are p and some vertices of $\{u_i^{(\ell)} \mid 1 \leq \ell \leq k, 1 \leq i \leq n\}$. Note that p is not a leaf node of (T, r) as p has two neighbours, i.e., \star and q , in T . Also, for every $1 \leq \ell \leq k$ and every $1 \leq i \leq n$, $u_i^{(\ell)}$ is not a leaf node of (T, r) because $u_i^{(\ell)}$ has at least two neighbours, i.e., $p_i^{(\ell)}$ and $q_i^{(\ell)}$, in T . Therefore, \star has no leaf neighbours in (T, r) . So, \star is not an internal node of (T, r) . Hence, we have $r = \star$.

Let $1 \leq \ell \leq k$. The only neighbours of w_ℓ in T are some vertices of $\{u_i^{(\ell)} \mid 1 \leq i \leq n\}$. As argued earlier, no red partner vertex is a leaf node of (T, r) . So, w_ℓ has no leaf neighbours in (T, r) . Thus, w_ℓ is not an internal node of (T, r) . That is, w_ℓ is a leaf node of (T, r) . Hence, there exists a unique integer (say $g(\ell)$) in $[n]$ such that $u_{g(\ell)}^{(\ell)}$ is the neighbour of w_ℓ in T .

Now, it suffices to show that every vertex in B has a neighbour in $\{v_{g(1)}^{(1)}, \dots, v_{g(k)}^{(k)}\}$. Let $b \in B$. There exist $1 \leq \ell \leq k$ and $1 \leq i \leq n$ such that $v_i^{(\ell)} \in N_T(b)$. As shown above, $u_i^{(\ell)}$ is not a leaf node of (T, r) . That is, $u_i^{(\ell)}$ is an internal node of (T, r) . So, $u_i^{(\ell)}$ has a leaf neighbour (say z) in (T, r) . No vertex of $V(T) \setminus \{p_i^{(\ell)}, q_i^{(\ell)}, \star, w_\ell, v_i^{(\ell)}\}$ is adjacent to $u_i^{(\ell)}$ in T . Note that

- $p_i^{(\ell)}$ is not a leaf node of (T, r) as $p_i^{(\ell)}$ has at least two neighbours, i.e., $u_i^{(\ell)}$ and $r_i^{(\ell)}$, in T .
- $q_i^{(\ell)}$ is not a leaf node of (T, r) as $q_i^{(\ell)}$ has at least two neighbours, i.e., $u_i^{(\ell)}$ and $s_i^{(\ell)}$, in T .
- If $v_i^{(\ell)}$ is a neighbour of $u_i^{(\ell)}$ in T , then $v_i^{(\ell)}$ is not a leaf node of (T, r) because in such a case, $v_i^{(\ell)}$ has at least two neighbours, i.e., b and $u_i^{(\ell)}$, in T .

Therefore, we have $z = w_\ell$ and hence, $i = g(\ell)$.

This completes the argument for equivalence. ◀

4.2 Directed Acyclic Graphs

We now prove Theorem 5 for DAGs.

Proof. Let φ be a given instance of 3-SAT, with clauses C_1, C_2, \dots, C_m over variables x_1, x_2, \dots, x_n .

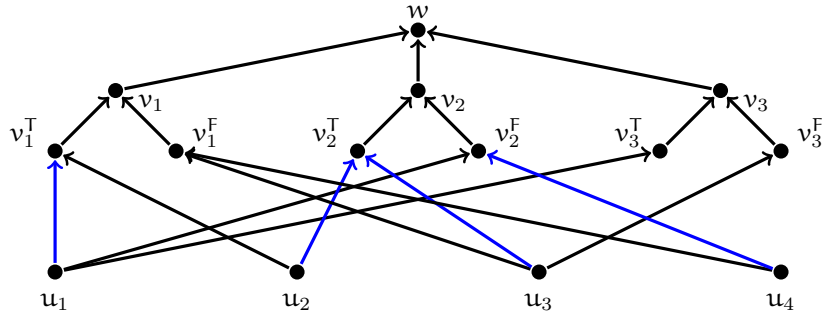
We construct the following directed graph $G = (W, E)$, where $W = U \cup V \cup \{w\}$; $U = \{u_1, u_2, \dots, u_m\}$ and $V = \{v_1, v_2, \dots, v_n\} \cup \{v_1^T, v_2^T, \dots, v_n^T\} \cup \{v_1^F, v_2^F, \dots, v_n^F\}$. Intuitively, each vertex in U represents a clause and vertices v_i^T, v_i^F correspond to an assignment of T, F respectively to x_i .

The edge set is $E = E_1 \cup E_2$, where

$$E_1 = \{(u_i, v_j^T) \mid x_j \in C_i, 1 \leq i \leq m, 1 \leq j \leq n\} \\ \cup \\ \{(u_i, v_j^F) \mid \neg x_j \in C_i, 1 \leq i \leq m, 1 \leq j \leq n\}$$

and

$$E_2 = \{(v_i^T, v_i) \mid 1 \leq i \leq n\} \cup \{(v_i^F, v_i) \mid 1 \leq i \leq n\} \cup \{(v_i, w) \mid 1 \leq i \leq n\}.$$



■ **Figure 11** The DAG corresponding to the set of clauses $C_1 = \{x_1, \neg x_2, x_3\}$, $C_2 = \{x_2, x_3\}$, $C_3 = \{\neg x_1, x_2, \neg x_3\}$, $C_4 = \{\neg x_2, \neg x_3\}$. The blue edges indicate the captures in Phase 1 for the satisfying assignment $x_1 = T$, $x_2 = T$, $x_3 = F$.

The graph G can clearly be computed in time polynomial in the input size (number of variables and clauses).

It thus suffices to show that φ is satisfiable iff G is solvable.

First, suppose that φ is satisfiable and let A be a satisfying assignment for φ . Consider the following sequence of captures:

Phase 1: For each $i \in \{1, 2, \dots, m\}$, let j be the least index such that C_i is satisfied by x_j or $\neg x_j$ in A . Then the token at u_i captures the token at v_j^T (if $x_j = T$) or the token at v_j^F (if $x_j = F$).

Phase 2: Now, for each $i \in \{1, 2, \dots, n\}$: if $x_i = T$, then the token at v_i^T captures the token at v_i , and then the token at v_i^F captures the token at v_i ; otherwise $v_i = F$ and the token at v_i^F captures the token at x_i , and then the token at v_i^T captures the token at v_i ;

Phase 3: For each $i \in \{1, 2, \dots, n\}$, the token at v_i captures the token at w .

We show the validity of this capture sequence by considering each phase.

At the end of Phase 1, there are no tokens remaining at any of the u_i s, and further for each $i \in \{1, 2, \dots, n\}$, exactly one of the tokens among $\{v_i^T, v_i^F\}$ has 1 move remaining, and the other token has 2 moves remaining.

In Phase 2, the token among $\{v_i^T, v_i^F\}$ with 2 moves remaining is the last to capture at v_i ; thus at the end of Phase 2, there is one token at each v_i with one move remaining and further one more token at w ; there are no other tokens.

In Phase 3, it is thus feasible for each token at v_i to successively capture at w and finally there is exactly one token remaining - at the vertex w .

Now, we suppose that G is solvable; let σ be a valid sequence of captures. We claim that for each $i \in \{1, 2, \dots, n\}$, captures were not made at both v_i^T and v_i^F in σ . For contradiction, suppose that captures were made both at v_i^T and at v_i^F ; let the last of these captures be at move t_1 of σ . Since all the tokens in $\{v_i^T, v_i^F\}$ must make their last capture at v_i , there must be a capture from $\{v_i^T, v_i^F\}$ to z that appears in σ later than t_1 ; let the last such capture happen at move $t_2 > t_1$. After move t_2 , there are no tokens in $\{v_i^T, v_i^F\}$ and the token at v_i has zero moves remaining; this implies that the token at v_i cannot be cleared, which is the desired contradiction.

Now, let I be the set of indices i such that a capture was made at v_i^T . Consider the assignment: for each $i \in \{1, 2, \dots, n\}$, we set $x_i = T$ if $i \in I$ and $x_i = F$ otherwise. We claim that every clause is satisfied by this assignment. Let C_i be an arbitrary clause; if the token at u_i made a capture at some v_j^T , then C_i contains the literal x_j , and $j \in I$ so that $x_j = T$

and C_i is satisfied. If the token at u_i made a capture at some v_j^F , then C_i contains the literal $\neg x_j$. Also, by the claim in the previous paragraph, no capture was made at v_j^T , therefore $j \notin I$ and $x_j = F$, so that C_i is satisfied. ◀

5 Concluding Remarks

We introduced GENERALIZED SOLO CHESS based on the Solo Chess game that is played on a 8×8 board. We focused mostly on scenarios that involve only pieces of one type, and showed that determining if a given instance is solvable is intractable when playing with rooks, bishops, and queens; while it is tractable for pawns. While we leave the case of knights open, we do show that a natural generalization of GENERALIZED SOLO CHESS restricted to knights, GRAPH CAPTURE, is hard even on DAGs and general undirected graphs. We also show that solvable instances of GENERALIZED SOLO CHESS played by rooks only admits an efficient characterization when the game is restricted to one-dimensional boards.

Our work leaves open a few concrete open problems, which we enlist below:

1. What is the complexity of GENERALIZED SOLO CHESS played by rooks only, for the special case when all rooks are allowed to capture at most twice initially? Notice that if we replace rooks by queens in this question, we show NP-completeness (Theorem 3).
2. What is the complexity of GENERALIZED SOLO CHESS played by pawns only, when the pawns are allowed at most a designated number of captures? Recall that if all pawns can capture at most twice, we have an efficient characterization (Theorem 4).
3. What is the complexity of GENERALIZED SOLO CHESS played by knights only?

There are also several broad directions for future work, and we suggest some that we think are both natural and interesting problems to consider:

1. For NO-instances of GENERALIZED SOLO CHESS, a natural optimization objective is to play as many moves as possible, or, equivalently, leave as few pieces as possible on the board. It would also be interesting to find the smallest d for which a board can be cleared if every piece was allowed to capture at most d times. These are natural optimization versions that we did not explicitly consider but we believe would be interesting to explore.
2. Does GENERALIZED SOLO CHESS become easier if the number of pieces in every row or column is bounded? Note that the reduction in Theorem 2 can be used to show that GENERALIZED SOLO CHESS(\mathbb{N} , 2) is NP-complete even when the number of rooks per column is a constant, if we initiate the reduction from an instance of Red-Blue Dominating Set where every red vertex has constant degree.
3. What is the complexity of GENERALIZED SOLO CHESS when played on boards of dimension $M \times N$, where one of M or N is a constant? It is not hard to generalize Theorem 1 to $2 \times N$ boards, however, a general result – say parameterized by one of the dimensions – remains open.
4. Variants of GENERALIZED SOLO CHESS where the pieces are limited not by the number of captures but the total distance moved on the board, or the distance moved per step, are also interesting to consider. Note that if the distance moved per step is lower bounded, then this forbids “nearby captures”, while if it is upper bounded, then “faraway captures” are disallowed.
5. It would also be interesting to restrict the number of capturing pieces instead of the number of captures per piece. For example, it seems intuitive to posit that if we are permitted only one capturing piece, then it must trace a “Hamiltonian path” of sorts among the pieces on the board.

6. Finally, we did not explicitly study GENERALIZED SOLO CHESS with multiple piece types on the board – although most such variants would be hard given the complexity results established already for pieces of one kind, it would be interesting to investigate special cases and exact algorithms in this setting.

References

- 1 N. R. Aravind, Neeldhara Misra, and Harshil Mittal. Chess is hard even for a single player, 2022. doi:10.48550/ARXIV.2203.14864.
- 2 Elwyn R Berlekamp, John H Conway, and Richard K Guy. *Winning ways for your mathematical plays*. A K Peters, Natick, MA, 2 edition, January 2001.
- 3 Josh Brunner, Erik D Demaine, Dylan Hendrickson, and Julian Wellman. Complexity of Retrograde and Helpmate Chess Problems: Even Cooperative Chess is Hard. *arXiv preprint arXiv:2010.09271*, 2020.
- 4 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*, volume 5. Springer, 2015.
- 5 Aviezri S Fraenkel and David Lichtenstein. Computing a Perfect Strategy for $n \times n$ Chess Requires Time Exponential in n . *Journal of Combinatorial Theory, Series A*, 31(2):199–214, 1981.
- 6 Michael R Garey and David S Johnson. *Computers and Intractability*, volume 174. freeman San Francisco, 1979.
- 7 Douglas Brent West et al. *Introduction to Graph Theory*, volume 2. Prentice hall Upper Saddle River, 2001.

Rolling Polyhedra on Tessellations

Akira Baes  

Département d'Informatique, Université libre de Bruxelles, Belgium

Erik D. Demaine  

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

Martin L. Demaine  

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

Elizabeth Hartung  

Department of Mathematics, Massachusetts College of Liberal Arts, North Adams, MA, USA

Stefan Langerman  

Département d'Informatique, Université libre de Bruxelles, Belgium

Joseph O'Rourke  

Department of Computer Science, Smith College, Northampton, MA, USA

Ryuhei Uehara  

School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa, Japan

Yushi Uno 

Graduate School of Informatics, Osaka Metropolitan University, Japan

Aaron Williams  

Department of Computer Science, Williams College, Williamstown, MA, USA

Abstract

We study the space reachable by *rolling* a 3D convex polyhedron on a 2D periodic tessellation in the xy -plane, where at every step a face of the polyhedron must coincide exactly with a tile of the tessellation it rests upon, and the polyhedron rotates around one of the incident edges of that face until the neighboring face hits the xy plane. If the whole plane can be reached by a sequence of such rolls, we call the polyhedron a *plane roller* for the given tessellation. We further classify polyhedra that reach a constant fraction of the plane, an infinite area but vanishing fraction of the plane, or a bounded area as *hollow-plane rollers*, *band rollers*, and *bounded rollers* respectively. We present a polynomial-time algorithm to determine the set of tiles in a given periodic tessellation reachable by a given polyhedron from a given starting position, which in particular determines the roller type of the polyhedron and tessellation. Using this algorithm, we compute the reachability for every regular-faced convex polyhedron on every regular-tiled (≤ 4)-uniform tessellation.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Design and analysis of algorithms; Mathematics of computing \rightarrow Discrete mathematics

Keywords and phrases polyhedra, tilings

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.6

Supplementary Material *Software*: <https://github.com/akirbaes/RollingPolyhedron/>
archived at `swh:1:dir:94cacdc90902f3129be316e189a4d64f94ad4537`

Funding *Stefan Langerman*: Directeur de Recherches du F.R.S.-FNRS.

Ryuhei Uehara: JSPS KAKENHI Grant Numbers JP18H04091, JP20H05961, JP20H05964, JP20K11673

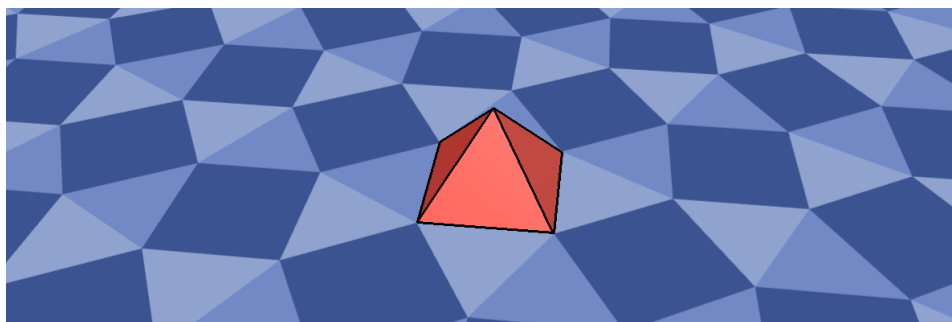


© Akira Baes, Erik D. Demaine, Martin L. Demaine, Elizabeth Hartung, Stefan Langerman, Joseph O'Rourke, Ryuhei Uehara, Yushi Uno, and Aaron Williams;
licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).
Editors: Pierre Fraigniaud and Yushi Uno; Article No. 6; pp. 6:1–6:16



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



■ **Figure 1** Screenshot from an interactive 3D rolling visualization program on the subject of this paper [3].

Acknowledgements Part of this work appeared in the first author’s Master’s Thesis. Part of this work was done at the 1st and 2nd Virtual Workshops on Computational Geometry (2020 and 2021). The authors would like to thank all participants of those workshops. Renders of prisms and antiprisms are by Robert Webb’s Stella software. Other polyhedron renders are from Wikimedia Commons under Creative Commons Attribution license.

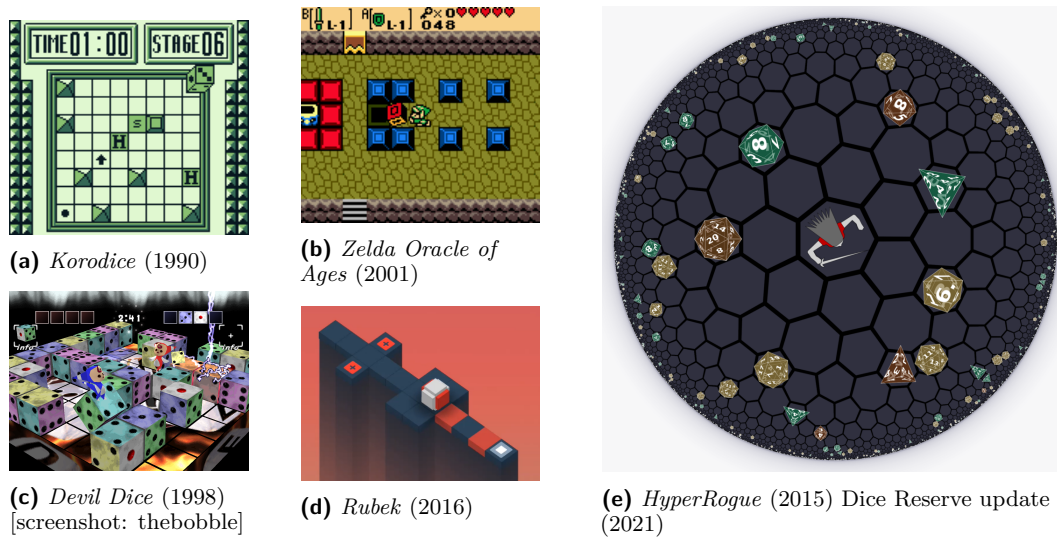
1 Introduction

Dice rolling puzzles feature a cube rolling around on the square grid. The goal is often to match a given face with a given tile. Such puzzles were popularized by Martin Gardner [11, 12, 13], and are featured in a variety of computer games, such as *Korodice* (Gameboy, 1990), *Super Mario 64* (Nintendo 64, 1996), *Devil Dice* (Playstation, 1998), *Legacy of Kain: Soul Reaver* (Playstation, 1999), *Legend of Zelda Oracle of Ages* (Gameboy Color, 2001), *Bombastic* (Playstation 2, 2002), *Legend of Zelda Spirit Tracks* (Nintendo DS, 2009), *Rubek* (Windows, 2016), *Roll The Box* (Mobile, 2021), and *The Last Cube* (Windows, 2022); see Figure 2. Cube rolling puzzles have been occasionally generalized to rolling other polyhedra on other grids. For example, computer game *HyperRogue* (Windows, 2015) involves hexagonal and heptagonal tiles in a hyperbolic space, and in its 2021 update, rolling tetrahedron, octahedron, or icosahedron dice on a triangular lattice; see Figure 2e. With various constraints, rolling puzzles can be NP-complete [6, 18], and when rolling multiple shapes, they can be PSPACE-complete [5, 16].

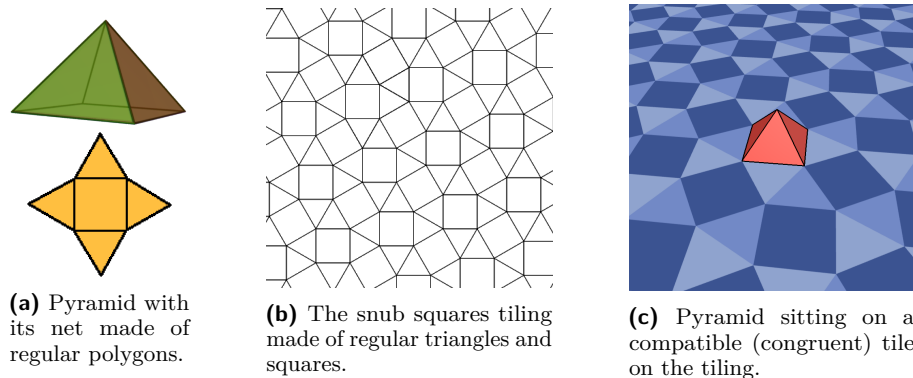
Previous work has explored rolling a polyhedron to reach any position and orientation in the plane [8, 4]. Akiyama [1] defined a *frame-stamper* as a regular polyhedron that covers the whole plane with a tiling by rolling the polyhedron in arbitrary directions, and a *tile-maker* as a polyhedron whose unfoldings all tile the plane. A more relaxed definition in [2] determines all *tessellation polyhedra* – regular-faced convex polyhedra that have at least one unfolding that tiles the plane.

1.1 Rolling Rollers

We formalize the concept of rolling any convex 3D polyhedron P on any tessellation T , which we imagine as lying in the (horizontal) xy -plane; refer to Figure 3. Recall that a plane tessellation is a partition of the plane into a collection T of polygons called *tiles* [15]. We restrict our attention to *edge-to-edge* tilings where two touching tiles share either a whole polygon edge or a vertex. When a tile of T is congruent to a face of P , we call them *compatible*.



■ **Figure 2** Cube and dice-rolling puzzles in video games.



■ **Figure 3** A polyhedron and a tessellation with compatible faces are required for rolling.

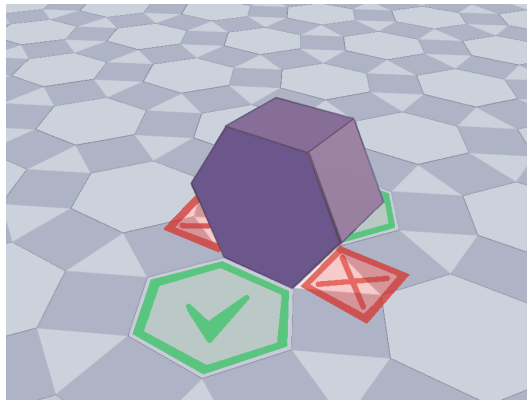
To start, we place the polyhedron P on the tessellation so that one of its faces *rests* on (i.e., coincides exactly with) a compatible tile. In a *rolling step*, we rotate the polyhedron about one of the edges of its resting face, until another face rests on the tessellation. For the roll to be *valid*, we insist that, at the end of the motion, the adjacent face of P across the rolling edge rests on another (adjacent) compatible tile. See Figure 4 for an example.

Valid sequences of rolls form paths in the *rolling graph* of possible configurations; see Section 2.2 for a formal definition. If the rolling graph contains a connected component that includes every tile of T , then we call the polyhedron a *plane roller* (denoted by the 🎲 icon) for that tessellation and starting position, as it can eventually roll to cover the entire plane. Other possibilities are 🎲 *hollow-plane rollers*, which cover a constant fraction of the plane while leaving holes; 🎲 *band rollers*, which cover an infinite area that is a vanishing fraction of the plane; and 🎲 *bounded rollers*, which are confined to a finite area.

1.2 Our Results

In this paper, we develop a polynomial-time algorithm to identify whether a polyhedron is a plane roller, hollow-plane roller, band roller, or bounded roller for a given plane tessellation and starting location, provided the tessellation is *periodic* meaning that its tiles have two linearly

6:4 Rolling Polyhedra on Tessellations



■ **Figure 4** Valid and invalid rolls, marked by green checks and red Xs respectively.



independent translational symmetries. The running time of our algorithm is polynomial in the number of faces of the polyhedron and the number of tiles in the fundamental domain of the two translational symmetries. We essentially take advantage of the periodicity of the tessellation, coupled with the structure of the polyhedron, to prove that the resulting rolling graph also has a periodic structure that we can exploit.

We then apply this algorithm to completely categorize a natural finite set of interesting



■ **Figure 5** Screenshot of the rolling-pair reachable-area classification interactive table available at <https://akirabaes.com/polyrollly/resulttable/>.

special cases, compiled on the website <https://akirabaes.com/polyrolly/resulttable/> shown in Figure 5. For polyhedra, we consider the *regular-faced* convex polyhedra where every face is a regular polygon: the 5 Platonic solids [9], the 13 Archimedean solids [10], the 92 Johnson solids and their chiral variations [14, 17, 19], the n -prisms for $n \in \{3, 5, 6, 8, 10, 12\}$, and the n -antiprisms for $n \in \{4, 5, 6, 8, 10, 12\}$, as higher-sided polygons cannot be used to tile the plane [15]. For periodic plane tessellations, we consider all “ k -uniform” tilings for $k \geq 4$, as listed in [7]. A plane tessellation is *k-uniform* if its tiles are regular polygons and it is *k-isogonal*, meaning that there are k equivalence classes of vertices (called *orbits*) formed by applying all transformations in the symmetry group to the vertices. All k -uniform tilings are periodic [15].

Including chiral variations of polyhedra that have one, these cases consist of 129 polyhedra and 131 tilings. For each case, we tried all possible starting positions to find the largest connected reachable area, thereby characterizing every pair of polyhedron and tiling as  plane roller,  hollow-plane roller, band roller, or bounded roller. See Figures 6, 7, 8, and 9 for examples of each respective type, and Tables 2, 3, and 4 in Appendix A for a condensed view of all results.

The figures and tables use standard notation for k -uniform tilings based on vertex types [7]. The type of a regular-polygon tile is the number of its sides, and the type of a vertex is the clockwise cyclic order of tile types that surround a vertex. For a k -uniform tiling, there are finitely many vertex types, so the tiling can be labeled by the list of vertex types, with duplicate names differentiated by a subscript. See Figure 10.

The rest of this paper is organized as follows. Section 2 describes our algorithm. Section 3 shows how the results from this algorithm can also assist puzzle designers. Section 4 describes our implementation.

2 The Algorithm

2.1 Tilings

First we review some basics about tilings, following Grünbaum and Shephard [15].

There are uncountably infinitely many tilings, even when restricted to edge-to-edge tilings with regular polygons. For example, the tiling in Figure 6b can be modified to follow any binary sequence of triangle and square rows, and there are uncountably many such binary sequences. We restrict our attention to *periodic tilings* T , which have two linearly independent translational symmetries (say, \vec{a} and \vec{b}) that act on the tiles of T . What this means is that applying the translation vector \vec{a} (respectively \vec{b}) on any tile $t \in T$ produces another tile of T . The symmetry group generated by \vec{a} and \vec{b} decomposes the set of tiles of T into equivalence classes, also called *orbits*, where two tiles are in the same class if there is a symmetry in the group (an integer linear combination $i\vec{a} + j\vec{b}$ for some $i, j \in \mathbb{Z}$) that matches one to the other.

The tiling can then be described by a *fundamental domain* for the action of these symmetries. Figure 11 shows an example. The fundamental domain is a connected subset of the tiles (one tile for each orbit), which glued together form a *supertile* S . We denote by $|S|$ the number of tiles in the supertile. The supertile (and the tiles that compose it) can be repeated by the action of the two translations to obtain the original tiling. As S tiles the plane isohedrally by translation, its boundary can be decomposed into six pieces, denoted by $A, B, C, \bar{A}, \bar{B}, \bar{C}$, counterclockwise, where \bar{A}, \bar{B} , and \bar{C} are translations of A by the action of \vec{a} , B by the action of \vec{b} , and C by the action of $\vec{b} - \vec{a}$, respectively. See Figure 16 (right).

6:6 Rolling Polyhedra on Tessellations

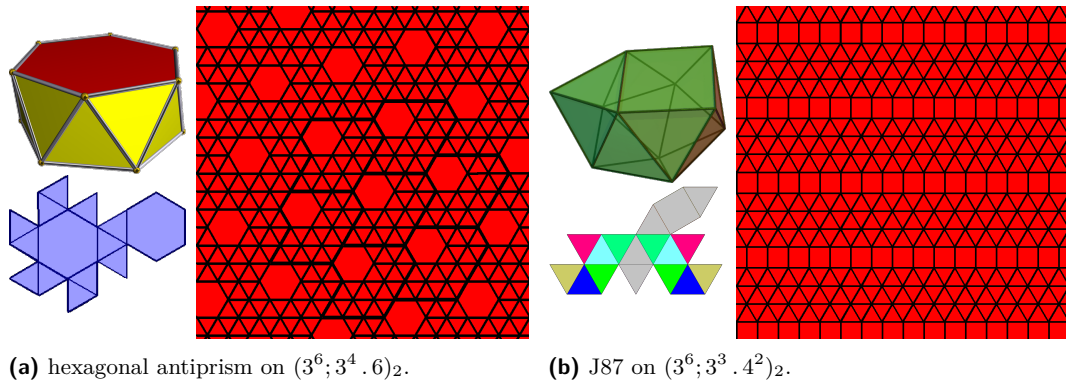



Figure 6 Examples of reachable-area patterns generated by  plane rollers which can reach the entire plane.

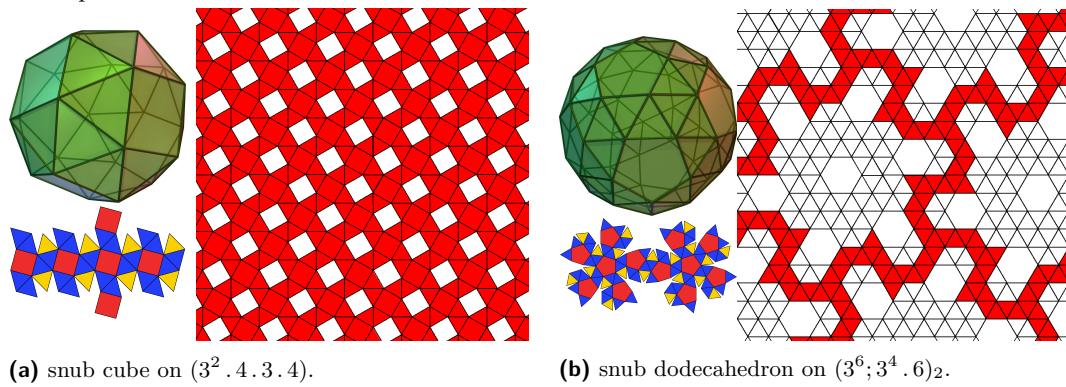



Figure 7 Examples of reachable area patterns generated by  hollow-plane rollers which reach a constant fraction of the plane while leaving holes.

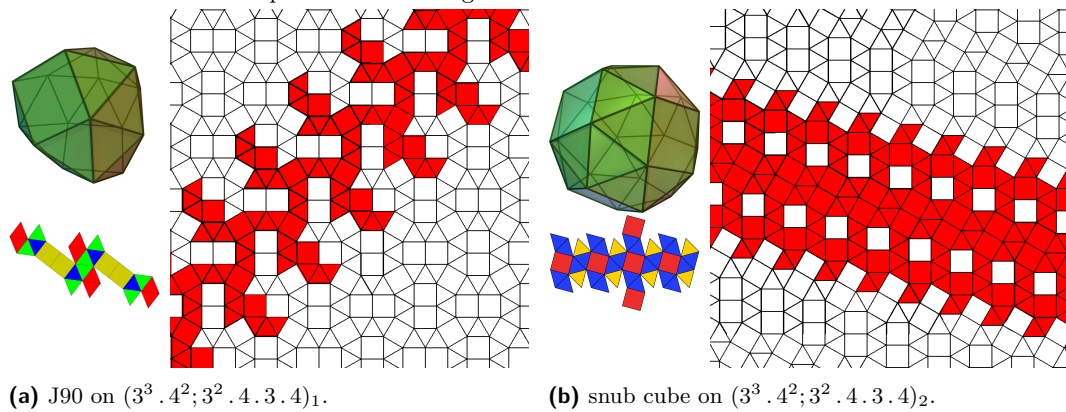



Figure 8 Examples of reachable areas patterns generated by  band rollers which reach an infinite area but a vanishing fraction of the plane, being restricted to an infinite band.

A copy of the supertile can be identified by its integer coordinates in the *basis* formed by the translation vectors \vec{a} and \vec{b} . That is, the copy (i, j) corresponds to the application of the translation $i\vec{a} + j\vec{b}$ to S . An individual tile t of the tiling T can then be uniquely identified by $\langle (i, j), s \rangle$: the coordinates (i, j) of the copy of S it is located in and its representative tile s within S . See Figure 12(a),(c)

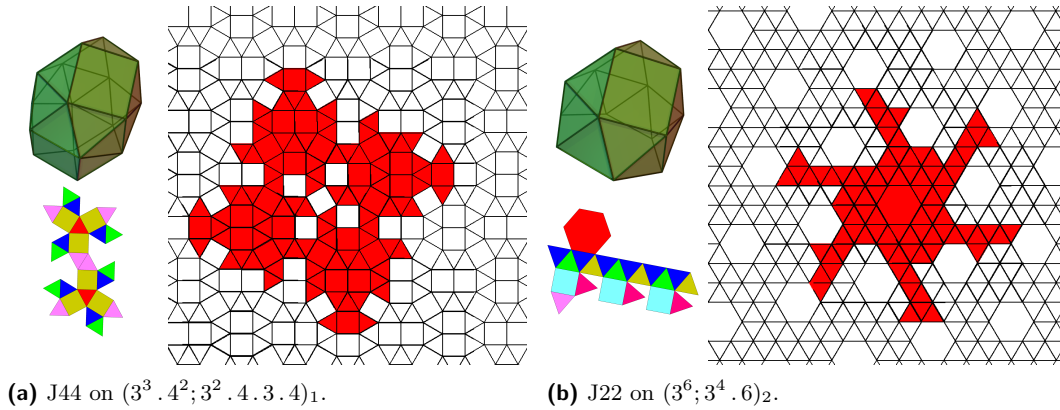


Figure 9 Examples of reachable area patterns generated by \blacktriangleright bounded rollers which are restricted to a finite area containing the start.

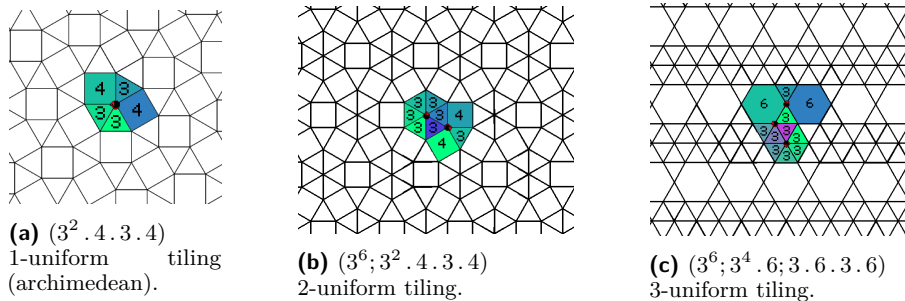


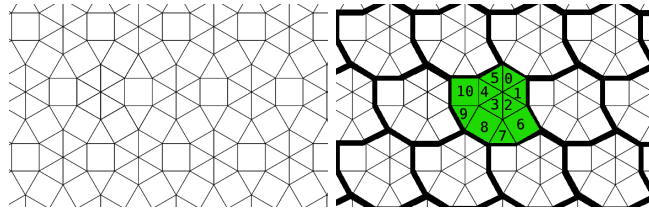
Figure 10 Examples of the naming convention of uniform tilings in the standardized “isogonal vertex type” notation, each point belonging to an orbit describing vertex types around it.

A tiling T can also be represented by its (infinite) dual graph G_T ,¹ where each tile is a vertex of G_T , and two vertices are connected by an edge if the two corresponding tiles are adjacent. When T is a periodic tiling, it is represented by the dual multigraph G_S of its supertile S . For tiles touching the boundary of S , we connect them to the tiles to which they are adjacent in the other copy or copies of the supertile, and mark the dual edges by $A, B, C, \bar{A}, \bar{B}$, or \bar{C} depending on the portion of the boundary they cross, see Figure 12(b). The graph G_S is in fact the quotient of G_T by the action of the symmetries \vec{a} and \vec{b} (also denoted $G_T/\{\vec{a}, \vec{b}\}$). The graph G_S can be used to navigate the tiling T or the graph G_T by updating the representation $\langle (i, j), s \rangle$ when moving to an adjacent tile. The tile s is updated to the adjacent tile s' in G_S , and the coordinates (i, j) need to be updated when crossing a boundary of the supertile S , using the edge marks.

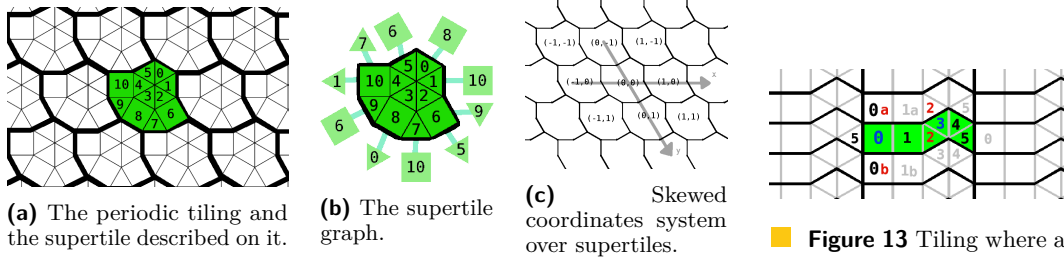
2.2 Rolling Graphs

Let P be a convex polyhedron in \mathbb{R}^3 . We denote by $|P|$ the number of faces of P . The face structure of P can be represented by its dual graph G_P where each face of P is a vertex in G_P and two vertices are connected by an edge if the two corresponding faces of P share an edge (Figure 14).

¹ This can be a multigraph, with parallel edges when two tiles are adjacent on more than one edge; see Figure 13.



■ **Figure 11** The same tiling as Figure 10b ($3^6; 3^2 \cdot 4 \cdot 3 \cdot 4$) in its supertile tiling representation.



(a) The periodic tiling and the supertile described on it.

(b) The supertile graph.

(c) Skewed coordinates system over supertiles.

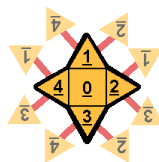
■ **Figure 13** Tiling where a multigraph is necessary; see tiles 3 and 2.

■ **Figure 12** Infinite tiling to supertile multigraph.

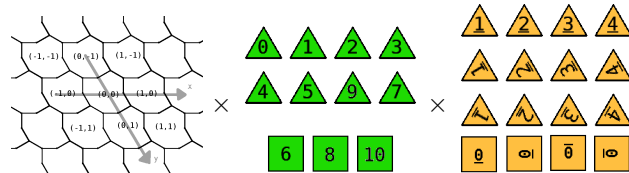
For a face $f \in P$ or a tile $t \in T$, denote by $|f|$ and $|t|$ its number of edges. We number the edges of every face f of polyhedron P counter-clockwise starting from one arbitrary edge that will serve as the reference edge. We do the same for every tile t of the supertile S (and the corresponding tessellation T), with one edge being the reference edge, and the next edges being numbered in clockwise order. A face $f \in P$ is *compatible* with $t \in T$ in the *orientation* o if $|f| = |t|$ and the counter-clockwise sequence of edge lengths and angles in f starting at edge number o matches exactly the clockwise sequence of edge lengths and angles in t starting from the reference edge. This means that f can be placed in the plane with edge number o overlapping with the reference edge of t so that the two polygons overlap perfectly.

We say polyhedron P *rests on the tile* t in the tessellation T with its face f at orientation o if f and t completely overlap and the edge number o of f overlaps the reference edge of t . The *position* of P is then represented by the tuple $\langle t, f, o \rangle$. When T is a periodic tiling with supertile S , and $t = \langle (i, j), s \rangle$ for $s \in S$, then this position can be written as $\langle (i, j), s, f, o \rangle$ (Figure 15). The *state* associated with this position is the tuple $\langle s, f, o \rangle$.

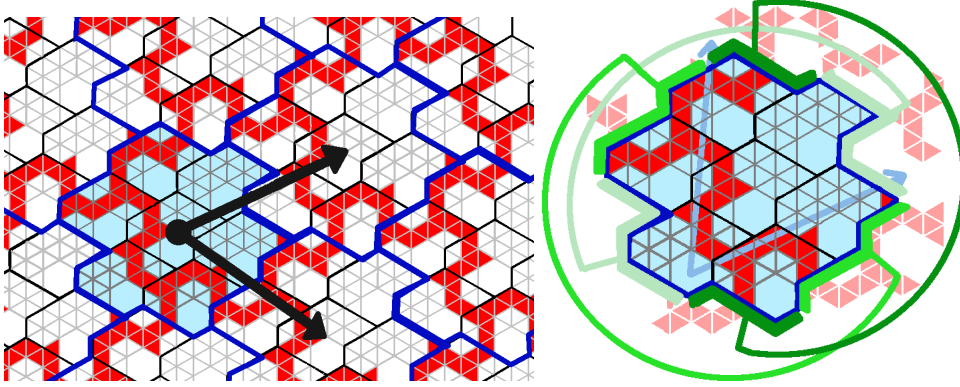
The *rolling graph* $G_{P,T}$ for P and T is an infinite graph whose vertex set is the set of all possible positions $\langle t, f, o \rangle$, and two nodes are connected by an edge if there is a valid roll between them. The positions adjacent to $\langle t, f, o \rangle$ can be easily explored by using the dual graphs of P and T . We write $\langle t, f, o \rangle \sim \langle t', f', o' \rangle$ if the two positions are connected by a path in the rolling graph. In that case, we say that the two positions are *reachable* from one another.



■ **Figure 14** Dual graph of a pyramid with information about the relative orientations of its faces.



■ **Figure 15** A vertex of the rolling graph is composed of $\langle (i, j), (tile, face, orientation) \rangle$



■ **Figure 16** By finding the symmetry vectors in a connected component, we can describe a compact representation of the connected component's periodic graph (over the rolling graph).

2.3 Symmetries of Rolling Graphs

In this section, we show that any large connected subgraph of the rolling graph $G_{P,T}$ has a translational symmetry. We start by bounding the number N of possible states $\langle s, f, o \rangle$ of a rolling graph.

$$\begin{aligned} N &= \sum_{s \in S} \sum_{f \in P} (\text{number of compatible orientations between } f \text{ and } s) \\ &\leq \sum_{s \in S} \sum_{f \in P} |f| \leq 6|S||P|. \end{aligned}$$

The last inequality is by Euler's formula. Note that the rolling graph in itself has the same translational symmetries as the tiling T , because the validity conditions are the same in both positions.

► **Fact 1.** *If $\langle (i, j), s_0, f_0, o_0 \rangle$ has a valid roll to $\langle (i+i_1, j+j_1), s_1, f_1, o_1 \rangle$, then $\langle (i', j'), s_0, f_0, o_0 \rangle$ has a valid roll to $\langle (i' + i_0, j' + j_0), s_1, f_1, o_1 \rangle$ for all $i', j' \in \mathbb{Z}$.*

This however does not mean that the same symmetries apply to the connected components of the rolling graph, that is, $\langle (i, j), s_0, f_0, o_0 \rangle$ and $\langle (i', j'), s_0, f_0, o_0 \rangle$ might not be reachable, even if the connected components are infinite. However, the following lemma shows that if two distinct reachable positions have the same state, then we obtain a translational symmetry on their connected components in the rolling graph.

► **Lemma 1.** *If $\langle (i, j), s, f, o \rangle \sim \langle (i+u, j+v), s, f, o \rangle$, then for all $\langle (i', j'), s', f', o' \rangle \sim \langle (i, j), s, f, o \rangle$, we have $\langle (i', j'), s', f', o' \rangle \sim \langle (i' + u, j' + v), s', f', o' \rangle$.*

That is, $u\vec{a} + v\vec{b}$ defines a translational symmetry on the connected component of $\langle (i, j), s, f, o \rangle$ in the rolling graph.

Proof. Write the path from $\langle (i', j'), s', f', o' \rangle$ to $\langle (i, j), s, f, o \rangle$ in the rolling graph as $\langle (i, j), s, f, o \rangle = \langle (i+i_0, j+j_0), s_0, f_0, o_0 \rangle, \dots, \langle (i+i_k, j+j_k), s_k, f_k, o_k \rangle = \langle (i', j'), s', f', o' \rangle$. Since, by Fact 1, $\langle (i+u+i_\ell, j+u+j_\ell), s_\ell, f_\ell, o_\ell \rangle$ to $\langle (i+u+i_{\ell+1}, j+u+j_{\ell+1}), s_{\ell+1}, f_{\ell+1}, o_{\ell+1} \rangle$ is a valid roll, we can construct the path $\langle (i', j'), s', f', o' \rangle = \langle (i+i_k, j+j_k), s_k, f_k, o_k \rangle, \dots, \langle (i+i_0, j+j_0), s_0, f_0, o_0 \rangle = \langle (i, j), s, f, o \rangle \sim \langle (i+u, j+v), s, f, o \rangle = \langle (i+u+i_0, j+v+j_0), s_0, f_0, o_0 \rangle, \dots, \langle (i+u+i_k, j+v+j_k), s_k, f_k, o_k \rangle = \langle (i' + u, j' + v), s', f', o' \rangle$ ◀

► **Lemma 2.** *There is an algorithm which, in $O(|P||S|)$ time either finds a base of the translational symmetries of the connected component of the rolling graph containing a given position $\langle(i, j), s, f, o\rangle$, or decides that the connected component is of finite size.*

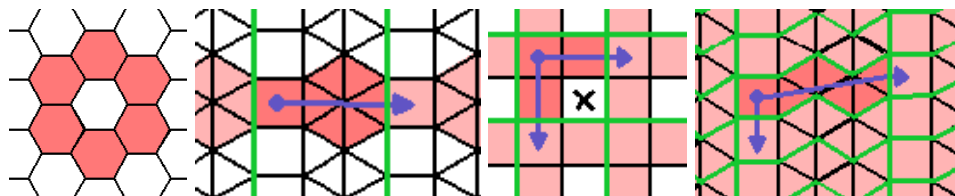
Proof. Run a depth first search on the rolling graph starting from $\langle(i, j), s, f, o\rangle$, for N steps. If the depth first search stops, then the connected component containing $\langle(i, j), s, f, o\rangle$ in the rolling graph is of finite size. Otherwise, by the pigeonhole principle, we have found two positions with the same state. By Lemma 1, we obtain a translational symmetry $u\vec{a} + v\vec{b}$ of the connected component.

Next, factor the rolling graph by this symmetry vector, that is, $G_{P,T}/\{u\vec{a} + v\vec{b}\}$ identifies any pair of positions $\langle(i, j), s, f, o\rangle$ and $\langle(i + ku, j + kv), s, f, o\rangle$ for all $k \in \mathbb{Z}$. Run again a depth first search in $G_{P,T}/\{u\vec{a} + v\vec{b}\}$ starting from $\langle(i, j), s, f, o\rangle$, for N steps. If the depth first search stops, then there are only a finite number of orbits for this symmetry vector, and so only one translational symmetry in this connected component. Otherwise, again by the pigeonhole principle and Lemma 1, we have found a second linearly independent translational symmetry $u'\vec{a} + v'\vec{b}$ for this connected component. ◀

The algorithm in the above lemma finds a basis of two, one or zero translational symmetries in the connected component. We can factor the rolling graph by those symmetries by identifying symmetric tiles. As the symmetries are multiples of the supertile symmetries, this is easily done by performing a coordinate change from the (i, j) coordinates to coordinates in the new basis. When there is no symmetry, the algorithm identifies a bounded connected component in $G_{P,T}$. When there is one symmetry vector, the algorithm finds a finite number of orbits for this symmetry. Finally, when there are two symmetry vectors in the basis, the factored rolling graph $G_{P,T}/\{u\vec{a} + v\vec{b}, u'\vec{a} + v'\vec{b}\}$ is of size polynomial in N and the connected component can be explored completely by depth first search. In all three cases, a compact representation of the connected component has been found. In the two latter cases, it takes the form of a polynomially-sized fundamental domain and one or two translational symmetry vectors.

2.3.1 Results on reachability

The arguments above show how to identify the connected components in the rolling graph. In order to find the set of tiles that can be reached from a starting position, we only need to look at the first part $(i, j), s$ of the positions in the connected component. Because this is a projection, it preserves the symmetry vectors. We obtain the following classification for the reachable area.



■ **Figure 17** No vector, one vector, two vectors but fail to cover, two vectors and full cover.

- If the rolling graph does not have symmetry vectors, the reachable area is bounded and P on T starting at $\langle t, f, o\rangle$, is a *bounded roller*.

- If the rolling graph only has one linearly independent vector, the reachable area is a band and P on T starting at $\langle t, f, o \rangle$ is a 🌀 *band roller*.
- If the rolling graph has two linearly independent vectors, the reachable area extends infinitely in all directions. If not every tile t is present in the reachable supertiles, the reachable tiles forms a plane with holes and P on T starting at $\langle t, f, o \rangle$ is a 🏠 *hollow-plane roller*.
- If every tile t is present in the reachable supertiles, the reachable tiles cover the entire plane and P on T starting at $\langle t, f, o \rangle$ is a 🏠 *plane roller*.

3 Toolbox for Puzzle Designers

As mentioned in the Introduction, a rolling puzzle game typically includes a playing area with obstacles and/or paths, a polyhedron that will navigate that space, a starting position, and a goal position. The starting and/or goal positions sometimes specify a specific polyhedron face to match with a specific tile, in addition to just the tile. Once a polyhedron and a tessellation have been selected, there are several additional properties that can facilitate puzzle design. The rolling graph defined above can also be used to compute them.

3.1 Properties

Unused tiles in the playing area

The first and most crucial piece of information is provided directly by the reachability computed in the previous section. For the puzzle to be solvable, the goal tile should be in the reachable area from the start tile. Also, when the game includes interactive elements, they cannot not be usefully placed on tiles that cannot be reached (except as misdirection).

Unused faces on the polyhedron

For face-matching puzzles, determining which faces of the polyhedron are usable in the puzzle is also important. Some faces might not be compatible with the tiling, while others might not appear in the connected rolling graph despite being compatible. For example, puzzle designers should avoid placing the goal on a polyhedron face that cannot be rolled on. Unused faces of the polyhedron can easily be detected while computing the reachable area.

Guaranteed starting points

When using a plane roller, we must select a starting state (tile, face, and orientation) from which the polyhedron can reach the whole plane. This task can be simplified by selecting a *guaranteed starting point*, which has the property that every tile in the plane is reachable from that starting tile, no matter what polyhedron face and orientation is used as a starting state.

► **Definition 3.** *Given a plane roller pair (P, T) , a tile $t \in T$ is a guaranteed starting point if, for every $f \in P$ with $|f| = |t|$, and for every $o \in f$, we have P on T starting at $\langle t, f, o \rangle$ is a plane roller.*

► **Definition 4.** *Given a rolling pair (P, T) with reachable area RA , a tile $t \in T$ is a guaranteed starting point if, for every $f \in P$ with $|f| = |t|$, for every $o \in f$, and for every $t_i \in RA$, there is a face f_i and orientation o_i such that $\langle t, f, o \rangle \sim \langle t_i, f_i, o_i \rangle$.*

Which faces reach which tiles: face-completeness

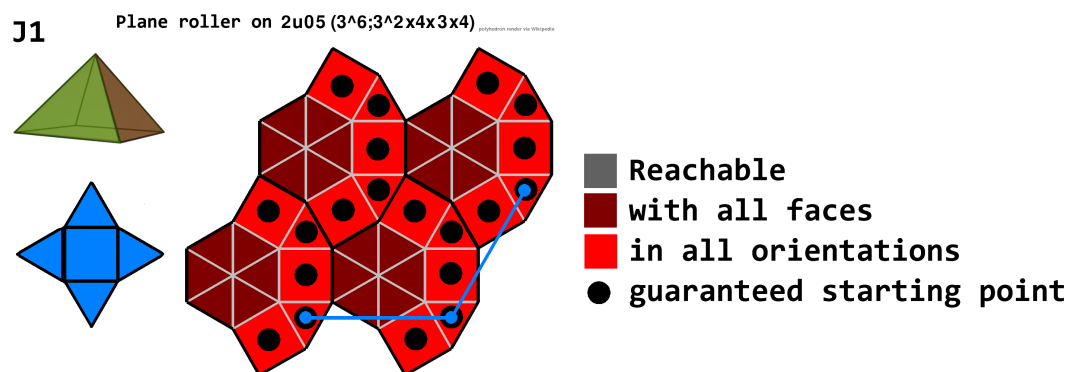
In a face-matching rolling puzzle game, the objective is to reach a specific tile with a specific face on the polyhedron (often marked by a different color). In some cases, not every face of a particular shape can reach every tile. When using a polyhedron/tiling pair in a puzzle game, it can help to know which face can reach which tile. We can track specific tiles that can be reached by every compatible face during our computation. We call such tiles *face-complete* tiles. Refer to Figure 18.

► **Definition 5** (face-complete tile). A rolling pair (P, T) with starting state $\langle t_0, f_0, o_0 \rangle$ has a face-complete tile $t \in T$ if all compatible faces of the polyhedron can roll on t with some orientation, that is, for all $f \in P$ with $|f| = |t|$, there is an orientation o such that $\langle t, f, o \rangle \sim \langle t_0, f_0, o_0 \rangle$.

► **Definition 6** (face-orientation-complete tile). A tile is face-orientation-complete if it can be visited with all compatible faces in every orientation within a connected component.

3.2 Puzzlemaker's Reference Image

We can combine all of the above results into one image that serves as a reference point for puzzlemakers. Figure 18 shows an example. This image allows one to select a tessellation/polyhedron pair very easily depending on the puzzle's needs.



■ **Figure 18** Puzzlemaker's reference image: Left: polyhedron and its used faces (net). Right: Tiling and tile properties.

The full set of reference images can be found on our website: <https://akirabaes.com/polyroll/>.

4 Implementation

The roller classification algorithm was implemented in Python 3.8 and is available on GitHub at <https://github.com/akirabaes/RollingPolyhedron/>. It uses NumPy and SymPy for creating a minimal linearly independent base, and pygame to produce images. The implemented version performs further manipulations, such as aggregating connected rolling graph states grouped by supertile into superstates, to lower processing time and avoid dealing with individual tile positions calculations by only looking at the supertile cartesian coordinates. The result table can be consulted at <https://akirabaes.com/polyroll/resulttable/>.

We defined the supertiles of each tiling by hand in a custom periodic tessellation drawing tool, as we lacked code to automatically convert vertex-type orbits (isohedral, edges) notation to dual-graph supertile (isogonal, tiles) notation, but we did have a list of n -uniform tessellation drawings [7].

An interactive 3D visualization of the rolling logic was implemented by Université libre de Bruxelles Computer Science Bachelor students [3]; see Figure 1.

5 Open Problem

It is left to determine, for the 87 polyhedra out of the 129 considered that did not generate a plane roller with the 131 considered tilings, if there exists a tiling on which they would be able to roll on the 2D plane.

■ **Table 1** Considered polyhedra which did not generate a plane roller with considered tilings.

dodecahedron, truncated cube, truncated octahedron, rhombicuboctahedron, truncated cuboctahedron, snub cube, snub cube c, icosidodecahedron, truncated dodecahedron, truncated icosahedron, rhombicosidodecahedron, truncated icosidodecahedron, snub dodecahedron, snub dodecahedron c, j2, j4, j5, j6, j7, j9, j18, j19, j20, j21, j23, j24, j25, j32, j33, j34, j35, j36, j38, j39, j40, j41, j42, j43, j45, j45 c, j46, j46 c, j47, j47 c, j48, j48 c, j49, j52, j53, j55, j57, j58, j59, j60, j61, j63, j64, j66, j67, j68, j69, j70, j71, j72, j73, j74, j75, j76, j77, j78, j79, j80, j81, j82, j83, j91, j92, triangular prism, pentagonal prism, hexagonal prism, octagonal prism, decagonal prism, dodecagonal prism, pentagonal antiprism, octagonal antiprism, decagonal antiprism, dodecagonal antiprism

References


- 1 Jin Akiyama. Tile-makers and semi-tile-makers. *The American Mathematical Monthly*, 114(7):602–609, 2007.
- 2 Jin Akiyama, Takayasu Kuwata, Stefan Langerman, Kenji Okawa, Ikuro Sato, and Geoffrey C. Shephard. Determination of all tessellation polyhedra with regular polygonal faces. In *International Conference on Computational Geometry, Graphs and Applications*, pages 1–11, 2010.
- 3 Rachel Aouad Albashara, Luca Insisa, Quentin Magron, Dan Ngongo, Dang Phi L Pham, and Simon Yousfi. Rouler des polyèdres sur des tessellations. Université libre de Bruxelles Computer Science Bachelor *Printemps des Sciences* showcase, 2022. URL: <https://akirabaes.com/polyrollly/printempsdessciences2022>.
- 4 Antonio Bicchi, Yacine Chitour, and Alessia Marigo. Reachability and steering of rolling polyhedra: a case study in discrete nonholonomy. *IEEE Transactions on Automatic Control*, 49(5):710–726, 2004.
- 5 Kevin Buchin and Maike Buchin. Rolling block mazes are pspace-complete. *Journal of Information Processing*, 20(3):719–722, 2012. doi:10.2197/ipsjjip.20.719.
- 6 Kevin Buchin, Maike Buchin, Erik D. Demaine, Martin L. Demaine, Dania El-Khechen, Sándor Fekete, Christian Knauer, André Schulz, and Perouz Taslakian. On rolling cube puzzles. In *Proceedings of the 19th Canadian Conference on Computational Geometry (CCCG 2007)*, Ottawa, Canada, August 2007.

6:14 Rolling Polyhedra on Tessellations

- 7 D. Chavey. Tilings by regular polygons – II: A catalog of tilings. *Computers & Mathematics with Applications*, 17(1–3):147–165, 1989.
- 8 Yacine Chitour, Alessia Marigo, Domenico Prattichizzo, and Antonio Bicchi. Rolling polyhedra on a plane, analysis of the reachable set. In *Algorithms for Robotic Motion and Manipulation (WAFR 1996)*, pages 277–285. CRC Press, 1997.
- 9 Euclid. *Elements*, volume I, Circa 300 BCE.
- 10 Judith V. Field. Rediscovering the archimedean polyhedra: Piero della francesca, luca pacioli, leonardo da vinci, albrecht dürer, danielle barbaro, and johannes kepler. *Archive for History of Exact Sciences*, 50(3/4):241–289, 1997.
- 11 Martin Gardner. *Martin Gardner’s Sixth Book of Mathematical Diversions from Scientific American*. W H Freeman, San Francisco, 1971. Chapter 8.
- 12 Martin Gardner. *Mathematical Carnival*. Borzoi / Alfred A Knopf, New York, 1975. Chapter 9, Problem 1: The red-faced cube.
- 13 Martin Gardner. *Time Travel and Other Mathematical Bewilderments*. W H Freeman, New York, 1988. Chapter 9, Problem 8: Rolling cubes.
- 14 B. Grünbaum and N. W. Johnson. The faces of a regular-faced polyhedron. *Journal of the London Mathematical Society*, 1(1):577–586, 1965.
- 15 Branko Grünbaum and G. C. Shephard. *Tilings and Patterns*. W. H. Freeman and Company, 1987.
- 16 Markus Holzer and Sebastian Jakobi. On the complexity of rolling block and Alice mazes. In *Proceedings of the 6th International Conference on Fun with Algorithms*, pages 210–222, 2012.
- 17 Norman W Johnson. Convex polyhedra with regular faces. *Canadian Journal of Mathematics*, 18:169–200, 1966.
- 18 Akihiro Uejima and Takahiro Okada. NP-completeness of rolling dice puzzles using octahedral and icosahedral dices (in Japanese). *IEICE Trans. Fund.*, J94-A(8):621–628, 2011.
- 19 Viktor Abramovich Zalgaller. Convex polyhedra with regular faces. *Zapiski Nauchnykh Seminarov POMI*, 2:5–221, 1967.

A Result Tables

tetrahedron with (3^6) • **cube** with (4^4) • **octahedron** with (3^6) • **icosahedron** with (3^6) • **truncated tetrahedron** with $(3^6; 3^2.6^2)$ • **cuboctahedron** with $(3^2.4.3.4)$, $(3^6; 3^2.4.3.4)$, $(3^3.4^2; 3^2.4.3.4)1$, $(3^6; 3^2.4.3.4; 3^2.4.3.4)$ • **j1** with $(3^2.4.3.4)$, $(3^6; 3^2.4.3.4)$, $(3^3.4^2; 3^2.4.3.4)1$, $(3^6; 3^3.4^2; 3^2.4.3.4)$, $(3^6; 3^2.4.3.4; 3^2.4.3.4)$ • **j3** with $(3^6; 3^2.4.3.3.4; 3.4^2.6)$, $(3^6; 3^2.4.3.4; 3.4^2.6; 3.4.6.4)$ • **j8** with (4^4) , $(3^6; 3^3.4^2; 4^4)1$, $(3^6; 3^3.4^2; 4^4)3$, $(3^6; 3^3.4^2; 3^2.4.3.4; 4^4)$ • **j10** with (3^6) , $(3^6; 3^3.4^2)1$, $(3^6; 3^3.4^2)2$, $(3^6; 3^2.4.3.4)$, $(3^6; 3^3.4^2; 3^2.4.3.4)$, $(3^6; 3^6; 3^3.4^2)1$, $(3^6; 3^6; 3^3.4^2)2$, $(3^6; 3^3.4^2; 3^2.4.3.4; 4^4)$ • **j11** with (3^6) • **j12** with (3^6) • **j13** with (3^6) • **j14** with $(3^6; 3^3.4^2)1$ • **j15** with $(3^6; 3^3.4^2)1$ • **j16** with $(3^6; 3^3.4^2)1$ • **j17** with (3^6) • **j22** with $(3^6; 3^4.6)1$, $(3^6; 3^4.6; 3.6.3.6)2$, $(3^6; 3^4.6; 3.6.3.6)3$, $(3^6; 3^6; 3^4.6^2)$ • **j26** with $(3^2.4.3.4)$, $(3^3.4^2; 3^2.4.3.4)2$, $(3^6; 3^2.4.3.4; 3^2.4.3.4)$ • **j27** with $(3^3.4^2)$, $(3^3.4^2; 3^2.4.3.4)1$, $(3^6; 3^3.4^2; 3^2.4.3.4)$, $(3^3.4^2; 3^2.4.3.4; 3^2.4.3.4)$ • **j28** with $(3^3.4^2)$, $(3^3.4^2; 4^4; 4^4)1$ • **j29** with $(3^2.4.3.4)$, $(3^6; 3^2.4.3.4; 3^2.4.3.4)$ • **j30** with $(3^3.4^2)$ • **j31** with $(3^2.4.3.4)$, $(3^6; 3^2.4.3.4; 3^2.4.3.4)$ • **j37** with (4^4) • **j44** with $(3^6; 3^2.4.3.4; 3^2.4.3.4)$, $(3^3.4^2; 3^2.4.3.4; 3^2.4.3.4)$ • **j44 chiral** with $(3^6; 3^2.4.3.4; 3^2.4.3.4)$ • **j50** with (3^6) , $(3^6; 3^3.4^2)1$, $(3^6; 3^3.4^2)2$, $(3^6; 3^2.4.3.4)$, $(3^6; 3^6; 3^3.4^2)1$, $(3^6; 3^6; 3^3.4^2)2$, $(3^6; 3^3.4^2; 3^2.4.3.4; 4^4)$ • **j51** with (3^6) • **j54** with $(3.4.6.4)$ • **j56** with $(3.4.6.4)$ • **j62** with (3^6) • **j65** with $(3.6.3.6)$ • **j84** with (3^6) • **j85** with (3^6) , $(3^6; 3^3.4^2)1$, $(3^6; 3^3.4^2)2$, $(3^6; 3^2.4.3.4)$, $(3^6; 3^6; 3^3.4^2)1$, $(3^6; 3^6; 3^3.4^2)2$, $(3^6; 3^3.4^2; 3^3.4^2)1$, $(3^6; 3^3.4^2; 3^3.4^2)2$, $(3^6; 3^3.4^2; 3^2.4.3.4; 4^4)$ • **j86** with (3^6) , $(3^6; 3^3.4^2)1$, $(3^6; 3^3.4^2)2$, $(3^6; 3^2.4.3.4)$, $(3^6; 3^3.4^2; 3^2.4.3.4)$, $(3^6; 3^3.4^2; 4^4)1$, $(3^6; 3^3.4^2; 4^4)2$, $(3^6; 3^6; 3^3.4^2)1$, $(3^6; 3^6; 3^3.4^2)2$, $(3^6; 3^3.4^2; 3^2.4.3.4; 4^4)$ • **j87** with (3^6) , $(3^6; 3^3.4^2)1$, $(3^6; 3^3.4^2)2$, $(3^6; 3^2.4.3.4)$, $(3^6; 3^6; 3^3.4^2)1$, $(3^6; 3^6; 3^3.4^2)2$, $(3^6; 3^3.4^2; 3^2.4.3.4; 4^4)$ • **j88** with (3^6) , $(3^6; 3^3.4^2)1$, $(3^6; 3^3.4^2)2$, $(3^6; 3^3.4^2; 3^2.4.3.4)$, $(3^6; 3^3.4^2; 4^4)1$, $(3^6; 3^3.4^2; 4^4)2$, $(3^6; 3^6; 3^3.4^2)1$, $(3^6; 3^6; 3^3.4^2)2$, $(3^6; 3^3.4^2; 3^3.4^2)1$, $(3^6; 3^3.4^2; 3^3.4^2)2$ • **j89** with (3^6) , $(3^6; 3^3.4^2)1$, $(3^6; 3^3.4^2)2$, $(3^6; 3^2.4.3.4)$, $(3^6; 3^3.4^2; 3^2.4.3.4)$, $(3^6; 3^3.4^2; 4^4)3$, $(3^6; 3^3.4^2; 4^4)4$, $(3^6; 3^6; 3^3.4^2)1$, $(3^6; 3^6; 3^3.4^2)2$, $(3^6; 3^3.4^2; 3^3.4^2)1$, $(3^6; 3^3.4^2; 3^3.4^2)2$ • **j90** with (3^6) , $(3^6; 3^3.4^2)1$, $(3^6; 3^3.4^2)2$, $(3^6; 3^2.4.3.4)$, $(3^6; 3^3.4^2; 3^2.4.3.4)$, $(3^6; 3^3.4^2; 4^4)1$, $(3^6; 3^3.4^2; 4^4)2$, $(3^6; 3^6; 3^3.4^2)1$, $(3^6; 3^6; 3^3.4^2)2$, $(3^6; 3^3.4^2; 3^3.4^2)1$, $(3^6; 3^3.4^2; 3^3.4^2)2$, $(3^6; 3^2.4.3.4; 3^2.4.3.4)$, $(3^6; 3^3.4^2; 3^2.4.3.4; 4^4)$ • **square antiprism** with $(3^3.4^2)$ • **hexagonal antiprism** with $(3^4.6)$, $(3^6; 3^4.6)1$, $(3^6; 3^4.6)2$, $(3^4.6; 3^2.6^2)$, $(3^6; 3^4.6; 3^2.6^2)2$, $(3^6; 3^4.6; 3.6.3.6)1$, $(3^6; 3^4.6; 3.6.3.6)2$, $(3^6; 3^4.6; 3.6.3.6)3$, $(3^6; 3^6; 3^4.6^2)$, $(3^6; 3^4.6; 3^4.6)$, $(3^4.6; 3^4.6; 3.6.3.6)1$, $(3^4.6; 3^4.6; 3.6.3.6)2$, $(3^6; 3^4.6; 3^2.6^2; 3.6.3.6)$, $(3^4.6; 3^2.6^2; 3^2.6^2; 3.6.3.6)$

■ **Table 2**  Plane-roller polyhedra and tilings (42 polyhedra and 145 pairings).

tetrahedron (x7) • octahedron (x7) • icosahedron (x7) • truncated tetrahedron (x12)
 • cuboctahedron (x4) • truncated cube • truncated octahedron (x10) •
 rhombicuboctahedron (x8) • truncated cuboctahedron (x6) • snub cube (x13)
 • snub cube chiral (x12) • truncated icosahedron (x4) • rhombicosidodecahedron (x4)
 • truncated icosidodecahedron (x5) • snub dodecahedron (x5) •
 snub dodecahedron chiral (x4) • j1 (x6) • j3 (x8) • j7 (x5) • j8 • j10 (x15) •
 j11 (x6) • j12 (x7) • j13 (x7) • j14 (x15) • j15 (x15) • j16 (x15) • j17 (x7) • j18 (x7) •
 j19 (x4) • j22 • j26 (x2) • j27 (x13) • j28 (x8) • j29 (x2) • j30 (x6) • j31 • j35 (x11) •
 j37 (x3) • j38 (x7) • j44 (x6) • j44 chiral (x5) • j45 (x2) • j45 chiral (x2) • j49 (x8)
 • j50 (x16) • j51 (x7) • j53 (x6) • j54 (x14) • j55 (x10) • j56 (x16) • j57 (x14) •
 j62 (x4) • j65 (x3) • j66 • j72 (x4) • j74 (x10) • j75 (x6) • j76 (x4) • j78 (x4) •
 j79 (x6) • j81 (x4) • j84 (x7) • j85 (x16) • j86 (x16) • j87 (x18) • j88 (x18) • j89 (x20)
 • j90 (x16) • triangular prism (x12) • hexagonal prism (x18) • octagonal prism
 • dodecagonal prism (x4) • square antiprism (x5) • hexagonal antiprism (x2) •
 dodecagonal antiprism (x2)

■ **Table 3** 🧩 Hollow-plane-roller polyhedra and tilings (76 polyhedra and 588 pairings).

tetrahedron (x35) • cube (x41) • octahedron (x35) • icosahedron (x35) •
 truncated tetrahedron (x34) • cuboctahedron (x3) • truncated octahedron (x15)
 • rhombicuboctahedron (x43) • snub cube (x7) • snub cube chiral (x8) •
 truncated icosahedron (x13) • rhombicosidodecahedron (x2) • snub dodecahedron (x7)
 • snub dodecahedron chiral (x7) • j1 (x7) • j3 (x2) • j7 (x43) • j8 (x39) • j9 (x42) •
 j10 (x29) • j11 (x31) • j12 (x35) • j13 (x35) • j14 (x42) • j15 (x42) • j16 (x42) • j17 (x35)
 • j18 (x43) • j19 (x41) • j20 (x42) • j21 (x42) • j22 (x30) • j23 (x30) • j24 (x30) • j25 (x30)
 • j26 (x7) • j27 (x17) • j28 (x46) • j29 (x4) • j30 (x15) • j31 (x3) • j35 (x44) • j36 (x49) •
 j37 (x46) • j38 (x44) • j39 (x47) • j40 (x42) • j41 (x42) • j42 (x42) • j43 (x42) • j44 (x32)
 • j44 chiral (x33) • j45 (x31) • j45 chiral (x32) • j46 (x32) • j46 chiral (x32) • j47 (x30) •
 j47 chiral (x30) • j48 (x30) • j48 chiral (x30) • j49 (x20) • j50 (x27) • j51 (x35) • j54 (x8)
 • j55 (x10) • j56 (x15) • j57 (x11) • j62 (x17) • j65 (x25) • j67 • j72 • j73 (x5) • j76 •
 j77 (x5) • j80 (x5) • j84 (x35) • j85 (x32) • j86 (x27) • j87 (x28) • j88 (x23) • j89 (x21)
 • j90 (x25) • triangular prism (x45) • pentagonal prism (x42) • hexagonal prism (x36)
 • octagonal prism (x42) • decagonal prism (x42) • dodecagonal prism (x43) •
 square antiprism (x37) • pentagonal antiprism (x30) • hexagonal antiprism (x40) •
 octagonal antiprism (x30) • decagonal antiprism (x30) • dodecagonal antiprism (x30)

■ **Table 4** 🧩 Band-roller polyhedra and tilings (94 polyhedra and 2623 pairings).

Beedroids: How Luminous Autonomous Swarms of UAVs Can Save the World?

Quentin Bramas ✉ 


University of Strasbourg, ICUBE, CNRS, France

Stéphane Devismes ✉ 

Université de Picardie Jules Verne, MIS UR 4290, Amiens, France

Anaïs Durand ✉ 

University Clermont Auvergne, CNRS UMR 6158, LIMOS, Aubière, France

Pascal Lafourcade ✉ 

University Clermont Auvergne, CNRS UMR 6158, LIMOS, Aubière, France

Anissa Lamani ✉ 

University of Strasbourg, ICUBE, CNRS, France

Abstract

Bee extinction is a great risk for humanity. To circumvent this ineluctable disaster, we propose to develop beedroids, *i.e.*, small UAVs mimicking the behaviors of real bees. Those beedroids are endowed with very weak capabilities (short-range visibility sensors, no GPS, light with a few colors, ...). Like real bees, they have to self-organize together into swarms. Beedroid swarms will be deployed in cuboid-shaped greenhouse. Each beedroid swarm will have to indefinitely search for flowers to pollinate in its greenhouse. We model this problem as a perpetual exploration of a 3D grid by a swarm of beedroids. In this paper, we propose two optimal solutions to solve this problem and so to save humanity.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Bee extinction, luminous swarms of beedroids, perpetual flower pollination problem, greenhouse

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.7

Funding This study was partially supported by the French ANR project ESTATE (ANR-16 CE25-0009-03).

Anissa Lamani: This work was supported by the University of Strasbourg Initiative of Excellence.

1 Introduction

Bees are crucial for human beings. They have limited life span: only few weeks for the workers and up to 6 years for the queen. The United Nation (UN) dedicates the 20th of May as the “Bee day”.¹ UN says that “we all depend on the survival of bees”. Indeed, pollination is a fundamental process for the survival of our ecosystems. Nearly 90% of the world’s wild flowering plant species depend on pollination. There are more than 800 wild bee species, seven of which are classified by the International Union for Conservation of Nature² (IUCN) as critically endangered. A further 46 are endangered, 24 are vulnerable, and 101 are near threatened. Many associations like Greenpeace³ or World Wild Foundation⁴ (WWF) are protecting bees and helping to avoid their extinction.

¹ <https://www.un.org/en/observances/bee-day>

² <https://www.iucn.org/>

³ <https://cdn.buglife.org.uk/2019/08/CM-EofE-bee-report-2019-Headlines-FINAL-CJ.pdf>

⁴ <https://www.greenpeace.org/static/planet4-international-stateless/2013/04/66f3eb6b-beesindecline.pdf>



© Quentin Bramas, Stéphane Devismes, Anaïs Durand, Pascal Lafourcade, and Anissa Lamani; licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 7; pp. 7:1–7:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

7:2 Beedroids: How Luminous Autonomous Swarms of UAVs Can Save the World?

Our goal here is to anticipate by considering the worst-case: the bee extinction. We propose solutions to save humanity using *Beedroids*. Beedroids are artificial bees that aim at pollinating flowers autonomously in a greenhouse. Such a technology can also be used in the Mars colonization. For example, the Biosphere 2 project⁵ was meant to demonstrate the viability of closed ecological systems to support and maintain human life in outer space. Beedroids are mandatory to implement such a project since it is not yet clear whether bees can survive interstellar trips.

A beedroid is a small autonomous Unmanned Aerial Vehicles (UAV) that mimics the behavior of a real bee. A famous ability of bees is stigmergy, *i.e.*, indirect communication through the movements. Implementing stigmergy requires perfect synchronization and visibility sensors. Consequently, we consider here a fully synchronous look-compute-move model of computation [20] (FSYNC). As explained before, beedroids will be deployed in greenhouses. So, each beedroid swarm will have to perpetually explore its greenhouse to find flowers and pollinate them. We assume greenhouses are finite cuboids. Each of these cuboids should be divided into cells to visit. Thus, we conveniently discretized a cuboid-shaped greenhouse as a finite 3D grid. Then, the problem we have to solve consists in coordinating a swarm of beedroids to perpetually explore a finite 3D grid in an exclusive manner (meaning that two beedroids cannot occupy the same place simultaneously).

The world bee population constantly decreases but is still incredibly huge as compared to human population. As a matter of facts, a bee colony is constituted of around 50 000 bees and in 2018 it was estimated that there were around 800 000 bee colonies in Canada for example. So, to eventually be able to replace bees, we should (1) solve pollination of greenhouses with the smallest possible number of beedroids and (2) also be able to massively produced them. Consequently, the design of beedroids should be as simple as possible. Below we list and motivate our main design choices.

Visibility Sensors: To maximize their flight time, we should save energy. Hence, the visibility range of beedroids should be as small as possible.

Communication: As previously explained, the communication between beedroids, like real bees, is indirect and based on positions of other bees. To vastly increase stigmergic communication, without compromising production cost and energy consumption, we have only endowed them with LED lights of a few colors that can be sensed by other beedroids within a short distance.

Memory: Still to save energy and manufacturing costs, beedroids have no permanent memory, except the color of their lights. They only have a short-term working memory allowing them to compute a decision (destination and new light color) at each step of their algorithm.

Orientation: Manufacturing costs and energy consumption also prevent us from endowing beedroids with GPS. Instead, we use chirality facilities making beedroids able to distinguish the two sides of a symmetrically reflexive panorama.

Contribution. To bring our own stone to the world safeguarding, we propose to implement pollination into cuboid-shaped greenhouses using swarms of artificial bees, so-called beedroids. Solving this problem requires to coordinate each swarm so that it perpetually explores a 3D grid. As motivated before, we should both minimize the size of the swarm (*i.e.*, the number of beedroids that compose it) and the capabilities used by those beedroids.

⁵ <https://biosphere2.org>

We first study the problem in the FSYNC model assuming the optimal visibility range one. Under this assumption, we show that three beedroids are necessary and sufficient to solve the problem. For the sufficient part, we propose an algorithm that requires only five colors. Then, we look for another solution optimal in terms of colors, still in FSYNC model. Actually, the solution we propose works with oblivious beedroids, *i.e.*, beedroids endowed with only one light color. This second solution requires five beedroids and visibility range two.

In order to help the reader, online animations illustrating the behavior of our algorithms are available for our two solutions: [3] and [4].

Roadmap. In the next section, we formally define the model, the beedroid skills, and the problem to solve, so-called the *Perpetual Flower Pollination Problem* (PFPP). In Section 3, we present the lower bound on the number of beedroids necessary to solve the PFPP under visibility range 1. In Sections 4 and 5, we present two algorithms solving the PFPP. Section 6 is dedicated to related work. Finally, we make concluding remarks in Section 7. Due to the lack of space, several proofs have been omitted.

2 Preliminaries

We consider a swarm of $n > 0$ beedroids (*n.b.*, n is *a priori* unknown by beedroids) evolving in a greenhouse modeled as a *finite 3D grid* of size $S_x \times S_y \times S_z$, with $S_x \geq n$, $S_y \geq n$, $S_z \geq n$, *i.e.*, an undirected graph $G(V, E)$ where $V = \{(i, j, k) : i \in [0, S_x - 1], j \in [0, S_y - 1], k \in [0, S_z - 1]\}$ and $E = \{\{(i, j, k), (i', j', k')\} : |i - i'| + |j - j'| + |k - k'| = 1\}$. Note that coordinates are used for the analysis only, *i.e.*, beedroids cannot access them.

We assume discrete time and at each *round*, the beedroids synchronously perform a *Look-Compute-Move* cycle. In the *Look* phase, a beedroid gets a snapshot of the subgraph induced by the nodes within distance $\Phi \in \mathbb{N}^*$ from its position. Φ is called the *visibility range* of the beedroids. The snapshot is not oriented in any way as the beedroids do not agree on the orientation of any of the three axes of the coordinate system. However, it is implicitly ego-centered since the beedroid that performs a *Look* phase is located at the center of the subgraph in the obtained snapshot. Then, each beedroid *computes* a destination in its local coordinate system (either Front, Back, Left, Right, Above, Below, or Idle) based on the received snapshot only. Finally, it *moves* towards its computed destination.

We forbid any two beedroids to occupy the same node simultaneously. A node is *occupied* when a beedroid is located at this node, otherwise it is *empty*. Beedroids have *lights* with maybe different colors that can be seen by beedroids within distance Φ from them. We denote by \mathcal{Cl} the set of all possible colors ($|\mathcal{Cl}| = 1$ corresponds to the case of oblivious beedroids).

The *state* of a node is either the color of the light of the beedroid located at this node, if it is occupied, or \perp otherwise. In the *Look* phase, the snapshot includes the state of the nodes (within distance Φ). During the *compute* phase, a beedroid may decide to change the color of its light (of course, if $|\mathcal{Cl}| > 1$).

In all our algorithms, we also prevent any two beedroids from traversing the same edge simultaneously. Since we already forbid them to occupy the same position simultaneously, this means that we additionally prevent beedroids from swapping their positions. Algorithms verifying this property are said to be *exclusive*. However, to be as general as possible, we do not make this additional assumption in our impossibility result.

Configurations. A *configuration* C in a 3D grid $G(V, E)$ is a set of pairs (p, c) , where $p \in V$ is an occupied node and $c \in Cl$ is the color of the beedroid located at p . A node p is empty if and only if $\forall c, (p, c) \notin C$. We sometimes just write the set of occupied nodes when the colors are clear from the context.

Views. We denote by G_r the *globally oriented view* centered at the beedroid r , *i.e.*, the subset of the configuration containing the states of the nodes at distance at most Φ from r , translated so that the coordinates of r is $(0, 0)$. We use this globally oriented view in our analysis to describe the movements of the beedroids (see, for example, Figure 1): when we say “the beedroid moves Left”, it is according to the globally oriented view. However, since beedroids do not agree on any axis, they have no access to the globally oriented view. When a beedroid looks at its surroundings, it instead obtains a *local view*. To model this, we assume that the local view acquired by a beedroid r in the Look phase is the result of an arbitrary *indistinguishable transformation* on G_r . Here, we assume that beedroids are *self-inconsistent*, meaning that different transformations may be applied at different rounds. An indistinguishable transformation consists of applying to each of the three axes (x -axis, y -axis, and z -axis) passing through r a rotation (maybe different for each axis) picked in the set $\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$. We denote by \mathcal{IT} the set of all possible indistinguishable transformations. Nevertheless, beedroids share a common chirality, which means that their local view is not a reflection (also called mirroring) of the global view. Having a common chirality allows a beedroid to distinguish a local view from its reflection and so take different decisions in such cases (*e.g.*, chirality allows to discriminate Above from Below in Rule R_1 of Figure 1). In other words, having a common chirality allows, given two axes, to determine a third one using the right-hand rule.

It is important to note that when a beedroid r computes a destination d , it is relative to its local view $f(G_r)$, which is the globally oriented view transformed by some $f \in \mathcal{IT}$. So, the actual movement of the beedroid in the *globally oriented view* is $f^{-1}(d)$. For example, if $d = \text{Above}$ but the beedroid sees the 3D grid upside-down (f is the π -rotation along the y -axis), then the beedroid moves $\text{Below} = f^{-1}(\text{Above})$. In a configuration C , $V_C(i, j)$ denotes the globally oriented view of a beedroid located at (i, j) .

A beedroid is said to be *lost* when it sees no wall and no other beedroids. Observe that in this case, if the beedroid decides to move, the destination is entirely determined by the choice of the transformation f done by the adversary.

Algorithm. An algorithm A is a tuple $(Cl, Init, T)$ where Cl is the set of possible colors, $Init$ is a mapping from any considered 3D grid to a non-empty set of initial configurations in that 3D grid, and T is the transition function $Views \rightarrow \{\text{Idle}, \text{Front}, \text{Back}, \text{Left}, \text{Right}, \text{Above}, \text{Below}\} \times Cl$, where $Views$ is the set of local views. When the beedroids are in Configuration C , a configuration C' obtained after one round satisfies: $((i, j, k), c') \in C'$, if and only if there exists a color $c \in Cl$ and a transformation $f \in \mathcal{IT}$ such that one of the following conditions holds:

- $((i, j, k), c) \in C$ and $f^{-1} \circ T \circ f \circ V_C(i, j, k) = (\text{Idle}, c')$,
- $((i - 1, j, k), c) \in C$ and $f^{-1} \circ T \circ f \circ V_C(i - 1, j, k) = (\text{Right}, c')$,
- $((i + 1, j, k), c) \in C$ and $f^{-1} \circ T \circ f \circ V_C(i + 1, j, k) = (\text{Left}, c')$,
- $((i, j - 1, k), c) \in C$ and $f^{-1} \circ T \circ f \circ V_C(i, j - 1, k) = (\text{Front}, c')$,
- $((i, j + 1, k), c) \in C$ and $f^{-1} \circ T \circ f \circ V_C(i, j + 1, k) = (\text{Back}, c')$,
- $((i, j, k - 1), c) \in C$ and $f^{-1} \circ T \circ f \circ V_C(i, j, k - 1) = (\text{Above}, c')$, or
- $((i, j, k + 1), c) \in C$ and $f^{-1} \circ T \circ f \circ V_C(i, j, k + 1) = (\text{Below}, c')$.

We denote by $C \rightsquigarrow C'$ the fact that C' can be reached in one round from C (*n.b.*, \rightsquigarrow is then a binary relation over configurations). An *execution* of Algorithm A in a 3D grid G is then a sequence $(C_i)_{i \in \mathbb{N}}$ of configurations such that $C_0 \in \text{Init}(G)$ and $\forall i \geq 0, C_i \rightsquigarrow C_{i+1}$.

The Perpetual Flower Pollination Problem. An execution $(C_i)_{i \in \mathbb{N}}$ in a 3D grid $G = (V, E)$ achieves the *Perpetual Flower Pollination Problem* (PFPP) in 3D grids if for every node $u \in V$ and for every time t , there exists a time $t' \geq t$ such that u is occupied in $C_{t'}$.

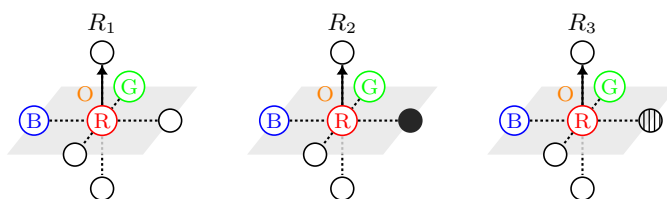
An algorithm A that uses n beedroids solves the PFPP problem if, for every finite 3D grid $G = (V, E)$ of size at least $n \times n \times n$ and every initial configuration $C_0 \in \text{Init}(G)$, we have every execution of A in G starting from C_0 that achieves the PFPP.

An Algorithm as a Set of Rules. We write an algorithm as a set of rules, where a *rule* is a triplet $(V, d, c) \in \text{Views} \times \{\text{Idle}, \text{Front}, \text{Back}, \text{Left}, \text{Right}, \text{Above}, \text{Below}\} \times \text{Cl}$. We say that an algorithm $(\text{Cl}, \text{Init}, T)$ includes the rule (V, d, c) , if $T(V) = (d, c)$. By extension, the same rule applies to indistinguishable views, *i.e.*, $\forall f \in \mathcal{IT}, T(f(V)) = (f(d), c)$. Consequently, we forbid an algorithm to contain two rules (V, d, c) and (V', d', c') such that $V' = f(V)$ for some $f \in \mathcal{IT}$.

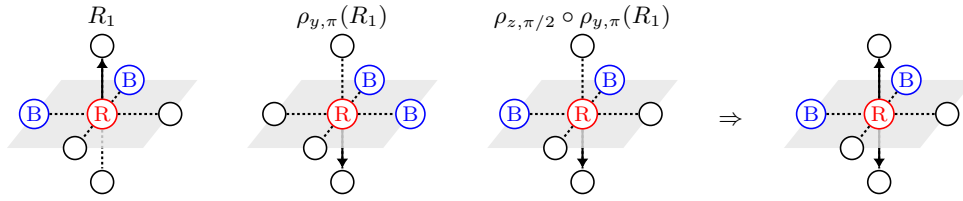
As an illustrative example, consider the rule R_1 given in Figure 1. This rule is defined for beedroids having a visibility range of one. This rule means that, when a beedroid sees two beedroids, one with Color B on its left and the other with color green in front of it, then the red beedroid is dictated to move Above and change its color to O . By extension, the same rule applied if the view is rotated by π on the z -axis, but in that case, the destination would be Below.

In the same figure, Rule R_2 is a rule where a black node represent a part of the outer boundary of the 3D grid, that we call *a wall* in the remaining of the paper. In our algorithms, we often define similar rules that apply regardless of the presence of a wall in some part of the view. Thus, to avoid defining several time rules with very similar views, we propose a notation to represent several rules in just one picture. For example, Rule R_3 in Figure 1 has one node hatched with vertical lines, which means that the rule applies regardless of the presence of a wall located at this node. In practice, every rule that contains such vertical (resp. horizontal) hatched lines, represents a set of rules obtained by replacing each of those lines either by walls, or by empty nodes. For example, Rule R_3 in Figure 1 is a concise representation of Rules R_1 and R_2 .

Figure 2 shows an ambiguous rule. The beedroid has a symmetric view so that, depending on the transformation f chosen by the adversary, the beedroid executing this rule moves either above, or below. In the following, we represent in ambiguous rules all possible destinations that can be dictated by the adversary.



■ **Figure 1** Examples of rules. Colored letters inside nodes indicate the color of the beedroids occupying the nodes. The arrow indicates the destination and when a colored letter is given next to an arrow, this means that the rule dictates the beedroid to switch to that color.



■ **Figure 2** An ambiguous rule. Since the destination depends on the choice of the adversary, we represent the rule abusively with multiple destinations, as illustrated on the right. Observe that if blue beedroids had different colors (as in Figure 1) the rule would not have been ambiguous. Indeed, having a common chirality allows the beedroid to determine a third direction, given two direction obtained from the view.

3 Impossibility Result

In this section, we establish that there is no algorithm solving the PFPP problem in 3D grids using two beedroids with visibility range one, whatever the finite number of available colors.

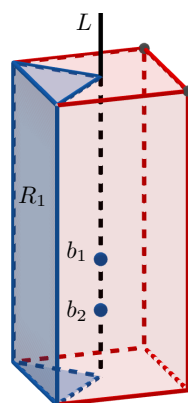
We first observe that in large enough grids, if beedroids travel a long distance without seeing a wall, then they execute a periodic sequence of movements. Indeed, in our settings, there are at most $B = \binom{|Cl|}{2} = \frac{|Cl|(|Cl|+1)}{2}$ different views without wall, and so at most B associated rules, where Cl is the set of available colors. Thus, if the two beedroids travel a distance at least B without seeing a wall, then they are executing a periodic sequence of movements. The definition of B , which depends on the algorithm, will be used throughout this section.

The above observation is important to prove our impossibility results. Actually, the outline of our proof is similar to the 2D case for luminous non-chiral robots studied in [19]. However, the existence of an axis of symmetry is replaced here by an axis of rotational symmetry. Indeed, robots in a 2D grid that do not agree on a common chirality cannot distinguish between two destinations that are symmetric with respect to an axis of symmetry. Similarly, two beedroids that do not agree on a common coordinate system, but agree on a common chirality, cannot distinguish between four destinations that are symmetric with respect to a rotational symmetry. Nevertheless, the proof of the main argument, given in Lemma 6 of [19], cannot be adapted directly since it is more complex than in the 2D case.

The overview of the proof is as follows. We proceed by contradiction assuming that an algorithm solving the PFPP in 3D grids exists. Then, we show that once beedroids move far away from the walls, their possible movements are restricted. In more details, they can only move straight, otherwise they may not explore the whole grid. Next, we show that, once the exploration has reached specific places, the beedroids must always stay close to at least one wall (Lemma 6), leading to the final contradiction (Theorem 7).

The two first lemmas are technical results that are in particular used in the main lemma, Lemma 6. The first one states that to explore the 3D grid, beedroids should stay neighbors when they do not see any wall. The second one shows that if a beedroid b_1 is lost, at distance 2 from a wall, and at distance at least 4 from other walls, then the other beedroid b_2 should be adjacent to a wall; moreover b_1 should wait for b_2 which, in turn, should eventually leave the wall to meet b_1 .

► **Theorem 1.** *Let A be an algorithm solving the PFPP in 3D grids using two beedroids under visibility range one. In a 3D grid of size at least $4 \times 4 \times 4$, no execution of A reaches a configuration where the two beedroids are lost.*



■ **Figure 3** If beedroids are on a line L , the adversary can decide from which side of the square cuboid the beedroids will exit. In particular, we can decide that the beedroids will exit toward R_1 , the blue area.

► **Theorem 2.** *Let A be an algorithm solving the PFPP in 3D grids using two beedroids under visibility range one. Consider an execution E of A in a 3D grid of size at least $8 \times 8 \times 8$. If E contains a configuration C where a beedroid b_1 is lost, at distance 2 from a wall, and at distance at least 4 from the other walls, then*

- b_1 is idle, moreover
- the other beedroid b_2 is adjacent to a wall and is either idle or moves away from the wall during the next step.

The two next lemmas are also technical results used in the proof of Lemma 6. They establish important properties related to long-term travels during which beedroids see no wall.

► **Theorem 3.** *Let A be an algorithm solving the PFPP in 3D grids using two beedroids under visibility range one. If there exists an execution that reaches a configuration C where beedroids are at distance at least $2B$ from any wall and, from C , the beedroids perform a periodic sequence of movements without ambiguous rules, then there is a straight line of the 3D grid that contains the two beedroids while none of them sees a wall.*

► **Theorem 4.** *Let A be an algorithm solving the PFPP in 3D grids using two beedroids under visibility range one. There exists no execution that reaches a configuration C where beedroids are at distance at least $2B$ from any wall and, from C , the beedroids perform a periodic sequence of movements that includes an ambiguous rule.*

The next lemma states that if two beedroids are on the same line and inside an area that is rotationally symmetric, then they cannot break this symmetry without executing an ambiguous rule (due to the lack of agreement on the coordinate system). Hence, the adversary can decide on which side of the line the beedroids move. More formally, if beedroids move out of the area through a node u , then there exists an execution where the beedroids move out of the area through another node u' that is symmetric to u .

We need to define a few concepts beforehand; see Figure 3 for an illustration. Consider a configuration C where the two beedroids are located in some line of the 3D grid. We call *blurred area* of L in C any subset S of nodes including the two nodes where beedroids are

located and such that the subgrid induced by S shapes a square cuboid for which L is an axis of rotational symmetry. A blurred area is non-trivial if it does not contain all nodes of the 3D grid. We denote by $EH(S)$ the external hull of a blurred area S , *i.e.*, the set of nodes in $V \setminus S$ at distance one from a node of S .

► **Theorem 5.** *Let A be an algorithm solving the PFPP in 3D grids using two beedroids under visibility range one. Consider an execution E reaching a configuration C where the two beedroids are on the same line L . Let S be a blurred area of L in C . Let $R_1 \subseteq EH(S)$ such that the union of R_1 and its symmetric w.r.t. the rotations around L is equal to $EH(S)$. If S is non-trivial and L is also an axis of rotational symmetry of $EH(S)$, then there exists an execution from C where a beedroid reaches $EH(S)$ for the first time at a node $u \in R_1$.*

In particular, we can choose R_1 to be the union of one side and two triangles, as shown in Figure 3 (page 7), where R_1 and its symmetric form the red external hull of the square cuboid. Assume the square cuboid leans against a wall (as illustrated in Figure 5 if the gray plan is a wall). Then, R_1 can consist only in one face and one triangle. By applying the lemma, we obtain that there is an execution where the beedroids escape from the blurred area either through a rectangular face or the top triangle.

The lemma below is the cornerstone of our impossibility result. It states that there are configurations from which the two beedroids can remain forever at bounded distance from walls. To see this, we need to define a few concepts beforehand. We say that a beedroid is *2-close* if it is at distance at most $2B$ from at least two walls. We say that beedroids are in a *T-configuration* if there is a line L of the 3D grid such that

- L contains the two beedroids,
- one of them is adjacent to a wall that is orthogonal to L , and
- robots are at distance at most $4B$ from another wall.

► **Theorem 6.** *Let A be an algorithm solving the PFPP in 3D grids using two beedroids under visibility range one. Assume some execution E reaches at a given time t (i) a configuration where a beedroid is 2-close or (ii) a T-configuration. Let C be the configuration of E at time t . Then, there exists an execution E' and a time $t' > t$ such that*

- C is reached in E' at time t ,
- at time t' in E' , a beedroid is 2-close or the system is in a T-configuration, and
- between time t and t' in E' , the beedroids remain at distance at most $4B$ from a wall.

Proof. We consider a 3D grid whose size is more than $8B \times 8B \times 8B$. The lemma otherwise trivially holds: by definition of A , a beedroid is infinitely often 2-close; moreover every beedroid is always at distance at most $4B$ from a wall in a 3D grid where at least one side is less or equal to $8B$.

Assume first that a beedroid is 2-close in a configuration C (the case where beedroids are in a T-configuration will be treated in the last paragraph of this proof). To explore the 3D grid, the two beedroids must sometimes be not 2-close. Indeed, if a beedroid remains 2-close forever, the other beedroid should in particular explore nodes at distance more than $2B + 2$ from every wall. In this case, it would be lost and consequently, the adversary can make it alternating between two nodes forever, making the exploration fail.

Consider now an execution E' such that, whenever a beedroid executes an ambiguous moves that could make it not 2-close, then the adversary chooses a destination that is 2-close. This is possible because, in case of an ambiguous move, the adversary can chose among at least 4 destinations and if one destination would make the beedroid not 2-close, then the opposite destination (with respect to the beedroid's position) would keep it 2-close (because when a beedroids makes a move, only the distance to one wall increases).

In E' , let $t_0 > t$ be the first time when no beedroid is 2-close. By assumption, at least one beedroid that is 2-close at time $t_0 - 1$, say b_2 , makes an unambiguous move. To make this unambiguous move, b_2 necessarily moves toward the other one, b_1 that is not 2-close. So, at time $t_0 - 1$, only b_2 is 2-close, *i.e.*, at distance at most $2B$ from two walls W_1 and W_2 . Without loss of generality, at time t_0 , b_2 is at distance $2B + 1$ from wall W_1 and at distance at most $2B$ from W_2 (b_2 is at distance more than $2B$ from other walls).

Since at time $t_0 - 1$, b_2 is moving towards b_1 , then, at time $t_0 - 1$, the two beedroids are on a line parallel to the wall W_2 . Assume first that beedroids are not adjacent to W_2 . Two cases can occur (both cases are represented in Figure 4) (page 10).

Case (1): They remain on the same line parallel to W_2 , moving away from W_1 , until a beedroid is at distance $3B + 1$ from W_1 .

If they do so, since they traveled a distance B since $t_0 - 1$, they are executing a periodic sequence of movements, hence, they continue to move on the same line until reaching the wall opposite to W_1 (Lemmas 3 and 4), in a T -configuration, while remaining at distance at most $2B$ from W_2 , and the lemma holds in this case.

Case (2): Before being at distance $3B + 1$ from W_1 , one or two beedroids move away from the line they were traveling through.

These moves are necessarily ambiguous. If two beedroids moves away simultaneously, we get a contradiction because the adversary can choose the destination so that the two beedroids become lost (Lemma 1). So, only one beedroid, say b_1 , moves away from the line. Again, since beedroids cannot become lost (Lemma 1) and the destination of b_1 is chosen by the adversary, we can consider the case where the two beedroids end up, at time $t_1 \geq t_0$, in a line L orthogonal to W_2 .

Consider now the case where the beedroids are adjacent to W_2 at time $t_0 - 1$ when moving away from W_1 . Similar things occur.

Case (a): They perform a periodic movement while remaining adjacent to W_2 and travel along the wall (but not necessarily in straight line) until reaching another wall (so they become 2-close), and the lemma holds in this case;

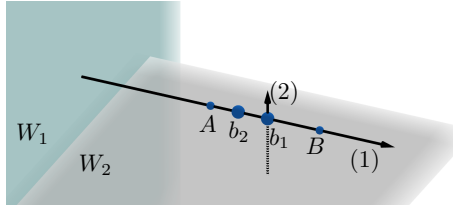
Case (b): a beedroid moves away from W_2 and forms with the other beedroid a line L orthogonal to W_2 before being at distance $3B + 1$ from W_1 ; or

Case (c): both beedroids move away from W_2 (simultaneously or one after the other, as a lost beedroid must wait the other beedroid, by Lemma 2) and they end up in a line L parallel to W_2 .

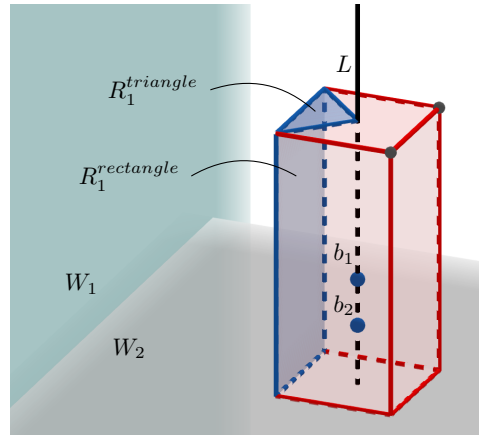
In this case, since they traveled at most B from t_0 , they can travel again a distance at most B before either performing a periodic movement (Case (1)) while staying at distance 2 from W_2 (and the lemma holds in this case), or making an ambiguous move to end up in a line L orthogonal to W_2 (Case (2)). In this latter case, the beedroids end up at distance at most $4B$ from W_1 .

So, the only cases that remains to consider are those where the beedroids are in a line L orthogonal to W_2 at distance at most $4B$ from W_1 .

Consider the set of nodes $R_1 = R_1^{rectangle} \cup R_1^{triangle}$ where $R_1^{rectangle}$ is a rectangle of nodes at distance $2B$ from the wall W_1 and $R_1^{triangle}$ a triangle of nodes at distance $2B$ from W_2 and at most $4B$ from W_1 , such that the union of R_1 and its image by the three rotations around L form the external hull of a square cuboid containing the two beedroids for which (1) one face is at distance 1 from W_2 and (2) L is axis of rotational symmetry; see Figure 5 for an illustration (page 10). Using Lemma 5, there exists an execution such that a beedroid reaches R_1 .



■ **Figure 4** Position of the bedroids when they become not 2-close.



■ **Figure 5** Position of the bedroids if they execute an ambiguous move after becoming not 2-close.

If a bedroid reaches $R_1^{rectangle}$, then a bedroid becomes 2-close and the lemma is proven. If a bedroid reaches $R_1^{triangle}$, then the bedroids have traveled a distance at least B without seeing a wall, hence are executing a periodic sequence of movements. The sequence cannot contain an ambiguous rule (using Lemma 4) because the bedroids are at distance at least $2B$ from any wall, so they are moving in a straight line (by Lemma 3), and they end up in the wall opposite to W_2 and reach a T -configuration, while remaining at distance at most $4B$ from W_1 .

Now we consider that the bedroids are in a T -configuration in configuration C . Then, they are on a line L orthogonal to a wall, say W_2 , and at distance at most $4B$ from another wall, say W_1 . Using a similar argument, we know that either the bedroids become 2-close, or move in a straight line to the opposite wall until they reach a T -configuration (while remaining at distance at most $4B$ from W_1). ◀

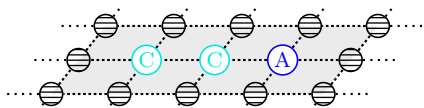
Using the previous lemma, we can now conclude.

► **Theorem 7.** *The PFPP is not solvable using two bedroids under visibility range 1 and any finite number of colors.*

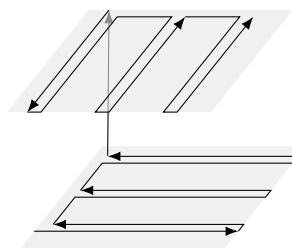
Proof. Assume that algorithm A solves the problem. Consider a grid of size $10B \times 10B$. Since the bedroids explore the entire grid, there exists a round where a bedroid is 2-close. By applying Lemma 6 repeatedly, we can construct an execution from there where bedroids forever remain at distance at most $4B$ from a wall, so that nodes at distance more than $4B$ from all the walls are not visited anymore, a contradiction. ◀

4 Visibility range one: Vone_5^3

In this section, we address the PFPP using bedroids under visibility range one. We present an algorithm, denoted by Vone_5^3 , that solves the PFPP using three bedroids endowed with five colors. By Theorem 7, under visibility range one, Vone_5^3 is optimal with respect to the number of bedroids. We encourage the reader to follow the overview of Vone_5^3 while looking at the animations available online [3], published as an additional material.



■ **Figure 6** Initial configuration of the beedroids.



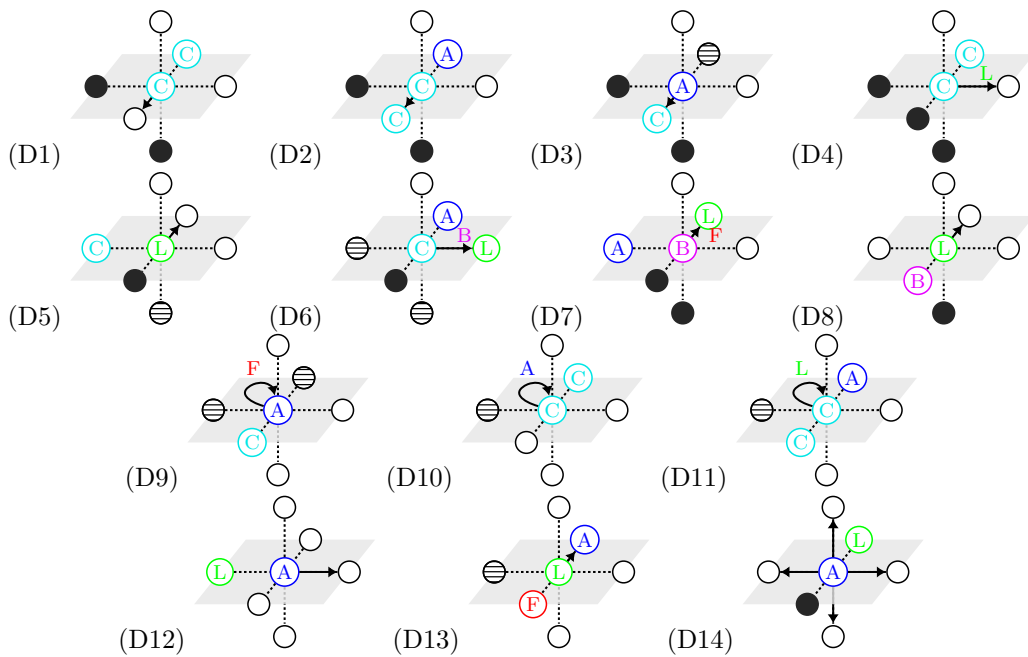
■ **Figure 7** Overview of the journey made by the exploring beedroids.

The 3D grid can be seen as a building with several floors. The overall idea of the proposed algorithm is to make the beedroids explore the 3D grid floor by floor, as illustrated in Figure 7. On a given floor, two beedroids will be in charge of exploring the floor line by line while the third one will be used as a landmark to keep track of the exploration direction: it will either designate the next line or the next floor to explore. Thus, the algorithm defines three main roles for the beedroids using colors: *Leader* (L), *Follower* (F) and *Landmark*. We use three different landmarks A, B, and C to distinguish different situations. Notice that a beedroid can change its role several times during the execution.

Initially, beedroids respectively have colors C, C, and A, as shown in Figure 6. They are aligned and one beedroid with color C should be adjacent to the two others. This pattern can be arbitrary placed on the 3D grid. In that sense, the set of all possible initial configurations is locally-defined [6]. Starting from any such locally-defined initial configuration, beedroids first move towards a wall. If they are aligned along an edge of the 3D grid, they do so while keeping their respective color. Otherwise, they first switch to the color sequence A, L, F. Once a wall is reached, beedroids coordinate together to reach a particular kind of configurations, denoted by C_p in the following, which will correspond to the effective start of the exploration. In other word, the initial prefix leading to a configuration C_p occurs only once. Then, the system periodically goes through Configurations C_p and all nodes are visited between two occurrences of them in the execution.

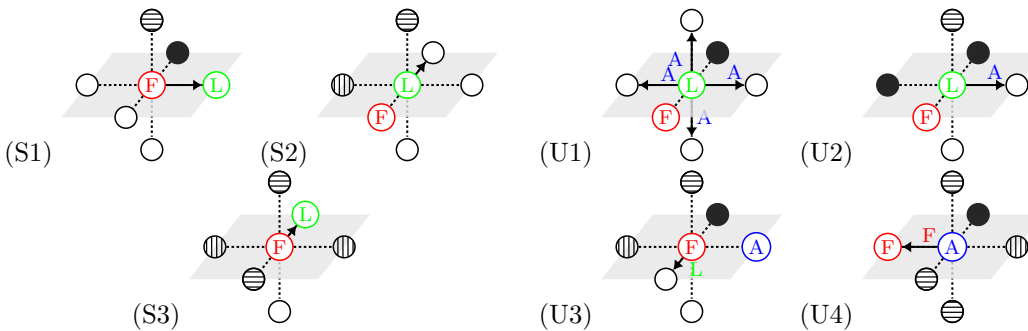
A configuration is of type C_p if the beedroids are located on two adjacent lines ℓ_i and ℓ_{i+1} of the 3D grid such that ℓ_i hosts two adjacent beedroids colored F and L respectively at distance 2 and 3 from the same wall \mathcal{W} ; and ℓ_{i+1} hosts a single beedroid that is colored A and adjacent to \mathcal{W} . The rules to reach a configuration C_p from a locally-defined initial configuration are given in Figure 8.

As explained before, from Configuration C_p , the beedroids will perform a periodic exploring journey around the 3D grid visiting all nodes floor by floor. As each floor of the 3D grid is a finite 2D grid, the strategy used to explore a given floor is similar to the one of [19], *i.e.*, two beedroids move and explore a given line while the third one remains idle next to a wall to indicate the next line to explore. More precisely, let ℓ_i be the current line of the floor f_i being explored. The leader moves away from the follower and the follower just follows the leader using the rules given in Figure 9. At the beginning, both the leader and the follower move away from the landmark which has color A, and move along the nodes of ℓ_i until reaching a wall. When the leader sees the wall, it does not see the landmark (since the landmark was left on the opposite wall), then the leader and the follower exchange their respective roles and move back along the same line ℓ_i . This role exchange is done in two rounds: first, the leader moves to one of its adjacent nodes changing its color to A to notify the follower that they have to change their role. As the follower does not sense the wall yet,



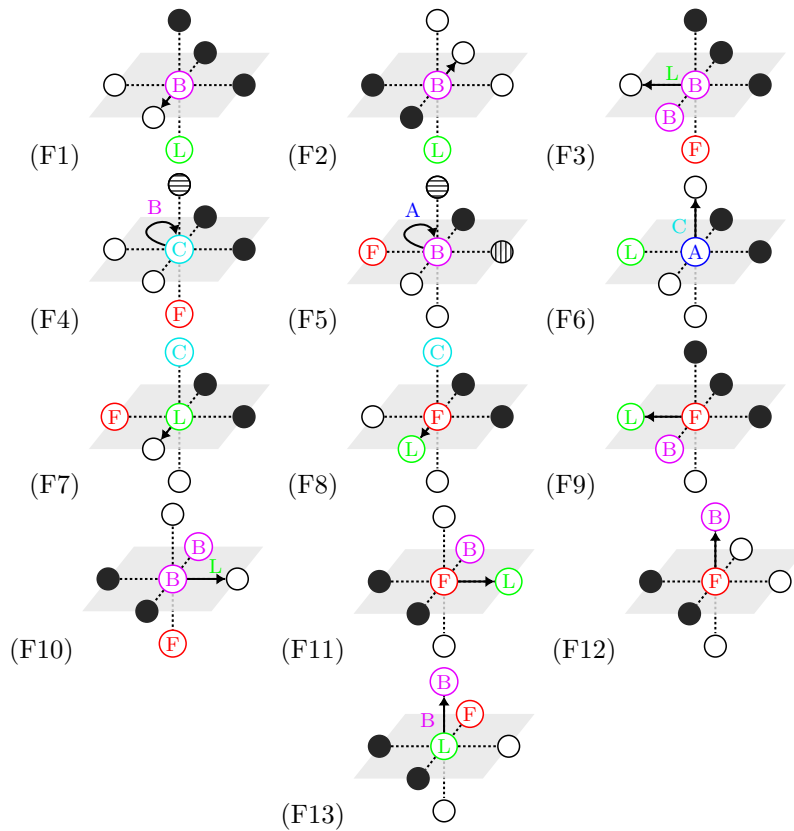
■ **Figure 8** Rules executed to reach a configuration C_p from a locally-defined configuration. Colored letters inside nodes indicate the color of the bedroids occupying the nodes. The arrow indicates the destination and when a colored letter is given next to an arrow, this means that the rule dictates the bedroid to switch to that color. Finally, self-loops indicate when a bedroid stays idle.

it continues to follow the leader and hence it becomes neighbor to a wall. Next, by observing a bedroid with color A, the follower moves back to its previous position and changes its color to the leader’s one L, while the ex-leader, the bedroid with color A, starts following the new leader and updates its color to become a follower. This u-turn is done by executing the rules of Figure 10.

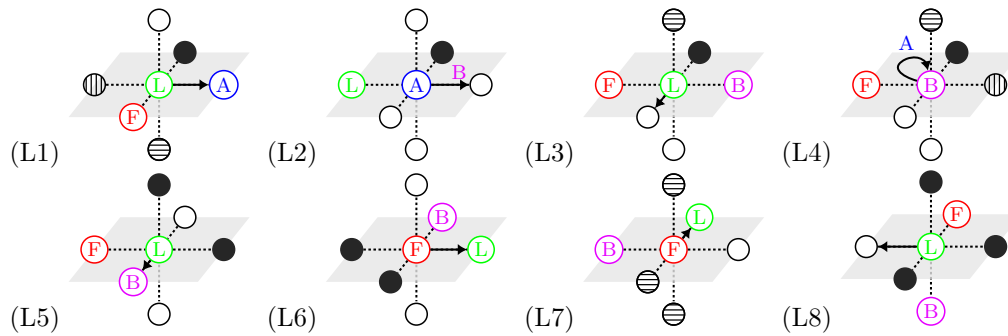


■ **Figure 9** Moving in a straight line. ■ **Figure 10** U-turn.

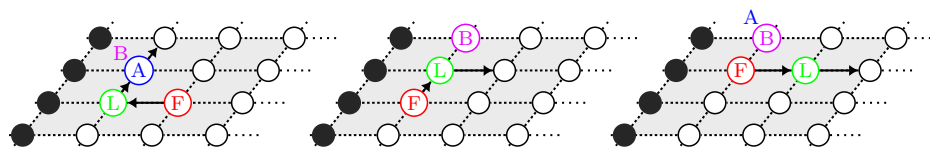
As the bedroids have switched their roles, they proceed again at the exploration of ℓ_i but this time, in the reverse direction. When the leader reaches the opposite wall, it sees this time the landmark and hence knows that the current line ℓ_i has been fully explored. Let ℓ_{i+1} be the line that hosts the landmark. Line ℓ_{i+1} is the next line to be explored. For this purpose, both the leader and the follower need to move to line ℓ_{i+1} while the landmark moves to another line ℓ_{i+2} , the line to visit after ℓ_{i+1} . This line switch is done in three rounds by



■ **Figure 11** Moving to the next floor.

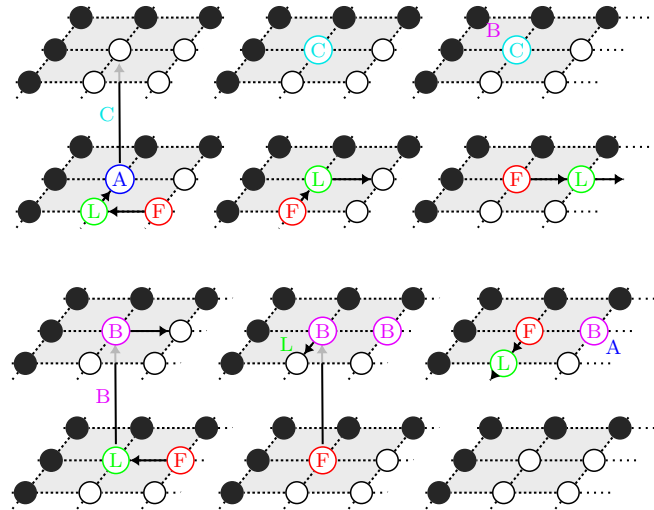


■ **Figure 12** Moving to the next line.

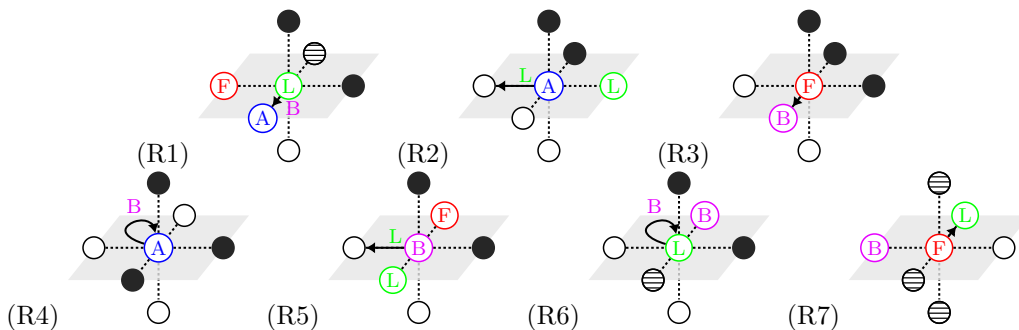


■ **Figure 13** Sequence of configurations during a line change on a floor.

executing the rules of Figure 12. Figure 13 illustrates the sequence of configurations reached during this latter process. The leader and the follower then simply proceed at the exploration of line l_{i+1} in the same manner as line l_i .



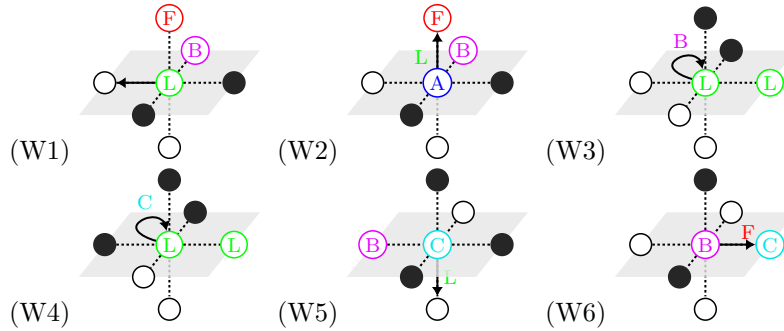
■ **Figure 14** Sequence of configurations during a change of floor. The three first configurations replace the line change that cannot occur because the beedroid A is in a corner, then the moving group explore the last line of the floor. The three last configurations occurs when the moving group comes back.



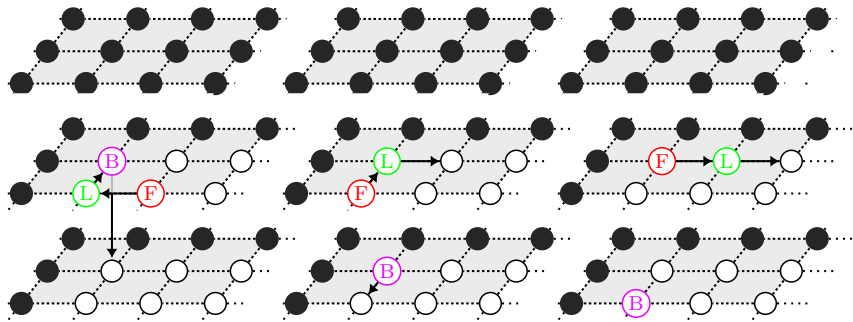
■ **Figure 15** Changing line on a roof.

In the case where l_{i+1} is the last line to be explored on f_i , the landmark moves to the next floor f_{i+1} to be explored when the leader moves to l_{i+1} . Note that f_{i+1} is determined thanks to chirality. Indeed, as the landmark is at a corner and sense the leader at one side, it can identify the upper floor from the lower floor. Both the leader and the follower then proceed in the same manner as previously. That is, they explore the nodes of l_{i+1} , exchange their roles and then move back on l_{i+1} until they reach the wall again. When the leader and the follower reach the wall after exploring l_{i+1} , they also move to Floor f_{i+1} indicated by the landmark. This is done by executing the rules of Figure 11. The beedroids will repeat the same process to explore Floor f_{i+1} . Note that in order to keep the same exploration direction, the landmark moves to the line such that the leader remains on the same side. This direction is again chosen using the chirality (recall that the beedroid is at the corner

and senses a beedroid at Floor f_i). Figure 14 shows the sequence of configurations of this floor change.



■ **Figure 16** Exploration direction switching.



■ **Figure 17** Sequence of configurations occurring when the landmark initiates a change of the exploration direction.

By doing so, the beedroids eventually explore each floor until they reach the last floor, called a roof in the sequel. This latter is explored similarly except that the beedroids update their respective color differently at line changes using the rules shown in Figure 15. This is done to ensure the beedroids remain on the roof without using additional colors.

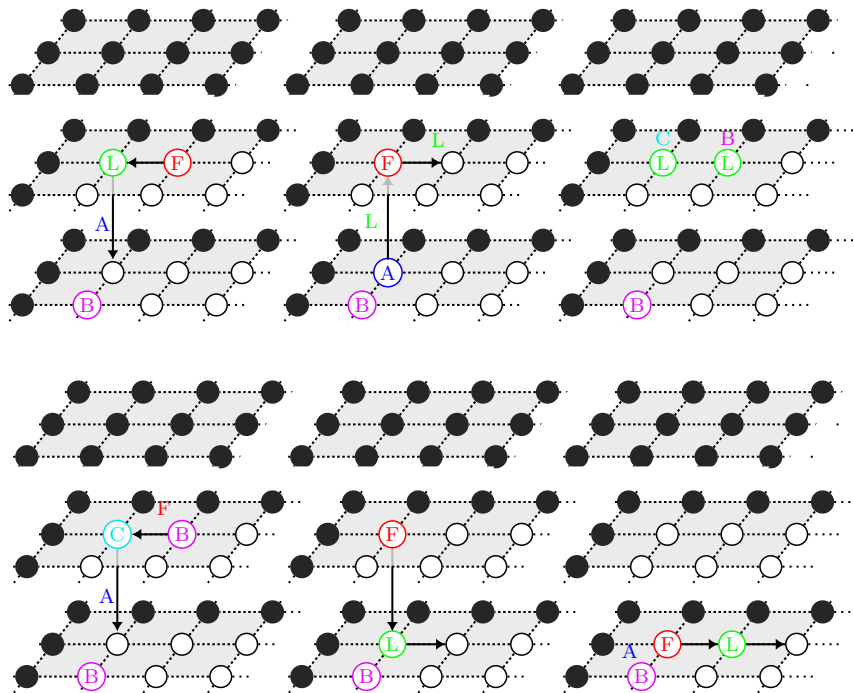
The beedroids then change the exploring direction to explore the 3D grid in the reverse direction. This is done by executing the rules of Figure 16. Figure 17 presents a sequence of configurations occurring when the landmark initiates a change of the exploration direction. Figure 18 presents the sequence of configurations occurring at the termination of this process, *i.e.*, when the leader and the follower come back from exploring the last line of the roof.

We have validated several base cases using our simulation tool [3]. Then, we have generalized the reasoning by induction to show that $Vone_5^3$ solves the PFPP.

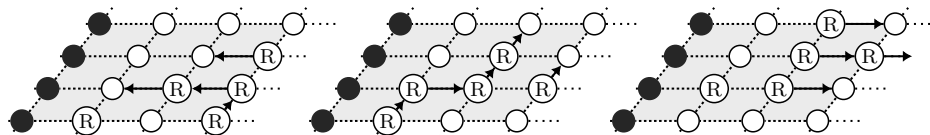
► **Theorem 8.** *Under visibility range one, $Vone_5^3$ solves the perpetual flower pollination problem with three beedroids endowed with five colors.*

5 Visibility range two: $Vtwo_1^5$

In this section, we present an algorithm that uses five oblivious beedroids to solve the PFPP. The beedroids are oblivious in the sense that the lights of all the beedroids have the same color and cannot change, so they do not have any bit of persistent memory.

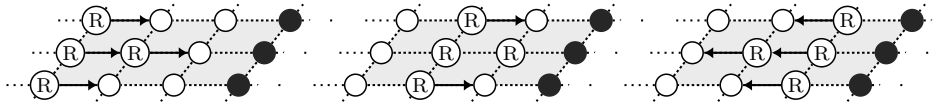


■ **Figure 18** Sequence of configurations occurring when the leader and follower come back to complete the exploration direction change.



■ **Figure 19** In the initial configuration, bedroids should be near a wall, as in the leftmost configuration. The bedroids then start a line change.

This algorithm works with a specific set of initial configurations that are not locally-defined. The initial configurations are those where the bedroids are close to a wall, like in the first configuration of Figure 19 but they must not be all adjacent to the same wall. Then, the principle is similar to $Vone_3^5$, *i.e.*, bedroids explore the 3D grid floor by floor. Within a given floor, bedroids are able to fly in straight line when they form a \perp pattern, called *moving group*, composed of four bedroids. Initially, the moving group makes a U-turn while changing line (Figure 19). Then, they explore one line leaving a landmark bedroid adjacent to the wall. Upon reaching the opposite wall, they make a U-turn again and explore the same line again (Figure 20). When they meet the landmark again, bedroids are in the same configuration as initially, translated by one node.

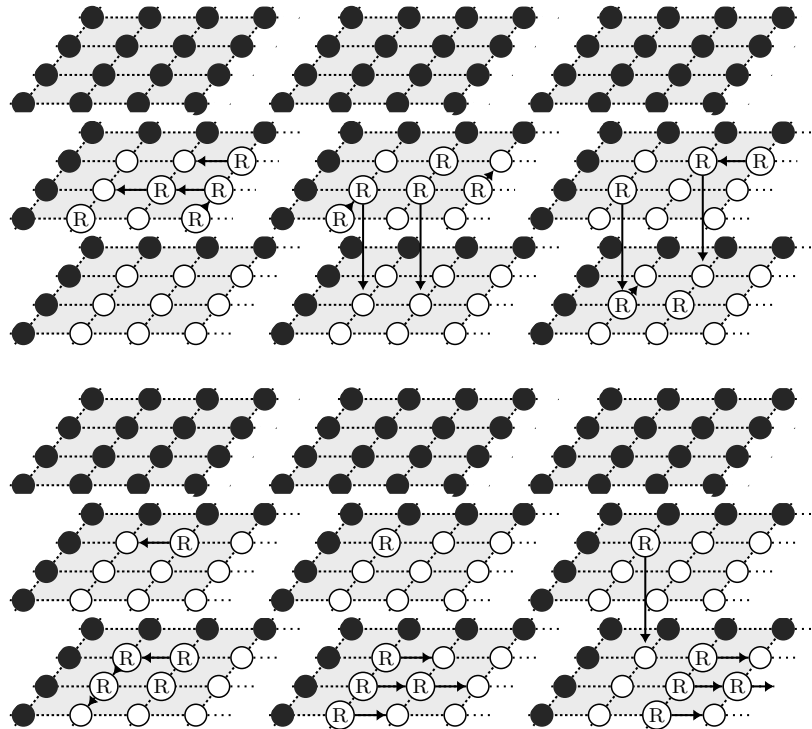


■ **Figure 20** Sequence of configuration during a U-turn.

Once they reach the corner, they move one floor above and explore this floor with the same pattern, rotated by $\frac{\pi}{2}$ (refer to Figure 22). Eventually, the beedroids finish exploring the roof and then switch the exploring direction to start exploring the floors of 3D grid in the reverse sense (refer to Figure 21). The animation illustrating beedroids' behavior is available online [4].

► **Theorem 9.** *Under visibility range two, $\forall \text{two}_1^5$ solves the perpetual flower pollination problem with five oblivious beedroids.*

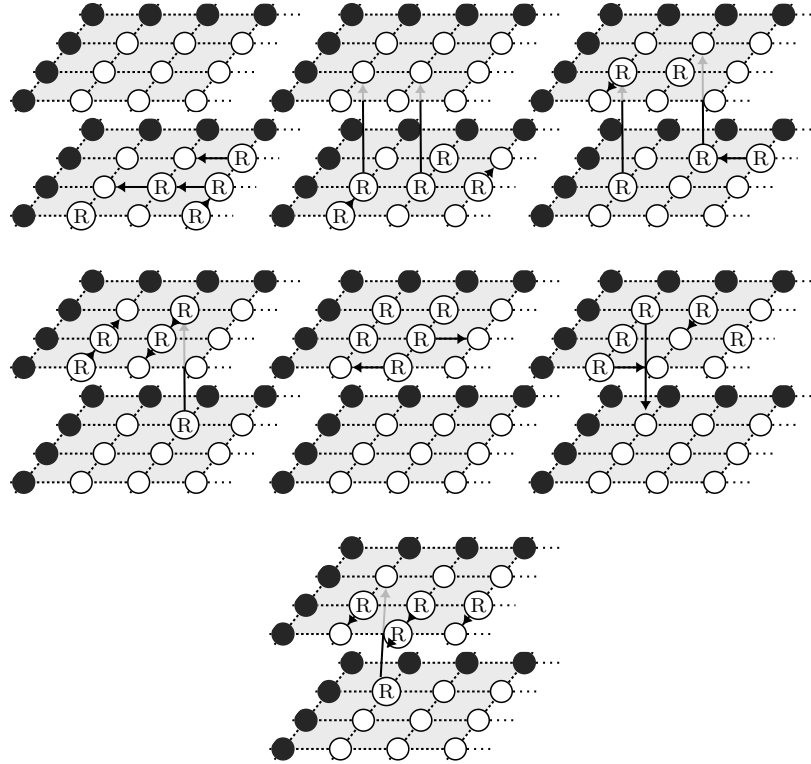
The proof of Theorem 9 is an induction similar to the one of Theorem 8.



■ **Figure 21** Sequence of configurations when beedroids switch the exploring direction.

6 Related Work

The problem of perpetual flower pollination by a beedroid swarm is actually known in the literature as the *perpetual exploration* of a 3D grid by a swarm of *luminous robots* [18]. Exploration of discrete environment by a robot swarm has been widely studied. Various topologies have been already considered including lines [15], rings [1, 8, 11, 16, 17], trees [14],



■ **Figure 22** Sequence of configurations when beedroids move to the next floor.

torus [10], finite [2, 6, 9, 19], infinite 2D grids [5, 7], and even infinite n -dimensional grids [13]. (In the infinite case, the exploration problem requires that each node is visited within finite time by at least one robot.) In the context of finite graphs, two main variants of the exploration problem have been studied: the *terminating* and *perpetual* exploration. The terminating exploration requires every possible location to be eventually visited by at least one robot, with the additional constraint that all robots stop moving after task completion. In contrast, the perpetual exploration requires each location to be visited infinitely often by all or a part of robots. Terminating exploration has been tackled in [8, 9, 10, 11, 14, 15, 16], while [1, 2, 6, 19] deal with the perpetual exploration problem. Notice that Ooshita and Tixeuil consider the two variants of the problem in [17]. In contrast with the present paper, a large part of the literature is devoted to “non-myopic” robots, *i.e.*, robots with an unbounded visibility range, meaning that the snapshot of each robot captures in the whole system configuration; see [1, 2, 9, 10, 11, 14, 15, 16]. In such a context, robots are always assumed to be anonymous and oblivious, *i.e.*, they have no state and cannot remember the past. Furthermore, chirality has never been considered under such settings. Exploration algorithms satisfying exclusiveness are proposed in both finite [1, 2, 6, 19] and infinite graphs [5, 7]. Assuming a common chirality is pretty usual in the 2D Euclidean plan; see *e.g.*, [12]. However, up to now only a few works dedicated to discrete environments, *e.g.*, infinite [7] and finite [6] 2D grids, assume robots have a common chirality. Now, the common chirality has an impact on the number of robots necessary to solve exploration: for example, with visibility range one and three colors, two (resp. three) synchronous robots are necessary and sufficient to explore a finite 2D grid with (resp. without) the common chirality assumption [6, 19]. To the best of

our knowledge, perpetual exploration has been never addressed in finite 3D grids. However, the exploration of an infinite n -dimensional grid has been investigated in [13]. In that paper, authors consider robots operating in two models: the semi-synchronous and synchronous ones. However, they do not impose the exclusivity at all since their robots can only sense the states of the robots located at the same node (in that sense, the visibility range is zero). Moreover, in contrast with our work, they assume all robots agree on a *global compass*, *i.e.*, they all agree on the same directions North-South and East-West. They propose several solutions and bounds, in particular they show that, in the semi-synchronous model, four deterministic robots are necessary and sufficient to explore an infinite 3D grid.

7 Conclusion

We have studied how typically small swarms of chiral luminous beedroids can solve the perpetual flower pollination problem in 3D grids assuming the FSYNC model. Under the optimal visibility range one, we have shown that three beedroids are necessary and sufficient to solve the problem. For the sufficient part, we have proposed an algorithm that requires only five colors. Then, we have proposed another solution that is optimal in terms of colors: an algorithm working with five oblivious beedroids under visibility range two.

However, our industrial partners are still not fully satisfied by our proposal. Even if our solutions require a very few number of weak beedroids, they believe that we can still achieve some economies of scale. Like the character Peter Isherwell in the movie “*Don’t look up*”, they want to both save humanity and win money... So, we have to study whether we can reduce the number of colors used by the first algorithm. We should also study whether the number of beedroids and the visibility range of the second algorithm can be decreased. For this latter, we are pessimistic: we conjecture that the visibility range cannot be lowered to one in the oblivious case. Our idea is that skills necessary to solve the perpetual flower pollination problem in 3D grids with chiral oblivious beedroids are similar to those necessary to solve the 2D grid exploration problem with non-chiral oblivious beedroids. So, we expect that the impossibility proof given in [7] can be adapted to the context of chiral oblivious beedroids evolving in a 3D grid.




References

- 1 Lélia Blin, Alessia Milani, Maria Potop-Butucaru, and Sébastien Tixeuil. Exclusive perpetual ring exploration without chirality. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, volume 6343 of *Lecture Notes in Computer Science*, pages 312–327, Boston, Massachusetts, USA, September 2010. Springer.
- 2 François Bonnet, Alessia Milani, Maria Potop-Butucaru, and Sébastien Tixeuil. Asynchronous exclusive perpetual grid exploration without sense of direction. In Antonio Fernández Anta, editor, *Proceedings of International Conference on Principles of Distributed Systems (OPODIS 2011)*, number 7109 in *Lecture Notes in Computer Science (LNCS)*, pages 251–265, Toulouse, France, December 2011. Springer Berlin / Heidelberg. URL: <http://www.springerlink.com/content/913v424157681707/>.
- 3 Quentin Bramas. Animation of the first algorithm, 2022. URL: <https://bramas.pages.unistra.fr/robot-grid-exploration-simulator/?/robot-grid-exploration-simulator/algo/finite-grid/chirality/range-1/3-robots-5-colors.web-algo>.
- 4 Quentin Bramas. Animation of the second algorithm, 2022. URL: <https://bramas.pages.unistra.fr/robot-grid-exploration-simulator/?/robot-grid-exploration-simulator/algo/finite-grid/chirality/range-2/algo-5-robots-oblivious.web-algo>.

- 5 Quentin Bramas, Stéphane Devismes, and Pascal Lafourcade. Infinite grid exploration by disoriented robots. In Chryssis Georgiou and Rupak Majumdar, editors, *Networked Systems - 8th International Conference, NETYS 2020, Marrakech, Morocco, June 3-5, 2020, Proceedings*, volume 12129 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2020. doi:10.1007/978-3-030-67087-0_9.
- 6 Quentin Bramas, Stéphane Devismes, and Pascal Lafourcade. Optimal Exclusive Perpetual Grid Exploration by Luminous Myopic Opaque Robots with Common Chirality. In *ICDCN'21: International Conference on Distributed Computing and Networking, Virtual Event*, pages 76–85, Nara, Japan, 5-8 January 2021. ACM.
- 7 Quentin Bramas, Pascal Lafourcade, and Stéphane Devismes. Finding water on poleless using melomaniac myopic chameleon robots. In Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara, editors, *10th International Conference on Fun with Algorithms, FUN 2021, May 30 to June 1, 2021, Favignana Island, Sicily, Italy*, volume 157 of *LIPICs*, pages 6:1–6:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 8 Ajoy Kumar Datta, Anissa Lamani, Lawrence L. Larmore, and Franck Petit. Enabling ring exploration with myopic oblivious robots. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop, IPDPS 2015*, pages 490–499, Hyderabad, India, May 25-29, 2015 2015. IEEE Computer Society.
- 9 Stéphane Devismes, Anissa Lamani, Franck Petit, Pascal Raymond, and Sébastien Tixeuil. Terminating exploration of A grid by an optimal number of asynchronous oblivious robots. *Comput. J.*, 64(1):132–154, 2021. doi:10.1093/comjnl/bzz166.
- 10 Stéphane Devismes, Anissa Lamani, Franck Petit, and Sébastien Tixeuil. Optimal torus exploration by oblivious robots. *Computing*, 101(9):1241–1264, 2019.
- 11 Stéphane Devismes, Franck Petit, and Sébastien Tixeuil. Optimal probabilistic ring exploration by semi-synchronous oblivious robots. *Theoretical Computer Science (TCS)*, 498:10–27, 2013.
- 12 Yoann Dieudonné, Franck Petit, and Vincent Villain. Leader election problem versus pattern formation problem. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing, 24th International Symposium, DISC 2010*, volume 6343 of *Lecture Notes in Computer Science*, pages 267–281, Cambridge, MA, USA, september 13-15 2010. Springer.
- 13 Stefan Dobrev, Lata Narayanan, Jaroslav Opatrny, and Denis Pankratov. Exploration of high-dimensional grids by finite automata. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPICs*, pages 139:1–139:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 14 Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. *Theor. Comput. Sci.*, 411(14-15):1583–1598, 2010. doi:10.1016/j.tcs.2010.01.007.
- 15 Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. How many oblivious robots can explore a line. *Inf. Process. Lett.*, 111(20):1027–1031, 2011. doi:10.1016/j.ip1.2011.07.018.
- 16 Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. *Algorithmica*, 65(3):562–583, 2013.
- 17 Fukuhito Ooshita and Sébastien Tixeuil. Ring exploration with myopic luminous robots. In Taisuke Izumi and Petr Kuznetsov, editors, *Stabilization, Safety, and Security of Distributed Systems - 20th International Symposium, SSS 2018*, volume 11201 of *Lecture Notes in Computer Science*, pages 301–316, Tokyo, Japan, november 4-7 2018. Springer.
- 18 David Peleg. Distributed coordination algorithms for mobile robot swarms: New directions and challenges. In Ajit Pal, Ajay D. Kshemkalyani, Rajeev Kumar, and Arobinda Gupta, editors, *Distributed Computing - IWDC 2005*, pages 1–12, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

- 19 Arthur Rauch, Quentin Bramas, Stéphane Devismes, Pascal Lafourcade, and Anissa Lamani. Optimal exclusive perpetual grid exploration by luminous myopic robots without common chirality. In Karima Echihabi and Roland Meyer, editors, *Networked Systems - 9th International Conference, NETYS 2021, Virtual Event, May 19-21, 2021, Proceedings*, volume 12754 of *Lecture Notes in Computer Science*, pages 95–110. Springer, 2021.
- 20 Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999.

Priority Queues with Decreasing Keys

Gerth Stølting Brodal   

Department of Computer Science, Aarhus University, Denmark

Abstract

A priority queue stores a set of items with associated keys and supports the insertion of a new item and extraction of an item with minimum key. In applications like Dijkstra’s single source shortest path algorithm and Prim-Jarník’s minimum spanning tree algorithm, the key of an item can decrease over time. Usually this is handled by either using a priority queue supporting the deletion of an arbitrary item or a dedicated `DecreaseKey` operation, or by inserting the same item multiple times but with decreasing keys.

In this paper we study what happens if the keys associated with items in a priority queue can decrease over time *without* informing the priority queue, and how such a priority queue can be used in Dijkstra’s algorithm. We show that binary heaps with bottom-up insertions fail to report items with unchanged keys in correct order, while binary heaps with top-down insertions report items with unchanged keys in correct order. Furthermore, we show that skew heaps, leftist heaps, and priority queues based on linking roots of heap-ordered trees, like pairing heaps, binomial queues and Fibonacci heaps, work correctly with decreasing keys without any modifications. Finally, we show that the post-order heap by Harvey and Zatloukal, a variant of a binary heap with amortized constant time insertions and amortized logarithmic time deletions, works correctly with decreasing keys and is a strong contender for an implicit priority queue supporting decreasing keys in practice.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases priority queue, decreasing keys, post-order heap, Dijkstra’s algorithm

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.8

Supplementary Material *Software*: <https://www.cs.au.dk/~gerth/papers/fun22code.zip>

Funding Supported by Independent Research Fund Denmark, grant 9131-00113B.

Acknowledgements The author wants to thank Rolf Fagerberg for insightful discussions.

1 Introduction

A priority queue is a data structure storing a set of items, where each item has an associated key. A basic priority queue supports the two operations `Insert` and `ExtractMin`, which insert a new item into the priority queue and extract an item with minimum key from the priority queue. A classic example of a data structure supporting these operations is the binary heap by Williams from 1964 [20]. Although many priority queues exist supporting a more comprehensive list of operations or having better asymptotic bounds, the binary heap is the standard priority queue implementation in many languages, like in Python (module `heapq`) and Java (class `java.util.PriorityQueue`). The popularity of binary heaps is due to its simplicity, it can be stored implicitly in an (extendable) array only storing the n items currently in the heap, and the number of comparisons is relatively low. Insertions require at most $\log_2 n$ comparisons, and minimum extractions at most $2 \log_2 n$ comparisons, but the number of comparisons performed are often lower in practice.

Two graph algorithms fundamentally relying on efficient priority queue implementations are Dijkstra’s algorithm [5] for finding shortest paths from a source node in directed graphs with non-negative edge weights, and Prim-Jarník’s algorithm [11, 16] for finding the minimum spanning tree in a graph. Both maintain a priority queue over the nodes not included yet in the shortest path tree and minimum tree, respectively. A node in the priority queue has an



© Gerth Stølting Brodal;

licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 8; pp. 8:1–8:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

associated key equal to the shortest distance to the node discovered so far and the lightest edge connecting the node to the minimum spanning tree constructed so far, respectively. For both algorithms the key of a node in the priority queue can decrease over time, which challenges the interface of the basic priority queue. One solution is to apply a more specialized priority queue, like Fibonacci heaps [9], which support a dedicated `DecreaseKey` operation – but this structure is more complicated, pointer based, and often not part of a standard library. Although theoretically worst-case superior, the overhead of supporting `DecreaseKey` only pays off when a large fraction of the edges cause a `DecreaseKey` operation to be performed. A simpler solution is to stay with a basic priority queue and just insert a node multiple times, once whenever the key decreases. This leaves outdated copies of nodes in the priority queue, but these can be skipped whenever they are extracted from the priority, since only the first extraction of a node is not outdated. Two possible implementations of this idea for Dijkstra’s algorithm are shown as algorithms `Dijkstra3` and `Dijkstra4` in Figure 1. In the following we do not discuss Prim-Jarník’s algorithm any further.

In a typical implementation of Dijkstra’s algorithm one maintains an array $dist$, where $dist[v]$ is the currently shortest known distance from the source node to node v , and the items inserted into the priority queue are pairs $\langle dist[v], v \rangle$, where $dist[v]$ is the key of the item. In this paper we consider adopting the idea of only storing v as an item in the priority queue *without* an explicitly associated key. The comparison between two items in the priority instead compares the current distances $dist[v]$. This will reduce the space usage for the priority queue, e.g., a binary heap only needs to store an array of node ids. The challenge is now that keys of inserted items can decrease over time, i.e., the ordering of the items in the priority queue changes over time and potentially invalidates the internal invariants maintained by a priority queue. In this paper we identify comparison based priority queues working correctly with decreasing keys. In particular we show that skew heaps [18], leftist heaps [4], binomial queues [19], pairing heaps [8], Fibonacci heaps [9], and post-order heaps [10] work correctly even with decreasing keys. For binary heaps [20] we show that the standard implementation with bottom-up insertions fails to support decreasing keys, whereas binary heaps work correctly if operations are performed top-down.

Model

We define a *priority queue with decreasing keys* as follows. It stores a set of items where each item has an associated key from some totally ordered universe. Over time the key of an item can decrease an arbitrary number of times. We let the *original key* of an item refer to the key when the item was inserted, whereas the *current key* refers to the key at the current time. If the current key equals the original key we say that the item has an *unchanged* key. The priority queue is not informed when keys decrease, and whenever two items are compared by the priority queue the comparison is performed with respect to their current keys. The priority queue has no access to the original keys of the items; it can only compare two items and get the relative order of their current keys. Note that the answer to the comparison between two items can vary over time depending on how keys decrease.

A priority queue with decreasing keys should support the following two operations:

- `Insert(x)` inserts an item into the priority queue.
- `ExtractMin()` returns an item from the priority queue with *current key* less than or equal to the *original keys* of all items in the priority queue.

It follows that if `ExtractMin` returns an item with unchanged key, the item has smallest key among all items with unchanged key. Furthermore, if several items in the priority queue have current key less than or equal to the smallest original key in the priority queue, then the

priority queue is allowed to return any of these items, i.e. its behavior is non-deterministic. As an example, consider a priority queue with four inserted items A , B , C and D with original keys 5, 2, 6 and 4, respectively. Assume C and D have had their keys decreased to have current keys 3 and 1, respectively. Then `ExtractMin` should return either B or D , with current keys 2 and 1, respectively, since B has smallest original key equal to 2, and A and C have current keys 5 and 3, respectively. In Section 2 we discuss how an implementation of Dijkstra’s algorithm can benefit from priority queues with decreasing keys.

Contributions

This paper introduces no new data structure. Only existing data structures are analyzed in the context of decreasing keys. Our contributions are:

- Section 2: We show that Dijkstra’s algorithm [5] (`Dijkstra4` in Figure 1) works correctly when using a priority queue with decreasing keys, i.e., items in the priority queue only store a node v instead of the pair $\langle dist[v], v \rangle$.
- Section 3: Binary heaps [20] with bottom-up insertions do not support decreasing keys, in particular we show that sorting (with interleaved key decreases) and Dijkstra’s algorithm fail on small examples.
- Section 4: Binary heaps with top-down insertions support decreasing keys, i.e., inserting a new item considers the ancestors of the new leaf top-down until the first ancestor is found with key greater than or equal to the new key. The central invariant used in the analysis, and also used in Sections 5 and 6, is *decreased heap order* requiring that that any ancestor of a node v in a tree must store an item with current key less than or equal to the original key of v .

Without decreasing keys, bottom-up and top-down insertions cause the nodes of the resulting heaps to store identical keys, but for random insertions the number of comparisons increases from average $O(1)$ [15] to $\Theta(\log n)$. In Section 7 we do an experimental comparison of the two variants of a binary heap, and in particular the overhead introduced by performing insertions top-down.

- Section 5: Skew heaps [18], leftist heaps [4], pairing heaps [8], binomial queues [19], and Fibonacci heaps [9] work correctly with decreasing keys.
- Section 6: The post-order heap by Harvey and Zatloukal [10] supports decreasing keys. The post-order heap is a simple implicit heap based on binary heaps that supports insertions in amortized constant time and extractions in amortized logarithmic time (like, e.g., binomial queues). In Section 7 our experimental evaluation shows that the post-order heap is a strong contender for an efficient implicit priority queue supporting decreasing keys.
- Section 7: We supplement our theoretical results with an experimental evaluation of priority queues supporting decreasing keys and compare the number of key comparisons performed to sort and for running Dijkstra’s algorithm on cliques.

Related work

The literature on priority queues is comprehensive, see, e.g., the survey by Brodal [1]. Fibonacci heaps [9] support `DecreaseKey` in amortized constant time and their discovery initiated the study of data structures supporting efficient `DecreaseKey` operations. Subsequently, e.g., relaxed heaps [6] were introduced, which support `DecreaseKey` in worst-case constant time. In this paper we focus on simpler data structures, not supporting `DecreaseKey` operations. Many priority queues can be extended to support an arbitrary `Remove` operation,

by having a separate index keeping track of where each item is stored in the data structure. This introduces a space overhead for the index and a time overhead to keep the index updated, e.g., swapping two items in a binary heap requires two entries in the index to be updated. If only `Insert` and `ExtractMin` need to be supported, many simple constructions exist – a few commonly used are mentioned and evaluated in this paper. A special interesting class of priority queues are those that can be stored in a single array containing the items, known as *implicit* priority queues. The classic example is the binary heap [20], but other examples are, e.g., the implicit binomial trees by Carlsson, Munro and Poblette [2], and the post-order heap by Harvey and Zatloukal [10] that is our focus in Section 6. Many priority queues maintain a forest of trees of sizes corresponding to digits in the binary or skew binary representation of the total number of items stored. Elmasry, Jensen and Katajainen [7] give an overview of this relationship for various constructions.

Background

The motivation for studying priority queues with decreasing keys arose from experience with undergraduate students having problems translating Dijkstra’s shortest path algorithm into correct Java programs based on the description in the standard text book by Cormen *et al.* [3, Section 24.3]. Students are challenged by the fact that the priority queue implementation supported by the Java standard library¹ does not support `DecreaseKey`, the `Remove` operation requires linear time, and the ordering of values is provided using a comparator (or the natural ordering of the values). A priority queue with decreasing keys allows Dijkstra’s algorithm to store node identifiers only in the priority and using a comparator to compare two nodes by comparing their currently best known distances – the Java solution many students implement, but fails since their priority queue does not support decreasing keys.

2 Dijkstra’s algorithm with decreasing keys

Assume we are given a directed graph $G = (V, E)$ with non-negative edge weights δ and a source node $s \in V$, and we want to compute the shortest distance from s to all nodes in the graph. This problem can be solved using Dijkstra’s algorithm [5]. The basic idea of Dijkstra’s algorithm is to visit nodes in increasing distance from s . For each node v not visited yet its currently known distance $dist[v]$ is stored in an array $dist$, i.e., the distance to v along paths only containing v and already visited nodes. The next node to visit is a node u not visited so far and with smallest $dist[u]$ value. When visiting a node u we *relax* along its outgoing edges (u, v) by performing the update $dist[v] := \min(dist[v], dist[u] + \delta(u, v))$. To obtain an efficient solution, the set of nodes not visited yet are stored in a priority queue, with $dist[v]$ as the key of v . Fibonacci heaps [9] provide a dedicated `DecreaseKey` operation to update (decrease) the key of a node whenever the known distance to a node decreases. Using a Fibonacci heap Dijkstra’s algorithm can be implemented as shown in `Dijkstra1` in Figure 1 obtaining running time $O(|E| + |V| \log |V|)$. If no `DecreaseKey` is available, but an arbitrary item can be removed by a `Remove` operation, we can simulate `DecreaseKey` by first removing the node using `Remove` and then reinserting the node with its smaller distance as key using `Insert` as shown in `Dijkstra2` in Figure 1. If `Remove` takes logarithmic time, the resulting running time is $O(|E| \log |V|)$. For sparse graphs, i.e., $|E| = O(|V|)$, the two running times are asymptotically identical.

¹ <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/PriorityQueue.html>

```

proc Dijkstra1( $V, E, \delta, s$ )
   $dist[v] = +\infty$  for all  $v \in V \setminus \{s\}$ 
   $dist[s] = 0$ 
  Insert( $Q, \langle dist[s], s \rangle$ )
  while  $Q \neq \emptyset$  do
     $\langle d, u \rangle = \text{ExtractMin}(Q)$ 
    for  $(u, v) \in E \cap (\{u\} \times V)$  do
      if  $dist[u] + \delta(u, v) < dist[v]$  then
         $dist[v] = dist[u] + \delta(u, v)$ 
      if  $v \in Q$  then
        DecreaseKey( $Q, v, dist[v]$ )
      else
        Insert( $Q, \langle v, dist[v] \rangle$ )
  return  $dist$ 

proc Dijkstra2( $V, E, \delta, s$ )
   $dist[v] = +\infty$  for all  $v \in V \setminus \{s\}$ 
   $dist[s] = 0$ 
  Insert( $Q, \langle dist[s], s \rangle$ )
  while  $Q \neq \emptyset$  do
     $\langle d, u \rangle = \text{ExtractMin}(Q)$ 
    for  $(u, v) \in E \cap (\{u\} \times V)$  do
      if  $dist[u] + \delta(u, v) < dist[v]$  then
         $dist[v] = dist[u] + \delta(u, v)$ 
      if  $v \in Q$  then
        Remove( $Q, v$ )
      Insert( $Q, \langle dist[v], v \rangle$ )
  return  $dist$ 

proc Dijkstra3( $V, E, \delta, s$ )
   $dist[v] = +\infty$  for all  $v \in V \setminus \{s\}$ 
   $dist[s] = 0$ 
  Insert( $Q, \langle dist[s], s \rangle$ )
  while  $Q \neq \emptyset$  do
     $\langle d, u \rangle = \text{ExtractMin}(Q)$ 
    if  $d = dist[u]$  then
      for  $(u, v) \in E \cap (\{u\} \times V)$  do
        if  $dist[u] + \delta(u, v) < dist[v]$  then
           $dist[v] = dist[u] + \delta(u, v)$ 
          Insert( $Q, \langle dist[v], v \rangle$ )
  return  $dist$ 

proc Dijkstra4( $V, E, \delta, s$ )
   $dist[v] = +\infty$  for all  $v \in V \setminus \{s\}$ 
   $dist[s] = 0$ 
   $visited = \emptyset$ 
  Insert( $Q, \langle dist[s], s \rangle$ )
  while  $Q \neq \emptyset$  do
     $\langle d, u \rangle = \text{ExtractMin}(Q)$ 
    if  $u \notin visited$  then
       $visited = visited \cup \{u\}$ 
      for  $(u, v) \in E \cap (\{u\} \times V)$  do
        if  $dist[u] + \delta(u, v) < dist[v]$  then
           $dist[v] = dist[u] + \delta(u, v)$ 
          Insert( $Q, \langle dist[v], v \rangle$ )
  return  $dist$ 

```

■ **Figure 1** Four variations of Dijkstra’s algorithm for the single source shortest path problem on a digraph with nodes V , edges E , edge weights δ , and source node s . The main result of this paper is that Dijkstra₄ still works correctly if we adopt a priority with decreasing keys.

Here we consider a simpler implementation using a binary heap only supporting **Insert** and **ExtractMin**, but also achieving running time $O(|E| \log |V|)$. Whenever a shorter distance is found to a node v , we insert the item $\langle dist[v], v \rangle$ into the heap, i.e., the same node v can be inserted multiple times, but with decreasing keys. All instances of v in the heap, except for the one with key equal to the current $dist[v]$, are outdated and should be ignored/skipped when extracted from the heap. We can identify nodes to be skipped by either comparing the extracted distance with the currently best known distance or by keeping a set of all visited nodes, e.g., a bit-vector. Algorithms Dijkstra₃ and Dijkstra₄ in Figure 1 contain the pseudo code for these implementations of Dijkstra’s algorithm. We will not argue further about the correctness of these variations of Dijkstra’s algorithm (leaving outdated items in the priority queue is also a common approach to external memory algorithms for the single source shortest path problem, see, e.g., [12, Section 4.2]).²

² In the worst-case the priority queue stores $O(|E|)$ items, but this can be reduced to $O(|V|)$ items by rebuilding the heap whenever it contains $> (1 + \varepsilon)|V|$ items, for some constant $\varepsilon > 0$, where all outdated items are removed. The time for rebuilding the heap can be charged to the removed items, i.e., the asymptotic running time remains unchanged.

Crucial to the above implementations of Dijkstra’s algorithm is that each item we insert into the heap is a pair $\langle dist[v], v \rangle$, where the key $dist[v]$ is fixed when inserted. These keys require space in the heap, that could be tempting to save. The (potentially dangerous) idea is now: *Skip storing the keys explicitly in the items and instead use the current value $dist[v]$ as the current key for all items storing v in the heap.*

In the following we argue that `Dijkstra4` still works correctly if we adopt a priority with decreasing keys, e.g., those in Sections 4–6. The only changes to `Dijkstra4` is that `Insert` should only take the node to insert (without the distance), and `ExtractMin` does not return the key/distance d (that anyway was not used by `Dijkstra4` after the item was extracted), and whenever the priority queue compares the keys of two nodes u and v it compares the currently known distances $dist[u]$ and $dist[v]$.

The invariant maintained by the algorithm is that only nodes v with $dist[v] < +\infty$ are stored in the priority queue, and for all nodes with $dist[v] < +\infty$ and $v \notin visited$, the priority queue contains an item containing v with unchanged key equal to the current $dist[v]$. This is true since we insert an item containing v with original key $dist[v]$ whenever $dist[v]$ decreases.

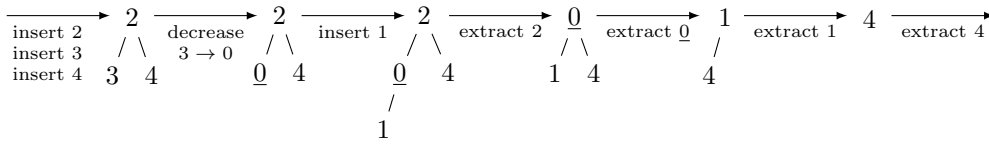
Whenever an item with a node v is extracted from the priority queue, we have three cases: *i*) $v \in visited$, *ii*) $v \notin visited$ and the current key of the item equals the original key, and *iii*) $v \notin visited$ and current key of the item is less than the original key. In case *i*) we extract a node that has already been visited, and therefore should be skipped. In case *ii*) we extract an item with current key equal to its original key. Since the priority queue guarantees that the current key of the extracted item is less than or equal to all original keys stored in the priority queue, the item has minimum original key among all items in the priority queue. From the invariant it follows that v has smallest $dist[v]$ value among all nodes not visited yet – as expected by Dijkstra’s algorithm. Finally, in case *iii*) an item is extracted containing a node v not visited yet and with current key less than its original key. By the priority queue specification, its current key, i.e., $dist[v]$, is less than all original keys in the priority queue, in particular those stored with unchanged keys. Since $v \notin visited$ the invariant implies there must exist another item in the priority queue storing v with original and current key equal to $dist[v]$ – i.e., v is a node not visited yet with minimum $dist$ value as expected by Dijkstra’s algorithm. It follows that a priority queue supporting decreasing keys extracts unvisited nodes in increasing order of distance as required by Dijkstra’s algorithm.

3 Binary heaps fail with decreasing keys

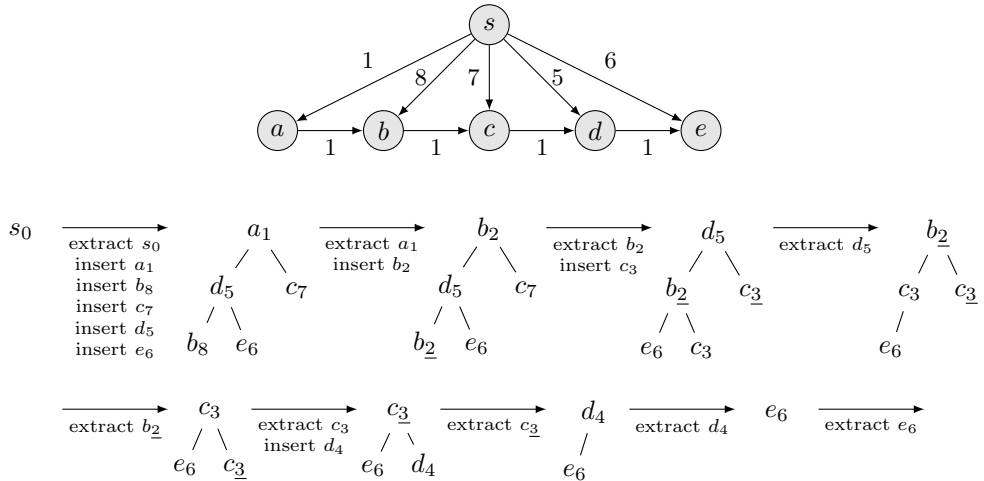
In this section we show how binary heaps with bottom-up insertions [20] fail to support decreasing keys on two simple examples: Sorting and Dijkstra’s single source shortest path algorithm.

We first briefly recall the structure of a binary heap. A binary heap stores n items in an array $H[1..n]$, that can be viewed as a binary tree where node i has children $2i$ and $2i + 1$.³ The items are stored such that *heap order* is satisfied, i.e., the key of item $H[i]$ is greater than or equal to the key of its parent $H[\lfloor i/2 \rfloor]$. A bottom-up insertion places the new item as the last item in H and repeatedly swaps it with its parent (sift-up) as long as its key is less than the parent’s *current* key. A minimum extraction returns the item at the root $H[1]$, and moves the last item x from $H[n]$ to $H[1]$, and sifts-down x by repeatedly swapping x with the item with smallest key among its children until no child stores an item with key less than x .

³ In the paper we assume arrays start at index 1. In our implementation we adapt to Python lists, which start at index 0.



■ **Figure 2** Binary heap failing to sort $\langle 2, 3, 4, 1 \rangle$ if 3 is decreased to 0 before inserting 1.



■ **Figure 3** Execution of Dijkstra’s single source shortest path algorithm (Dijkstra_4), using $\text{dist}[v]$ as keys and a binary heap with bottom-up insertions, incorrectly computing the distance to e as 6. Top is input graph and below the content of the binary heap. Subscripts are keys, and underlined keys are decreased keys.

Sorting

Consider sorting items by inserting them into a priority queue and then extracting them in increasing key order. If an item gets its key decreased during the sequence of operations, a priority queue with decreasing keys guarantees that the items with unchanged keys are still reported in increasing key order. A binary heap with bottom-up insertions fails to do so when inserting four items with keys 2, 3, 4 and 1, and where key 3 is decreased to 0 before inserting 1, as illustrated in Figure 2. Recall that the heap is not informed when keys decrease, causing the current keys to violate heap order. Instead of reporting the items with unchanged keys in order $\langle 1, 2, 4 \rangle$ they are reported in order $\langle 2, 1, 4 \rangle$. Note that when inserting 1 as the last leaf, it is compared to its parent with current key 0 (but original key 3), where the sift-up terminates, and incorrectly leaves 1 in the subtree of 2, causing 2 to be the first item extracted by ExtractMin .

Dijkstra’s algorithm

Figure 3 shows a graph with 6 nodes and 9 edges, where Dijkstra’s algorithm (Dijkstra_4) fails when using a binary heap with bottom-up insertions and using $\text{dist}[v]$ as the current key for node v . Whenever a smaller distance to a node is discovered, the node is inserted with the new distance as its original key. The previously inserted copies of the node (if any) get their current keys decreased to the new smaller distance, like b where b_2 is the copy of b

in the heap with original key 8 and current key 2. When extracting b_2 in the example, e_6 is sifted down and leaving the leftmost path top-down with nodes d_5 , b_2 and e_6 , causing the inserted node c_3 to stay at a leaf when compared with b_2 . The algorithm incorrectly visits node d before node c , causing the distance to node e to be computed incorrectly as 6. Note that node d is extracted twice from the heap, both with original keys, but only the first time we visit d and consider paths with d as the second to last node on the paths (the test $u \notin \text{visited}$ prevents us from visiting d a second time).

It should be noted that if we skipped the test $u \notin \text{visited}$, the algorithm would compute the correct distances by revisiting nodes whenever a shorter distance to a node has been discovered. This is against the principle idea of Dijkstra's algorithm to only visit nodes once in increasing order of distance from the source, so that each edge is considered at most once. See the discussion in Section 7.3 on Figure 9 for further details.

4 Binary heaps with top-down insertions

In a binary heap with top-down insertions $\text{Insert}(x)$ creates a new empty leaf at position n , and items on the path from the root to the new leaf are compared with x top-down until the first node u is found with current key greater than or equal to the new key. The items on the path from u to the new leaf are sifted one level down and item x is inserted in node u . The ancestor at depth $d = 0, \dots, \lfloor \log_2 n \rfloor$ is node $\lfloor n/2^{\lfloor \log_2 n \rfloor - d} \rfloor$. If keys are distinct and do not decrease, then top-down insertions and bottom-up insertions yield identical structures.

In the following we let key_{org} denote an original key and key_{cur} a current key. For a tree structure, where each node stores an item, we say that the tree satisfies the *decreased heap order* if and only if $u.key_{\text{cur}} \leq v.key_{\text{org}}$ for all ancestors u of a node v . Note that if decreased heap order is satisfied, then the item at the root of a tree satisfies the conditions to be returned by ExtractMin , and decreased heap order remains satisfied when current keys decrease.

During $\text{Insert}(x)$ a node v can only get one new ancestor, namely x . This happens when x is compared with an ancestor u of v with the result $x.key_{\text{cur}} \leq u.key_{\text{cur}}$. Since before the insertion decreased heap order ensures $u.key_{\text{cur}} \leq v.key_{\text{org}}$, we have $x.key_{\text{cur}} \leq v.key_{\text{org}}$ and the tree satisfies decreased heap order after the insertion.

An ExtractMin operation returns the root of the tree. By the decreased heap order the item returned has current key less than or equal to all original keys in the tree. Before returning the answer, the last item x is moved to the root and sifted down, where x is swapped with the item at the child with smallest current key until the items of both children have current key $\geq x.key_{\text{cur}}$. Whenever two siblings v and w are compared, say $v.key_{\text{cur}} \leq w.key_{\text{cur}}$, we have two cases. If x becomes the parent of v and w , since $x.key_{\text{cur}} \leq v.key_{\text{cur}} \leq w.key_{\text{cur}}$, then $x.key_{\text{cur}}$ is less than or equal to the original keys in all nodes below x , since either $v.key_{\text{cur}}$ or $w.key_{\text{cur}}$ was so before the operation. Otherwise, v becomes the parent of w , and x and all nodes in w 's subtree get v as a new ancestor. Since $v.key_{\text{cur}} \leq w.key_{\text{cur}}$, then $v.key_{\text{cur}}$ is less than or equal to all original keys in w 's subtree, and $v.key_{\text{cur}} \leq x.key_{\text{cur}} \leq x.key_{\text{org}}$. It follows that after ExtractMin the tree still satisfies decreased heap order.

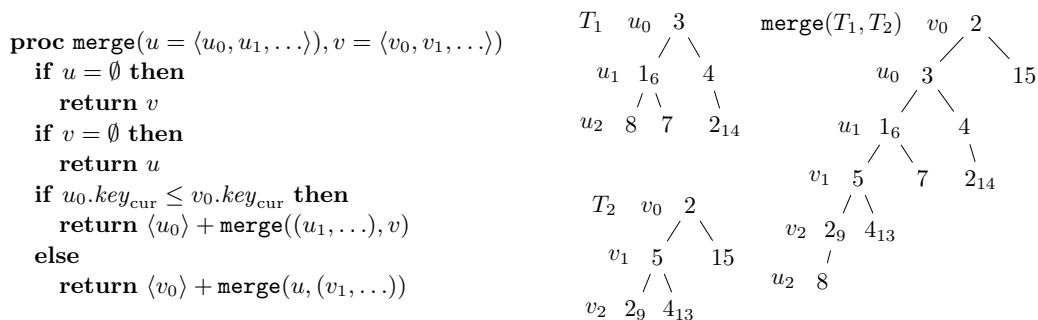
5 Existing heaps supporting decreasing keys

In the following we argue that the internal workings of several existing priority queues ensure decreased heap order to be maintained in the presence of decreasing keys.

Skew heaps and Leftist heaps

Skew heaps [18] and leftist heaps [4] support **Insert** and **ExtractMin** on a heap storing n items in time $O(\log n)$, where the time for skew heaps is amortized and for leftist heaps worst-case. Both data structures represent a priority queue by a (decreased) heap ordered binary tree, and support **ExtractMin** by removing the root and returning its item. All other structural changes consist of merging two root-to-leaf paths in two (decreased) heap ordered binary trees, and potentially swapping the left and right subtrees at nodes of the resulting tree. Swapping the left and right subtrees of a node does not change ancestor relationships, i.e., does not affect decreased heap order. To argue that the two data structures maintain decreased heap order, we only need to argue that merging two paths ensures that the resulting tree satisfies decreased heap order.

Normally the keys along the two paths would appear in increasing key order, but this is not necessarily the case when keys can decrease (in fact the current keys can appear in any order). Assume the nodes along the two root-to-leaf paths to be merged are $\langle u_0, u_1, u_2, \dots \rangle$ and $\langle v_0, v_1, v_2, \dots \rangle$, and the merging is performed top-down recursively as in Figure 4. If u_i ends up before v_j , i.e., u_i is a new ancestor of v_j , then there exists $j' \leq j$ where u_i and $v_{j'}$ have been compared and $u_i.key_{\text{cur}} \leq v_{j'}.key_{\text{cur}}$. Since by assumption $v_{j'}.key_{\text{cur}} \leq v_j.key_{\text{org}}$, it follows that u_i satisfies decreased heap order with v_j and all its descendants. It follows that the resulting tree after merging two root-to-leaf paths in two decreased heap ordered trees also satisfies decreased heap order.



■ **Figure 4** Top-down merging two paths u and v . In the example values are current keys and subscripts original keys (subscripts are omitted if current and original keys are equal).

Pairing heaps, Binomial queues, and Fibonacci heaps

Many priority queues represent a priority queue by one or more heap ordered trees of arbitrary degree. Pairing heaps [8], binomial queues [19], and Fibonacci heaps [9] are examples of such priority queues supporting **Insert** in amortized time $O(1)$ and **ExtractMin** in amortized time $O(\log n)$. Here we prove that these data structures maintain decreased heap order when used with decreasing keys. Pairing heaps only represent a priority queue by a single tree, whereas binomial queues and Fibonacci heaps maintain a forest, and the item removed by **ExtractMin** is the root with minimum current key, i.e., the returned item has current key less than or equal to all original keys in all trees. Nodes only get new ancestors when two roots are *linked*, that makes the root v with greatest current key a child of the root u with smallest current key. Since $u.key_{\text{cur}} \leq v.key_{\text{cur}}$ and any node w in the tree rooted at

v has $v.key_{cur} \leq w.key_{org}$, it follows that $u.key_{cur} \leq w.key_{org}$, i.e., the linked tree satisfies decreased heap order. It follows that the resulting heaps satisfy decreased heap order.

For Fibonacci heaps, the operations `DecreaseKey` replaces the key of an item by a smaller key. In our context this corresponds to lowering the original key. In the context of decreasing keys, it is important that `DecreaseKey` is implemented to always cut the edge from the node to its parent (without comparing with the current key of the parent, since that could have been decreased arbitrarily), and adds the node as a new root to the forest.

6 Post-order heap

Binary heaps with bottom-up insertions often benefit from the fact that insertions do not sift-up items far in the tree in practice. With top-down insertions this property cannot be exploited. In this section we consider the post-order heap by Harvey and Zatloukal [10]. In our experiments it appears to be a strong contender for an efficient implicit priority queue supporting decreasing keys, which is why we consider it in more detail in this section. Harvey and Zatloukal [10] did an experimental comparison of C# implementations of post-order heaps and binary-heaps with bottom-up insertions and found that post-order heaps had faster insertions but slower deletions.

A post-order heap consists of a forest of complete heap ordered binary trees, where all trees have distinct size, except for possibly the two trees of smallest size. Since a complete binary tree has size $2^i - 1$, for some i , the number of trees of each size corresponds to the digits in the skew binary number representation of n . Myers [14] proved that the set of tree sizes is unique for a given n . The trees are laid out consecutively in a single array H in decreasing size order, and each tree in post-order. For a subtree of size s with root $H[i]$, the subtree is stored in $H[i - s + 1, i]$, the subtrees at the children have size $\lfloor s/2 \rfloor$, the right child is $H[i - 1]$, and the left child is $H[i - 1 - \lfloor s/2 \rfloor]$. See Figure 5.

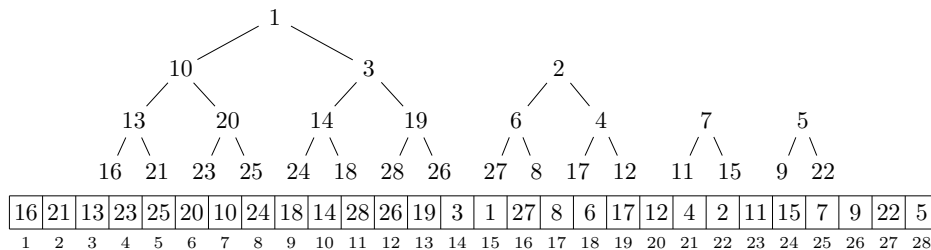


Figure 5 Top: A post-order heap storing 28 items in four trees of size 15, 7, 3 and 3. Bottom: The implicit post-order layout in a single array.

`Insert(x)` inserts the item x as the last item of H . If the last two trees had different size, x becomes a tree of size one. Otherwise, the last two trees of size s together with x are combined to a new tree of size $2s + 1$ with x as the root, and we apply the sift-down operation `Heapify(|H|, 2s + 1)` (where the first argument is the node position, and the second argument is the size of the subtree). Except for node indexing, `Heapify` is implemented as for binary heaps. To support decreasing keys, it is important that `Heapify` is performed top-down. `ExtractMin()` identifies the root min with minimum (current) key to return, and removes the root x from the rightmost tree (causing the two subtrees to become new trees). If $x \neq min$, x replaces the root min , and x is sifted down using `Heapify`. Pseudo-code for the operations is given in Figure 6 (in [10] it is discussed how the list of tree sizes S of length $O(\log n)$ can be stored using only $O(\log n)$ bits). Since a post-order heap structurally is


```

proc Insert( $x$ )
  push( $H, x$ )
  if  $|S| \geq 2$  and  $S[|S|] = S[|S| - 1]$  then
     $size = \text{pop}(S) + \text{pop}(S) + 1$ 
    push( $S, size$ )
    Heapify( $|H|, size$ )
  else
    push( $S, 1$ )

proc Heapify( $i, size$ )
  if  $size > 1$  then
     $size = \lfloor size/2 \rfloor$ 
     $right = i - 1$ 
     $left = right - size$ 
     $smallest = H[left] < H[right] ? left : right$ 
    if  $H[smallest] < H[i]$  then
      swap  $H[i]$  and  $H[smallest]$ 
      Heapify( $smallest, size$ )

proc ExtractMin()
   $min = +\infty$ 
   $i = |H|$ 
  for  $j = 1$  to  $|S|$  do
     $size = S[|S| - j + 1]$ 
    if  $H[i] < min$  then
       $min = H[i]$ 
       $i_{min} = i$ 
       $size_{min} = size$ 
     $i = i - size$ 
   $size = \lfloor \text{pop}(S)/2 \rfloor$ 
  if  $size > 0$  then
    push( $S, size$ )
    push( $S, size$ )
   $x = \text{pop}(H)$ 
  if  $i_{min} < |H|$  then
     $H[i_{min}] = x$ 
    Heapify( $i_{min}, size_{min}$ )
  return  $min$ 

```

■ **Figure 6** Post-order heap operations, where H stores the items and S is a list of tree sizes.

just a collection of binary heaps updated only using top-down **Heapify**, the discussion from Section 4 carries over to prove that post-order heaps support decreasing keys.

Harvey and Zatloukal [10] proved that post-order heaps support **Insert** in amortized time $O(1)$ and **ExtractMin** in amortized time $O(\log n)$. If we consider the worst-case number of comparisons for binary heaps with top-insertions, then **Insert** uses at most $\lfloor \log_2 n \rfloor$ whereas **ExtractMin** at most $2 \lfloor \log_2 n \rfloor$, i.e., sorting using a binary heap requires at most $3 \lfloor \log_2 n \rfloor$ comparisons. The worst-case number of comparisons for operations on post-order heaps is not competitive, since in the worst-case **Insert** performs **Heapify** on a tree containing all items, i.e., requiring at most $2 \lfloor \log_2 n \rfloor$ comparisons, and **ExtractMin** first must find the root with minimum value, and then perform **Heapify** on this tree, requiring at most $3 \lfloor \log_2 n \rfloor$ comparisons. Using these bounds for deriving a bound on sorting using a post-order heap gives us an upper bound of $5n \log_2 n$ comparisons. But for sorting we can derive a better bound. During the n insertions at most $n/2^h$ roots are created at height h each requiring $2h$ comparisons for a sift-down, causing a total of at most $\sum_{h=0}^{\infty} 2h \cdot n/2^h = O(n)$ comparisons for insertions. Furthermore, during the sequence of minimum extractions the average number of trees is $\frac{1}{2} \log_2 n + O(1)$, causing the total number of comparisons for finding the minimum roots to be at most $\frac{1}{2} n \log_2 n + O(n)$. Together with the upper bound of $2 \log_2 n$ comparisons for each **Heapify** caused by an **ExtractMin** gives a total bound of $2.5n \log_2 n + O(n)$ comparisons for sorting using a post-order heap. The advantage of post-order heaps over binary heaps with top-down insertions is studied in Section 7 (Figure 7).

7 Experimental evaluation

The previous sections state that many priority queue data structures work in the setting with decreasing keys. Any implementation of these data structures will also work if explicit keys can be removed and handled by implicit decreasing keys, e.g., implemented by a comparator

accessing the array *dist* in Dijkstra’s algorithm. The worst-case analysis of these data structures carries over to the setting with decreasing keys and the worst-case running time analysis of Dijkstra’s algorithm remains unchanged, though in practice the picture could be different. In particular items to be skipped (i.e., with decreased keys) in Dijkstra’s algorithm can be extracted earlier when allowing decreasing keys.

We implemented various priority queues and Dijkstra’s algorithm in Python 3.9 to have code as close as possible to pseudo code, and to focus on measuring counts that were hardware, language, and compiler independent.⁴ A priority queue was implemented as a class with `extract_min` and `insert` methods to update the priority queue, and a method `empty` to test for emptiness. Items are compared using the `<` operator, i.e., using the `__lt__` method of the items. Finally, each priority queue has a method `validate` to check the structural integrity of its current content, e.g., a recursive traversal checking heap order, number of children, and balance conditions. The experimental evaluation was done on a Lenovo T460s laptop (Intel i7-6600U CPU, 12 GB RAM) running Python 3.9.4 under Windows 10.

The following priority queues were implemented: Skew heaps [18], leftist heaps [4], binomial queues [19], pairing heaps [8], post-order heaps [10], and binary heaps [20] with bottom-up and top-down insertions. Finally, we made a wrapper class around Python’s builtin module `heapq` that is a C implementation⁵ of binary heaps with bottom-up insertions. We did not implement Fibonacci heaps [9], since we do not consider dedicated `DecreaseKey` operations, and without `DecreaseKey` Fibonacci heaps are identical to binomial queues.

We considered four versions of binary heaps, where the first two do not support decreasing keys: `BinaryHeap` is a standard binary heap with bottom-up insertions and top-down heapify to sift-down the new root value during minimum extractions. `BinaryHeapHeapifyBottomup` improves typical performance by letting heapify first recursively pull up the child with smallest (current) key until an empty leaf is created, where the item from the last leaf is inserted and sifted up (our experiments confirm that module `heapq` implements this idea). The last two variants support decreasing keys. `BinaryHeapTopdown` supports insertions by performing comparisons top-down along the root-to-new-leaf path, until the first node is reached with greater or equal (current) key. The new item is inserted in this node, and all remaining nodes on the path to the last leaf are sifted one level down. `BinaryHeapTopdownHeapify` has a slightly naïver insertion implementation, where all items on the root-to-new-leaf path are sifted one level down, and the new item is inserted at the root and sifted down by `heapify`.

Experiments were parameterized by the priority queue class to be tested, to ensure identical testing overhead for the different classes. In our experiments we have measured the number of comparisons performed, various counts and running time. All priority queues were tested with exactly the same set of inputs. Data structures not supporting decreasing keys (`Heapq`, `BinaryHeap` and `BinaryHeapHeapifyBottomup`) are shown with dashed lines. In all plots `Heapq` and `BinaryHeapHeapifyBottomup` have identical curves, except for the time for sorting in Figure 7(e).

7.1 Correctness of implementation

To have some evidence for the correctness of our implementations, we performed two simple sorting tests: The first checks if `Insert` and `ExtractMin` work correctly if no keys decrease, and the second checks if decreasing keys are supported. The second stress test was in fact

⁴ Python source code used for experiments and data visualized in figures is available at <https://www.cs.au.dk/~gerth/papers/fun22code.zip>

⁵ https://github.com/python/cpython/blob/master/Modules/_heapqmodule.c

used to identify which priority queues supported decreasing keys, before knowing if they did so, directing the search for the arguments presented in Sections 3–6.

Sorting

Each priority queue implementation was used to sort various input sequences of n numbers, $1 \leq n \leq 1000$, with input being the increasing sequence $1, \dots, n$, the decreasing sequence $n, \dots, 1$, a uniform random permutation of $1, \dots, n$, and n uniformly selected random integers from $1, \dots, n$ (with possible repetitions). Only the random integer inputs can contain duplicates, where the expected number of distinct integers among n integers is $n(1 - (1 - 1/n)^n) \approx n(1 - 1/e) \approx 0.632n$. Each input was first inserted using n calls to `insert` followed by calls to `extract_min` until `empty` returned true. The output was checked against Python’s built-in function `sorted`. After each update the priority queue’s `validate` method was called to check for internal integrity.

Decrease key support

Each priority queue implementation was tested for the support of decreasing keys by performing the following experiment 1000 times with $n = 100$: Insert a random permutation of $1, \dots, n$ using n calls to `Insert`, followed by n calls to `ExtractMin`. After each operation, with probability $1/2$ a random inserted item has its key decreased to zero. At the end it is checked if all items with unchanged key were reported in sorted order. As expected by the theory, all priority queues not supporting decreasing keys failed this test, whereas the others succeeded on all inputs

7.2 Sorting performance

We evaluated the performance of the different priority queues by using them to sort n integers for different n (powers of two, $n \leq 2^{20}$) and different input distributions: increasing sequences, decreasing sequences, uniformly permuted sequences, uniformly selected random integers from $1, \dots, n$. Each input was run at least 3 times and until at least 0.2 seconds were passed. For random inputs 10 different inputs were generated and the average computed. The measured average number of comparisons for each input size and type is shown in Figure 7(*a, b, c, d*). In the plots we on the y -axis have number of comparisons performed divided by $n \log_2 n$, i.e., the theoretical asymptotic worst-case bound of sorting.

That `Heapq` is equivalent to the Python implementation `BinaryHeapHeapifyBottomup` follows from the plots, where the two priority queues achieve coinciding number of comparisons (it was checked that the number of comparisons performed were identical). Among the implicit constructions based on binary heaps we have a clear ordering (except for decreasing sequences) where `BinaryHeapHeapifyBottomup` (and `Heapq`) performs the fewest comparisons, followed by `BinaryHeap`, `BinaryHeapTopdown` and `BinaryHeapTopdownHeapify`, where only the last two support decreasing keys. In all cases the implicit `PostOrderHeap` achieves a better performance than the other implicit binary heaps supporting decreasing keys. In particular we see that `PostOrderHeap` performs about $\frac{1}{2}n \log_2 n$ fewer comparisons as `BinaryHeapTopdown` for random input, as expected by the discussion in Section 6. The only priority queues supporting decreasing keys that achieve significant better bounds on the number of comparisons are all pointer based (`SkewHeap`, `LeftistHeap`, `PairingHeap` and `BinomialQueue`). This leaves the post-order heap as a strong contender for an implicit priority queue supporting decreasing keys – at least with respect to comparisons.

With respect to running times the built-in `Heapq` consistently achieved the best time-wise performance, which is not surprising since it is implemented in `C`, whereas the other implementations are clearly penalized by the overhead of Python being interpreted. Running times for random input is shown in Figure 7(e). Interestingly, for large random inputs the running time of post-order heaps is only outmatched by the builtin `heapq`, although the overhead of using Python makes the results less conclusive. Since the motivation for studying priority queues with decreasing keys is to achieve space efficiency by avoiding storing keys with the items, the implicit post-order heap appears to be a good choice of data structure in this context.

7.3 Dijkstra’s algorithm performance

We implemented a generic version of Dijkstra’s algorithm for the single-source shortest path problem corresponding to `Dijkstra4` in Figure 1. This algorithm was selected since it allows to be executed both with a priority queue supporting decreasing keys (and no explicit keys in the items) and with a standard priority queue (with explicit keys in the items), and to study if allowing decreasing keys caused the number of comparisons performed to increase or decrease. We also tested what happens if we in addition to removing the explicit keys from the items also removed the visited array from the algorithm. This further reduces the space requirement for the algorithm, but breaks the $O(|E| \log |V|)$ running time guarantee.

As arguments the function takes the graph, the priority queue to be used, and two flags `use_visited` and `use_dist`. If `use_visited` is false, the algorithm will not check if extracted nodes from the priority queue have been visited before, i.e., we could save the space for having a bit-vector for the visited nodes, the cost being that we might revisit nodes (and relax their outgoing edges) multiple times (once for each shorter distance discovered to the node). If `use_dist` is true, `dist[v]` is used as the key of v when comparing v , otherwise an item in the priority queue consists of a pair (distance, node).

As input we tested directed cliques (including self loops) with n nodes and n^2 edges, where $10 \leq n \leq 250$, with random integer weights from $1, \dots, n$, and weights forcing the worst-case number of key decreases. In the latter case, the edge from node u to v has weight $\max(0, 2(v - u) - 1)$, where $0 \leq u, v \leq n - 1$, and node 0 is the source node, i.e., the path $0 \xrightarrow{1} 1 \xrightarrow{1} \dots \xrightarrow{1} v - 1 \xrightarrow{1} v$ is the shortest path to node v with distance v .

In the experiments we measured the number of comparisons performed by the priority queues, the number of nodes inserted into the priority queue, the number of nodes visited, and the number of edges relaxed. The results are summarized in Figures 8–10, where different combinations of `use_visited` and `use_dist` were tested. E.g., “+visited –dist” is when `use_visited` is true and `use_dist` is false, i.e., `Dijkstra4`. The y -axis in Figures 8 and 9 is the measured cost divide by n , i.e., the average cost per node.

Since we only consider cliques, the number of edges relaxed is exactly n times the number of nodes visited. Furthermore, when a visited bit-vector is used, each node is visited and each edge relaxed exactly once, whereas if visited is not used, then each node inserted into a priority queue is also visited. For cliques with worst-case edge weights each edge (u, v) , where $u < v$, will cause a new shorter distance of $2v - u - 1$ to v to be discovered, causing the number of insertions into the priority queue to be $\binom{n}{2}$, and the priority queue to grow to size $\Theta(n^2)$. When visited is not used, the total number of nodes visited is $\binom{n}{2} \approx n^2/2$ and the number of edges relaxed is $n \binom{n}{2} \approx n^3/2$. We therefore only show data for the number of nodes inserted into the priority queues for cliques with random weights in Figure 9 and omit data for the number of edges relaxed, nodes visited, and insertions for worst-case graphs.

For random edge weights, the probability is at most $1/k$ that the k ’th edge considered

with node v as target decreases the distance to v (this follows by a simple backwards analysis argument), i.e., the expected number of times node v is inserted into the priority queue is at most $\sum_{k=1}^{n-1} \frac{1}{k} \approx \ln n$. This explains the nature of the curves in Figure 9(a), where the priority queue stores $\langle \text{distance}, \text{node} \rangle$ pairs. When using $\text{dist}[v]$ as key, it follows from Section 2 that the same applies to the priority queues supporting decreasing keys (non-dashed plots in Figure 9(b)).

Priority queues not supporting decreasing keys (Heapq, BinaryHeap and BinaryHeap-HeapifyBottomup) might return the wrong items when using the current dist as keys. In combination with using a visited bit-vector, the algorithm might fail to compute the correct shortest distances. This was the case for random weights. Surprisingly, the algorithm worked correctly for worst-case weights. If the visited bit-vector is not used, then incorrect nodes returned by the priority queue will just be revisited later with a shorter distance, so the algorithm will still correctly find shortest paths – but there is no guarantee on the number of edge relaxes performed (except for an exponential upper bound that holds for any relax based approach). For our inputs the potential increase in number of revisits to nodes appears neglectable though, cf. Figure 9(b).

Finally, the number of comparisons performed in the priority queues is depicted in Figure 8. Interestingly, the number of comparisons performed appears to be a little bit lower if we use dist as key, cf. (e) versus (g), and (f) versus (h). In Figure 10 we have plotted the number of comparisons performed when using dist as the key relative to using $\langle \text{distance}, \text{node} \rangle$ pairs, for priority queues supporting decreasing keys and when using visited. The gain is largest for random cliques where the number of comparisons is reduced by typically at least 10%. Interestingly, it appears that *using a priority queue supporting decreasing keys one can both save the space for storing explicit keys in the priority queue and reduce the number of comparisons performed.*

8 Conclusion

We have considered using priority queues supporting decreasing keys in Dijkstra’s single source shortest path algorithm, motivated by the idea of saving space by omitting explicit keys for the items in the priority queue. Although standard binary heaps with bottom-up insertion fail to support decreasing keys, many other priority queues have been identified to do so, and in particular post-order heaps have been identified as a strong contender for a good alternative implicit priority queue in this context.

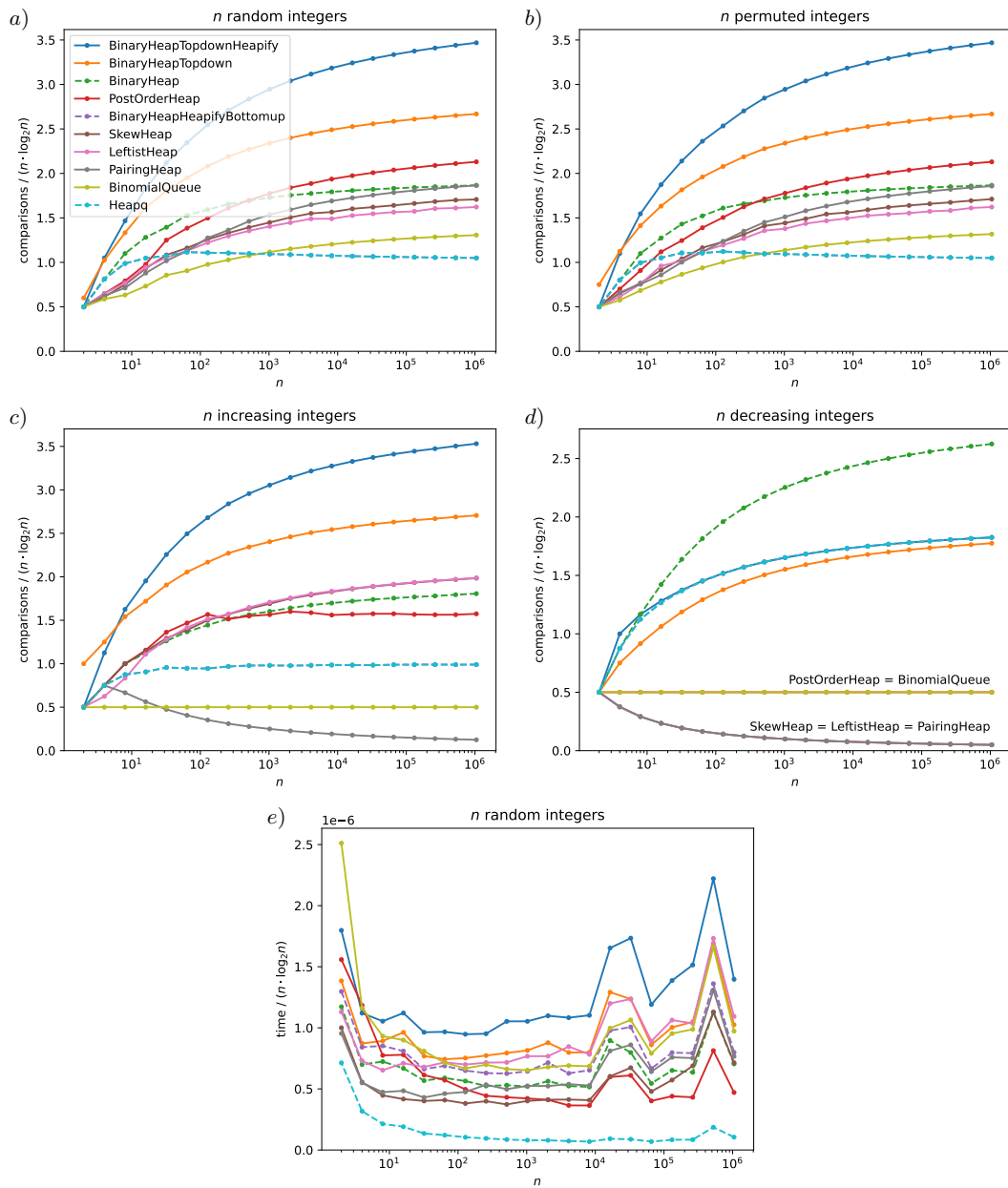
An open problem is to do a detailed experimental evaluation of the priority queues supporting decreasing keys in a low-level programming language like C. This is beyond the scope of this paper. Optimizing the running time of such implementations would require to carefully tune the code to take into account, e.g., caching, paging, branch mispredictions, and exploiting SIMD instructions [17]. Since the cache performance of binary heaps is known to be improvable by increasing the degree of the heap [13], one should also consider post-order heaps of higher degree.

References

- 1 Gerth Stølting Brodal. A survey on priority queues. In Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 150–163. Springer, 2013. doi: 10.1007/978-3-642-40273-9_11.

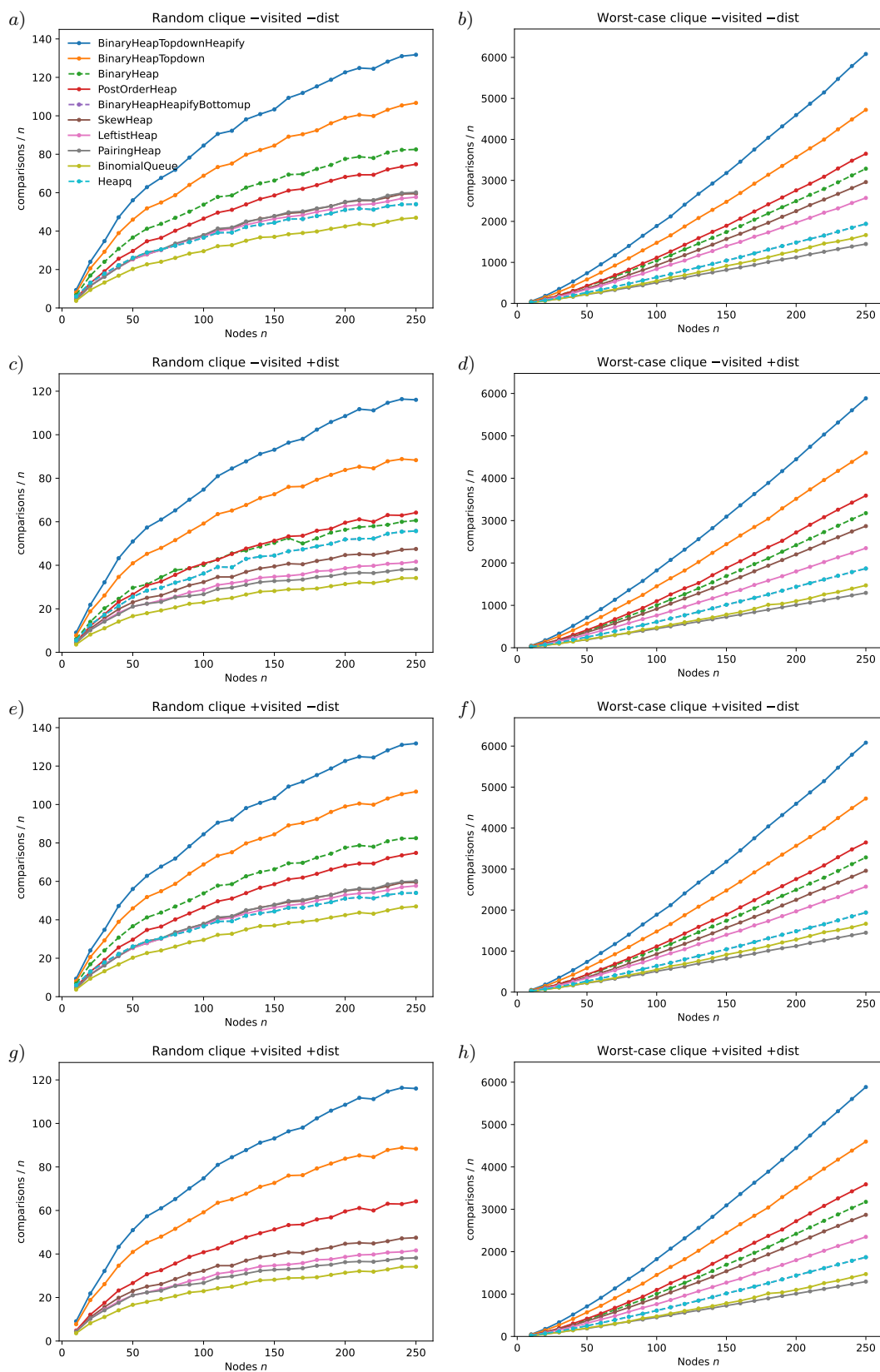
- 2 Svante Carlsson, J. Ian Munro, and Patricio V. Poblete. An implicit binomial queue with constant insertion time. In Rolf G. Karlsson and Andrzej Lingas, editors, *SWAT 88, 1st Scandinavian Workshop on Algorithm Theory, Halmstad, Sweden, July 5-8, 1988, Proceedings*, volume 318 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1988. doi:10.1007/3-540-19487-8_1.
- 3 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: <http://mitpress.mit.edu/books/introduction-algorithms>.
- 4 Clark A. Crane. *Linear Lists and Priority Queues as Balanced Binary Trees*. PhD thesis, Department of Computer Science, Stanford University, 1972.
- 5 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.
- 6 James R. Driscoll, Harold N. Gabow, Ruth Shrairman, and Robert Endre Tarjan. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. *Commun. ACM*, 31(11):1343–1354, 1988. doi:10.1145/50087.50096.
- 7 Amr Elmasry, Claus Jensen, and Jyrki Katajainen. Two skew-binary numeral systems and one application. *Theory Comput. Syst.*, 50(1):185–211, 2012. doi:10.1007/s00224-011-9357-0.
- 8 Michael L. Fredman, Robert Sedgewick, Daniel Dominic Sleator, and Robert Endre Tarjan. The pairing heap: A new form of self-adjusting heap. *Algorithmica*, 1(1):111–129, 1986. doi:10.1007/BF01840439.
- 9 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.
- 10 Nicholas J. A. Harvey and Kevin C. Zatloukal. The post-order heap. In *Proceedings Third International Conference on Fun with Algorithms (FUN 2004)*, 2004. URL: <http://people.csail.mit.edu/nickh/Publications/PostOrderHeap/FUN04-PostOrderHeap.pdf>.
- 11 Vojtěch Jarník. O jistém problem minimálním. *Práce Moravské Pridovedecké Spolecnosti v Brně*, 4:57–63, 1930.
- 12 Irit Katriel and Ulrich Meyer. Elementary graph algorithms in external memory. In Ulrich Meyer, Peter Sanders, and Jop F. Sibeyn, editors, *Algorithms for Memory Hierarchies, Advanced Lectures*, volume 2625 of *Lecture Notes in Computer Science*, pages 62–84. Springer, 2002. doi:10.1007/3-540-36574-5_4.
- 13 Anthony LaMarca and Richard E. Ladner. The influence of caches on the performance of heaps. *ACM J. Exp. Algorithmics*, 1:4, 1996. doi:10.1145/235141.235145.
- 14 Eugene W. Myers. An applicative random-access stack. *Inf. Process. Lett.*, 17(5):241–248, 1983. doi:10.1016/0020-0190(83)90106-0.
- 15 Thomas Porter and István Simon. Random insertion into a priority queue structure. *IEEE Trans. Software Eng.*, 1(3):292–298, 1975. doi:10.1109/TSE.1975.6312854.
- 16 R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957. doi:10.1002/j.1538-7305.1957.tb01515.x.
- 17 Peter Sanders. Fast priority queues for cached memory. *ACM J. Exp. Algorithmics*, 5:7, 2000. doi:10.1145/351827.384249.
- 18 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting heaps. *SIAM J. Comput.*, 15(1):52–69, 1986. doi:10.1137/0215004.
- 19 Jean Vuillemin. A data structure for manipulating priority queues. *Commun. ACM*, 21(4):309–315, 1978. doi:10.1145/359460.359478.
- 20 J. W. J. Williams. Algorithm 232 heapsort. *Commun. ACM*, 7(6):347–348, 1964. doi:10.1145/512274.512284.

A Plots

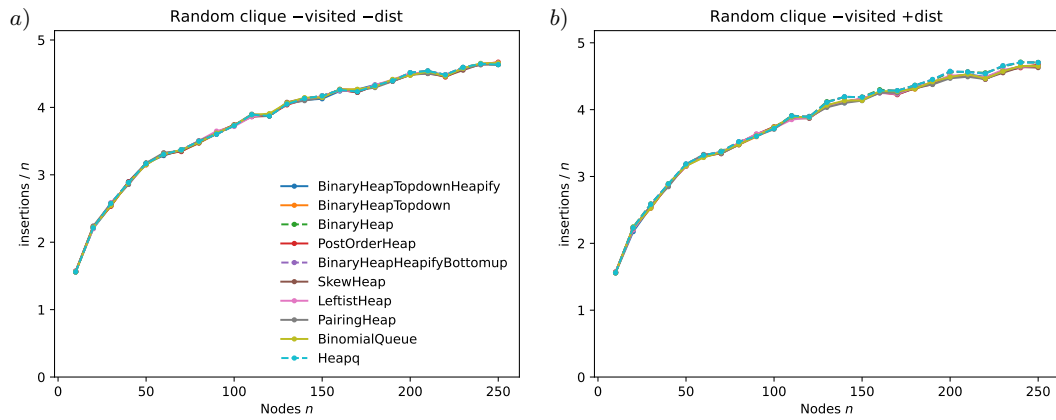


■ **Figure 7** Sorting n integers using various priority queues.

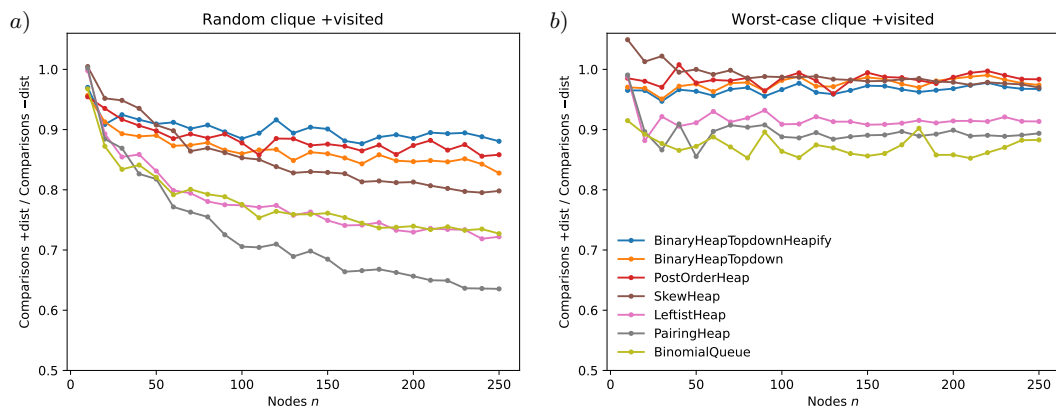
8:18 Priority Queues with Decreasing Keys



■ **Figure 8** Dijkstra, comparisons performed.



■ Figure 9 Dijkstra, priority queue insertions.



■ Figure 10 Dijkstra, number of comparisons performed when using *dist* as key divided by the number of comparisons performed when storing explicit keys in the priority queues.

Zero-Knowledge Proof of Knowledge for Peg Solitaire

Xavier Bultel  

INSA Centre Val de Loire, Laboratoire d'informatique fondamentale d'Orléans, Bourges, France

Abstract

Peg solitaire is a very popular traditional single-player board game, known to be NP-complete. In this paper, we present a zero-knowledge proof of knowledge for solutions of peg solitaire instances. Our proof is straightforward, in the sense that it does not use any reduction to another NP-complete problem, and uses the standard design of sigma protocols. Our construction relies on cryptographic commitments, which can be replaced by envelopes to make the protocol physical. As a side contribution, we introduce the notion of isomorphisms for peg solitaire, which is the key tool of our protocol.

2012 ACM Subject Classification Theory of computation → Interactive proof systems

Keywords and phrases Zero-Knowledge Proof, Peg Solitaire

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.9

1 Introduction

Zero-knowledge proofs are fascinating protocols introduced by Goldreich, Micali, and Rackoff [8], in which a *prover*, knowing a secret needed to verify some property, tries to convince a verifier that the property actually holds, but without revealing anything about the secret. Although these surprising features seem difficult to achieve, these protocols are often elegant and very simple to understand. For this reason, zero-knowledge proofs are usually considered as a beautiful concept.

In [7], Goldreich, Micali, and Wigderson present a protocol for proving the knowledge of a graph 3-coloring, without revealing anything about the coloring. Such a protocol is said to be a *zero-knowledge proof of knowledge*. This protocol is amazingly simple: the prover randomly permutes the three colors and commits the new color of each vertex, the verifier chooses one of the edges of the graph, the prover reveals the color of its two endpoints, and the verifier checks that these two colors are different. On the one hand, the verifier learns nothing else than the fact that the two endpoints have different (random) colors. On the other hand, if the prover does not know any 3-coloring, then at least one edge has two endpoints of the same color. In this case, the prover succeeds its proof with probability of at most $(n-1)/n$, where n is the number of edges. By repeating the protocols λ times, its probability of success falls to $((n-1)/n)^\lambda$. By adjusting the parameter λ , the probability of deceiving the verifier can be reduced as much as desired. This protocol can be used physically (*i.e.* without computer and without cryptography) by replacing the commitments with paper envelopes.

This construction allows at the same time to show the existence of a zero-knowledge proof of knowledge for any problem in NP: by reducing the required problem to the graph 3-coloring, the prover can turn the instance of the problem into an instance of the graph 3-coloring, and can prove its knowledge of this 3-coloring. However, using this generic method results in heavy and unclear proof protocols, and it is often better to design tailor-made proofs for specific problems in NP that are understandable, elegant and efficient.

If building tailor-made zero-knowledge proofs of knowledge for NP problems is fun in itself, building such proofs for fun problems is even more fun. Therefore, many proofs of knowledge for logic puzzles have been proposed, either in a cryptographic or a physical setting. For



© Xavier Bultel;

licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 9; pp. 9:1–9:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

instance, cryptographic and physical zero-knowledge proofs of knowledge for sudoku puzzles are presented in [9] and [16], physical proofs for various logical grid puzzles (namely akari, takuzu, kakuro and ken-ken) are presented in [4], and cryptographic proofs problems based on the Rubik's cube are presented in [19]. Some works use physical properties of playing cards to optimize the design of the physical proofs [13, 16]. As a fun application, such proofs can be used as authentication or signature protocols, where authentication amounts to proving the knowledge of a secret solution for a public logic game instance [19].

Peg solitaire is a traditional board game for one player known all over the world. This game was already known in the 17th century, and is probably older. Peg solitaire rules are very simple but the game is deeply complex, which makes it an exciting research topic for mathematicians and computer scientists, as shown by the many scientific papers that have already been published about it [1, 2, 10, 11, 12, 15, 18]

The game is played on a board with holes that can contain pegs. The player can jump a peg over an adjacent peg (which is removed from the board) into a hole. The goal is to reach a given winning position from a given initial position. Peg solitaire is proven to be NP-complete [10]. A weaker variant of the game, where the goal is to remove all the pegs but one, was previously proven to be NP-complete in [18]. In [15], authors show that the game is no longer NP-complete when one of the dimensions of the board is fixed.

In this paper, we give the first tailor-made cryptographic zero-knowledge proof of knowledge for solutions of peg solitaire, in the sense that our protocol does not rely on a reduction to another problem but uses the specific properties of the peg solitaire. While most logic games used to build zero-knowledge proofs consist of filling in a grid with a pen within certain constraints, peg solitaire has a different mechanism, since its solution is a series of successive moves on a board. The method that we use seems generic and should be applicable to other similar games: at each proof round, the prover shows to the verifier that one of its moves is legal.

Our construction

Because of its structure, a peg solitaire board can be formalized as a graph (the link between the graphs and the peg solitaire is quite natural and has already been studied in [2]). This representation allows us to extend the notion of graph isomorphisms to define the notion of peg solitaire isomorphisms. Loosely speaking, two peg solitaires are isomorphic when there is a bijection that allows to pass from one to the other preserving the existence and the general structure of their solutions. Considering two isomorphic peg solitaires, we give an efficient method to transform a solution of one into a solution of the other using the isomorphism. Our definition of peg solitaire isomorphism is generic and could be of independent interest.

Our proof protocol uses the standard method introduced in [7] with the proof for 3-coloring: the prover commits its solution, the verifier challenges the prover to show that one of the constraints of the problem holds in the solution, and the prover shows this by decommitting one part of the solution. Obviously, the revealed part should not leak anything else than the respect of the constraint.

More precisely, the prover first chooses an isomorphism of peg solitaire at random, and computes the image of the peg solitaire instance and its solution by this isomorphism. This step *randomizes* the structure of the solution. The prover then commits each part of this randomized peg solitaire and its solution. The verifier has two ways to challenge the prover:

- Either it requests the prover to reveal the isomorphism. In this case the prover also decommits the isomorphic peg solitaire (but not its solution). The verifier then checks that the peg solitaire has been correctly randomized, but it learns nothing about the solution.

- Or it requests the prover to reveal the i^{th} move of the randomized solution. In this case the prover decommits the corresponding part of the randomised solution, and the verifier checks that the move is legal (*i.e.* it is a jump from a peg over an adjacent peg landing in a hole). Since this move is isolated from the others and takes place on a randomized board, the verifier does not learn anything else about the solution.

If the prover knows how to answer each challenge, then it knows each move of the solution for the isomorphic peg solitaire instance, and it knows the isomorphism allowing to pass from this solution to that of the initial peg solitaire instance. By contrapositive, if the prover is not able to answer all the challenges, then it does not know a solution. The probability that the prover deceives the verifier is at most $(S - 1)/S$, where S is the number of steps in the solution. By repeating the protocol several times, this probability becomes as small as desired. Note that this protocol can also be turned into physical proof by replacing each commitment with a paper envelope.

2 Cryptographic background

In this section, we recall the definitions of negligible functions, zero-knowledge proofs of knowledge, sigma protocols, and commitments.

We use the notion of *negligible function* to formalize the concept of *unrealistic event*. A function is negligible when it tends to zero faster than the inverse of any polynomial. We consider that an attack is not feasible in practice if its probability is negligible in some security parameter.

► **Definition 1.** A positive function $\epsilon : \mathbb{N} \rightarrow \mathbb{R}$ is said to be negligible if for any positive polynomial t , there exists an integer n such that for all integer $x > n$, $|\epsilon(x)| < \frac{1}{t(x)}$.

A *zero-knowledge proof of knowledge* [3] is a protocol where two parties interact, a *prover* and a *verifier*. The prover knows a secret value (in our case, the solution of a peg solitaire) and tries to convince the verifier of this knowledge, without leaking any other information about the solution.

A zero-knowledge proof of knowledge has the three following properties:

Completeness: If the prover actually knows the secret and runs the protocol honestly, then the proof is accepted.

Validity: If the prover does not know the secret, then the probability that its proof is accepted in a reasonable time is negligible. This probability is called the *knowledge error*. More precisely, assuming the existence of a prover able to perform an accepted proof with a reasonable probability, the validity ensures that the secret of that prover can be extracted in a similar running time.

Zero-knowledge: The verifier learns nothing about the secret of the prover. More precisely, the verifier is able to simulate its interaction with the prover in such a way that no efficient algorithm can distinguish the simulated transcripts from the real transcripts with non-negligible probability.

The protocol that we present in this paper is a *sigma protocol* [5], *i.e.* it is a repetition of the three following interactions: the prover sends a commitment, the verifier sends a challenge, and the prover sends a response. Due to their structure, validity of such protocols can be proven using the definition of the *n-Special-soundness*. This property holds when it is possible to extract the secret from n transcripts containing different challenges but sharing the same commitment. A *n-special-sound* sigma protocol is valid with a knowledge error of $(n - 1)/n$ [5]. By repeating the protocol λ times, the knowledge error falls to $((n - 1)/n)^\lambda$, which is negligible in λ .

► **Definition 2** (Sigma protocol [3, 5]). Let R_L be a relation for a language L in NP, we say that an element x is a witness for the instance $y \in L$ if $(x, y) \in R_L$. Let λ be a security parameter; in what follows, p.p.t algorithm means probabilistic polynomial-time algorithm in λ . Let us consider two p.p.t algorithms \mathcal{P} (the Prover) knowing a witness x and \mathcal{V} (the verifier) knowing an instance y interacting in a proof protocol $\langle \mathcal{P}(x), \mathcal{V}(y) \rangle$ using the following pattern: \mathcal{P} sends a commitment, \mathcal{V} sends a challenge, \mathcal{P} sends a response, after which \mathcal{V} accepts or rejects the proof. This proof protocol is said to be a sigma-protocol for R_L . We denote the set of the accepted transcripts for any instance y by $\text{Acc}(y)$. Moreover, a sigma protocol can have the following properties:

Completeness: For every $(x, y) \in R_L$:

$$\Pr[\langle \mathcal{P}(x), \mathcal{V}(y) \rangle \in \text{Acc}(y)] = 1.$$

n -Special-soundness: For any $y \in L$, there exists an efficient algorithm \mathcal{E} (polynomial in $|y|$) such that for any set of n valid transcripts $\tau \in \text{Acc}(y)^n$ sharing the same commitment but having n different challenges from each other, we have:

$$\Pr[x \leftarrow \mathcal{E}(\tau, y) : (x, y) \in R_L] = 1.$$

(Computational) Zero-knowledge: For every $(x, y) \in R_L$ and every p.p.t verifier \mathcal{V}^* there exists a probabilistic p.p.t algorithm \mathcal{S} (called simulator) such that for every p.p.t algorithm \mathcal{D} , the following is negligible in λ :

$$|\Pr[\text{tr} \leftarrow \langle \mathcal{P}(x), \mathcal{V}^*(y) \rangle; b \leftarrow \mathcal{D}(\text{tr}) : b = 1] - \Pr[\text{tr} \leftarrow \mathcal{S}(y); b \leftarrow \mathcal{D}(\text{tr}) : b = 1]|.$$

A sigma protocol which is complete, special-sound, and zero-knowledge is said to be a zero-knowledge proof of knowledge.

A cryptographic commitment scheme [6, 7] $C = (\text{Gen}, \text{Commit})$ is a pair of algorithms that allows a user to commit a value with a secret key. More precisely, $\text{Gen}(1^\lambda)$ generates a commitment key sk from a security parameter λ , and $\text{Commit}(\text{sk}, m)$ generates a commitment c from a key sk and a value m . To open the commitment c , the user reveals the committed value m and the key sk . The validity of the opened commitment is verified if $c = \text{Commit}(\text{sk}, m)$. Such a scheme should verify two properties:

Biding: The user is constrained to open the value that it actually committed. In other words, the user cannot open a commitment in two different ways with non-negligible probability.

If this probability is zero, we say that the commitment has the *perfect* binding property.

Hiding: Without the key, a commitment reveals nothing about the committed value. In other words, a user cannot distinguish which value is committed out of two chosen values in a given commitment with non-negligible probability.

► **Definition 3** (Commitment Scheme [6, 7]). A commitment scheme $C = (\text{Gen}, \text{Commit})$ is a pair of algorithms defined as follows:

- $\text{Gen}(1^\lambda)$ generates a commitment key sk from a security parameter λ .
- $\text{Commit}(\text{sk}, m)$ generates a commitment c from a key and a message.

A commitment scheme C has the binding (resp. perfect biding) security property if for any p.p.t algorithm \mathcal{A} , the following is negligible in λ (resp. null), where $\mathbf{Exp}_{C, \mathcal{A}}^{\text{Binding}}(\lambda)$ denotes the experiment given in Figure 1 (right side):

$$\Pr \left[1 \leftarrow \mathbf{Exp}_{C, \mathcal{A}}^{\text{Binding}}(\lambda) \right].$$

<p>Experiment $\text{Exp}_{C,\mathcal{A}}^{\text{Binding}}(\lambda)$:</p> <p>$(\text{sk}_0, \text{sk}_1, m_0, m_1) \leftarrow \mathcal{A}(1^\lambda)$</p> <p>if $\text{Commit}(\text{sk}_0, m_0) = \text{Commit}(\text{sk}_1, m_1)$</p> <p>then return $m_0 \neq m_1$</p> <p>else return 0</p>	<p>Experiment $\text{Exp}_{C,\mathcal{A},b}^{\text{Hiding}}(\lambda)$:</p> <p>$\text{sk} \leftarrow \text{Gen}(1^\lambda)$</p> <p>$(m_0, m_1, \text{st}) \leftarrow \mathcal{A}_1(\text{sk})$</p> <p>$c_b \leftarrow \text{Commit}(\text{sk}, m_b)$</p> <p>$b' \leftarrow \mathcal{A}_2(c_b, \text{st})$</p> <p>return $b = b'$</p>
---	--

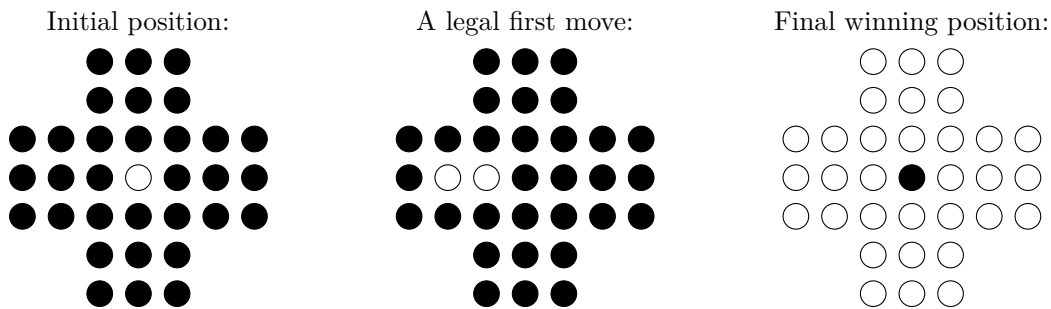
■ **Figure 1** The *binding* and the *hiding* experiments.

A commitment scheme has the hiding security property if for any two-party p.p.t algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following is negligible in λ , where $\text{Exp}_{C,\mathcal{A}}^{\text{Hiding}}(\lambda)$ denotes the experiment given in Figure 1 (left side):

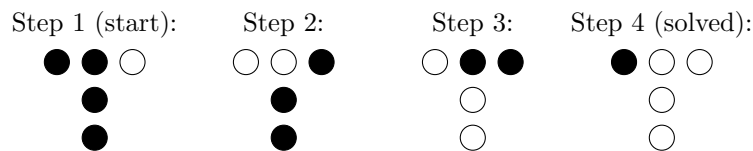
$$\left| \Pr \left[1 \leftarrow \text{Exp}_{C,\mathcal{A},1}^{\text{Hiding}}(\lambda) \right] - \Pr \left[0 \leftarrow \text{Exp}_{C,\mathcal{A},0}^{\text{Hiding}}(\lambda) \right] \right|.$$

In [7], the authors show how to construct a commitment scheme that has both hiding and perfect binding properties under the hypothesis of the existence of an ideal hash function, and use it as a key tool to construct zero-knowledge proofs of knowledge for NP problems. A perfect hiding commitment scheme is given in [14]. Note that a commitment scheme cannot be both perfect hiding and perfect binding.

3 The game



■ **Figure 2** The english peg solitaire.



■ **Figure 3** The solution of the tee peg solitaire.

Peg solitaire is a board game for one player, which is played on a board containing holes where pegs can be inserted. Each hole can hold at most one peg. At the beginning of the game, the pegs are placed in a given initial position, and the goal is to move them in order to reach a given winning position. The moves are jumps: if two pegs and an empty hole are

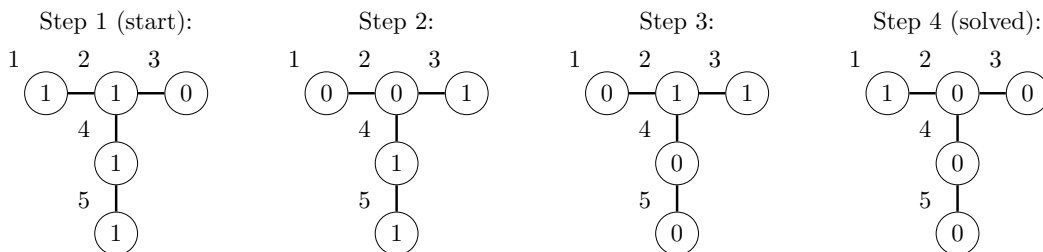
aligned and adjacent, the first peg can jump over the second one to land in the empty hole. The jumped peg is removed from the game. Since at each move a single peg is removed from the game, the number of moves to reach the winning position is equal to the number of pegs in the initial position less the number of pegs in the winning one.

In Figure 2 we present the English peg solitaire, which is one of the most popular variants of this game. The black circles are holes containing pegs, and the white circles are empty holes. In this variant, the holes are placed orthogonally and form a cross. In the initial position, only the central hole is empty. The goal is to remove all the pegs except one, which must be moved into the center hole. Note that the possibilities of boards are endless, and are not limited to orthogonal boards.

Throughout this paper, we will use a simple example of peg solitaire to illustrate our definitions: the tee peg solitaire. The initial board contains 5 holes that form a tee, with the rightmost hole being the only empty hole. The winning position contains only one peg in the leftmost hole. In Figure 3, we show how to solve this peg solitaire, *i.e.* we show successive moves that allow to reach the winning position from the initial one.

4 Formalism

In this section, we present how we formalize the peg solitaires. In a first step, we represent the state of the game board by a graph, where each hole corresponds to a vertex labeled by an index. The value of a vertex is 1 if the corresponding hole contains a peg, 0 otherwise. The vertices corresponding to adjacent holes on the board are linked by an edge labeled by a direction (horizontal or vertical). Note that this representation can be generalized for a number of directions greater than two (for example for hexagonal boards).



■ **Figure 4** The graph representation of the steps of the tee peg solitaire solution.

For instance, the steps of the solution of the tee peg solitaire can be represented as in Figure 4, where the direction of each edge implicitly corresponds to its actual direction on the diagram.

A peg solitaire instance is defined by the shape of the board, the position of the pegs at the beginning of the game, and the position of the pegs that must be reached to win. We can therefore represent any instance by the graph of its initial position and the graph of its winning position. Note that only the values of the vertices differ between these two graphs, so we can give only the structure of the graph and the values of the vertices in the initial position and in the winning position.

Finally, we can encode an instance of peg solitaire by giving a set of triplets, containing the ordered combinations of 3 indexes of vertices that are adjacent and aligned in the graph (*i.e.* the triplets (a, b, c) such that (a, b) and (b, c) are edges labeled by the same direction). This is sufficient to rebuild the entire structure of the graph. For instance, the graph of the

tee peg solitaire and the graph of the English peg solitaire are encoded by 2 and 38 triplets of vertices respectively. The encoding must also contain two vectors f (for *first*) and l (for *last*), containing the values of the vertices in the initial position and in the winning position respectively, ordered by the indexes of the vertices. This leads us to the following definition, where N denotes the number of vertices, T the number of triplets, and S the number of steps in the solution.

► **Definition 4** (Peg solitaire instance). *Let N , S , and T be three positive integers. A (N, T, S) -peg solitaire instance is a tuple (t, f, l) where*

- $t = (t_i)_{1 \leq i \leq T}$ is a vector of T triplets of integers, where:
 - Each t_i is in $\llbracket 1, N \rrbracket^3$.
 - For each i , parsing t_i as $(t_{i,1}, t_{i,2}, t_{i,3})$, we have $t_{i,1} \neq t_{i,2}$, $t_{i,2} \neq t_{i,3}$ and $t_{i,1} \neq t_{i,3}$ (the values in the triplets are different).
 - For each $i \neq i'$, parsing t_i as $(t_{i,1}, t_{i,2}, t_{i,3})$ and $t_{i'}$ as $(t_{i',1}, t_{i',2}, t_{i',3})$, we have $\{t_{i,1}, t_{i,2}, t_{i,3}\} \neq \{t_{i',1}, t_{i',2}, t_{i',3}\}$ (each triplet contains different values).
- $f = (f_i)_{1 \leq i \leq N}$ and $l = (l_i)_{1 \leq i \leq N}$ are vectors of N bits and $S = 1 + \sum_{i=1}^N (f_i - l_i)$ (S is the difference between the number of 1 in f and l plus one, because at each step, a peg is removed).

Note that our definition is generic, and accepts instances that could not be represented by graphs.

► **Example 5** (Tee peg solitaire instance). Using the previous definition, the tee peg solitaire is encoded by a $(5, 2, 4)$ -peg solitaire instance (t, f, l) defined as follows:

- $t = ((1, 2, 3), (2, 4, 5))$.
- $f = (1, 1, 0, 1, 1)$ and $l = (1, 0, 0, 0, 0)$.

We now show how to encode a solution and how to verify its validity for a given peg solitaire instance. A solution is simply the set of vectors representing the values of the vertices at each step of the game. Thus, the first vector of the solution must be f , and the last vector must be l (*i.e.* the sequence of vectors must show how to go from the position of f to the position of l).

To be considered as valid, the solution must match steps that respect the rules of the game. To formalize this, let us notice two properties:

- At each step, there are exactly three aligned holes whose state changes: the hole containing the jumping peg, the hole containing the jumped peg, and the hole receiving the jumping peg.
- The state of these three aligned holes is $(\bullet\bullet\circ)$ or $(\circ\bullet\bullet)$ before the move, and $(\circ\circ\bullet)$ or $(\bullet\circ\circ)$ respectively after the move. The direction (horizontal or vertical) does not matter here, since both cases are symmetric.

Thus, we must verify that exactly three bits are different between each vector of the solution, and that these three bits are the values of three vertices indexed in one of the triplets of the instance. Moreover, we must verify that :

- either the two first vertices of the triplet have the value 1 and the last one has the value 0 in the first vector,
- or the first vertex of the triplet has the value 0 and the two last ones have the value 1 in the first vector.

Finally, we obtain the following definition.

► **Definition 6** (Solution for a peg solitaire instance). Let (t, f, l) be a (N, T, S) -peg solitaire instance. A solution for this instance is a binary matrix $Z = (Z[i, j])_{1 \leq i \leq S; 1 \leq j \leq N}$. Throughout this paper, by abuse of notation, for any solution Z and any index i , we will denote the i^{th} line of a solution by $Z[i]$, i.e. $Z[i] = (Z[i, j])_{1 \leq j \leq N}$.

A solution is said to be valid if $f = Z[1]$, $l = Z[S]$, and for any $i \in \llbracket 1, S-1 \rrbracket$, there exists $j \in \llbracket 1, T \rrbracket$ such that, parsing t_j as $(t_{j,1}, t_{j,2}, t_{j,3})$:

- For any $k \in \llbracket 1, N \rrbracket$, $(k \in \{t_{j,1}, t_{j,2}, t_{j,3}\} \Leftrightarrow Z[i, k] \neq Z[i+1, k])$ (exactly three bits, corresponding to a triplet of vertices, are different).
- $Z[i, t_{j,1}] \neq Z[i, t_{j,3}]$ and $Z[i, t_{j,2}] = 1$ (the three vertices are in a state where the jump is possible).

► **Example 7** (Solution for the tee peg solitaire). Using the previous definition, the solution for the tee peg solitaire instance given in Example 5 is encoded by the following matrix:

$$Z = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We now define the notion of peg solitaire isomorphism. Loosely speaking, two peg solitaires are isomorphic when one can be obtained by permuting the vertices, permuting the triplets, and reversing the order of the triplets of the other.

► **Definition 8** (Peg solitaire isomorphism). A (N, T, S) -peg solitaire isomorphism is a triplet of bijections $\xi = (\phi, \chi, \psi)$ defined as follows:

- $\phi : \llbracket 1, N \rrbracket \rightarrow \llbracket 1, N \rrbracket$,
- $\chi : \llbracket 1, T \rrbracket \rightarrow \llbracket 1, T \rrbracket$, and
- $\psi : \{1, 3\} \rightarrow \{1, 3\}$.

For any (N, T, S) -peg solitaire instance $\Sigma = (t, f, l)$, we define the (N, T, S) -peg solitaire instance $\Sigma' = (t', f', l')$ isomorphic to Σ by ξ as follows:

- for any $i \in \llbracket 1, T \rrbracket$, $t'_{\chi(i)} = (\phi(t_{i,\psi(1)}), \phi(t_{i,\psi(2)}), \phi(t_{i,\psi(3)}))$, and
- for any $i \in \llbracket 1, N \rrbracket$, $f'_i = f_{\phi^{-1}(i)}$ and $l'_i = l_{\phi^{-1}(i)}$.

► **Remark 9.** A more generic definition of peg solitaire isomorphism can be obtained by choosing a different function ψ for each triplet t_i . However, as this property is not required for our purpose and would make the notations more cumbersome, we do not mention it explicitly in our definition.

An isomorphism can be used to transform a solution for a peg solitaire instance into a solution for its isomorphic instance. We show this result in the following theorem.

► **Theorem 10.** Let $\Sigma = (t, f, l)$ be a (N, T, S) -peg solitaire instance, $Z = (Z[i, j])_{1 \leq i \leq S; 1 \leq j \leq N}$ be a solution for Σ , $\xi = (\phi, \chi, \psi)$ be a (N, T, S) -peg solitaire isomorphism, and $\Sigma' = (t', f', l')$ be the (N, T, S) -peg solitaire instance isomorphic to Σ by ξ . Let $Z' = (Z'[i, j])_{1 \leq i \leq S; 1 \leq j \leq N}$ be a solution for Σ' such that, for any $i \in \llbracket 1, S \rrbracket$ and $j \in \llbracket 1, N \rrbracket$, $Z'[i, j] = Z[i, \phi^{-1}(j)]$. We have that Z' is a valid solution for Σ' if and only if Z is a valid solution for Σ .

Proof. Throughout this proof, we use the same notation as in Definition 4, Definition 6 and Definition 8. We first prove the three following claims:

▷ **Claim 11.** $f = Z[1] \Leftrightarrow f' = Z'[1]$ and $l = Z[S] \Leftrightarrow l' = Z'[S]$.

Proof. For any $j \in \llbracket 1, N \rrbracket$, $f'_j = f_{\phi^{-1}(j)}$ and $Z'[1, j] = Z[1, \phi^{-1}(j)]$ and $f_{\phi^{-1}(j)} = Z[1, \phi^{-1}(j)]$, so:

$$f = Z[1] \Leftrightarrow f' = Z'[1].$$

Moreover, $l'_j = l_{\phi^{-1}(j)}$ and $Z'[S, j] = Z[S, \phi^{-1}(j)]$ and $l_{\phi^{-1}(j)} = Z[S, \phi^{-1}(j)]$, so:

$$l = Z[S] \Leftrightarrow l' = Z'[S]. \quad \triangleleft$$

▷ Claim 12. for any $i \in \llbracket 1, S-1 \rrbracket$, any $j \in \llbracket 1, T \rrbracket$ and any $k \in \llbracket 1, N \rrbracket$:

$$\begin{aligned} (k \in \{t_{j,1}, t_{j,2}, t_{j,3}\} \Leftrightarrow Z[i, k] \neq Z[i+1, k]) \\ \Leftrightarrow (k \in \{t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3}\} \Leftrightarrow Z'[i, k] \neq Z'[i+1, k]) \end{aligned}$$

Proof. For any $i \in \llbracket 1, S-1 \rrbracket$, any $j \in \llbracket 1, T \rrbracket$ and any $k \in \llbracket 1, N \rrbracket$:

$$\begin{aligned} (k \in \{t_{j,1}, t_{j,2}, t_{j,3}\} \Leftrightarrow Z[i, k] \neq Z[i+1, k]) & \quad (1) \\ \Leftrightarrow (\phi^{-1}(k) \in \{t_{j,1}, t_{j,2}, t_{j,3}\} \Leftrightarrow Z[i, \phi^{-1}(k)] \neq Z[i+1, \phi^{-1}(k)]) & \quad (2) \\ \Leftrightarrow (k \in \{\phi(t_{j,1}), \phi(t_{j,2}), \phi(t_{j,3})\} \Leftrightarrow Z[i, \phi^{-1}(k)] \neq Z[i+1, \phi^{-1}(k)]) & \quad (3) \\ \Leftrightarrow (k \in \{\phi(t_{j,1}), \phi(t_{j,2}), \phi(t_{j,3})\} \Leftrightarrow Z'[i, k] \neq Z'[i+1, k]) & \quad (4) \\ \Leftrightarrow (k \in \{\phi(t_{j,\psi(1)}), \phi(t_{j,2}), \phi(t_{j,\psi(3)})\} \Leftrightarrow Z'[i, k] \neq Z'[i+1, k]) & \quad (5) \\ \Leftrightarrow (k \in \{t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3}\} \Leftrightarrow Z'[i, k] \neq Z'[i+1, k]) & \quad (6) \end{aligned}$$

Equivalency (2) holds because ϕ is bijective, so $\{\phi^{-1}(k)\}_{k \in \llbracket 1, N \rrbracket} = \llbracket 1, N \rrbracket$. Equivalency (3) holds because $(\phi^{-1}(k) \in \{t_{j,1}, t_{j,2}, t_{j,3}\}) \Leftrightarrow (k \in \{\phi(t_{j,1}), \phi(t_{j,2}), \phi(t_{j,3})\})$. Equivalency (4) holds because by definition of Z' , for all i and k , $Z[i, \phi^{-1}(k)] = Z'[i, k]$. Equivalency (5) holds because $(\psi(1), \psi(3)) = (1, 3)$ or $(3, 1)$, which implies $\{\phi(t_{j,\psi(1)}), \phi(t_{j,2}), \phi(t_{j,\psi(3)})\} = \{\phi(t_{j,1}), \phi(t_{j,2}), \phi(t_{j,3})\}$. Finally, Equivalency (6) holds because by definition of Σ' we have that $(\phi(t_{j,\psi(1)}), \phi(t_{j,2}), \phi(t_{j,\psi(3)})) = (t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3})$. \triangleleft

▷ Claim 13. For any $i \in \llbracket 1, S-1 \rrbracket$, and any $j \in \llbracket 1, T \rrbracket$:

$$(Z[i, t_{j,1}] \neq Z[i, t_{j,3}] \text{ and } Z[i, t_{j,2}] = 1) \Leftrightarrow (Z'[i, t'_{\chi(j),1}] \neq Z'[i, t'_{\chi(j),3}] \text{ and } Z'[i, t'_{\chi(j),2}] = 1)$$

Proof. For any $i \in \llbracket 1, S-1 \rrbracket$, and any $j \in \llbracket 1, T \rrbracket$:

$$\begin{aligned} (Z[i, t_{j,1}] \neq Z[i, t_{j,3}] \text{ and } Z[i, t_{j,2}] = 1) & \quad (7) \\ \Leftrightarrow (Z'[i, \phi(t_{j,1})] \neq Z'[i, \phi(t_{j,3})] \text{ and } Z'[i, \phi(t_{j,2})] = 1) & \quad (8) \\ \Leftrightarrow (Z'[i, \phi(t_{j,\psi(1)})] \neq Z'[i, \phi(t_{j,\psi(3)})] \text{ and } Z'[i, \phi(t_{j,2})] = 1) & \quad (9) \\ \Leftrightarrow (Z'[i, t'_{\chi(j),1}] \neq Z'[i, t'_{\chi(j),3}] \text{ and } Z'[i, t'_{\chi(j),2}] = 1) & \quad (10) \end{aligned}$$

Equivalency (8) holds because by definition of Z' , for all i and k , $Z[i, \phi^{-1}(k)] = Z'[i, k]$, so $Z[i, k] = Z'[i, \phi(k)]$. Equivalency (9) holds because $(\psi(1), \psi(3)) = (1, 3)$ or $(3, 1)$, so $(Z'[i, \phi(t_{j,1})] \neq Z'[i, \phi(t_{j,3})]) \Leftrightarrow (Z'[i, \phi(t_{j,\psi(1)})] \neq Z'[i, \phi(t_{j,\psi(3)})])$. Finally, Equivalency (10) holds because by definition of Σ' , we have that $(\phi(t_{j,\psi(1)}), \phi(t_{j,2}), \phi(t_{j,\psi(3)})) = (t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3})$. \triangleleft

We show that $(Z \text{ is a valid solution for } \Sigma) \Leftrightarrow (Z' \text{ is a valid solution for } \Sigma')$. By definition, $(Z \text{ is a valid solution for } \Sigma)$ is equivalent to the following property :

9:10 Zero-Knowledge Proof of Knowledge for Peg Solitaire

Property 1: $f = Z[1]$, $l = Z[S]$, and for any $i \in \llbracket 1, S-1 \rrbracket$, there exists $j \in \llbracket 1, T \rrbracket$ such that, parsing t_j as $(t_{j,1}, t_{j,2}, t_{j,3})$:

- for any $k \in \llbracket 1, N \rrbracket$, ($k \in \{t_{j,1}, t_{j,2}, t_{j,3}\} \Leftrightarrow Z[i, k] \neq Z[i+1, k]$), and
- $Z[i, t_{j,1}] \neq Z[i, t_{j,3}]$ and $Z[i, t_{j,2}] = 1$.

By Claim 11, 12 and 13, Property 1 is equivalent to the following property:

Property 2: $f' = Z'[1]$, $l' = Z'[S]$, and for any $i \in \llbracket 1, S-1 \rrbracket$, there exists $j \in \llbracket 1, T \rrbracket$ such that, parsing $t'_{\chi(j)}$ as $(t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3})$:

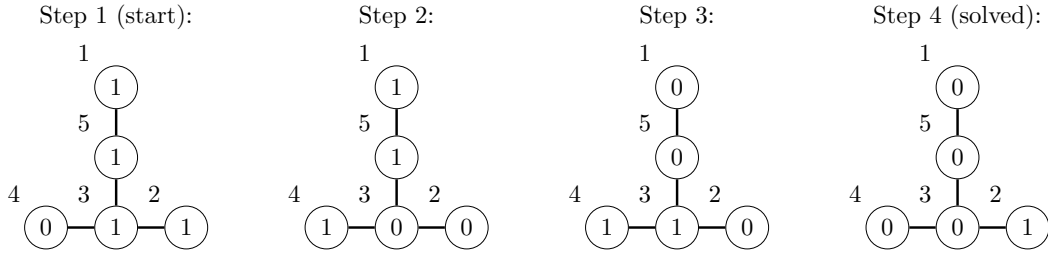
- for any $k \in \llbracket 1, N \rrbracket$, ($k \in \{t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3}\} \Leftrightarrow Z'[i, k] \neq Z'[i+1, k]$), and
- $Z'[i, t'_{\chi(j),1}] \neq Z'[i, t'_{\chi(j),3}]$ and $Z'[i, t'_{\chi(j),2}] = 1$.

χ is a bijection, so $\{\chi(j)\}_{j \in \llbracket 1, T \rrbracket} = \llbracket 1, T \rrbracket$, which implies that Property 2 is equivalent to the following property:

Property 3: $f' = Z'[1]$, $l' = Z'[S]$, and for any $i \in \llbracket 1, S-1 \rrbracket$, there exists $j \in \llbracket 1, T \rrbracket$ such that, parsing t'_j as $(t'_{j,1}, t'_{j,2}, t'_{j,3})$:

- for any $k \in \llbracket 1, N \rrbracket$, ($k \in \{t'_{j,1}, t'_{j,2}, t'_{j,3}\} \Leftrightarrow Z'[i, k] \neq Z'[i+1, k]$), and
- $Z'[i, t'_{j,1}] \neq Z'[i, t'_{j,3}]$ and $Z'[i, t'_{j,2}] = 1$.

Finally, by definition, Property 3 is equivalent to (Z' is a valid solution for Σ'), which concludes the proof. ◀



■ **Figure 5** The graph representation of the steps of the solution for a peg solitaire which is isomorphic to the tee peg solitaire, by the isomorphism given in Example 14.

► **Example 14.** Let $\xi = (\phi, \chi, \psi)$ be a $(5, 2, 4)$ -peg solitaire isomorphism defined by:

- For any $x \in \llbracket 1, N \rrbracket$, $\phi(x) = (x \bmod 5) + 1$.
- $\chi(1) = 2$ and $\chi(2) = 1$.
- $\psi(1) = 3$ and $\psi(3) = 1$.

The $(5, 2, 4)$ -peg solitaire instance Σ' isomorphic to the tee peg solitaire instance (given in Example 5) by ξ is defined by:

- $t' = ((1, 5, 3), (4, 3, 2))$.
- $f' = (1, 1, 1, 0, 1)$ and $l' = (0, 1, 0, 0, 0)$.

Using Theorem 10 on the valid solution for the tee peg solitaire (given in Example 7), we obtain the following valid solution Z' for Σ' :

$$Z' = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

We give the graph representation of this solution in Figure 5.

5 Protocol

In this section we present our proof protocol for the peg solitaire. This protocol has a sigma structure, it therefore consists of three interactions between a prover, knowing a solution Z for a (N, T, S) -peg solitaire instance Σ , and a verifier, knowing only Σ . We first give a high-level description of the protocol :

- **The prover** chooses an (N, T, S) -isomorphism ξ and computes the peg solitaire instance Σ' , isomorphic to Σ by the isomorphism ξ , together with a valid solution Z' of Σ' (using the method given in Theorem 10). The prover commits each part of Σ' and Z' , and sends the commitments to the verifier.
- **The verifier** chooses $\sigma \in \llbracket 1, S \rrbracket$ at random, and sends it to the prover.
- **The prover** computes the response in terms of σ :
 - **If** $\sigma = S$, then the prover reveals the isomorphism ξ and the peg solitaire instance Σ' to the verifier. The verifier checks the validity of the commitments of Σ' , and checks that Σ' is isomorphic to Σ by the isomorphism ξ .
 - **Else**, the prover reveals the integer $\gamma \in \llbracket 1, T \rrbracket$, which is the index of the triplet t'_γ in Σ' that contains the indexes of the vertices that change their values between step σ and step $(\sigma + 1)$ of the solution. Such an integer always exists by definition of a valid solution. The prover reveals t'_γ and reveals the lines σ and $(\sigma + 1)$ of the solution Z' (*i.e.* the vectors that correspond to the state of the peg solitaire before and after the σ^{th} move) to the verifier. The verifier checks the validity of the commitments of t'_γ and of the two revealed lines of Z' , and checks that the σ^{th} move is legal according to the triplet t'_γ .

If all the checks are valid, then the verifier accepts the transcript, else it rejects it.

A classical variant of peg solitaire consists in not imposing a final position, but only the number of pegs that must be removed from the board. We note that our protocol can be adapted to this (less constraining) version of the game simply by not revealing the vector of the final position in the case $\sigma = S$.

The formal definition of the protocol is given in Protocol 1. In Corollary 16, we show that this protocol is complete, valid, and zero-knowledge under the hypothesis that the commitment scheme has both perfect binding and hiding properties.

► **Protocol 1.** *Let \mathcal{P} be a prover knowing a solution Z for a (N, T, S) -peg solitaire instance $\Sigma = (t, f, l)$, and \mathcal{V} be a verifier knowing Σ . Our protocol uses a commitment scheme $C = (\text{Gen}, \text{Commit})$ and a security parameter λ . Our protocol is a sigma protocol having the following successive interactions, repeated λ times:*

- **The prover** \mathcal{P} chooses an isomorphism $\xi = (\phi, \chi, \psi)$ at random, and computes the (N, T, S) -peg solitaire instance $\Sigma' = (t', f', l')$, isomorphic to Σ by ξ . The prover \mathcal{P} computes a valid solution Z' for Σ' by using the method given in Theorem 10 on Σ, Z and ξ . For all $i \in \llbracket 1, T \rrbracket$, and all $j \in \llbracket 1, S \rrbracket$:
 - \mathcal{P} generates $\text{skt}_i \leftarrow \text{Gen}(1^\lambda)$ and computes the commitment $\hat{t}_i = \text{Commit}(\text{skt}_i, t'_i)$.
 - \mathcal{P} generates $\text{skz}_j \leftarrow \text{Gen}(1^\lambda)$ and computes $\hat{Z}_j = \text{Commit}(\text{skz}_j, Z'[j])$.
 Finally, \mathcal{P} sends $(\hat{t}_i)_{1 \leq i \leq T}$ and $(\hat{Z}_j)_{1 \leq j \leq S}$ to the verifier.
- **The verifier** \mathcal{V} chooses $\sigma \in \llbracket 1, S \rrbracket$ at random, and sends it to the prover.
- **The prover** \mathcal{P} computes the response in terms of σ :
 - **If** $\sigma = S$, then \mathcal{P} sends $\xi, (\text{skt}_i, t'_i)_{1 \leq i \leq T}, (\text{skz}_1, Z'[1]),$ and $(\text{skz}_S, Z'[S])$ to the verifier.
 - **Else**, the prover \mathcal{P} chooses the integer $\gamma \in \llbracket 1, T \rrbracket$, which is the index of the triplet t'_γ containing the indexes of the vertices that change their values between step σ and step

$(\sigma + 1)$ of the solution (such an integer always exists by definition of a valid solution).
 \mathcal{P} then sends $(\text{skt}_\gamma, t'_\gamma)$, $(\text{skz}_\sigma, Z'[\sigma])$ and $(\text{skz}_{\sigma+1}, Z'[\sigma + 1])$ to the verifier.

- **The verifier's verification depends on the value of σ :**
 - **If $\sigma = S$, then \mathcal{V} verifies the following commitments:**
 - * For $i \in \llbracket 1, T \rrbracket$, \mathcal{V} checks that $\hat{t}_i = \text{Commit}(\text{skt}_i, t'_i)$,
 - * \mathcal{V} checks that $\hat{Z}_1 = \text{Commit}(\text{skz}_1, Z'[1])$.
 - * \mathcal{V} checks that $\hat{Z}_S = \text{Commit}(\text{skz}_S, Z'[S])$.

\mathcal{V} then checks that $((t'_i)_{1 \leq i \leq T}, Z'[1], Z'[S])$ is isomorphic to Σ by ξ .
 - **Else, \mathcal{V} verifies the following commitments:**
 - * \mathcal{V} checks that $\hat{t}_\gamma = \text{Commit}(\text{skt}_\gamma, t'_\gamma)$.
 - * \mathcal{V} checks that $\hat{Z}_\sigma = \text{Commit}(\text{skz}_\sigma, Z'[\sigma])$.
 - * \mathcal{V} checks that $\hat{Z}_{\sigma+1} = \text{Commit}(\text{skz}_{\sigma+1}, Z'[\sigma + 1])$.

\mathcal{V} then checks that:

 - * for any $k \in \llbracket 1, N \rrbracket$, $(k \in \{t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3}\} \Leftrightarrow Z'[\sigma, k] \neq Z'[\sigma + 1, k])$, and
 - * $Z'[\sigma, t'_{\gamma,1}] \neq Z'[\sigma, t'_{\gamma,3}]$ and $Z'[\sigma, t'_{\gamma,2}] = 1$.

If all the checks are valid, then \mathcal{V} accepts the transcript, else \mathcal{V} rejects it.

► **Theorem 15.** *If the sigma protocol given in Protocol 1 is instantiated with a commitment scheme having the hiding and the perfect binding properties, then it is complete, S -special-sound, and (computational) zero-knowledge, where S is the number of steps in the solution of the peg solitaire instance.*

As mentioned in Section 2, if a sigma protocol repeated λ times is n -special-sound, then it is a valid proof of knowledge whose the knowledge error is $((n - 1)/n)^\lambda$, which implies the following corollary.

► **Corollary 16.** *If Protocol 1 is instantiated with a commitment scheme having the hiding and the perfect binding properties, then it is a complete, valid, and (computational) zero-knowledge proof of knowledge whose the knowledge error is $((S - 1)/S)^\lambda$, where S is the number of steps in the solution of the peg solitaire instance and λ is the number of rounds of the protocol.*

In what follows, we prove Theorem 15.

Proof. First, let us present a sketch of the proof.

Completeness: The prover knows a valid solution for Σ , so it is able to compute a peg solitaire instance isomorphic to Σ and its valid solution using the method described in Theorem 10. Since the solution of the isomorphic instance is valid, the prover can always answer the verifier correctly.

S -Special-soundness: Assume that the prover is able to respond correctly whatever the challenge it receives for the same commitment. Since the commitment scheme has the perfect binding property, the prover opens its commitments in a consistent way (*i.e.* the same commitment is not valid for two different values). We have that:

- The response to the challenge S reveals an isomorphism ξ and a peg solitaire instance Σ' that is isomorphic to Σ by ξ .
- The response to each challenge $\sigma < S$ reveals one move of the valid solution Z' for Σ' . By putting all the moves of Z' together, we can extract the whole solution Z' . Using Theorem 10, we extract a valid solution Z for Σ from Z' and the isomorphism ξ .

Zero-knowledge: First, assume that the commitments perfectly hide the committed values.

- If the verifier chooses the challenge S , then it receives an isomorphism ξ randomly chosen, and the image of Σ by ξ .

In this case the verifier can simulate the protocol by choosing a random isomorphism.

- If the challenger chooses a challenge $\sigma < S$, then it receives a triplet $t'_\gamma = (x, y, z)$ of Σ' and two successive vectors of the solution Z' . The index of the triplet depends on the unknown random permutation χ , and seems to be randomly chosen in $\llbracket 1, T \rrbracket$ from the verifier point of view. Since the indexes of the nodes are shuffled by the unknown random permutation ϕ , the triplet (x, y, z) contains three different integers in $\llbracket 1, N \rrbracket$ that seem randomly chosen from the verifier point of view. Let F be the number of 1 in the vector f of Σ ; since at each step of the solution only one peg is removed from the board, the number of 1 in the first revealed vector of the solution is $F + \sigma - 1$. Note that this property does not depend on the valid solution, and does not need to be hidden from the verifier. Except for the bits indexed by x, y and z , the 1 seem to be randomly distributed in the vector from the verifier point of view, because the indexes of the nodes are shuffled by the unknown random permutation ϕ . According to the verification requirements, the three bits indexed by x, y and z have a specific structure in the two vectors: they must match one of the two positions ($\bullet\bullet\circ$) or ($\circ\bullet\bullet$) in the first vector, and ($\circ\circ\bullet$) or ($\bullet\circ\circ$) respectively in the second one. The chosen position depends on the unknown random permutation ψ and seems to be randomly chosen from the verifier point of view. The other bits are the same in the two vectors. In this case the verifier can simulate the protocol by choosing the index of a triplet in $\llbracket 1, T \rrbracket$ at random, choosing three random indexes in $\llbracket 1, N \rrbracket$ at random, putting the bits of these three indexes as in one of the two legal positions (chosen at random) in the two vectors of the solution, and choosing the joint positions of the other 1 in the two vectors at random.

In both case, the verifier simulates the unrevealed commitments by committing a random bit-string, so the real protocol is perfectly simulated. Now, consider that the commitments do not perfectly hide the committed values. In this case, the zero-knowledge property depends on the ability of the verifier to distinguish which value is hidden in a commitment. Zero-knowledge is thus ensured by the hiding property of the commitment scheme.

We now give the technical details for the proof of each propertie.

▷ **Claim 17.** The protocol is complete.

Proof. The honest prover knows the (N, T, S) -isomorphism ξ and the committed (N, T, S) -peg solitaire instance Σ' , which is isomorphic to Σ by ξ . It also knows the committed valid solution Z' for Σ' . We recall that according to Theorem 10, a honest prover knowing Z, Σ and ξ is able to efficiently generate a valid solution Z' for Σ' , because it is isomorphic to Σ by ξ .

If the challenge is S , then it is clear that the verifier accepts the proof of an honest prover revealing its isomorphism.

By definition of a valid solution (Definition 6), if $Z' = (t', f', l')$ is a solution for Σ' , then for any $\sigma \in \llbracket 1, S - 1 \rrbracket$, there exists $\gamma \in \llbracket 1, T \rrbracket$ such that, parsing t'_γ as $(t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3})$:

- For any $k \in \llbracket 1, N \rrbracket$, $(k \in \{t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3}\} \Leftrightarrow Z'[\sigma, k] \neq Z'[\sigma + 1, k])$.
- $Z'[\sigma, t'_{\gamma,1}] \neq Z'[\sigma, t'_{\gamma,3}]$ and $Z'[\sigma, t'_{\gamma,2}] = 1$.

Thus, for any challenge $\sigma \in \llbracket 1, S - 1 \rrbracket$, by choosing the index γ , the honest prover convinces the verifier, which concludes the proof of completeness. ◁

▷ **Claim 18.** If the commitment scheme C has the perfect binding property, then the protocol is S -special-sound.

9:14 Zero-Knowledge Proof of Knowledge for Peg Solitaire

Proof. Assume that a prover gives an accepted response for any challenge $\sigma \in \llbracket 1, S \rrbracket$ on the same commitment. Since the commitment scheme is assumed to have the binding property, the same commitment cannot be opened in several different ways in the S responses of the prover. We show how to build a polynomial time knowledge extractor that computes a valid solution Z for Σ from these transcripts.

Since all responses are accepted, all checks required during the protocol are valid, which implies that:

- $\Sigma' = ((t'_i)_{1 \leq i \leq T}, Z'[1], Z'[S])$ is isomorphic to Σ by $\xi = (\phi, \chi, \psi)$.
- for each challenge $\sigma \in \llbracket 1, S-1 \rrbracket$, the prover reveals an index $\gamma \in \llbracket 1, T \rrbracket$ such that, parsing t'_γ as $(t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3})$:
 - For any $k \in \llbracket 1, N \rrbracket$, $(k \in \{t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3}\} \Leftrightarrow Z'[\sigma, k] \neq Z'[\sigma+1, k])$.
 - $Z'[\sigma, t'_{\gamma,1}] \neq Z'[\sigma, t'_{\gamma,3}]$ and $Z'[\sigma, t'_{\gamma,2}] = 1$.

By definition of a valid solution (Definition 6), we deduce that Z' is a valid solution for Σ' .

Let $Z = (Z[i, j])_{1 \leq i \leq S; 1 \leq j \leq N}$ be a solution for Σ such that, for any $i \in \llbracket 1, S \rrbracket$ and $j \in \llbracket 1, N \rrbracket$, $Z[i, j] = Z'[i, \phi(j)]$. Our knowledge extractor computes and returns this solution Z . We have that for any $i \in \llbracket 1, S \rrbracket$ and $j \in \llbracket 1, N \rrbracket$, $Z'[i, j] = Z[i, \phi^{-1}(j)]$, we deduce that Z is a valid solution for Σ according to Theorem 10, which concludes the proof. \triangleleft

\triangleright **Claim 19.** If the commitment scheme C has the hiding property, then the protocol is zero-knowledge.

Proof. In what follows, by abuse of language, we use the expression "*pick in X at random*" for "*pick from the uniform distribution on X* ". Let \mathcal{V}^* be a p.p.t verifier, Σ be a (N, T, S) -peg solitaire instance, and Z be a valid solution for Σ . Let str be a bit-string and p be a polynomial. We define the simulator $\text{Sim}(\Sigma)$ for \mathcal{V}^* as follows, where $\Sigma = (t, f, l)$ is a (N, T, S) -peg solitaire instance.

The simulator. Sim picks $\sigma \in \llbracket 1, S \rrbracket$ at random. We distinguish the two following cases:

- If $\sigma = S$, Sim chooses an isomorphism $\xi = (\phi, \chi, \psi)$ at random, and computes the (N, T, S) -peg solitaire instance $\Sigma' = (t', f', l')$ isomorphic to Σ by ξ . Sim then generates the following commitments:
 - For all $i \in \llbracket 1, T \rrbracket$, Sim generates $\text{skt}_i \leftarrow \text{Gen}(1^\lambda)$ and computes the commitment $\widehat{t}_i = \text{Commit}(\text{skt}_i, t'_i)$.
 - Sim generates $\text{skz}_1 \leftarrow \text{Gen}(1^\lambda)$ and computes $\widehat{Z}_1 = \text{Commit}(\text{skz}_1, f'_1)$.
 - For all $j \in \llbracket 2, S-1 \rrbracket$, Sim generates $\text{skz}_j \leftarrow \text{Gen}(1^\lambda)$ and computes $\widehat{Z}_j = \text{Commit}(\text{skz}_j, \text{str})$.
 - Sim generates $\text{skz}_S \leftarrow \text{Gen}(1^\lambda)$ and computes $\widehat{Z}_S = \text{Commit}(\text{skz}_S, l'_S)$.

Finally, Sim sets the prover commitment as $\text{com} = ((\widehat{t}_i)_{1 \leq i \leq T}, (\widehat{Z}_j)_{1 \leq j \leq S})$, the verifier challenge as σ , and the prover response as $\text{res} = (\xi, (\text{skt}_i, t'_i)_{1 \leq i \leq T}, (\text{skz}_1, f'_1), (\text{skz}_S, l'_S))$.

- Else, Sim picks $\gamma \in \llbracket 1, T \rrbracket$ and picks a triplet $t'_\gamma = (t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3}) \in \llbracket 1, N \rrbracket^3$ at random that verifies $t'_{\gamma,1} \neq t'_{\gamma,2}$, $t'_{\gamma,2} \neq t'_{\gamma,3}$ and $t'_{\gamma,1} \neq t'_{\gamma,3}$. Let F be the number of 1 in the vector f of Σ . Sim picks a bit b at random, then Sim picks a vector $Z'[\sigma]$ of N bits at random such that $Z'[\sigma]$ verifies the following properties:
 - $Z'[\sigma]$ contains exactly $F + \sigma - 1$ times the bit 1, and
 - $Z'[\sigma, t'_{\gamma,1}] = b$, $Z'[\sigma, t'_{\gamma,2}] = 1$, and $Z'[\sigma, t'_{\gamma,3}] = 1 - b$. Sim sets a vector $Z'[\sigma+1]$ of N bits such that for all $k \in \llbracket 1, N \rrbracket$, $(k \in \{t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3}\} \Leftrightarrow Z'[\sigma, k] \neq Z'[\sigma+1, k])$. Sim then generates the following commitments:
 - For all $i \in \llbracket 1, T \rrbracket \setminus \{\gamma\}$, Sim generates $\text{skt}_i \leftarrow \text{Gen}(1^\lambda)$ and computes the commitment $\widehat{t}_i = \text{Commit}(\text{skt}_i, \text{str})$.

- Sim generates $\text{skt}_\gamma \leftarrow \text{Gen}(1^\lambda)$ and computes the commitment $\hat{t}_\gamma = \text{Commit}(\text{skt}_\gamma, t'_\gamma)$.
 - For all $j \in \llbracket 1, S \rrbracket \setminus \{\sigma, \sigma + 1\}$, Sim generates $\text{skz}_j \leftarrow \text{Gen}(1^\lambda)$ and computes $\hat{Z}_j = \text{Commit}(\text{skz}_j, \text{str})$.
 - Sim generates $\text{skz}_\sigma \leftarrow \text{Gen}(1^\lambda)$ and computes $\hat{Z}_\sigma = \text{Commit}(\text{skz}_\sigma, Z'[\sigma])$.
 - Sim generates $\text{skz}_{\sigma+1} \leftarrow \text{Gen}(1^\lambda)$ and computes $\hat{Z}_{\sigma+1} = \text{Commit}(\text{skz}_{\sigma+1}, Z'[\sigma + 1])$.
- Finally, Sim sets the prover commitment as $\text{com} = ((\hat{t}_i)_{1 \leq i \leq T}, (\hat{Z}_j)_{1 \leq j \leq S})$, the verifier challenge as σ , and the prover response as $\text{res}((\text{skt}_\gamma, t'_\gamma), (\text{skz}_\sigma, Z'[\sigma]), (\text{skz}_{\sigma+1}, Z'[\sigma + 1]))$.

Finally, Sim runs a proof protocol with $\mathcal{V}^*(\Sigma)$, sends com , receives a challenge σ' , and aborts the protocol. If $\sigma' = \sigma$, then Sim returns $(\text{com}, \sigma, \text{res})$, else, the simulator starts all over again with a new random σ . After $p(\lambda)$ unsuccessful tries, Sim returns the failure symbol \perp .

We show that no p.p.t distinguisher \mathcal{D} is able to distinguish between real transcripts from the protocol and transcripts simulated by Sim. We use a sequence of game between \mathcal{D} and a p.p.t challenger \mathcal{C} as described by Shoup in [17]. In what follows, $\epsilon(\lambda)$ denotes the probability of the best algorithm attacking the hiding property of the commitment scheme (which is assumed to be negligible).

Game \mathbf{G}_0 : In this game, the challenger \mathcal{C} runs the real protocol $\text{tr} \leftarrow \langle \mathcal{P}(Z), \mathcal{V}^*(\Sigma) \rangle$, and runs $b \leftarrow \mathcal{D}(\text{tr})$.

Game \mathbf{G}_1 : We define the following algorithm:

- $\mathcal{P}_1(Z)$ is the same algorithm as \mathcal{P} except that it starts by picking a challenge $\sigma \in \llbracket 1, S \rrbracket$ at random, then it aborts if the verifier chooses a challenge σ' such that $\sigma' = \sigma$.

In this game, the challenger \mathcal{C} runs the protocol $\text{tr} \leftarrow \langle \mathcal{P}_1(Z), \mathcal{V}^*(\Sigma) \rangle$ until the prover does not abort. If \mathcal{P}_1 aborts $p(\lambda)$ times successively, then \mathcal{C} sets $\text{tr} = \perp$. Since \mathcal{P}_1 aborts with probability $(S - 1)/S$, the challenger \mathcal{C} sets $\text{tr} = \perp$ with probability $((S - 1)/S)^{p(\lambda)}$. We have that:

$$|\Pr[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G}_0] - \Pr[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G}_1]| \leq \Pr[\mathcal{C} \text{ sets } \text{tr} = \perp \text{ in } \mathbf{G}_1] = \left(\frac{S-1}{S}\right)^{p(\lambda)}.$$

We note that from this game, **the prover knows in advance (i.e., before generating the commitments) which commitments will be revealed and which will not**. We also recall that the prover generates $S + T$ commitments during the protocol.

Game $\mathbf{G}_{1,i}$ (for $0 \leq i \leq S + T$): We define the following algorithm:

- $\mathcal{P}_{1,i}(Z)$ is the same algorithm as \mathcal{P}_1 except that for the first i commitments that will not be revealed to the verifier in the response phase, the prover $\mathcal{P}_{1,i}$ commits the value str instead of the value that is committed in the real protocol.

This game is the same as \mathbf{G}_1 except that it uses $\mathcal{P}_{1,i}$ instead of \mathcal{P}_1 . Clearly, $\mathbf{G}_1 = \mathbf{G}_{1,0}$. We claim that, for all $i \in \llbracket 1, S + T - 1 \rrbracket$:

$$|\Pr[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G}_{1,i}] - \Pr[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G}_{1,i+1}]| \leq \epsilon(\lambda).$$

We prove this claim by reduction. We use \mathcal{D} to build a p.p.t algorithm \mathcal{A} that plays the experiment $\text{Exp}_{\mathcal{C}, \mathcal{A}, b'}^{\text{Hiding}}(\lambda)$. The algorithm \mathcal{A} plays the role of \mathcal{C} and simulates $\text{tr} \leftarrow \langle \mathcal{P}_{1,i}(Z), \mathcal{V}^*(\Sigma) \rangle$ exactly as in $\mathbf{G}_{1,i}$, except that \mathcal{A} sets m_0 as the value that is committed in the $(i + 1)^{\text{th}}$ unrevealed commitment produced by $\mathcal{P}_{1,i}$, sets $m_1 = \text{str}$, sends (m_0, m_1) to the experiment $\text{Exp}_{\mathcal{C}, \mathcal{A}, b'}^{\text{Hiding}}(\lambda)$, receives the commitment $c_{b'}$ of $m_{b'}$, and replaces the $(i + 1)^{\text{th}}$

unrevealed commitment of $\mathcal{P}_{1,i}$ by $c_{b'}$ in tr . (Obviously, \mathcal{C} replaces this commitment in the transcript of the protocol instance that did not abort). Finally, \mathcal{A} runs $b \leftarrow \mathcal{D}(\text{tr})$ and returns b to the experiment. We have that:

$$\begin{aligned} & |\Pr[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G}_{1,i}] - \Pr[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G}_{1,i+1}]| = \\ & \left| \Pr \left[0 \leftarrow \mathbf{Exp}_{\mathcal{C},\mathcal{A},0}^{\text{Hiding}}(\lambda) \right] - \Pr \left[1 \leftarrow \mathbf{Exp}_{\mathcal{C},\mathcal{A},1}^{\text{Hiding}}(\lambda) \right] \right| \leq \epsilon(\lambda), \end{aligned}$$

which concludes the proof of the claim. We note that from the game $\mathbf{G}_{1,T+S}$, **all the commitments that are not revealed to the verifier commit the value str as in our simulator.**

Game \mathbf{G}_2 : In this game, \mathcal{C} runs $\text{tr} \leftarrow \mathcal{S}(\Sigma)$ and $b \leftarrow \mathcal{D}(\text{tr})$. We have that $\mathbf{G}_{1,T+S} = \mathbf{G}_2$ (the justification of this equivalence has already been given in the sketch of the proof). Finally, using each of these game hops, we deduce that:

$$\begin{aligned} & |\Pr[\text{tr} \leftarrow \langle \mathcal{P}(Z), \mathcal{V}^*(\Sigma) \rangle; b \leftarrow \mathcal{D}(\text{tr}) : b = 1] - \Pr[\text{tr} \leftarrow \text{Sim}(\Sigma); b \leftarrow \mathcal{D}(\text{tr}) : b = 1]| \\ & = |\Pr[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G}_0] - \Pr[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G}_2]| \leq (S + T)\epsilon(\lambda) + \left(\frac{S-1}{S} \right)^{p(\lambda)}. \end{aligned}$$

Since ϵ is negligible and p is a polynomial, this value is negligible, which concludes the proof. \triangleleft

6 Conclusion

In this paper, we have given a tailor-made zero-knowledge proof of knowledge for the peg solitaire, which does not use any reduction to another NP-complete problem. Our proof can be used physically by using envelopes instead of commitments. We also define the notion of peg solitaire isomorphism, which is of independent interest. In the future, it would be interesting to reduce the soundness error of our protocol, either by reducing the number of challenges in the cryptographic version of the protocol, or by using techniques that are specific to the physical proof protocols, such as the one introduced in [16]. Another possible future work would be to extend our protocol to the many variants of the peg solitaire, and more generally, to other single-player board games.

References

- 1 David Avis and Antoine Deza. On the solitaire cone and its relationship to multi-commodity flows. *Mathematical Programming*, 90:27–57, March 2001. doi:10.1007/PL00011419.
- 2 Robert A. Beeler and D. Paul Hoilman. Peg solitaire on graphs. *Discrete Mathematics*, 311(20):2198–2202, 2011. doi:10.1016/j.disc.2011.07.006.
- 3 Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 390–420, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.
- 4 Xavier Bultel, Jannik Dreier, Jean-Guillaume Dumas, and Pascal Lafourcade. Physical Zero-Knowledge Proofs for Akari, Takuzu, Kakuro and KenKen. In Erik D. Demaine and Fabrizio Grandoni, editors, *8th International Conference on Fun with Algorithms (FUN 2016)*, volume 49 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:20, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FUN.2016.8.

- 5 Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, pages 174–187, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.
- 6 Ivan Damgård. *Commitment Schemes and Zero-Knowledge Protocols*, pages 63–86. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. doi:10.1007/3-540-48969-X_3.
- 7 Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, July 1991. doi:10.1145/116825.116852.
- 8 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. Association for Computing Machinery. doi:10.1145/22145.22178.
- 9 Ronen Gradwohl, Moni Naor, Benny Pinkas, and Guy N. Rothblum. Cryptographic and physical zero-knowledge proof systems for solutions of sudoku puzzles. In *Proceedings of the 4th International Conference on Fun with Algorithms*, FUN'07, pages 166–182, Berlin, Heidelberg, 2007. Springer-Verlag.
- 10 Luciano Gualà, Stefano Leucci, Emanuele Natale, and Roberto Tauraso. Large Peg-Army Maneuvers. In Erik D. Demaine and Fabrizio Grandoni, editors, *8th International Conference on Fun with Algorithms (FUN 2016)*, volume 49 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FUN.2016.18.
- 11 Christopher Jefferson, Angela Miguel, Ian Miguel, and S. Armagan Tarim. Modelling and solving english peg solitaire. *Computers & Operations Research*, 33(10):2935–2959, 2006. Part Special Issue: Constraint Programming. doi:10.1016/j.cor.2005.01.018.
- 12 Masashi Kiyomi and Tomomi Matsui. Integer programming based algorithms for peg solitaire problems. In Tony Marsland and Ian Frank, editors, *Computers and Games*, pages 229–240, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- 13 Pascal Lafourcade, Daiki Miyahara, Takaaki Mizuki, Léo Robert, Tatsuya Sasaki, and Hideaki Sone. How to construct physical zero-knowledge proofs for puzzles with a “single loop” condition. *Theoretical Computer Science*, 888:41–55, 2021. doi:10.1016/j.tcs.2021.07.019.
- 14 Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.
- 15 Bala Ravikumar. Peg-solitaire, string rewriting systems and finite automata. *Theor. Comput. Sci.*, 321(2–3):383–394, August 2004. doi:10.1016/j.tcs.2004.05.005.
- 16 Tatsuya Sasaki, Daiki Miyahara, Takaaki Mizuki, and Hideaki Sone. Efficient card-based zero-knowledge proof for sudoku. *Theoretical Computer Science*, 839:135–142, 2020. doi:10.1016/j.tcs.2020.05.036.
- 17 Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. URL: <https://ia.cr/2004/332>.
- 18 Ryuhei Uehara and Shigeki Iwata. Generalized hi-q is np-complete. *Transactions of the IEICE*, 1990.
- 19 Emmanuel Volte, Jacques Patarin, and Valérie Nachev. Zero knowledge with rubik’s cubes and non-abelian groups. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *Cryptology and Network Security*, pages 74–91, Cham, 2013. Springer International Publishing.

Nimber-Preserving Reduction: Game Secrets And Homomorphic Sprague-Grundy Theorem

Kyle W. Burke ✉

Department of Computer Science, Plymouth State University, NH, USA

Matthew Ferland ✉

Department of Computer Science, University of Southern California, Los Angeles, CA, USA

Shang-Hua Teng ✉

Department of Computer Science, University of Southern California, Los Angeles, CA, USA

Abstract

The concept of *nimbers* – a.k.a. *Grundy-values* or *nim-values* – is fundamental to combinatorial game theory. Beyond the winnability, nimbers provide a complete characterization of strategic interactions among impartial games in disjunctive sums. In this paper, we consider *nimber-preserving* reductions among impartial games, which enhance the *winnability-preserving reductions* in traditional computational characterizations of combinatorial games. We prove that GENERALIZED GEOGRAPHY is complete for the natural class, \mathcal{I}^P , of *polynomially-short* impartial rulesets, under polynomial-time nimber-preserving reductions. We refer to this notion of completeness as *Sprague-Grundy-completeness*. In contrast, we also show that not every PSPACE-complete ruleset in \mathcal{I}^P is Sprague-Grundy-complete for \mathcal{I}^P .

By viewing every impartial game as an encoding of its nimber – a succinct game secret richer than its winnability alone – our technical result establishes the following striking cryptography-inspired homomorphic theorem: Despite the PSPACE-completeness of nimber computation for \mathcal{I}^P , there exists a polynomial-time algorithm to construct, for any pair of games G_1, G_2 in \mathcal{I}^P , a GENERALIZED GEOGRAPHY game G satisfying:

$$\text{nimber}(G) = \text{nimber}(G_1) \oplus \text{nimber}(G_2).$$

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Combinatorial Games, Nim, Generalized Geography, Sprague-Grundy Theory, Grundy value, Computational Complexity, Functional-Preserving Reductions

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.10

Related Version *Full Version*: <https://arxiv.org/abs/2109.05622>

Funding *Shang-Hua Teng*: Supported by the Simons Investigator Award for fundamental & curiosity-driven research and NSF grant CCF-1815254.

1 Introduction

Mathematical games are fun and intriguing. Even with *succinct* rulesets (which define game positions and the players' options from each position), they can grow *game trees* of size *exponential* in that of the starting positions. A game is typically formulated for two players. They take turns strategically selecting from the current options to move the game state to the next position. In the *normal-play convention*, the player that faces a *terminal position* – a position with no feasible options – loses the game. The *game tree* from a starting position – with the leaves as the terminal positions – naturally captures this alternation of all potential feasible moves.



© Kyle W. Burke, Matthew Ferland, and Shang-Hua Teng;
licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 10; pp. 10:1–10:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

10:2 Nimber-Preserving Reductions

Over the years, rulesets have been formulated based on graph theory [4, 24, 11], logic [28], topology [25, 19, 27, 30, 10], and other mathematical fields, often inspired by real-world phenomena [21, 1, 20, 13, 32, 8, 2, 33]. These rulesets distill fundamental mathematical concepts, structures, and dynamics. For example:

- **NODE KAYLES** [28] models a strategic game of growing a maximal independent set: Each position is an undirected graph, and each move consists of removing a node and its neighbors.
- **GENERALIZED GEOGRAPHY** [28, 24] models a two-player game of traversing maximal paths: Each position is defined by a token in a directed graph and a move consists of removing the current vertex from the graph and moving the token to an out-neighbor. (It is often just referred to as **GEOGRAPHY**.)
- **ATROPOS** [10] models the dynamic formation of discrete equilibria (panchromatic triangles) in topological maps: Positions are partially colored Sperner triangles, and a move consists of coloring a vertex.

The deep alternation of strategic reasoning also intrinsically connects optimal play in many games to highly intractable complexity classes, most commonly PSPACE. After Even and Tarjan proved this for a generalization of Nash's HEX [14], deciding *winnability* of many natural combinatorial games – including **NODE-KAYLES**, **GENERALIZED GEOGRAPHY**, **AVOID TRUE**, **PROPER-K-COLORING**, **ATROPOS**, **GRAPH NIM**, and **GENERALIZED CHOMP**¹ – have been shown to be PSPACE-complete [28, 24, 11, 3, 10, 30, 22].

1.1 A Classical Mathematical Theory for Impartial Games

Mathematical characterizations of combinatorial games emerged prior to the age of modern computational complexity theory. In 1901, Bouton [5] developed a complete theory for **NIM**, based on an ancient Chinese game *Jian Shi Zi* (撿石子 - picking stones). A **NIM** position is a collection of piles of (small) stones. On their turn, a player takes one or more stones from exactly one of the piles. Representing each **NIM** position by a list of integers, Bouton [5] proved that the current player has a winning strategy in the normal-play setting if and only if the *bitwise-exclusive-or* of these integers (as *binary representations*) is not zero. Note that although the game tree of a **NIM** position could be exponentially tall in the number of bits representing the position, Bouton's characterization provides a polynomial-time solution for determining the winnability of **NIM** games.

NIM is an example of an *impartial* ruleset, meaning both players have the same options at every position. Games that aren't impartial are known as *partisan*. The two graph games, **NODE-KAYLES** and **GENERALIZED GEOGRAPHY** aforementioned, are also impartial.

In the 1930s, Sprague [31] and Grundy [23] independently developed a comprehensive mathematical theory for impartial games. They introduced a notion of *equivalence* among games, characterizing their contributions in the disjunctive sums with other impartial games. Extending Bouton's theory for **NIM**, Sprague-Grundy Theory provides a complete mathematical solution to the disjunctive sums of impartial games.

► **Definition 1** (Disjunctive Sum). *For any two game positions G and H (respectively, of rulesets R_1 and R_2), their disjunctive sum, $G + H$, is a game position in which at each turn, the current player chooses to make a move in exactly one of G and H , leaving the other unchanged. A sum position $G + H$ is terminal if and only if both G and H are terminal according to their own rulesets.*

¹ We consider **GENERALIZED CHOMP** to be **CHOMP**, but on any Directed Acyclic Graph. This is equivalent to **FINITE ARBITRARY POSET GAME**, when the partial order can be evaluated in polynomial time.

Sprague and Grundy showed that every impartial game G can be equivalently replaced by a single-pile Nim game in any disjunctive sum involving G . Thus, they characterized each impartial game by a natural number – now known as the *number*, *Grundy value*, or *nim-value* – which corresponds to a number of stones in a single-pile of Nim. Mathematically, the number of G , which we denote by $\text{number}(G)$, can be recursively formulated via G 's game tree: (1) if G is terminal, then $\text{number}(G) = 0$; otherwise (2) if $\{G_1, \dots, G_k\}$ is the set options of G , letting mex returns the smallest value of $(\mathbb{Z}^+ \cup \{0\}) \setminus \{\text{number}(G_1), \dots, \text{number}(G_t)\}$, then:

$$\text{number}(G) = \text{mex}(\{\text{number}(G_1), \dots, \text{number}(G_k)\}). \quad (1)$$

Let \oplus denote the *bitwise xor (nim-sum)*. By Bouton's NIM theory [5]:

$$\text{number}(G + H) = \text{number}(G) \oplus \text{number}(H) \quad \forall \text{ impartial } G, H. \quad (2)$$

Thus, Sprague-Grundy Theory – using Bouton's NIM solution – provides an instrumental mathematical summary (of the much larger game trees) that enhances the winnability for impartial games: A position is a winning position if and only if its number is non-zero. This systematic framework inspired subsequent work in the field, including Berlekamp, Conway, and Guy's *Winning Ways for Your Mathematical Plays* [4], and Conway's *On Numbers And Games* [11], which laid the foundation for Combinatorial Game Theory (CGT). This 1930s theory also has an algorithmic implication. Equation (2) provides a polynomial-time framework for computing the number of a sum game – and the hence the winnability – from the numbers of its component games: If the numbers of two games G and H are tractable, then $\text{number}(G + H)$ is also tractable.

1.2 Our Main Contributions

Obviously, in spite of this algorithmic implication, Sprague-Grundy Theory does not provide a general-purpose polynomial-time solution for all impartial games, as witnessed by many PSPACE-hard rulesets, including NODE KAYLES and GENERALIZED GEOGRAPHY [28, 24]. If one views the number characterization of an impartial game as a reduction from that game to a single pile NIM, then Schaefer *et al*'s complexity results demonstrate that this reduction has intractable constructability. In fact, a recent result [9] proved that the number of polynomial-time solvable UNDIRECTED GEOGRAPHY – i.e., GENERALIZED GEOGRAPHY on undirected graphs – is also PSPACE-complete to compute. The sharp contrast between the complexity of winnability and number computation illustrates a fundamental mathematical-computational divergence in Sprague-Grundy Theory [9]: Numbers can be PSPACE-hard “*secrets of deep alternation*” even for polynomial-time solvable games.

Computational complexity theory often gives new perspectives of classical mathematical results. In this work, it also provides us with a new lens for understanding this classical mathematical characterization as well as tools for exploring and identifying new fundamental characterizations in combinatorial game theory.

1.2.1 Polynomial-Time Nimber-Preserving Reduction to Generalized Geography

In this paper, we consider the following natural concept of reduction among impartial games.

► **Definition 2** (Nimber-Preserving Reduction). ² A map ϕ is a *nimber-preserving reduction* from impartial ruleset R_1 to impartial ruleset R_2 if for every position G of R_1 , $\phi(G)$ is a position of R_2 satisfying $\text{nimber}(G) = \text{nimber}(\phi(G))$.

Because an impartial position is a losing position if and only if its nimber is zero, nimber-preserving reductions enhance *winnability-preserving reductions* in traditional complexity-theoretical characterizations of combinatorial games [14, 28]: Polynomial-time nimber-preserving reductions introduce the following natural notion of “universal” impartial rulesets.

► **Definition 3** (Sprague-Grundy Completeness). For a family \mathcal{J} of impartial rulesets, we say $R \in \mathcal{J}$ is a *Sprague-Grundy-complete ruleset for \mathcal{J}* if for any position Z of any ruleset of \mathcal{J} , one can construct, in polynomial time, a position $G \in R$ such that $\text{nimber}(G) = \text{nimber}(Z)$.

As the main technical contribution of this paper, we prove the following theorem regarding the expressiveness of GENERALIZED GEOGRAPHY. The natural family of rulesets containing GENERALIZED GEOGRAPHY is \mathcal{I}^P , the family of all impartial rulesets whose positions have game trees with height polynomial in the sizes of the positions. We call games of \mathcal{I}^P *polynomially-short* games. In addition to GENERALIZED GEOGRAPHY, \mathcal{I}^P contains many combinatorial rulesets studied in the literature, including NODE KAYLES, CHOMP, PROPER-K-COLORING, ATROPOS, and AVOID TRUE, as well as NIM and GRAPH NIM with polynomial numbers of stones.

► **Theorem 4** (A Complete Geography). *GENERALIZED GEOGRAPHY is a Sprague-Grundy complete ruleset for \mathcal{I}^P .*

In other words, for example, given any NODE KAYLES or AVOID TRUE game, we can, in polynomial time, generate a GENERALIZED GEOGRAPHY game with the same Grundy value, despite the fact that the Grundy value of the input game could be intractable to compute.

Because nimber-preserving reductions generalize winnability-preserving reductions, every Sprague-Grundy complete ruleset for \mathcal{I}^P must be PSPACE-complete to solve. However, for a simple mathematical reason, we have the following observation:

► **Proposition 5.** *Unless $P = \text{PSPACE}$, not every PSPACE-complete ruleset in \mathcal{I}^P is Sprague-Grundy complete for \mathcal{I}^P .*

In particular, ATROPOS [10] is PSPACE-complete but not Sprague-Grundy-complete for \mathcal{I}^P . Thus, together Theorem 4 and Proposition 5 highlight the fundamental difference between winnability-preserving reductions and nimber-preserving reductions. Our result further illuminates the central role of *Generalized Geography* – a classical PSPACE-complete ruleset instrumental to Lichtenstein-Sipser’s PSPACE-hard characterization of GO [16] – in the complexity-theoretical understanding of combinatorial games.

In Section 5, when discussing related questions, we also demonstrate the brief corollary:

² This natural concept of reduction in combinatorial game theory can be viewed as the analog of *functional-preserving reductions* in various fields. To name a few: approximation-preserving, gap-preserving, structure-preserving reductions in complexity and algorithmic theory, hardness-preserving and security-preservation in cryptography, dimension-preserving, metric-preserving, and topology-preserving reductions in data analytics, parameter-preserving reductions in dynamic systems, counterexample-preserving reductions in model checking, query-number-preserving, sample-preserving and high-order-moment-preserving in statistical analysis, and modularity-preserving reductions in network modeling. We are inspired by several of these works.

► **Corollary 6.** *EDGE GEOGRAPHY (formulated in [28] and studied in [17], see Section 5.1 for the ruleset) is also Sprague-Grundy-complete for \mathcal{I}^P .*

1.2.2 Game Secrets: Homomorphic Sprague-Grundy Theorem

In the framework of disjunctive sums, every impartial game G encodes a secret, i.e., its Grundy value $\text{nimber}(G)$, which succinctly summarizes G 's game tree and can be represented by a single-pile NIM. Once this secret is obtained, by Sprague-Grundy theory, one can replace G by its equivalent single-pile NIM in any disjunctive sum involving G . Even though NIM is expressive enough to provide natural game representations of these game secrets, it does not admit an efficient reduction, even for polynomial-time solvable games, such as UNDIRECTED GEOGRAPHY, in \mathcal{I}^P .

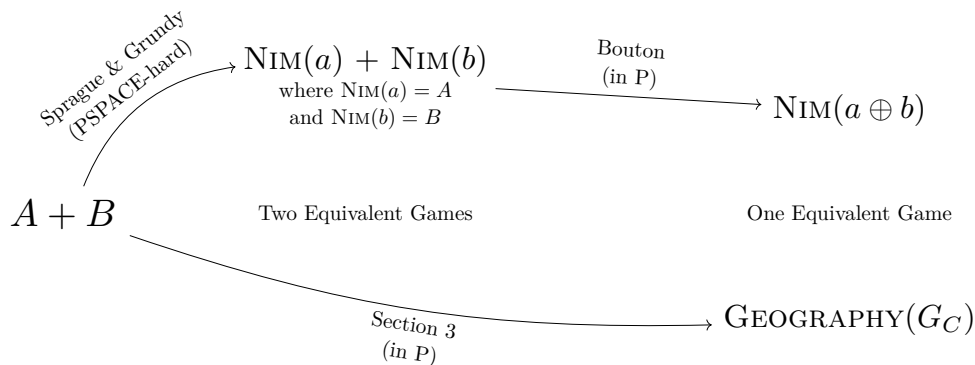
In contrast, for impartial games in \mathcal{I}^P , Theorem 4 shows that GENERALIZED GEOGRAPHY provides natural game representations of these game secrets. In conjunction with Sprague-Grundy theory, this GENERALIZED GEOGRAPHY-based encoding of numbers leads to a surprising cryptography-inspired homomorphic characterization of impartial games.

► **Theorem 7 (Homomorphic Sprague-Grundy-Bouton Theorem).** *\mathcal{I}^P enjoys the following two contrasting properties:*

- **Hard-Core Nimber Secret:** *The problem of computing the nimber – i.e., finding $\text{nimber}(G)$ given a position G of \mathcal{I}^P – is PSPACE-complete.*
- **Homomorphic Game Encoding:** *For any pair of positions G_1 and G_2 of \mathcal{I}^P , one can, in polynomial-time (in the sizes of G_1 and G_2), construct a GENERALIZED GEOGRAPHY game G , such that:*

$$\text{nimber}(G) = \text{nimber}(G_1) \oplus \text{nimber}(G_2).$$

Like the Sprague-Grundy theory – which represents the game values by natural games – Theorem 7 encodes “nimber secrets” with natural games. The former uses “single-pile” NIM and the later uses “single-graph” GENERALIZED GEOGRAPHY. For both, one can compute, in polynomial time, the representation of the disjunctive sum of any two representations. Because GENERALIZED GEOGRAPHY is not naturally closed under disjunctive sums, genuine computational effort – although feasible in polynomial-time – is required in the homomorphic encoding of the disjunctive sum.



■ **Figure 1** Two transformations of impartial games A and B into a single game equivalent to their disjunctive sum.

1.3 Remarks

This genuineness can be partially captured by Conway’s notion of *prime impartial games* in his studies of misère games. Let prime impartial games be ones that can’t be expressed as the disjunctive sum of any two other games (see Section 4.1 for the formal definition). Like the role of prime numbers in the multiplicative group over integers, these games are the basic building blocks in the disjunctive-sum-based monoid over impartial games.

In Theorem 20, we will prove that each GENERALIZED GEOGRAPHY game created in the proof of Theorem 7 is indeed a prime game. Thus, in Theorem 7, because G is a natural prime game, the algorithm for “Homomorphic Game Encoding” cannot trivially output the syntactic description of $G_1 + G_2$. Thus, it must use a more “elementary” position to encode $\text{nimber}(G_1) \oplus \text{nimber}(G_2)$. In other words, the bitwise-xor of the PSPACE-hard “nimber secrets” encoded in any two impartial games of \mathcal{I}^P can be efficiently re-encoded by a natural prime game of \mathcal{I}^P , whose game tree is not isomorphic to that of $G_1 + G_2$.

Conway’s notation of prime games only partially captures the genuineness of the homomorphic encoding in Theorem 7 because one may locally modify the game rule for some zero positions to make the sum game prime without changing the nimber. So the naturalness of GENERALIZED GEOGRAPHY captures more beyond the current concept of prime games. We continue to look for a more accurate characterization.

Note also that the characterization presented in Theorem 7 is cryptography-inspired rather than cryptographically-applicable. In *partially homomorphic encryption*, the encoding functions, such as RSA encryption and discrete-log, must be one-way functions. Here, we consider a game position as a “natural encoding” of its nimber-secret: The focus of Theorem 7 is on the complexity-based homomorphic property of this encoding (and that it arises naturally from impartial games) rather than the construction of homomorphic “one-way” game-based encryption of secret messages. (See Section, 5 – Conclusion and Open Questions – for more discussion on this.) Thus, in contrast to (partially) homomorphic cryptographic functions – such as discrete-log and RSA – whose secret messages can be recovered in NP (due to a one-way encoding of secrets), “decoding” the nimber-secrets – as they are naturally encoded in combinatorial games of \mathcal{I}^P – is PSPACE-complete in the worst case.

2 Impartial Games and Their Trees: Notation and Definitions

In this section, we review some background concepts and notation. In the paper, we use \mathcal{I} to denote the family of all impartial rulesets; we use \mathbb{I} to denote the family of all impartial games – positions – defined by rulesets in \mathcal{I} ; we use \mathbb{I}^P to denote the family of all polynomially-short impartial games, i.e., positions defined by rulesets in \mathcal{I}^P .

Each impartial ruleset $R \in \mathcal{I}$ has two mathematical components (\mathcal{B}_R, ρ_R) , where \mathcal{B}_R represents the set of all possible game positions in R and $\rho_R : \mathcal{B}_R \rightarrow 2^{\mathcal{B}_R}$ defines the options for each position in R . For each position $G \in \mathcal{B}_R$, the ruleset R defines a natural *game tree*, T_G , rooted at G . T_G recursively branches with feasible options. The root is associated with position G itself. The number of children that the root has is equal to the number of options, i.e., $|\rho_R(G)|$, at G . Each child is associated with a position from $\rho_R(G)$ and its sub-game-tree is defined recursively. Thus, T_G contains all *reachable positions* of G under ruleset R . The leaves of T_G are terminal positions under the normal-play setting. Up to isomorphism, the game tree for an impartial game is unique.

► **Definition 8** (Game isomorphism). *Two impartial games, G and H , are isomorphic to each other if T_G is isomorphic to T_H .*

For any two impartial games F and G , the game tree $T_{(F+G)}$ of their disjunctive sum can be more naturally characterized via $T_F \square T_G$, the *Cartesian product* of T_F and T_G . Clearly, $T_F \square T_G$ is a directed acyclic graph (DAG) rather than a rooted tree, as a node may have up to two parents. The game tree $T_{(F+G)}$ can be viewed as a tree-expansion of DAG $T_F \square T_G$. To turn it into a tree, one may simply duplicate all subtrees whose root has two parents, and give the parents edges to different roots of those two subtrees. We will use $T_F \blacksquare T_G$ to denote this tree expansion of the Cartesian product $T_F \square T_G$, and call it the *tree sum* of T_F and T_G .

In combinatorial game theory (CGT), each ruleset usually represents an infinite family of games, each defined by its starting position. For algorithmic and complexity analyses, a *size* is associated with each game position as the basis for measuring complexity [28, 15, 26, 7]. Examples include: (1) the number of vertices in the graph for NODE KAYLES and GENERALIZED GEOGRAPHY, (2) the board length of HEX and ATROPOS, and (3) the number of bits encoding NIM.

The size measure is assumed to be *natural*³ with respect to the key components of the ruleset. In particular, for each position G in a ruleset R :

- G has a binary-string representation of length polynomial in $\text{size}(G)$.
- Each position reachable from G has size upper-bounded by a polynomial function in $\text{size}(G)$.
- Determining if a position of $F \in \mathcal{B}_R$ is an option of G – i.e., whether $F \in \rho_R(G)$ – takes time polynomial in $\text{size}(G)$.

Recall that the family, \mathcal{I}^P , discussed in the introduction is formulated based on the sizes of game positions.

► **Definition 9** (Polynomially-Short Games). *A combinatorial ruleset $R = (\mathcal{B}_R, \rho_R)$ is polynomially short if the height of the game tree T_G of each position $G \in \mathcal{B}_R$ is polynomial in $\text{size}(G)$. Furthermore, we say $R \in \mathcal{I}^P$ is polynomially-wide if for each position $G \in \mathcal{B}_R$, the number of options $|\rho_R(G)|$ is bounded by a polynomial function in $\text{size}(G)$.*

We call games of \mathcal{I}^P *polynomially-short* games. GENERALIZED GEOGRAPHY and NODE KAYLES are among the many examples of games that are both polynomially-wide and polynomially-short. NIM, however, is neither polynomially-wide nor polynomially-short due to the binary encoding of the piles. In general, under the aforementioned assumption regarding the size function of game positions, $|\rho_R(G)|$ could be exponential in $\text{size}(G)$. However, positions in $\rho_R(G)$ can be enumerated in polynomial space. Therefore, by DFS evaluation of game trees and classical complexity analyses of NODE KAYLES, GENERALIZED GEOGRAPHY and AVOID TRUE:

► **Proposition 10** (PSPACE-Completeness). *For any polynomially-short impartial ruleset R and a position G in R , $\text{nimber}(G)$ can be computed in space polynomial in $\text{size}(G)$. Furthermore, under Cook-Turing reductions, nimber computation for **some** games in \mathcal{I}^P is PSPACE-hard.*

An impartial ruleset is said to be *polynomial-time solvable* – or simply, *tractable* – if there is a polynomial-time algorithm to identify a winning option whenever there exists one (i.e., for the search problem associated with the decision of winnability). If one doesn't exist, then the algorithm needs to only identify this.

³ In other words, the naturalness assumption rules out rulesets with embedded hard-to-compute predicate like – as a slightly dramatized illustration – “If $P \neq \text{PSPACE}$ is true, then the feasible options of a position include a special position.”

3 Star Atlas: A Complete Generalized Geography

In our analysis, we will use the standard CGT notation for numbers: $*k$ for k , except that $*$ is shorthand for $*1$ and 0 is shorthand for $*0$.⁴ Mathematically, one can view $*$ as a map from $\mathbb{Z}^+ \cup \{0\}$ to (infinite) subfamilies of impartial games in \mathbb{I} , such that for each $k \in \mathbb{Z}^+ \cup \{0\}$, $\text{number}(G) = k$, for all $G = *k$. In other words, $*$ is *nature's game encoding* of non-negative integers.

Sprague-Grundy Theory establishes that each impartial game's strategic relevance in disjunctive sums is determined by its number (i.e., its star value). In this section, we prove Theorem 4, showing how to use GENERALIZED GEOGRAPHY to efficiently “map out” games' numbers across \mathbb{I}^P .

For readability, we restate Theorem 4 to make the needed technical component explicit:

► **Theorem 11** (Sprague-Grundy-Completeness of GENERALIZED GEOGRAPHY). *There exists a polynomial-time algorithm ϕ such that for any game $G \in \mathbb{I}^P$, $\phi(G)$ is a GENERALIZED GEOGRAPHY position satisfying $\text{number}(\phi(G)) = \text{number}(G)$.*

Our proof starts with the following basic property of numbers, which follows from the recursive definition (given in Equation 1):

► **Proposition 12.** *For any impartial game G , $\text{number}(G)$ is bounded above by both the height of its game tree, h , and the number of options at G , l . In other words, $G = *k$, where $k \leq \min(h, l)$.*

To simplify notation, we let $g = \min(h, l)$. To begin the reduction, we will need a reduction for each decision problem $Q_i = \text{“Does } G = *i\text{?”}$ ($\forall i \in [g]$). By Proposition 10, the decision problem Q_i is in PSPACE. Thus, we can reduce each Q_i to an instance in the PSPACE-complete QSAT (Quantified SAT), then to a GEOGRAPHY instance using the classic reduction, f , from [28, 24]. Referring to the starting node of $f(Q_i)$ as s_i , we add two additional vertices, a_i and b_i , with directed edges (b_i, a_i) and (a_i, s_i) . Now,

- s_i has exactly two options, so the value of $f(Q_i)$ is either 0 , $*$, or $*2$. By the reduction, it is 0 exactly when $G \neq *i$, and in $\{*, *2\}$ when $G = *i$.
- a_i has exactly one option (s_i), so the value of the GEOGRAPHY position starting there (instead of at s_i) is 0 when $G = *i$ and $*$ otherwise.
- b_i has exactly one option (a_i), so the value of the GEOGRAPHY position starting there is $*$ when $G = *i$ and 0 otherwise.

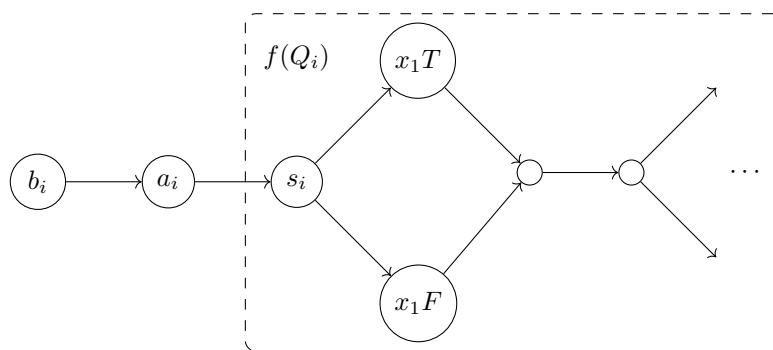
Each of these constructions from Q_i is shown in Figure 2.

We will combine these $g + 1$ GEOGRAPHY instances into a single instance, but first we need some utility vertices each equal to one of the number values $0, \dots, *(g - 2)$. We can build these using a single gadget as shown in figure 3. This gadget consists of vertices t_0, t_1, \dots, t_{g-2} with edges (t_i, t_j) for each $i > j$. Thus, each vertex t_i has options to t_j where $j < i$ and no other options, exactly fulfilling the requirements for t_i to have value $*i$.

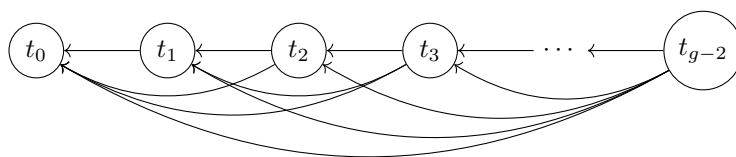
Now we build a new gadget to put it all together and combine the $f(Q_i)$ gadgets, as shown in figure 4:

- $\forall i \geq 1$: add a vertex c_i , as well as edges (c_i, b_i) and $\forall j \in [1, i - 2] : (c_i, t_j)$.
- $\forall i \geq 2$: add a vertex d_i as well as edges (d_i, b_1) and $\forall j \in [2, i - 1] : (d_i, c_j)$.
- Finally, add a vertex $start$ with edges $(start, b_0)$, $(start, c_1)$, and $\forall j \in [2, g] : (start, d_j)$.

⁴ The reason for the $*0 = 0$ convention is that it is equivalent to the integer zero in CGT.



■ **Figure 2** Result of the classic QSAT and GEOGRAPHY reductions of the question, Q_i , “Does $G = *i?$ ”, with the added vertices a_i and b_i .



■ **Figure 3** vertices t_0 through t_{g-2} . Each vertex t_i has edges to t_0, t_1, \dots, t_{i-1} . Thus, the number value of the GEOGRAPHY position at vertex t_i is $*i$.

► **Lemma 13.** *The GEOGRAPHY position starting at each vertex c_i has value $*(i - 1)$ if $G \neq *i$, and value 0 otherwise.*

Proof. The GEOGRAPHY position starting at c_i has options to $t_j, \forall t \in [1, i - 2]$. That means that c_i has options with values $*, \dots, *(i - 2)$. If the move to b_i has value 0, then there are moves to $0, *, \dots, *(i - 2)$, so the value at c_i is $*(i - 1)$. Otherwise, there is no option from c_i to a zero-valued position, so the value at c_i is 0. ◀

► **Lemma 14.** *If $G = 0$, then the GEOGRAPHY position starting at each vertex d_i has value $*i$.*

Proof. d_i has moves to b_1, c_2, \dots, c_i . Since $G = 0$, by Lemma 13 none of the vertices c_j have values 0 (and b_1 does have value 0), so the options have values $0, *, *2, \dots, *(i - 1)$, respectively. The mex of these i $*i$, so d_i has value $*i$. ◀

► **Lemma 15.** *Let $G = *k$, where $k > 0$. Then the GEOGRAPHY position starting at each vertex d_i has value $*i$ if $i < k$, and value $*(k - 1)$ if $i \geq k$.*

Proof. We need to prove this by cases. We’ll start with $k = 1$, then show it for $k \geq 2$.

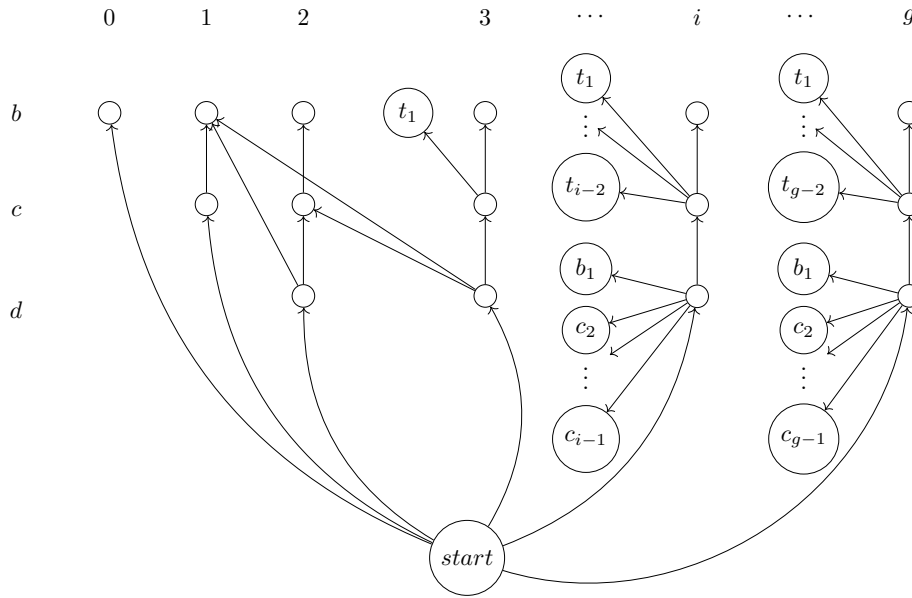
When $k = 1$, d_i has moves to b_1, c_2, \dots, c_i . (There is no d_0 or d_1 , so $i > k$.) The value at b_1 is $*$, and by Lemma 13, the remainder have values $*, *2, \dots, *(i - 1)$, respectively. 0 is missing from this list, so $d_i = 0 = *(1 - 1) = *(k - 1)$.

For $k \geq 2$, we will split up our analysis into the two cases: $i < k$ and $i \geq k$.

We will next consider the case where $k \geq 2$ and $i < k$. From d_i there are moves to b_1, c_2, \dots, c_i . Since $k > i$, these have values $0, *, *2, \dots, *(i - 1)$, respectively, by Lemma 13. The mex of these is $*i$, so d_i has value $*i$.

Finally, when $k \geq 2$ and $i \geq k$, d_i has options to $b_1, c_2, \dots, c_{k-1}, c_k, c_{k+1}, \dots, c_i$. By Lemma 13, these have values $0, *, *2, \dots, *(k - 2), 0, *k, \dots, *(i - 1)$, respectively. $*(k - 1)$ doesn’t exist in that list, so that’s the mex, meaning the value of d_i is $*(k - 1)$. ◀

10:10 Nimber-Preserving Reductions



■ **Figure 4** The gadget combining all of the $f(Q_i)$ gadgets into a single GEOGRAPHY instance. The vertices b_i , c_i , and d_i are in rows and columns indexed by the letters on the left and the numbers along the top. Each b_i has an edge to a_i as in figure 2.

► **Theorem 16.** *Let $G = *k$. Then the GEOGRAPHY position beginning at $start$ equals $*k$.*

Proof. The options from $start$ are b_0 , c_1 , and $\forall i \in [2, g] : d_i$. If $G = 0$, then b_0 is $*$, c_1 is $*$, and, by Lemma 14, each d_i is $*i$. Since 0 is missing from these options, the value at $start$ is $0 = *k$.

If $G = *$, then the move to b_0 is 0, the move to c_1 is also 0, and, by Lemma 15, the moves to d_i is each also 0, because each $i > k = 1$ and $*(k-1) = *(1-1) = 0$. All the options are to 0, so the value at $start$ is $* = *k$.

Finally, if $G = *k$, where $k \geq 2$, then the moves are to b_0 , c_1 , $d_2, \dots, d_{k-1}, d_k, d_{k+1}, \dots, d_g$. These have values, respectively, $0, *, *2, \dots, *(k-1), *(k-1), *(k-1), \dots, *(k-1)$, by Lemma 15. The mex of these is k , so the value of $start$ is $*k$. ◀

4 Nimber Secrets: A PSPACE-Complete Homomorphic Encoding

Mathematically, Sprague-Grundy Theory together with Bouton's NIM characterization provide an algebraic view of impartial games. Their framework establishes that the Grundy function, $\text{nimber}()$, is a morphism from the monoid $(\mathbb{I}, +)$ – impartial games with disjunctive sum – to the monoid $(\mathbb{Z}^+ \cup \{0\}, \oplus)$ – non-negative integers in binary representations with bitwise-xor:

$$\text{nimber}(G + H) = \text{nimber}(G) \oplus \text{nimber}(H) \quad \forall G, H \in \mathbb{I}.$$

Elegantly,

1. $\text{nimber}(G)$ can also be represented by a natural game, i.e., a single pile NIM with $\text{nimber}(G)$ stones, and
2. the sum of two single-pile NIM games can be represented by another single-pile NIM whose game tree can be significantly different from the game tree of the sum.

The only “blemish” – from computational perspective – is that the Grundy function can be intractable to compute [28], even for some tractable games in \mathbb{I}^P [9].

Theorem 11 provides an alternative natural game representation of $\text{nimber}(G)$, for $G \in \mathbb{I}^P$. In contrast to NIM, GENERALIZED GEOGRAPHY admits a polynomial-time algorithm for computing this representation from G without the need of computing $\text{nimber}(G)$. In fact, using Theorem 11, one can also compute, in polynomial time, a single-graph GENERALIZED GEOGRAPHY representation of the sum of any two (GENERALIZED GEOGRAPHY) games:

► **Theorem 17** (Homomorphic Game Encoding of Grundy Values). *For any pair of games $G_1, G_2 \in \mathbb{I}^P$, one can, in polynomial-time in $\text{size}(G_1) + \text{size}(G_2)$, construct a GENERALIZED GEOGRAPHY game G , such that:*

$$\text{nimber}(G) = \text{nimber}(G_1) \oplus \text{nimber}(G_2).$$

Proof. Given the game functions ρ_{G_1} and ρ_{G_2} , in time linear in $\text{size}(G_1) + \text{size}(G_2)$, one can construct a game function $\rho_{(G_1+G_2)}$ for their disjunctive sum $G_1 + G_2$ such that the game tree of $\rho_{(G_1+G_2)}$ is $G_1 \blacksquare G_2$. This theorem follows from Theorem 4 and the following basic fact:

The disjunctive sum $(G_1 + G_2)$ of two polynomially-short games $G_1, G_2 \in \mathbb{I}^P$ remains polynomially short in terms of $\text{size}(G_1) + \text{size}(G_2)$.

Now we can apply Theorem 4 to $\rho_{(G_1+G_2)}$ to construct a prime GENERALIZED GEOGRAPHY G in time polynomial in $\text{size}(G_1) + \text{size}(G_2)$. G satisfies: $\text{nimber}(G) = \text{nimber}(G_1) \oplus \text{nimber}(G_2)$. The correctness follows from that of Theorem 4 and Sprague-Grundy Theory. ◀

Figuratively, *every impartial game G encodes a secret, $\text{nimber}(G)$* . The game G itself can be viewed as an “*encryption*” of its nimber-secret. The players who can uncover this nimber-secret can play the game optimally. For every game $G \in \mathbb{I}^P$, this secret can be “*decrypted*” by a DFS-based evaluation of G ’s game tree in polynomial space. Thus, computing the nimber for \mathbb{I}^P is PSPACE-complete (under the Cook-Turing reduction).

Speaking of encryption, several basic cryptographic functions have homomorphic properties. For example, for every RSA encryption function ENC_{RSA} , for every pair of its messages m_1 and m_2 , the following holds:

$$\text{ENC}_{\text{RSA}}(m_1 \times m_2) = \text{ENC}_{\text{RSA}}(m_1) \times \text{ENC}_{\text{RSA}}(m_2).$$

Another example is the discrete-log function. For any prime p , any primitive element $g \in Z_p^*$, and any two messages $m_1, m_2 \in Z_p^*$:

$$g^{m_1+m_2} = g^{m_1} \times g^{m_2}.$$

Assuming RSA encryption and the discrete-log function are computationally intractable to invert, these morphisms state that without decoding the secret messages from their encoding, one can efficiently encode their product or sum, respectively, with the RSA and discrete-log functions. In cryptography, these functions are said to support *partially homomorphic encryption*.

Together, Theorem 17 and Theorem 11 establish that polynomially-short impartial games are themselves *partially homomorphic encodings* of their nimber-secrets: Without decoding their numbers, one can efficiently create a GENERALIZED GEOGRAPHY game encoding the \oplus of their numbers.

Note again that homomorphic cryptographic functions, such as discrete log and RSA encryption, satisfy an additional property: They are one-way functions, i.e., tractable to compute but are assumed to be intractable to invert. Theorem 7 (and hence Theorem 17)

10:12 Nimber-Preserving Reductions

is inspired by the concept of partially homomorphic encryption. However, its focus is not on a one-way encoding of targeted nimber-values with impartial games in \mathbb{I}^P . Rather, it characterizes the complexity-theoretical homomorphism in this classical and natural encoding for impartial games. Because of the one-way property, RSA and discrete-log functions are decodable by an NP-oracle. In contrast, the nimber-decoding of impartial games in \mathbb{I}^P is in general PSPACE-hard.

4.1 Natural Prime Games

Inspired by Conway's notation with *parts* within the context of misère games [11][29], we use the following terms to identify what game trees can be described as isomorphically the sum of two other games:

► **Definition 18** (Prime Games and Composite Games). *A game G is a composite game if it is a sum of two games that both have tree-height at least 1. Otherwise, it is prime.*

Note that prime games, in a similar manner to prime numbers, can only be summed by a game with tree-height of 0 (ie: just a single vertex) and itself. It follows from the basic property of Cartesian graph products that each composite game has a unique decomposition into prime games.

► **Proposition 19** (Decomposition in Prime Games). *A game G is isomorphic to a disjunctive sum of two games A and B if and only if its game tree T_G is isomorphic to $T_A \blacksquare T_B$.*

In Bouton theory for NIM, for any non-negative integers a, b , even though $\text{nimber}(\text{NIM}(a \oplus b)) = \text{nimber}(\text{NIM}(a)) \oplus \text{nimber}(\text{NIM}(b))$, $\text{NIM}(a \oplus b)$ is not isomorphic to $\text{NIM}(a) \oplus \text{NIM}(b)$. In fact, $\text{NIM}(a \oplus b)$ is a natural prime game. Similarly, even though in the proof of Theorem 17, $\rho_{(G_1+G_2)}$ simply copies the syntactic game transition function that can generate $G_1 \blacksquare G_2$, the construction in Theorem 4 generates the homomorphic prime game encoding of $\text{nimber}(G_1) \oplus \text{nimber}(G_2)$.

► **Theorem 20** (Prime Geography). *Each GENERALIZED GEOGRAPHY position as created in the reduction in Theorem 11 is a prime game.*

Proof. Suppose that our game tree is claimed to be $X \blacksquare Y$, with root vertices x_0 and y_0 , respectively, and both X and Y have height at least 1. We will find a contradiction.

Consider vertex b_0 , an option of *start*. Since b_0 has only one option, that means that it must correspond to a terminal move in either X or Y . WLOG, let it correspond to x_1 , a terminal vertex in X . Thus, $b_0 = (x_1, y_0)$ and is isomorphic to Y , because x_1 is terminal in X .

The move to c_1 must be available, since otherwise the *start* would have only one option and thus not be a tree sum. This position also has only one option from itself. There are two cases: it either corresponds to a terminal vertex in X , say x_2 , or a terminal vertex in Y , say y_1 . In the first case, then $c_1 = (x_2, y_0)$, which is isomorphic to Y . This causes a contradiction, however, because the subtrees generated by b_0 and c_1 are not isomorphic. (b_0 has 2 moves to reach s_0 , but c_1 has 3 moves to reach s_1 .)

In the second case, $c_1 = (x_0, y_1)$, and y_1 is terminal in Y . Then that means there must be a move from c_1 to a vertex, v , corresponding to (x_1, y_1) . Since both x_1 and y_1 are terminal (in X and Y), that means v will be terminal in the tree sum. However, c_1 doesn't have any options to a terminal vertex. This case cannot happen and, without any other possible cases, no such X and Y exist as factors for our tree. ◀

5 Final Remarks and Open Questions

It is expected that PSPACE-complete games encode some valuable secrets. And once revealed, those secrets can help players in their decision making (e.g., under the guidance of Sprague-Grundy Theory). In this work, through the lens of computational complexity theory, we see that all polynomially-short impartial games neatly encode their number-secrets, which can be efficiently transferred into prime GENERALIZED GEOGRAPHY games. The game encoding is so neat that the bitwise-xor of any pair of these number-secrets can be homomorphically re-encoded into another prime game in polynomial time, without the need to find the secrets first.

We are excited to discover this natural mathematical-game-based PSPACE-complete homomorphic encoding. *Recreational mathematics can be simultaneously serious and fun!*

The crypto-concept of (partially) homomorphic encryption has inspired us to identify these basic complexity-theoretical properties of this fundamental concept in CGT. It would have been more fulfilling if we could also make our findings useful in cryptography. Currently, we are exploring potential cryptographic applications of this “game encoding of strategic secrets,” particularly on *one-way* game generation for targeted numbers. In addition to finding direct cryptographic connections, we are still exploring several concrete CGT questions. Below, we share some of them.

5.1 Expressiveness of Intractable Games: Sprague-Grundy Completeness

In this paper, we have proved that the PSPACE-complete polynomially-short GENERALIZED GEOGRAPHY is prime Sprague-Grundy complete for \mathcal{I}^P . We observe that not all games in \mathcal{I}^P with PSPACE-hard number computation are Sprague-Grundy complete for the family because:

1. Some intractable games can't encode numbers polynomially related to the input size
2. Some games with intractable number computation have some nim values which are tractable.

For (1), our first example is GENERALIZED GEOGRAPHY ON DEGREE-THREE GRAPHS. In [24], Lichtenstein and Sipser proved that GENERALIZED GEOGRAPHY is PSPACE-complete to solve even when the game graph is planar, bipartite, and has a maximum degree of three. These graph properties are essential to their analysis of the two-dimensional grid-based GO. Mathematically, the maximum achievable number in GENERALIZED GEOGRAPHY ON DEGREE-THREE GRAPHS is three. Thus, there is no number-preserving reduction from higher number position in \mathbb{I}^P to these low-degree GENERALIZED GEOGRAPHY games. For the same reason, the PSPACE-complete ATROPOS introduced in [10] cannot be Sprague-Grundy complete.

► **Lemma 21.** *The value of any ATROPOS position must be one of these numbers: 0, *, *2, ..., *7. (And thus, ATROPOS cannot be Sprague-Grundy complete.)*

Proof. For the details of how ATROPOS is played, please see [10]. If the last (played) vertex has uncolored neighbors, then there are at most six neighbors, so the highest number value is *6.

If the previously-played vertex is fully surrounded by colored vertices⁵, then there are two possibilities: either all playable vertices have uncolored neighbors, or some of the playable vertices are also fully surrounded. In the first case, there may be options to all numbers 0, *, *2, ..., *6, so the value here could be up to *7.

⁵ In this case, the next player gets a “jump” and gets to play anywhere on the board.

10:14 Nimber-Preserving Reductions

In the second case, the current position, say G , is equal to the sum of the portion of the board (say, H) without those fully-surrounded (but playable) vertices and the portion of the board with only those vertices. Each of those vertices in $G \setminus H$ changes the value by $*$. Thus, if there are k of them, $G = H + k \times *$. Thus, either $G = H$ or $G = H + *$. By the previous case, the number of H can be up to $*7$, so the value of G can also be at most $*6$ or $*7$.

ATROPOS has a bounded number, so it cannot be Sprague-Grundy complete. ◀

For (2), both UNDIRECTED GEOGRAPHY [17] and UNCOOPERATIVE UNO [12]⁶ are not Sprague-Grundy complete for \mathcal{I}^P – unless $P = PSPACE$ – despite their number intractability, with UNDIRECTED GEOGRAPHY being known to have polynomially high number positions [9]. For UNDIRECTED GEOGRAPHY, Fraenkel, Scheinerman, and Ullman [17] presented a matching-based characterization to show these games are polynomial-time solvable. For UNCOOPERATIVE UNO, Demaine *et al* [12] presented a polynomial-time reduction to UNDIRECTED GEOGRAPHY. Thus, any polynomial-time number-preserving reduction from \mathcal{I}^P to UNDIRECTED GEOGRAPHY (or UNCOOPERATIVE UNO) would yield a polynomial-time algorithm for solving \mathcal{I}^P .

Rulesets which have number preserving reductions from GENERALIZED GEOGRAPHY are Sprague-Grundy complete. A simple example is the vertex version of DIGRAPH NIM [18], in which each node has a NIM pile and players can only move to a reachable node in a directed graph from the current node to pick stones. When every pile has one stone, the game is equivalent to GENERALIZED GEOGRAPHY with the underlying graph. An interesting question is whether NEIGHBORING NIM (with a polynomial number of stones) – a PSPACE-complete version of NIM played on an undirected graph [6] – is Sprague-Grundy complete.

The edge variant of GENERALIZED GEOGRAPHY, known as the EDGE-GEOGRAPHY, considered in the literature [28, 24, 17] presents a natural extension. This is a version of GEOGRAPHY where instead of deleting the current node after the token moves away, it is the edge traversed by the token that is deleted. EDGE-GEOGRAPHY and its undirected sub-family, UNDIRECTED EDGE-GEOGRAPHY are both PSPACE-complete. The following proof sketch shows that EDGE-GEOGRAPHY remains Sprague-Grundy complete.

► **Corollary 22.** *EDGE-GEOGRAPHY is prime Sprague-Grundy-complete for \mathcal{I}^P .*

Proof. We can follow the early parts of the proof for GENERALIZED GEOGRAPHY. We reduce from all polynomially-short games, creating a game of EDGE-GEOGRAPHY for each. Then, for each game, we again append two “filler” moves to the beginning, to ensure that it is exactly 0 or $*$.

We can then reuse our scheme from figure 2. Since there are no cycles in that gadget, play between both EDGE-GEOGRAPHY and GENERALIZED GEOGRAPHY is identical.

Of course, the primality section required knowing that the main GEOGRAPHY game didn’t start with an out degree of only one. To fix this, we can simply have v_b go to v_{a1} and v_{a2} which both only have a single edge to v_s . ◀

It remains open whether UNDIRECTED EDGE-GEOGRAPHY is Sprague-Grundy complete.

In addition to these rulesets adjacent to GENERALIZED GEOGRAPHY, we are interested in the following three well-studied games:

⁶ In this game, there are two hands, H_1 and H_2 , which each consist of a set of cards. This is a perfect information game, so both players may see each other’s hands. Each card has two attributes, a color c and a rank r . Each card then thus be represented (c, r) . A card can only be played in the center (shared) pile if the previous card matches either the c of the current card or the r of the current card.

- Is NODE KAYLES Sprague-Grundy complete for \mathcal{I}^P ? Is AVOID TRUE Sprague-Grundy complete for \mathcal{I}^P ? Is GENERALIZED CHOMP Sprague-Grundy complete for \mathcal{I}^P ?

In our proof for GENERALIZED GEOGRAPHY, we critically use the “locality” in this graph-theoretical game: The options are defined by the graph-neighbors of the current node. Both NODE-KAYLES and AVOID TRUE are far more “global”; there is no need for moves to be near the previous move. We are also interested in GENERALIZED CHOMP because the hierarchical structures from partial orders could be instrumental to analyses.

NODE-KAYLES – see below for more discussion – also suggests the following basic structural question:

- Is there a natural ruleset in \mathcal{I}^P that is Sprague-Grundy-complete for \mathcal{I}^P but not *prime* Sprague-Grundy-complete for \mathcal{I}^P ?

5.2 Game Encoding and Computational Homomorphism

Let’s call a family \mathcal{H} of impartial rulesets satisfying Theorem 7 (in place of \mathcal{I}^P and with a prime game of \mathcal{I}^P in place of GENERALIZED GEOGRAPHY) a *computationally-homomorphic family*. Note that for any \mathcal{J} including UNDIRECTED GEOGRAPHY, \mathcal{J} satisfies Theorem 7.

Now suppose we “slightly” weaken Theorem 7 by removing the prime-game requirement (in the *Homomorphic Game Encoding* condition), and call \mathcal{H} satisfying the weakened version of Theorem 7 a *weakly computationally-homomorphic family*. Then, \mathcal{I}^P itself is a weakly computationally-homomorphic family, by Sprague-Grundy Theory and the fact that \mathcal{I}^P is closed under the disjunctive sum.

Indeed, if a ruleset in \mathcal{I}^P is PSPACE-complete and allows a simple way to express the sum of two positions as a single position, then the ruleset is a weakly computationally-homomorphic family. One of the most basic examples of this is NODE KAYLES. Here, two positions can be trivially summed into a single game by simply taking the two graphs and making them a single (disconnected) graph.

This is a very common property for combinatorial games to have. However, many impartial games with this property aren’t known to be intractable. As an example, CRAM is a game that is simply played by placing 2x1 dominoes in either horizontal or vertical orientation on unoccupied tiles of a 2-dimensional grid. Two CRAM positions can be added together by surrounding each with a boundary of dominoes, then concatenating the two boards together. Unfortunately, it is not currently known whether CRAM is intractable.

Related to the question we asked in Section 5.1, we are curious to know:

- Given a pair of NODE KAYLES positions G_1 and G_2 , can we construct, in polynomial time, a *prime* NODE KAYLES position satisfying $\text{number}(G) = \text{number}(G_1) \oplus \text{number}(G_2)$?

5.3 Beyond \mathcal{I}^P

More generally,

- Are there analog extensions of our results to polynomially-short partizan games?
- Is there a characterization of Sprague-Grundy completeness for \mathcal{I}^P ?
- Does the family of PSPACE-solvable impartial games have a natural Sprague-Grundy-complete ruleset?
- Does the family of all impartial games have a natural Sprague-Grundy-complete ruleset?
- What is the complexity of GRAPH NIM with an exponential number of stones?

For these last few questions, we may need to go beyond PSPACE as well as polynomially-short games to unlock the number secrets.

5.4 Finally

Is there a Bouton analog – i.e., a more clean and direct graph operator – to compute a GENERALIZED GEOGRAPHY game G from two GENERALIZED GEOGRAPHY games G_1 and G_2 such that $\text{nimber}(G) = \text{nimber}(G_1) \oplus \text{nimber}(G_2)$?

References

- 1 Selim G Akl. On the importance of being quantum. *Parallel processing letters*, 20(03):275–286, 2010.
- 2 D. Applegate, G. Jacobson, and D. Sleator. Computer analysis of Sprouts. Technical Report CMU-CS-91-144, Carnegie Mellon University Computer Science, 1991.
- 3 Gabriel Beaulieu, Kyle G. Burke, and Éric Duchêne. Impartial coloring games. *Theoret. Comput. Sci.*, 485:49–60, 2013. doi:10.1016/j.tcs.2013.02.032.
- 4 Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for your Mathematical Plays*, volume 1. A K Peters, Wellesley, Massachusetts, 2001.
- 5 Charles L. Bouton. Nim, a game with a complete mathematical theory. *Annals of Mathematics*, 3(1/4):pp. 35–39, 1901.
- 6 K. Burke and O. George. A PSPACE-complete graph nim. In Urban Larsson, editor, *Games of No Chance 5*, volume 70 of *Mathematical Sciences Research Institute Publications*, pages 259–269. Cambridge University Press, 2019.
- 7 Kyle Burke, Matthew Ferland, and Shang-Hua Teng. Quantum combinatorial games: Structures and computational complexity. *CoRR*, abs/2011.03704, 2020. arXiv:2011.03704.
- 8 Kyle Burke, Matthew Ferland, and Shang-Hua Teng. Transverse Wave: An impartial color-propagation game inspired by social influence and quantum nim. *Integers*, 21B:A3, 30, 2021.
- 9 Kyle Burke, Matthew Ferland, and Shang-Hua Teng. Winning the war by (strategically) losing battles: Settling the complexity of Grundy-values in undirected geography. In *Proceedings of the 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2021.
- 10 Kyle W. Burke and Shang-Hua Teng. Atropos: A PSPACE-complete Sperner triangle game. *Internet Mathematics*, 5(4):477–492, 2008. doi:10.1080/15427951.2008.10129176.
- 11 John H. Conway. *On Numbers and Games (2. ed.)*. A K Peters, 2000.
- 12 Erik D Demaine, Martin L Demaine, Nicholas JA Harvey, Ryuhei Uehara, Takeaki Uno, and Yushi Uno. UNO is hard, even for a single player. *Theoretical Computer Science*, 521:51–61, 2014.
- 13 Paul Dorbec and Mehdi Mhalla. Toward quantum combinatorial games. *arXiv preprint arXiv:1701.02193*, 2017.
- 14 S. Even and R. E. Tarjan. A combinatorial problem which is complete in polynomial space. *J. ACM*, 23(4):710–719, October 1976.
- 15 Aviezri S Fraenkel. Complexity, appeal and challenges of combinatorial games. *Theoretical Computer Science*, 313(3):393–415, 2004.
- 16 Aviezri S. Fraenkel and David Lichtenstein. Computing a perfect strategy for $n \times n$ chess requires time exponential in n . *J. Comb. Theory, Ser. A*, 31(2):199–214, 1981. doi:10.1016/0097-3165(81)90016-9.
- 17 Aviezri S. Fraenkel, Edward R. Scheinerman, and Daniel Ullman. Undirected edge geography. *Theor. Comput. Sci.*, 112(2):371–381, 1993.
- 18 Masahiko Fukuyama. A nim game played on graphs. *Theor. Comput. Sci.*, 1-3(304):387–399, 2003. doi:10.1016/S0304-3975(03)00292-5.
- 19 David Gale. The game of Hex and the Brouwer fixed-point theorem. *American Mathematical Monthly*, 10:818–827, 1979.
- 20 Adam Glos and Jarosław Adam Miszczak. The role of quantum correlations in cop and robber game. *Quantum Studies: Mathematics and Foundations*, 6(1):15–26, 2019.
- 21 Allan Goff. Quantum tic-tac-toe: A teaching metaphor for superposition in quantum mechanics. *American Journal of Physics*, 74(11):962–973, 2006.

- 22 Daniel Grier. Deciding the winner of an arbitrary finite poset game is PSPACE-complete. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming - Volume Part I, ICALP'13*, pages 497–503, Berlin, Heidelberg, 2013. Springer-Verlag.
- 23 P. M. Grundy. Mathematics and games. *Eureka*, 2:198—211, 1939.
- 24 David Lichtenstein and Michael Sipser. Go is polynomial-space hard. *J. ACM*, 27(2):393–401, 1980.
- 25 John F. Nash. *Some Games and Machines for Playing Them*. RAND Corporation, Santa Monica, CA, 1952.
- 26 C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, Reading, Massachusetts, 1994.
- 27 S. Reisch. Hex ist PSPACE-vollständig. *Acta Inf.*, 15:167–191, 1981.
- 28 Thomas J. Schaefer. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16(2):185–225, 1978.
- 29 Aaron N. Siegel. On the structure of misère impartial games, 2021. [arXiv:2012.08554](https://arxiv.org/abs/2012.08554).
- 30 Michael Soltys and Craig Wilson. On the complexity of computing winning strategies for finite poset games. *Theory Comput. Syst.*, 48:680–692, April 2011.
- 31 R. P. Sprague. Über mathematische Kampfspiele. *Tôhoku Mathematical Journal*, 41:438—444, 1935-36.
- 32 David Wolfe. Go endgames are PSPACE-hard. In Richard J. Nowakowski, editor, *More Games of No Chance*, volume 42 of *Mathematical Sciences Research Institute Publications*, pages 125–136. Cambridge University Press, 2002.
- 33 D. Zeilberger. Chomp, recurrences and chaos. *Journal of Difference Equations and Applications*, 10:1281–1293, 2004.

Quantum-Inspired Combinatorial Games: Algorithms and Complexity

Kyle W. Burke ✉

Department of Computer Science, Plymouth State University, NH, USA

Matthew Ferland ✉

Department of Computer Science, University of Southern California, Los Angeles, CA, USA

Shang-Hua Teng ✉

Department of Computer Science, University of Southern California, Los Angeles, CA, USA

Abstract

Recently, quantum concepts inspired a new framework in combinatorial game theory. This transformation uses *discrete* superpositions to yield beautiful new rulesets with succinct representations that require sophisticated strategies. In this paper, we address the following fundamental questions:

- **Complexity Leap:** *Can this framework transform polynomial-time solvable games into intractable games?*
- **Complexity Collapse:** *Can this framework transform PSPACE-complete games into ones with complexity in the lower levels of the polynomial-time hierarchy?*

We focus our study on how it affects two extensively studied polynomial-time-solvable games: NIM and UNDIRECTED GEOGRAPHY. We prove that both NIM and UNDIRECTED GEOGRAPHY make a *complexity leap* over NP, when starting with superpositions: The former becomes Σ_2^P -hard and the latter becomes PSPACE-complete. We further give an algorithm to prove that from *any* classical starting position, quantumized UNDIRECTED GEOGRAPHY remains polynomial-time solvable. Together they provide a nearly-complete characterization for UNDIRECTED GEOGRAPHY. Both our algorithm and its correctness proof require strategic moves and graph contraction to extend the matching-based theory for classical UNDIRECTED GEOGRAPHY.

Our constructive proofs for both games highlight the intricacy of this framework. The polynomial time robustness of UNDIRECTED GEOGRAPHY in this quantum-inspired setting provides a striking contrast to the recent result that the disjunctive sum of two UNDIRECTED GEOGRAPHY games is PSPACE-complete. We give a Σ_2^P -hardness analysis of quantumized NIM, even if there are no pile sizes of more than 1.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Quantum-Inspired Games, Combinatorial Games, Computational Complexity, Polynomial Hierarchy, $\text{class}\{\text{PSPACE}\}$, Nim, Generalized Geography, Snort

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.11

Related Version *Full Version:* <https://arxiv.org/abs/2011.03704> [6]

Supplementary Material We have implemented several games discussed in this paper as web games:

Quantum Nim: <https://turing.plymouth.edu/~kgb1013/DB/combGames/quantumNim.html>

Demi Version: <https://turing.plymouth.edu/~kgb1013/DB/combGames/demiQuantumNim.html>

Transverse Wave: <https://turing.plymouth.edu/~kgb1013/DB/combGames/transverseWave.html>

Funding *Shang-Hua Teng:* Supported by the Simons Investigator Award for fundamental & curiosity driven research and NSF grant CCF-1815254.



© Kyle W. Burke, Matthew Ferland, and Shang-Hua Teng;
licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 11; pp. 11:1–11:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In 2006, Allan Goff – a nuclear engineer – introduced a game, called QUANTUM TIC-TAC-TOE, as an educational tool for learning quantum mechanics [21]. He incorporated elements inspired by the quantum concepts of superpositions and entanglement into TIC-TAC-TOE, a popular two-player game traced back to ancient Egypt and Roman Empire. This variant’s popularity made it subject to further study, including exploring the entire game tree [24]. Since then, quantum-inspired rules have been derived for other combinatorial games¹, e.g., CHESS [1] and COPS AND ROBBERS [20]. In 2017, Dorbec and Mhalla [13] took a step further and presented a general *discrete* framework for turning combinatorial games into *quantum-inspired* variants, extending Goff’s formulation (of superposition of moves and entanglement among TIC-TAC-TOE positions).

In this paper, we study several fundamental questions addressing the computational impacts of this framework to combinatorial games. Superpositions of moves and entanglement among game positions introduce complex *nondeterminism* into the game space, providing a rich structure for algorithm design and complexity characterization.

1.1 Combinatorial Games and a Quantum-Inspired Transformation

Combinatorial games are mathematical games between two players with perfect information and no random elements. Traditionally, each game is defined by a *succinct ruleset*, specifying the domain of game *positions* that map to other positions one can move to (*options*). [4, 23, 12, 2, 32]. In order to obtain a meaningful semblance of entanglement, one needs to include another facet, *moves*: the natural descriptions for transitioning from positions to their options. Each move (e.g., “take two from the third pile” or “move the token to v ”) could apply to many positions, resulting in a different option for each. In the *normal-play* setting, two players take turns selecting their next move, and the player who is forced to a *terminal position* – a position with no feasible options – loses the game. A ruleset is *impartial* if both players have the same options at every position, and *partisan* otherwise.

The rapid expansion of strategic spaces makes games, e.g., GO, CHESS, and CHECKERS, both favorite intellectual pastimes in the real world and decision problems for study [28, 35, 17]. Many games, such as NIM, TIC-TAC-TOE, and MANCALA, have long historical roots and are taught to primary school students. Others are studied in mathematics and computer science, including theorized practical games and abstract formulations based on topology, logic, and graph theory, (e.g., HEX, POSET, ATROPOS, QSAT, AVOID TRUE, GEOGRAPHY², NODE-KAYLES, SPROUTS and HACKENBUSH [30, 19, 31, 4, 33, 8, 3]). In 1901, Bouton [5] developed a complete mathematical theory for NIM, an impartial game traced back to an ancient Chinese game known as Jian Shi Zi (撿石子 - picking stones). Each NIM position is a collection of piles of (small) stones. At their turn, players take at least one stone from one of the piles. Under normal play, the player taking the last stone wins. Three decades later, Bouton’s solution was instrumental in Sprague-Grundy Theory on impartial games [34, 22], which laid the foundation for Combinatorial Game Theory (CGT) [4].

CGT is now a research area drawing interest in both mathematics and computer science [4, 23, 12, 2, 32]. Beyond analyzing the structure and complexity of individual rulesets, CGT studies the system of games as a whole. This holistic approach provides a systematic

¹ Although different in details, these developments are parallel to other fields. For example, in economic game theory, researchers have explored what happens when agents are allowed to make quantum decisions and how quantum decisions impact game and economic equilibria [14].

² We use GEOGRAPHY to mean GENERALIZED GEOGRAPHY (and DIRECTED GEOGRAPHY).

framework for combining games and studying their relationships. The most basic way to combine games is the *disjunctive sum*: For any two games G and H , $G + H$ is a game in which the next player chooses to make a move in exactly one of G and H , leaving the other unchanged. Using this algebraic structure over games, the Sprague-Grundy theorem establishes that every impartial game is equivalent to a single NIM pile. With this equivalence, *a.k.a* the *Grundy value or nimber*, Bouton’s solution for NIM – based on bit-wise exclusive OR – provides a complete mathematical theory for the disjunctive sum of impartial games.

The disjunctive sum also acts as a transformation for defining new rulesets from existing ones. For example, let $\text{NIM}[(1)]$ denotes a single-pile one-stone NIM. For any position Z , $\text{OnePass}(Z) := Z + \text{NIM}[(1)]$ transforms Z into a new game of playing Z endowed with a single one-time “pass move” shared by the two players. Mathematically, the quantum-inspired framework [21, 13] can be viewed as a new *transformation* of combinatorial games. Before outlining this framework, we remark that by *quantum combinatorial games*, Goff, Dorbec and Mhalla don’t mean games defined on *continuous* qubits and unitary transformations as in quantum computing. Rather, quantum combinatorial games draw on quantum concepts of superposition and entanglement to enhance the classical combinatorial games with a “quantum-inspired” *discrete* framework of moves and positions. We think that “*quantum-inspired combinatorial games*” is the more precise and literal name for these games.

This generalization transforms each game Z into a new one $Z^{\mathcal{Q}}$ in which, at their turn, players can either make a classical move, or a superposition of classical moves.³ Following Goff, Dorbec and Mhalla, the latter type of moves will be referred to as *quantum moves*. Likewise, a *quantum game state* – a quantum position – is a superposition of multiple classical game states, each referred to as a *realization*. Mathematically, quantum moves introduce *nondeterminism* into game states, represented by quantum positions. We will use $\mathbb{M} := \langle m_1 \mid m_2 \mid \dots \mid m_w \rangle$ to denote a *w-wide superposition* of classical moves, and $\mathbb{G} = \langle g_1 \mid g_2 \mid \dots \mid g_s \rangle$ to denote an *s-wide quantum position*. With this notation, classical moves and positions correspond to $w = 1$ and $s = 1$, respectively.

In $Z^{\mathcal{Q}}$, \mathbb{M} is said to be *feasible* for \mathbb{G} if $\forall i \in [1 : w]$, m_i is feasible for a non-empty subset of realizations in \mathbb{G} , according to Z . \mathbb{M} takes \mathbb{G} to a new quantum position with “nondeterministic” game states resulting from feasible transitions in the “Cartesian product” of $\{m_1, \dots, m_w\}$ and $\{g_1, \dots, g_s\}$, according to Z : Realizations in \mathbb{G} , according to [21, 13], are “entangled” in that each move m_i applies to the entire nondeterministic composition. If m_i is not feasible for a realization, then the realization can no longer be factored in for making future moves, so we say it *collapses*. As a quantum game progresses, the number of realizations in the quantum state of the board can go up and down. When all possible moves of a player would cause a realization to collapse, we call it a *terminal* realization for that player. Under normal-play⁴, the game ends and the current player loses on a quantum game position when all its realizations are terminal for that player.

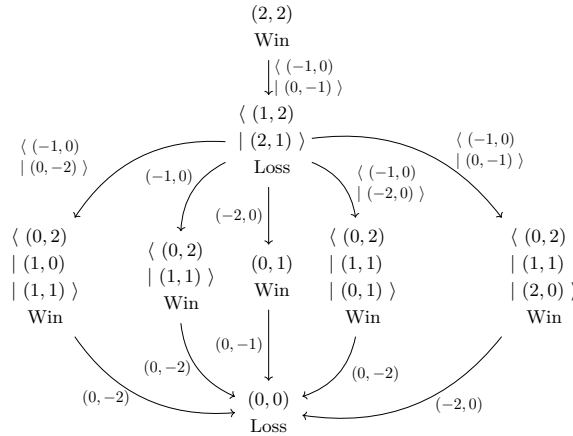
For positive integer w , let $Z^{\mathcal{Q}(w)}$ denote the quantumized Z in which all quantum moves have width at most w . For partizan games, there are also some subtleties in formalizing the outcome of the quantum extension. We will provide more detailed discussion in Section 4.

³ In [13], Dorbec and Mhalla postulated five quantum variants for extending classical rulesets, addressing possible restrictions on the interaction between classical moves and quantum positions. In this shorter version for FUN, we focus on the most natural variant, in which classical moves can be viewed as special cases of superposition moves. More on other variants can be found in the full version of this paper.

⁴ We do not apply quantumness to Misère play or Scoring games, the usual alternatives to Normal Play. These seem like excellent areas for future work.

1.2 An Illustration of “Quantum Impact” via Quantum Nim

We use NIM to provide a concrete example. In the classical setting, a well-known losing position for the current player is the configuration consisting of a pair of two-stone piles, which we denote by the 2D vector $(2, 2)$. The current player cannot win at this position because regardless of whether they pick one or two from a pile, the other player can simply do the same to the other pile to win.



■ **Figure 1** Winning strategy for current player in QUANTUM NIM $(2, 2)$, showing that quantum moves impact the game’s outcome. (There are four additional move options from $\langle (1, 2) | (2, 1) \rangle$ that are not shown because they are symmetric to moves given.)

A move is specified by the index of the pile and the amount to take. Here, we use $(-c, 0)$ and $(0, -c)$ to denote the moves of taking c stones from the first and second pile, respectively. From position $(2, 2)$, in addition to the classical moves of picking one or two stones from a pile, the current player can consider a quantum move, $\langle (-1, 0) | (0, -1) \rangle$, formed by the *superposition* of taking one stone from either pile. This *quantum move* yields a *quantum position*, $\langle (1, 2) | (2, 1) \rangle$, consisting of a superposition of two NIM positions. Subsequent quantum moves may further increase the number of realizations in the game’s superposition state. For example, if the other player makes the quantum move, $\langle (-1, 0) | (0, -2) \rangle$, then the next game position will be a superposition of three classical positions: $\langle (0, 2) | (1, 0) | (1, 1) \rangle$. (The realization $(2, -1)$ is not included because it’s not a legal NIM position.)

Remarkably, the quantum move $\langle (-1, 0) | (0, -1) \rangle$ is a winning move on QUANTUM NIM at $(2, 2)$. Figure 1 gives the complete game tree after that move from $(2, 2)$. Thus, quantum moves not only enrich players’ strategical domains, but also can alter the winnability. The sophisticated interactions (between superpositions of moves and superpositions of realizations) and the potential explosiveness (in the complexity of quantum configurations) make quantum games fascinating for computational complexity studies.

1.3 Highlights of Our Results

Combinatorial games are fun to play and it is intellectually stimulating to search for optimal moves. The *deep alternation* of elegant rules defines *game trees*, which can *exponentially* expand the space of game strategies, introducing a challenge to the basic decision problem of whether or not the current player has a winning move. Thus, it is fundamental to determine the complexity of a game. If it is *intractable*, this indicates that the game doesn’t

have a simple schema players can employ, so they must use complex strategies instead. *Elegant* combinatorial games with *simple rules* and *intractable complexity* are the gold standard for combinatorial game design [15]. In the digital age powered by AI, the lack of an efficient algorithm for optimally playing a combinatorial game gives human players a fighting chance to compete against computer programs. The deep challenge also gives computer programs a reason to learn and improve [8, 9].⁵ If it is *tractable*, then, in addition to being a pedagogical illustration, the efficient method can be a source of ideas for more general solution concepts and methodology. An exemplary case is Bouton’s polynomial-time NIM solution and its subsequent generalization in Sprague-Grundy theory. Polynomial-time-solvable games are also useful in capturing some practical phenomena, for example in modeling network dynamics [16, 10] or deriving other graph algorithms [25]. Furthermore, understanding the causes behind the transitions between tractable and intractable, through transformation of games, allows one to both turn “simple games” into “sophisticated” games [7] and transform intractable abstract games into real world solutions [11, 29].

Our research program here started with and expanded upon the following question:

- **Complexity Leap:** *Can the quantum-inspired framework transform a polynomial-time solvable combinatorial game into an intractable game?*

In our pursuit of this possible leap in the *quantumized complexity* of game, we have also obtained an affirmative answer to the following basic question:

- **Complexity Collapse:** *Are there PSPACE-complete combinatorial games whose quantum extensions fall to the lower levels of polynomial hierarchy?*

We focus on the effect on two extensively studied polynomial-time-solvable games: NIM and UNDIRECTED GEOGRAPHY, the undirected version of GEOGRAPHY. In the 1970s, recognizing the graph-theoretical background of a real-world “Word Chain” game called GEOGRAPHY, Richard Karp recommended the game to his then Ph.D. student Tom Schaefer for complexity study [31]. In GEOGRAPHY, a position is defined a directed graph and a starting node (for the token). The two players alternates turns moving the token to an adjacent node and then deleting the node it came from. In the normal-play setting, the player who cannot make a move loses the game. Schaefer proved that deciding the winnability of GEOGRAPHY is PSPACE-complete [31]. Lichtenstein and Sipser [28] then simplified his gadgets to prove that GO is PSPACE-hard. Indeed, both UNDIRECTED GEOGRAPHY and DIRECTED ACYCLIC GEOGRAPHY – i.e., GEOGRAPHY on DAGs – are tractable [18].

In this paper, we prove that superpositions not only enrich the structure, but also impact the complexity of combinatorial games: When starting at a quantum position, both QUANTUM NIM and QUANTUM UNDIRECTED GEOGRAPHY make a *complexity leap* over NP. The former becomes Σ_2^P -hard and the latter becomes PSPACE-complete.

Complexity Characterization of Quantum Undirected Geography. We show that this is PSPACE-complete, even in a position resulting from polynomially-many (quantum) moves from a classical position. We complement this with an algorithmic result, highlighting the fragility of this game. In Section 2.1, we present a polynomial-time algorithm for solving QUANTUM UNDIRECTED GEOGRAPHY when starting at any classical position. The proof of correctness requires carefully-designed strategic moves supported by graph contraction

⁵ The rapid exponential growth in decision time makes searching for winning strategies challenging even for game boards with moderate-sizes, such as 19 by 19 for GO or 14 by 14 for Nash’s “optimal” HEX size [30].

to extend the matching theory for classical UNDIRECTED GEOGRAPHY [18]. In addition to highlighting the fundamental difference between classical and quantum starts, this algorithmic result provides a sharp contrast to the recent result that the disjunctive sum of two UNDIRECTED GEOGRAPHY games is PSPACE-complete [7].

Complexity Leap in Quantum Nim and Beyond. Our Σ_2^p -hardness proof of QUANTUM NIM takes several turns, each of which provides new insight. It involves two logic games of Schaefer [31]: AVOID TRUE and PARTITION-FREE QBF.⁶ We establish the following: (1) QUANTUM AVOID TRUE and QUANTUM NIM with Boolean starting quantum position (i.e., each pile in a realization has one or zero stones) are isomorphic to each other. (2) QUANTUM PARTITION-FREE QBF is polynomial-time reducible to QUANTUM AVOID TRUE. (3) QUANTUM PARTITION-FREE QBF is Σ_2^p -complete. These technical steps have several basic implications: First, because AVOID TRUE is PSPACE-complete [31], this isomorphism represents a significant use of superpositions of NIM positions – individually polynomial-time solvable – to encode the intractable game. The encoding shows superposition can be more expressive than disjunctive sum. Second, the isomorphism aforementioned suggests an in-between transformation of rulesets: A *Demi-Quantum* combinatorial game lives in the classical world with an initial quantum endowment: It starts in a quantum position, but during the game, only classical moves are allowed. Our isomorphism result proves that DEMI-QUANTUM NIM is PSPACE-hard. Third, our isomorphism is not between QUANTUM BOOLEAN NIM and AVOID TRUE, but with QUANTUM AVOID TRUE.

The need to prove the latter’s intractability led us to a family of fundamental questions, centered around an intuitive conjecture: The quantum generalization of any combinatorial game is always as hard (computationally) to play as the game itself. Particularly, *does a PSPACE-hard combinatorial game remain PSPACE-hard in the quantum setting?*

We present a complexity-theoretical refutation to this question. Furthermore, motivated by Ko’s separation of the polynomial-time hierarchy [26],⁷ we prove that for any integer $k > 0$, there exists a PSPACE-complete game whose quantum extension is complete exactly for level k in the polynomial-time hierarchy (in the full version of the paper). Our Σ_2^p -hard proof naturally extends to the quantum generalization of SUBTRACTION games, a family of arithmetic strategy games widely used to teach young children about mathematical thinking. In Section 3.3, we formulate general “Robust Binary-Nim Encoding” properties sufficient for supporting a reduction from QUANTUM NIM, implying that several natural games including BRUSSELS SPROUTS, GO and DOMINEERING are Σ_2^p -hard in the quantum setting.

Practical Board Games. Our studies have inspired the design of a practical board game, called TRANSVERSE WAVE, based on the isomorphism between QUANTUM BOOLEAN NIM and AVOID TRUE. In our web-based version of the game (given in Supplementary Material), one can play either against another human player (sitting at the same computer) or some of AI programs. Being a PSPACE-complete impartial game with simple rules played on a colorful 2D-grid board, this board game has several desirable properties outlined by Eppstein [15]. This board game also enabled the implementation of QUANTUM NIM and DEMI-QUANTUM NIM.

⁶ See Section 3.1 for the rules of these two logic games.

⁷ Ko proved the following: For any integer $k > 0$, there exists an oracle O_k such that *relative to* O_k , the polynomial-time hierarchy collapses to exactly to level k .

2 Exploring Quantum Undirected Geography

We first focus on QUANTUM UNDIRECTED GEOGRAPHY. For this ruleset, each move is indicated by a vertex; a classic move is feasible if the indicated vertex is adjacent to the current vertex. Our two complementing algorithmic/complexity characterizations reveal a majestic computational landscape with PSPACE-hard peaks over quantum positions, dipping into Polynomial-Time valleys around classical instances. Furthermore, some PSPACE-hard positions are reachable from classical positions via polynomial-length quantum paths, making them realizable in QUANTUM UNDIRECTED GEOGRAPHY from classical starts.

2.1 Strategic Graph Contraction: Tractability of Classical Starts

► **Theorem 1** (Efficient Solution for Classical Starts). *For any UNDIRECTED GEOGRAPHY position Z , $Z^{\mathcal{Q}(2)}$ with the same starting position are solvable in polynomial time.*

We will prove the theorem by establishing that for any undirected graph $G = (V, E)$ and starting vertex $s \in V$, position (G, s) is a winning position (of the current player) in the quantum setting with 2-wide quantum moves if and only if (G, s) is a winning position in the classical UNDIRECTED GEOGRAPHY. The classical case can be solved by the Fraenkel-Scheinerman-Ullman algorithm guided by an elegant matching theory: (G, s) is a winning position if and only if s is in every maximum matching of G [18]. Our graph-contraction-based algorithm extends this matching theory to the quantum setting.

In order to prove this, we will first present the algorithm that the winner in the classical game (*hero*) will use to win under quantum play. The hero will always make classical moves, but we need to show how they respond to quantum moves by their opponent, the *villain*. If the villain makes a quantum move, the hero will try to make a winning collapsing move (described further below). If they cannot, they can still win by keeping track of the quantum superposition, treating the two quantumly-chosen vertices as one combined (*contracted*) vertex. The hero keeps track of an overlaid graph $G' = (V', E')$: (A) Initially, $V' = \{\{v\} \mid v \in V\}$. We refer to $c(v)$ (the contraction with v) as the element of V' that contains v . (B) E' will be updated so that $(X, Y) \in E' \Leftrightarrow X, Y \in V'$ and $\forall x \in X, y \in Y : (x, y) \in E$. (C) Whenever a player makes a classical move from $a \rightarrow b$ (meaning the current player makes a classical move after the previous player makes a classical move) the hero will remove $c(a)$ from V' and all incident edges from E' . (D) Whenever the villain makes a quantum move $a \rightarrow \langle v_1 \mid v_2 \rangle$, the hero will again remove $c(a)$ from V' and all incident edges from E' . (E) Whenever the hero follows a quantum move with a classical move, $\langle v_1 \mid v_2 \rangle \rightarrow b$, the hero will update G' based on whether they make a collapsing move: (E.1) If the hero collapses that quantum move, then they can remove the remaining v_i as though it had been a classical move by the villain. (E.2) If the hero does not collapse, but $c(v_1) = c(v_2)$, then all of those vertices have all been visited in all realizations. The hero can remove $c(v_1)$ from V' and all incident edges from E' . (E.3) If the hero does not collapse and $c(v_1) \neq c(v_2)$, then the hero removes both $c(v_1)$ and $c(v_2)$ from V' and replaces them with $c(v_1) \cup c(v_2)$. Then the hero will reset E' to match the definition given above.

Next, we describe how the hero chooses their move, using maximum matchings on G' . (In our notation, we consider a matching, M , as both a set of pairs and a function. So, $(a, b) \in M \Leftrightarrow M(a) = b \Leftrightarrow M(b) = a$.) In the classical winning position, the current vertex is in all maximum matchings on the graph [18], meaning after the classical winner's turn, the loser must start from a vertex not contained in some maximum matching. Our hero will maintain a similar invariant on G' : there is a maximum matching on G' such that the villain will be starting their turn on a vertex not in that matching. Our algorithm is as follows:

- If the villain makes a classical move to v , the hero considers any maximum matching, M on G' , and then moves to $x \in M(c(v))$ such that $(v, x) \in E$. This leaves us with a maximum matching, $M \setminus \{(c(v), c(x))\}$ on the remaining graph that does not include $c(x)$, thus upholding the invariant. The hero removes $c(v)$ and $c(x)$ from V' .
- If the villain makes a quantum move to $\langle a \mid b \rangle$, and $c(a) = c(b)$, then the hero acts as though the villain moved classically to only a , finds a maximum matching, M on G' , then moves to $x \in M(c(a))$ such that $(a, x) \in E$. Subtracting $(c(a), c(x))$ from M results in a maximum matching without $c(x)$, upholding the invariant. The hero removes $c(a) = c(b)$ and $c(x)$ from the partition V' they are keeping track of.
- If the villain moves to $\langle a \mid b \rangle$, and $c(a) \neq c(b)$, then the hero has additional work to do. Notably, they find a matching, M , such that $\exists x \in M(c(a))$ where $(a, x) \in E$, but $(b, x) \notin E$; or $\exists x \in M(c(b))$ where $(b, x) \in E$, but $(a, x) \notin E$, if one exists. There are three cases:
 1. If $\exists M$ with $x \in M(c(a))$ where $(a, x) \in E$, but $(b, x) \notin E$, then the hero moves to x . Since $(b, x) \notin E$, the realization where the villain moved to b collapses out. Subtracting $(c(a), c(x))$ from M yields a maximum matching without $c(x)$, upholding the invariant. The hero removes $c(a)$ and $c(x)$ from V' .
 2. If $\exists M$ with $x \in M(c(b))$ where $(b, x) \in E$, but $(a, x) \notin E$, then the hero moves to x as in the previous case, with a and b swapping roles, then removes $c(b)$ and $c(x)$ from V' .
 3. If no maximum matching exists with those requirements, then the hero can just use any maximum matching, M , to make their move. For any maximum matching M :

$$\forall x \in M(c(a)) : (b, x) \in E \quad \text{and} \quad \forall y \in M(c(b)) : (a, y) \in E.$$

The hero can now move to any x and update V' by removing $c(x)$ and contracting $c(a)$ and $c(b)$ into one element $c(a) \cup c(b)$. In order to continue safely, the hero needs this new contracted vertex to be adjacent to $c(y)$, for any $y \in M(c(b))$. Thus, the hero needs both that $(c(a), c(y)) \in E'$ and $(c(b), c(y)) \in E'$. The latter is already true because $M(c(b)) = c(y)$. For the former, by Lemma 2 below, (a, x) and (b, y) are in a maximum matching on G . Thus, there is another maximum matching with the swapped edges (a, y) and (b, x) . Then, by Lemma 3, $(c(a), c(y)) \in E'$.

► **Lemma 2.** *If $(c(a), c(b))$ is in a maximum matching of G' , then in any realization where neither a nor b has been used, (a, b) is part of a maximum matching on the unvisited graph.*

Proof. Let M be the maximum matching on G' containing $(c(a), c(b))$. Also let H be the remaining subgraph of G that hasn't been visited in the given realization. Then, for each $X \in V'$, there is exactly one vertex in H remaining. Due to the definition of E' , that vertex must be adjacent to the vertex of H inside the contraction $M(X)$. Thus, each matched pair in M corresponds to exactly one unique pair of neighbors in H , which creates a maximum matching on that graph. Thus, a and b must be neighbors and (a, b) is in a maximum matching on H . ◀

► **Lemma 3.** *If (a, b) is in a maximum matching on G and $c(a) \neq c(b)$, then at any point in the game where a and b are still included in contractions, $(c(a), c(b)) \in E'$.*

Proof. We prove this by contradiction. Assume $(c(a), c(b)) \notin E'$. Thus, $\exists (a', b') \notin E$, where $a' \in c(a)$ and $b' \in c(b)$. Without losing generality, we assume that the most recent contraction breaks the statement of the lemma; all prior contraction-graphs G' contained all edges from all maximum matchings in G . Assume that the villain's last quantum move was to $\langle x \mid y \rangle$ and the hero had to respond to a non-collapsing move at vertex z . Thus: (1) In the prior contraction, c' , $c'(x) \neq c'(y)$. (2) $c'(x) \cup c'(y) = c(a)$. (3) $\exists M$, a matching on G' that the hero used to choose z . (4) WLOG, $a' \in c'(x)$ and $b' \in M(c'(y))$. ◀

Proof of Theorem 1. The invariant the hero maintains is: at the end of the hero’s turn, after having moved to x , there is a maximum matching on G' that does not contain the contracted vertex $c(x)$. Thus, either (A) There are no more edges leaving x and the villain loses immediately, or (B) If there is a move and the villain moves from x to y , then y must be contained in one of the other matched pairs, meaning that the hero will be able to move to y ’s match. (Lemma 2 requires that y is part of one of those matches, because if it wasn’t, then (x, y) would be part of a maximum matching on G and that edge will be represented as an edge in a maximum matching in G' , which won’t work with the invariant.) When there are no moves left in G' , there are no moves left in G , since if the edge (x, y) has $c(x) = c(y)$ then only one vertex is remaining in each realization, and if $c(x) \neq c(y)$, then $(c(x), c(y))$ must be in G' . The invariant will be maintained because each turn the hero will start on a vertex in all maximum matchings of G' and will traverse the edge from one of them. Since the hero will always have a move to make on their turn, they will never lose the game. ◀

2.2 Intractability of Quantum Geography Positions



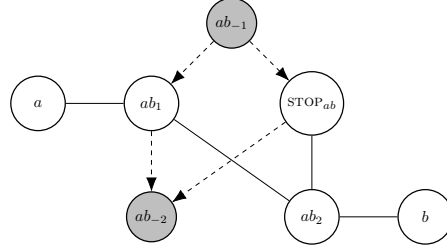
■ **Figure 2** Gadgets for reducing from a GEOGRAPHY edge (a, b) . The left is the gadget as it will appear in the *main-board* and all other boards except from (a, b) -board. The right is the edge as it appears in (a, b) -board. All other parts of the two boards will be the same.

► **Theorem 4** (Intractability of Quantum Starts). *QUANTUM UNDIRECTED GEOGRAPHY, when one begins with a superposition of polynomial width (“polynomially wide”) is PSPACE-complete.*

Proof. The game tree has height $\leq n = |V|$. As seen in the full version of this paper, the quantumization with poly-wide quantum start is in PSPACE. For hardness, we reduce from Quantum GEOGRAPHY, the original version which includes directed edges (and which we prove to be PSPACE-hard in Theorem 21). Consider a GEOGRAPHY instance where the underlying graph has n nodes and m edges. We will create an (undirected) superposition that consists of $m + 1$ entangled realizations. (1) One realization, *main-board*, has a very similar structure, but each arc (a, b) is replaced by a three-part path with two new vertices, ab_1 and ab_2 : (a, ab_1) , (ab_1, ab_2) , and (ab_2, b) . (2) For each arc (a, b) , we include (a, b) -board, which is exactly the same as *main-board* except there is an extra vertex, STOP, and the edge (ab_1, ab_2) is replaced with $(STOP, ab_2)$.

We show the two relevant edge gadgets in Figure 2. The *main-board* can never collapse out unless a player moves to STOP, since all other edges are also in other realizations. If a player does move to STOP, the game immediately ends, as all other realizations with STOP collapse. Thus, the game either ends when there are no moves left in the main game board, or STOP is entered. Additionally, a player can move to STOP iff their opponent reached the ab_2 vertex by moving from b , corresponding to going backwards on arc (a, b) in the Quantum GEOGRAPHY game. This cannot happen when moving from ab_1 since traversing the (ab_1, ab_2) edge collapses out the realization with an edge to STOP. Thus, traversing an edge backwards is always a losing move. Thus, winning strategies for QUANTUM UNDIRECTED GEOGRAPHY, correspond exactly to those for QUANTUM GEOGRAPHY. ◀

► **Theorem 5** (Intractability: Reachable Quantum Positions). *QUANTUM UNDIRECTED GEOGRAPHY is PSPACE-complete for positions reachable with polynomial number of moves after a classical start.*



■ **Figure 3** Gadget for a GEOGRAPHY edge (a, b) in UNDIRECTED GEOGRAPHY. Prior to the current position, the quantum moves $ab_{-1} \rightarrow \langle ab_1 \mid STOP_{ab} \rangle \rightarrow ab_{-2}$ were made.

Proof. To prove this, we use the reduction from Theorem 4, which contains what we refer to as the *core realizations*. Our new usage differs in that we have separate vertices $STOP_{ab}$ instead of a common STOP. We call the non-core (exponential number of) realizations *redundant realizations*, because the players can ignore them. This is because, in each redundant realization, R : (1) There is a core realization such that if it collapses, so does R , and (2) At any position in the game, there is a core realization that contains all available moves in R . To complete the proof, we show that we can reach a superposition consisting only of (1) all core boards and (2) redundant boards. For our new starting position, we take all vertices and edges in our main board from the Theorem 4 reduction, and add vertices and all new edges. As shown in Figure 3, for each arc (a, b) in Quantum GEOGRAPHY we can insert vertices ab_1 , ab_2 , and $STOP_{ab}$, along with edges (a, ab_1) , (ab_1, ab_2) , $(STOP_{ab}, ab_2)$, and (ab_2, b) .

In addition, we include two vertices, ab_{-1} and ab_{-2} , which will have already been previously visited in all realizations by the time we reach the start. These vertices are connected by edges (ab_{-1}, ab_1) , $(ab_{-1}, STOP_{ab})$, (ab_1, ab_{-2}) , and $(STOP_{ab}, ab_{-2})$. For some other arc, (c, d) , we can connect the gadgets by setting $ab_{-2} = cd_{-1}$. Now we prescribe a series of prior moves across all edges $\{(a_i, b_i) \mid i \in \{1, \dots, m\}\}$: $(a_1 b_1)_{-1} \rightarrow \langle (a_1 b_1)_1 \mid STOP_{a_1 b_1} \rangle \rightarrow (a_1 b_1)_{-2} = (a_2 b_2)_{-1} \rightarrow \dots \rightarrow (a_{m-1} b_{m-1})_{-2} = (a_m b_m)_{-1} \rightarrow \langle (a_m b_m)_1 \mid STOP_{a_m b_m} \rangle (a_m b_m)_{-2} \rightarrow x$ were made, where x is the starting vertex of Quantum GEOGRAPHY. By construction, no realizations were ever collapsed in these prior moves. Also, there is a realization for each of the core realizations, by just following the branch that moved to STOP for each edge gives us the main-board. Following the branch that moved to STOP for each edge gadget other than some edge $e = (a, b)$ gives us the (a, b) -board. We now only need to show that the rest of the realizations are redundant. All realizations have only the edges in either the main-board or an (a, b) -board fulfilling the first property for redundancy. For the second property, we note that each of the non-core realizations must contain at least two STOP vertices; its current vertex is either adjacent to one stop (all its moves are in the core realization with the stop) or not (all its moves are in the main board). ◀

3 Complexity Leap in Quantum Nim: Logic Connection

In this section and the next section, we prove the following theorem:

► **Theorem 6** (Quantum Leap Over NP). *QUANTUM NIM with quantum starts is Σ_2^P -hard.*

3.1 Proof Outline: The Logic Connection of Quantum Nim

Our proof of Theorem 6 is involved. We first give its high-level steps, connecting QUANTUM NIM to the quantumized AVOID TRUE. AVOID TRUE is an impartial logic game introduced in Schaefer’s landmark paper [31]. A position is given by a *positive* CNF formula, with all variables set to **false**. Two players take turns flipping a variable from **false** to **true**. A move is *feasible* if setting its variable **true** will not make the whole CNF evaluate to **true**.

► **Theorem 7.** *Let QUANTUM BOOLEAN NIM denote the game whose position is a superposition of Boolean NIM. in which each pile has either zero or one stone. Then, QUANTUM BOOLEAN NIM and QUANTUM AVOID TRUE are isomorphic rulesets, i.e. they have isomorphic game trees.*

In Section 4, we will demonstrate the complexity of QUANTUM AVOID TRUE through another logic game, PARTITION-FREE QBF, which is a *partisan* game also played on a formula. Player True wants to make the formula **true** while False wants to make it **false**. Boolean variables of the formula are *partitioned* into two sets, one for each player. At their turn, players can “freely” set any of their unassigned variables. Using a delicate argument to counter an “unwelcome” quantum impact, in Section 4.2 we extend Schaefer’s reduction from PARTITION-FREE QBF to AVOID TRUE in the quantum setting as well.

► **Theorem 8** (Partisan-Improvement Reduction in Quantum Setting). *There exists a polynomial-time reduction from QUANTUM PARTITION-FREE QBF to QUANTUM AVOID TRUE.*

In Section 4.1, we study the quantumized complexity of the family of PSPACE-complete QBF games as a whole, addressing the subtlety of transforming QBF-games into normal-play games in the quantum setting. In particular, we establish the following:

► **Theorem 9** (Complexity Collapses: Σ_2^P -Hardness). *QUANTUM PARTITION-FREE QBF with a classical start is Σ_2^P -complete for even alternation and Π_2^P -complete for odd alternation.*

Motivated by celebrated complexity characterizations of Lander’s NP-intermediate problems [27] and Ko’s intricate, meticulous separation of levels of the polynomial-time hierarchy [26], in the full version, we refine the last theorem to prove the following:

► **Theorem 10** (Complexity Collapses into Polynomial-Time Hierarchy). *For any integer $k > 0$, for complexity class Σ_k , Π_k , or both Σ_k and Π_k , there exists a classically PSPACE-complete combinatorial game whose quantum generalization is complete in that class.*

3.2 Isomorphism Between Two Natural Quantum-Inspired Games

Proof of Theorem 7. Crucial to our complexity analysis, this isomorphism between QUANTUM BOOLEAN NIM and QUANTUM AVOID TRUE is polynomial-time computable.

First note that QUANTUM AVOID TRUE has the following interesting self-isomorphic property. QUANTUM AVOID TRUE with quantum starts is isomorphic with QUANTUM AVOID TRUE with classical starts. This is because that the superposition of any two classical AVOID TRUE positions is “equivalent” to the classical position defined by the **and** for their CNFs. Thus, focusing on the logic-to-nim direction, we consider a classical AVOID TRUE position (F, V, T) , where $V = \{x_1, \dots, x_n\}$, F is the formula with m clauses, C_1, \dots, C_m , and $T \subseteq V$ denotes the subset of variables set to **true**. (T is empty at the start.)

We now reduce this position to a BOOLEAN NIM superposition $\mathbb{B}_{(F,V,T)}$ with m realizations – one for each clause – and n piles (encoding Boolean variables). In the realization for C_i , we set piles corresponding to variables in C_i to zero to set up the mapping between

11:12 Quantum-Inspired Games

collapsing the realization with making the clause **true**. We also set all piles associated with variables in T to zero, to set up the mapping between collapsing the realization with selecting a selected variable. We use these two mappings to inductively establish that the game tree for QUANTUM BOOLEAN NIM at $\mathbb{B}_{(F,V,T)}$ is isomorphic to the game tree for QUANTUM AVOID TRUE at (F, V, T) . We first demonstrate this reduction on the following example:

$$\underbrace{(x_1 \vee x_2 \vee x_3 \vee x_4)}_A \wedge \underbrace{(x_1 \vee x_5 \vee x_6 \vee x_7)}_B \wedge \underbrace{(x_1 \vee x_3 \vee x_6)}_C \wedge \underbrace{(x_2 \vee x_5 \vee x_8)}_D$$

and already-chosen variables, $T = \{x_8\}$. The reduction gives three NIM realizations: $A = (0, 0, 0, 0, 1, 1, 1, 0)$, $B = (0, 1, 1, 1, 0, 0, 0, 0)$, and $C = (0, 1, 0, 1, 1, 0, 1, 0)$. The clause corresponding to D was already true, so the corresponding realization has already collapsed.

To prove the correctness of the reduction, we prove that the current player has a winning strategy at (F, V, T) in QUANTUM AVOID TRUE iff the next player has a winning strategy at $\mathbb{B}_{(F,V,T)}$ in QUANTUM NIM. We will prove inductively that the following invariant holds if we play our NIM encoding and its original QUANTUM AVOID TRUE position in tandem: For every active clause in each realization of QUANTUM AVOID TRUE there is exactly one realization in the QUANTUM NIM encoding with the corresponding realization having piles of 0 for each variable in the clause and variables that are no longer **false** for that position.

As the basis of the induction, the stated invariant is true at the start. There is one realization of QUANTUM AVOID TRUE. In its NIM encoding constructed above, each realization has piles of 0 for the variables in the corresponding clause, and all variables are **false**, so the rest of the piles are at 1. We show that the invariant still holds after any move in QUANTUM NIM by examining each effect of its “coupled move” on any possible clause.

- Any classical or quantum move targeting a pile of size 1 in the QUANTUM NIM position does not collapse the NIM realization, consistent with the fact that selecting the corresponding variable results in the corresponding clause remaining active in the QUANTUM AVOID TRUE instance. After the move, the pile will now be at 0 in the QUANTUM NIM realization, consistent with the fact that after selecting the corresponding variable in the realization of QUANTUM AVOID TRUE, the variable is no longer **false**.
- Any classical or quantum move targeting a pile of 0 in QUANTUM NIM, that is not in the corresponding QUANTUM AVOID TRUE clause. Since this variable must have already been flipped, that means that it is 0 in all realizations and thus cannot be taken.
- If a classical or quantum move is made in the QUANTUM NIM position on a pile of 0 that corresponds to a variable in the associated QUANTUM AVOID TRUE clause, then the clause is satisfied, and the corresponding realization collapses. This matches the QUANTUM AVOID TRUE case where the clause is no longer active.

And it is through establishing this invariant that we get the desired structural morphism between playing QUANTUM AVOID TRUE and playing its encoding QUANTUM NIM.

The inverse encoding from QUANTUM BOOLEAN NIM to AVOID TRUE is the following: Given a position \mathbb{B} in QUANTUM BOOLEAN NIM, we create a clause from each realization in \mathbb{B} . Suppose \mathbb{B} has m realizations and n piles. We use n Boolean variables, $V = \{x_1, \dots, x_n\}$. For each realization in \mathbb{B} , the clause consists of all variables corresponding to piles with zero pebbles. The reduced CNF $F_{\mathbb{B}}$ is the **and** of all these clauses. Taking a stone from a pile collapses a realization for which the pile has no stone is mapped to selecting the corresponding Boolean variable making the clause associated with the realization **true**. Thus, playing QUANTUM BOOLEAN NIM at position \mathbb{B} is isomorphic to playing AVOID TRUE starting at position $(F_{\mathbb{B}}, V, \emptyset)$. Note that the reduction can be set up in polynomial time. ◀

3.3 Nim Encoding and Structural Witness of Σ_2^P -Hardness

In our reduction from Σ_2^P -hard QUANTUM AVOID TRUE, our QUANTUM NIM game uses a poly-wide superposition of perhaps the simplest NIM positions: all NIM positions that we used in our encoding are from the family of BOOLEAN NIM. So, our Σ_2^P -hard intractability proof of QUANTUM NIM holds for QUANTUM BOOLEAN NIM and can be extended broadly to the quantum extension of SUBTRACTION games. In this section, we present a more systematic theory to extend these hardness results. Particularly, we can use the Σ_2^P -hard result get results for several other games in the quantum setting. We show that if a game is able to classically properly embed a binary game with properties that we will describe, then its quantumized complexity at a poly-wide superposition is at least Σ_2^P -hard.

► **Definition 11** (Robust Binary-Nim Encoding). *A ruleset \mathcal{R} has a robust binary-Nim encoding if \mathcal{R} has a position b with the following properties for any n . (1) b has a set M of $n = |M|$ distinct feasible moves. (2) For each $\sigma \in M$, selecting σ moves the game from b to a losing position b_σ , such that position b_σ has $(n - 1)$ feasible moves given by $M \setminus \{\sigma\}$. In addition, b_σ recursively induces a robust binary-Nim encoding for $(n - 1)$. (3) When $n = 0$, the game will have no available moves.*

► **Lemma 12.** *If a game \mathcal{R} has a robust binary-Nim encoding, then it is Σ_2^P -hard to determine the winnability of poly-wide quantum positions in its quantum setting.*

Proof. We reduce from QUANTUM BOOLEAN NIM at a poly-wide superposition. Suppose there are n piles and a superposition with m realizations $\langle b_1 \mid \dots \mid b_m \rangle$. Now we focus on the position b in \mathcal{R} that induces the robust binary-Nim encoding for n , with set of moves $M = \{\sigma_1, \dots, \sigma_n\}$. For each pile i , we associate it with move σ_i . Each realization, $b \in \{b_1, \dots, b_m\}$, in QUANTUM BOOLEAN NIM defines a set $S \subseteq [n]$, corresponding to piles with value 1. In the reduced quantum position for \mathcal{R} , we make a realization with the position whose feasible moves are $\{\sigma_i \mid i \in S\}$. This is a direct encoding, where we are just relabeling move i to σ_i , setting up the desired morphism between playing QUANTUM BOOLEAN NIM and playing its encoding in the quantum setting of \mathcal{R} . ◀

We can easily embed BOOLEAN NIM in several games, even games with fixed outcomes like BRUSSELS SPROUTS. For “interesting” games, such as GO and DOMINEERING, this process is often simple, as one needs only to be able to make a board state that is a sum of combinatorial game value $*$ with only a single move in each of them.

4 On the Complexity of Quantum Avoid True

It is well known that the Quantified Boolean formula problem (QBF) – determining whether a quantified Boolean formula is true or false – can be viewed as a classical combinatorial game. Technically, combinatorial games must also fulfill the “normal play” requirement, meaning that a player loses *if and only if* they are unable to make a move. There are several equivalent ways to do this (transforming logical QBF decisions into combinatorial games) and are all classically polynomial-time reducible to each other. Complexity-theoretically, QBF is the canonical complete problem for PSPACE, and thus all these QBF variants are PSPACE combinatorial games; they form the bedrock of PSPACE reductions. However, because quantum games are more subtly dependent on the explicit move definitions in the ruleset, these known reductions from the “classical world” don’t necessarily continue to hold. So, we need to define a variant that we will use to prove the intractability of AVOID TRUE.

4.1 Quantum Collapse

In this subsection, we consider one natural “end-of-QBF” transformation that contributes to the complexity collapse in the quantum setting. This is critical for our AVOID TRUE proof, and thus allows us to get the hardness for NIM’s quantumized complexity. In the literature, the Quantified Boolean formula problem is also known as the Quantified Satisfiability Problem (QSAT). However, in this paper, we will – for clarity of presentation – use the following ruleset naming convention for QBF and QSAT, in order to denote two different ruleset families of combinatorial games rooted in the Quantified Boolean formula problem.

The QBF family: We will use QBF to denote the family of games that *textually* implements the Quantified Boolean formula problem: An instance of QBF is given by a CNF f , whose clauses may contain both positive and negative literals, over two ordered lists of Boolean variables. We have two cases: if there are even variables, then it has the form (T_1, \dots, T_n) and (F_1, \dots, F_n) . Otherwise, it has the form $(T_1, \dots, T_n, T_{n+1})$ and (F_1, \dots, F_n) . So, the quantified boolean formula is, respectively:

$$\begin{aligned} \text{Case Even:} \quad & \exists T_1 \forall F_1 \cdots \exists T_n \forall F_n \quad f(T_1, \dots, T_n, F_1, \dots, F_n) \\ \text{Case Odd:} \quad & \exists T_1 \forall F_1 \cdots \exists T_n \forall F_n \exists T_{n+1} \quad f(T_1, \dots, T_n, T_{n+1}, F_1, \dots, F_n) \end{aligned}$$

In this game, one player – *Player True* – aims to satisfy the CNF formula, while the other player – *Player False* – wants the formulae to be unsatisfied. Starting with True, the two players alternate turns setting their next variables to True or False. In other words, in their respective i^{th} turn, Player True sets variable T_i , then Player False sets variable F_i . The variants in this family, as we shall define later, differ in the details of the termination condition in transforming logic QBF into a combinatorial game.

The QSAT Family: We will use QSAT to denote the family of *partition free assignment* QBF games, with relaxations on the variables the players can choose to set at each turn. As in QBF, each instance of QSAT includes a CNF formulae f , and two players – Player True and Player False – who take turns to set their own variables; Player True can only set variables in $\{\cdots T_i \cdots\}$ and Player False can only set variables in $\{\cdots F_i \cdots\}$. Again, we have two cases – Case Even and Case Odd – depending on which player has one more variable. For the game, Player True aims to satisfy the CNF formulae, while Player False wants refute them. Unlike in QBF, the players can “freely” set any of their unassigned variables to True or False, as opposed to setting variables according to the prescribed order. In the variant we will call PHANTOM-MOVE QBF, after all variables have been assigned, the next player has a feasible (*phantom*) move available iff they have a winning assignment.

► **Theorem 13** (Quantum Collapses of QBF: Classical Starts). *QUANTUM PHANTOM-MOVE QBF with a classical start is Σ_2^P -complete in Case Even, and is Π_2^P -complete in Case Odd.*

Proof. Notice that the player who makes the final move (the “phantom move”) will be allowed to choose to collapse any quantum moves so that the variables are assigned in a way that they will win (if it is possible). Since quantum moves cannot collapse before the end of the game, they can choose the assignments for any variables made during assignments by either player. As such, the optimal move for this collapsing player will be to make exclusively quantum moves, and the optimal move for other player will be to never make a quantum move. So, one player makes a variable assignment of half of the variables, and then only wins if every other assignment of the other variables are winning positions for them. Otherwise, the phantom move player wins. If the phantom move player is False, then this is exactly the Σ_2^P SAT problem. Otherwise, it is exactly the Π_2^P SAT problem.

For formal completeness, we go through the trivial two-way reduction. In Case Even, we can create a Σ_2^P SAT instance, giving True’s variables as the “exists” variables, and False’s variables as the “for all” variables. In Case Odd, we can use a Π_2^P SAT instance by similarly giving False’s variables as the “for all” variables, and True’s variables as the “exists” variables. As previously described, both of these will output which player wins correctly.

For the other direction, we can do the same with both, only in the other direction. In other words, make all “exist” variables into True’s variables and all “for all” variables into False’s variables. Then, we have a corresponding QUANTUM PHANTOM MOVE QBF instance. ◀

We now turn our attention to PARTITION-FREE QBF and its quantumized complexity. In contrast to standard QBF games, the partition-free version relaxes the order requirement on setting the variables. As we discussed before, we will refer to this family of QBF-based games as the QSAT family. Classically, all variants of QSAT are PSPACE-complete games.

This variant of the game is implicitly invoked in several classic reductions, including Schaefer’s AVOID TRUE reduction [31], and we will use this fact to prove the quantum version.

► **Theorem 14** (Quantum Collapses of Partition-Free QBF). *PHANTOM MOVE QSAT with a classical start is Σ_2 -complete in Case Even and Π_2 -complete in Case Odd.*

Proof. We will call the player that makes the phantom move the PM Player and the other player as the NM Player, for short. Then, the theorem follows from the following observations:

► **Observation 15.** *If the PM Player has a winning strategy, then arbitrarily assigning each variable to $\langle \text{true} \mid \text{false} \rangle$ is also a winning strategy.*

Proof. Suppose for the sake of contradiction that the PM player has a winning strategy S_1 but the only quantum strategy S_2 isn’t a winning strategy. Then, that means that NM Player has a sequence of moves such that in no realizations, the PM player has fulfilled their winning condition, as otherwise they could move into the phantom move. But then, that exact sequence of moves is a winning sequence of moves against S_1 , and any possible deviations of it. Therefore S_1 isn’t a winning strategy. ◀

► **Observation 16.** *If the NM player has a winning strategy, then they have a winning strategy of selecting classic moves independent the PM player’s choices.*

Proof. If the NM player has a winning strategy against PM player, then they must also have one against the PM Player’s strategy of only selecting quantum assignments as mentioned in Observation 15. Regardless of what this strategy is, the fact that it wins means that it must win against all realizations of this strategy, since otherwise the PM Player could collapse to that realization on the phantom move, then win. The NM Player can do this with only classic moves, as any realization of a quantum strategy must also not have any winning realizations for the NM Player either. ◀

From here, we can form a trivial reduction to and from Σ_2^P -SAT and Π_2^P -SAT for instances where False gets the phantom move and True gets the phantom move, respectively. ◀

4.2 Quantum Lift of Schaefer’s Partisan-Impartial Reduction

We now prove Theorem 6, showing that QUANTUM BOOLEAN NIM is intractable, with complexity between NP and PSPACE. We will reduce to it – via QUANTUM AVOID TRUE – from the quantum generalization of PHANTOM-MOVE QSAT. In PHANTOM-MOVE QSAT, one can equivalently define the “phantom move” to be included with the final variable

selection, such that the player that would assign the final variable may only do so if they will have reached their win condition upon playing that variable. One can reduce from the other Phantom move variant by introducing a variable v_{n+1} which only appears in a clause as $(v_{n+1} \vee \neg v_{n+1})$ given to the phantom move player. So the hardness remains the same. We will be reducing from this variant of the game. Classically, this game, as proved by Schaefer [31], is PSPACE-complete. Below, we will use this fact to establish the following theorem.

► **Theorem 17** (Complexity of QUANTUM AVOID TRUE). *QUANTUM AVOID TRUE is Σ_2^P -hard.*

Proof. Consider a PHANTOM-MOVE QSAT instance Z with CNF. In the proof, we will focus on the case where both players have the same number of variables, which is an even number. This is acceptable since the hardness results remain even when fixed to any arbitrary parity. Recall that the player that goes first will be called the *True player* and the player that goes second the *False player*. Below in both Z and its quantum generalization Z^Q , we will denote the True Player’s variables by $\{T_1, \dots, T_m\}$ and the False Player’s variables by $\{F_1, \dots, F_m\}$.

In our proof, we will show that Schaefer’s reduction from PHANTOM-MOVE QSAT to AVOID TRUE in the classical setting can be quantum lifted into a QUANTUM AVOID TRUE instance to encode Z^Q : Schaefer’s reduction (in our notation) is the following: (1) For each True Player’s variable T_i , we introduce two new variables T_{i_1} and T_{i_2} , and create a positive clause $(T_{i_1} \vee T_{i_2})$. We will collectively call these clauses *TV clauses*. T_{i_1} will represent assigning a true literal, and T_{i_2} will represent assigning a false literal. (2) For each False Player’s variable F_i , we introduce three new variables F_{i_1} , F_{i_2} , and F_{i_G} , and create a clause $(F_{i_1} \vee F_{i_2} \vee F_{i_G})$. We will collectively call these clauses *FV clauses*. As before, F_{i_1} will represent assigning a true literal, and F_{i_2} will represent assigning a false literal. The F_{i_G} variable is simply an arbitrary variable for the purposes of ensuring a certain clause parity (which we will give the motivation for later). It will function as an alternate truth literal assignment, as it appears in every clause F_{i_1} does. (3) For each of the clauses in Z , we replace each instance of positive literal T_i and F_i with T_{i_1} and F_{i_1} , respectively; we replace each instance of negated variable $\neg T_i$ and $\neg F_i$ with T_{i_2} and F_{i_2} , respectively. We will collectively call these clauses *QBF clauses*. (4) For each instance of a variable T_{i_1} , F_{i_1} , T_{i_2} , and F_{i_2} in the QBF clauses, we add, to same clause, variables T'_{i_1} , $F_{i_G} T'_{i_2}$, or F'_{i_2} , respectively. We will collectively call these new variables *duplicate variables*. These serve both as a way to ensure the QBF clauses all have even parity, and for player strategy, as we will cover later.

Through our proof, we will let X denote the AVOID TRUE instance obtained from Schaefer’s reduction of PHANTOM-MOVE QSAT instance Z . We will call the first player in X Player True and the second player Player False. Schaefer proved that True can win (the impartial) X if and only if True can win (the partizan) Z . Below, we will extend Schaefer’s proof to show that their quantum generalization Z^Q and X^Q has the same winner (when optimally played). Before going on with our proof, we first recall one of the key properties, formulated by Schaefer, and extend it to the quantum setting.

► **Lemma 18** (Avoid-True Destiny). *For any classical position created by Schaefer’s reduction, if all unsatisfied clauses have an even number of variables, then Player True will win, under arbitrary play by both players for the remainder of the game. If instead all unsatisfied clauses have an odd number of variables, then Player False will win, under arbitrary play by both players for the remainder of the game.*

Lemma 18 captures the “parity” design used in Schaefer’s reduction. It also defines a scenario satisfying the condition of “quantumness doesn’t matter.”

► **Lemma 19** (When Quantum Doesn't Matter). *If all realizations of a game classically have only one possible winner, no matter what sequence of moves either player makes, then quantumness doesn't matter.*

Proof. In order for a game to classically have only a single possible winner, all paths to a leaf node in a game tree must be of the same parity. For a game to end, all realizations must have no moves remaining. Otherwise, the rule sets would allow for some kind of move to be made. As such, all games end with all remaining realizations at leaf nodes. Since the number of moves to get here is the same in all realizations, all paths to the game's end must have the same parity as the classical game's terminal nodes. Then, that same player wins no matter what moves either player makes. Therefore, quantumness doesn't matter. ◀

Combining Lemma 18 and Lemma Lemma 19, we get:

► **Corollary 20** (Quantum-Avoid True Destiny). *For any position reached in Schaefer's reduction, if in all realizations, there are only TV clauses and/or QBF clauses unsatisfied, then False wins; if in all realizations, there are only FV clauses unsatisfied, then True wins.*

To prove the correctness of this reduction in the quantum setting, we simply need to prove that if True has a winning strategy in the instance of Z^Q that we are reducing from, then True has a winning strategy in X^Q , and that if False has a winning strategy in Z^Q , then False has a winning strategy in X^Q . In the proof below, we will crucially use the following fact that we will establish in Observation 16: True has a winning strategy in QUANTUM PHANTOM-MOVE QSAT if and only if True has a single assignment (of only classical moves) that can always win, regardless of what moves False makes. So, if True is the winner in Z^Q , we prescribe the following strategy for True in X^Q : (1) Assign variables in the TV clauses according the winning strategy for Z^Q (2) Assign all of the F'_{i2} variables. Our proof is based on the following key observation is: If True is able to follow this strategy to completion, then all TV and QBF clauses must be satisfied, resulting in a True win by Corollary 20.

To proceed, we just need to show that True's strategy is always a legal set of moves. First, True must be able to satisfy all m TV clauses, because False can't assign all m FV clauses before True, thus whatever TV clause true assigns last can't be the last. Then, because that variable assignment in Z must have satisfied all clauses regardless of how the FV variables were assigned, that means in no realization is it legal to make a move that would satisfy all FV clauses, as all QBF clauses must be satisfied at the same time. Since all realizations still have the FV clauses unsatisfied, True can then assign all F'_{i2} variables, since none of them appear in an FV clause. So, that strategy is always achievable.

Now, for the second case. Suppose that False wins the instance Z^Q . Then, False applies the following strategy: (1) For False's move $i \leq m - 1$, if F_{i1} and F_{i2} are not yet classically assigned, make a quantum move of $[F_{i1} \mid F_{i2}]$. Otherwise, play arbitrarily on the FV clauses not yet assigned in all realizations. (2) For the final move, assign the variable in final FV to either **true** or **false**, whichever is a legal move. If both are, choose arbitrarily.

Note that if False is able to carry out this strategy, then by Corollary 20, they will win the game, as they will have satisfied all FV clauses leaving only TV and/or QBF clauses remaining. For the proof of the correctness of this strategy, first note that, as we will prove in Observation 15, if False has a winning strategy in QUANTUM PHANTOM-MOVE QSAT, then repeatedly choosing, for an arbitrary variable, a quantum assignment of $\langle \mathbf{true} \mid \mathbf{false} \rangle$ is also a winning strategy. If True has only played consistently, then False will be able to make a move on the final unsatisfied FV clause, as there exists a realization where not all QBF clauses are satisfied. If True ever makes a move with at least one realization that isn't

on a variable in a TV clause, then when False moves to the final FV clause, there exists a realization where True hasn't assigned all of the TV clauses, so False can make their final move arbitrarily on the clause. ◀

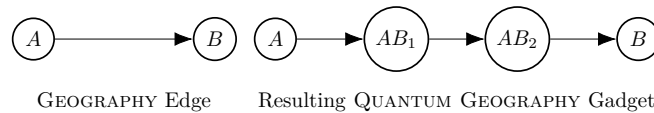
5 Quantum Graph Games

In this section, we consider quantum transformation of several well-studied PSPACE-complete combinatorial games. In addition to GEOGRAPHY, we will also analyze the following games:

- **NODE KAYLES:** An impartial game where players alternate turns placing tokens on vertices of a given graph. A player is only able to place a token on vertex if that vertex does not already contain a token and is not adjacent to any vertex with a token. As such, a player is unable to move when the tokens form an *maximal* independent set.
- **BIGRAPH NODE KAYLES:** A variant of NODE KAYLES in which nodes are partitioned into red nodes and blue nodes, where the blue player can only play on blue vertices and the red player can only play on red vertices.
- **SNORT:** A game where one player is a blue player, and the other is a red player. Players alternate placing tokens of their color onto vertices of a given graph. Players can't place tokens on vertices adjacent to vertices with a token of the opponent's color.

► **Theorem 21.** *QUANTUM GEOGRAPHY with a classical start is PSPACE-complete.*

Proof. We have a very simple reduction from classic GEOGRAPHY. We replace each edge in the GEOGRAPHY graph as shown in Figure 4 with a path through two new vertices.



■ **Figure 4** PSPACE Reduction to QUANTUM GEOGRAPHY is a simple transformation on the edges.

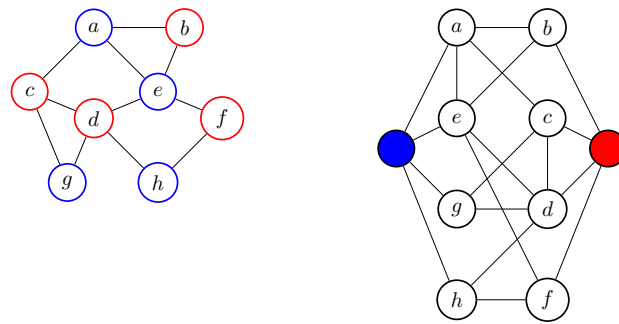
Now, if a player ever makes a quantum move from a classical move, e.g. from A to the super position of AB_1 and AC_1 , then the opponent can immediately collapse to either AB_2 or AC_2 , effectively choosing which of B and C will be moved to. Thus, making a quantum move only gives the next player the power to choose your move and will never give a classically-losing player a winning quantum strategy. ◀

In the full version, we prove the following theorem.

► **Theorem 22.** *QUANTUM NODE KAYLES and QUANTUM BIGRAPH NODE KAYLES are PSPACE-complete.*

► **Theorem 23.** *QUANTUM SNORT is PSPACE-complete.*

Proof. We reduce from BIGRAPH NODE KAYLES. We simply create an extra vertex connected to all red vertices and place a red token on it, and then create a vertex with a blue token on it connected to all blue vertices. See Figure 5. ◀



■ **Figure 5** Graph for QUANTUM BIGRAPHNODEKAYLES, followed by the result of the reduction to QUANTUM SNORT on that graph.

References

- 1 Selim G Akl. On the importance of being quantum. *Parallel processing letters*, 20(03):275–286, 2010.
- 2 M. H. Albert, R. J. Nowakowski, and D. Wolfe. *Lessons in Play: An Introduction to Combinatorial Game Theory*. A. K. Peters, Wellesley, Massachusetts, 2007.
- 3 D. Applegate, G. Jacobson, and D. Sleator. Computer analysis of Sprouts. Technical Report CMU-CS-91-144, Carnegie Mellon University Computer Science, 1991.
- 4 Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for your Mathematical Plays*, volume 1. A K Peters, Wellesley, Massachusetts, 2001.
- 5 Charles L. Bouton. Nim, a game with a complete mathematical theory. *Annals of Mathematics*, 3(1/4):pp. 35–39, 1901.
- 6 Kyle Burke, Matthew Ferland, and Shang-Hua Teng. Quantum combinatorial games: Structures and computational complexity. *CoRR*, abs/2011.03704, 2020. [arXiv:2011.03704](https://arxiv.org/abs/2011.03704).
- 7 Kyle Burke, Matthew Ferland, and Shang-Hua Teng. Winning the war by (strategically) losing battles: Settling the complexity of Grundy-values in undirected geography. In *Proceedings of the 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2021.
- 8 Kyle W. Burke and Shang-Hua Teng. Atropos: A PSPACE-complete Sperner triangle game. *Internet Mathematics*, 5(4):477–492, 2008.
- 9 Kyle Webster Burke. *Science for Fun: New Impartial Board Games*. PhD thesis, Boston University, USA, 2009.
- 10 Nathann Cohen, Mathieu Hilaire, Nicolas Martins, Nicolas Nisse, and Stéphane Pérennes. *Spy-game on graphs*. PhD thesis, Inria, 2016.
- 11 Nathann Cohen, Nicolas A. Martins, Fionn Mc Inerney, Nicolas Nisse, Stéphane Pérennes, and Rudini Sampaio. Spy-game on graphs: Complexity and simple topologies. *Theoretical Computer Science*, 725:1–15, 2018.
- 12 Erik D. Demaine and Robert A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory. In Michael H. Albert and Richard J. Nowakowski, editors, *Games of No Chance 3*, volume 56 of *Mathematical Sciences Research Institute Publications*, pages 3–56. Cambridge University Press, 2009.
- 13 Paul Dorbec and Mehdi Mhalla. Toward quantum combinatorial games. *arXiv preprint arXiv:1701.02193*, 2017.
- 14 Jens Eisert, Martin Wilkens, and Maciej Lewenstein. Quantum games and quantum strategies. *Physical Review Letters*, 83(15):3077, 1999.
- 15 David Eppstein. Computational complexity of games and puzzles, 2006. URL: <http://www.ics.uci.edu/~eppstein/cgt/hard.html>.
- 16 Stephen Finbow, Margaret-Ellen Messinger, and Martin F van Bommel. Eternal domination on $3 \times n$ grid graphs. *Australas. J Comb.*, 61:156–174, 2015.

- 17 Aviezri S. Fraenkel and David Lichtenstein. Computing a perfect strategy for $n \times n$ chess requires time exponential in n . *J. Comb. Theory, Ser. A*, 31(2):199–214, 1981. doi:10.1016/0097-3165(81)90016-9.
- 18 Aviezri S. Fraenkel, Edward R. Scheinerman, and Daniel Ullman. Undirected edge geography. *Theor. Comput. Sci.*, 112(2):371–381, 1993.
- 19 David Gale. The game of Hex and the Brouwer fixed-point theorem. *American Mathematical Monthly*, 10:818–827, 1979.
- 20 Adam Glos and Jarosław Adam Miszczyk. The role of quantum correlations in cop and robber game. *Quantum Studies: Mathematics and Foundations*, 6(1):15–26, 2019.
- 21 Allan Goff. Quantum tic-tac-toe: A teaching metaphor for superposition in quantum mechanics. *American Journal of Physics*, 74(11):962–973, 2006.
- 22 P. M. Grundy. Mathematics and games. *Eureka*, 2:198—211, 1939.
- 23 Robert A. Hearn and Erik D. Demaine. *Games, puzzles and computation*. A K Peters, 2009.
- 24 Takumi Ishizeki and Akihiro Matsuura. Solving quantum tic-tac-toe. *International Conference on Advanced Computing and Communication Technologies*, 2011.
- 25 Valerie King, Satish Rao, and Rorbert Tarjan. A faster deterministic maximum flow algorithm. *Journal of Algorithms*, 17(3):447–474, 1994.
- 26 Ker-I Ko. Relativized polynomial time hierarchies having exactly k levels. *SIAM J. Comput.*, 18(2):392–408, April 1989.
- 27 Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, January 1975.
- 28 David Lichtenstein and Michael Sipser. Go is polynomial-space hard. *J. ACM*, 27(2):393–401, 1980.
- 29 G Mahadevan, T Ponnuchamy, Selvam Avadayappan, and Jyoti Mishra. Application of eternal domination in epidemiology. In *Mathematical Modeling and Soft Computing in Epidemiology*, pages 147–171. CRC Press, 2020.
- 30 John F. Nash. *Some Games and Machines for Playing Them*. RAND Corporation, Santa Monica, CA, 1952.
- 31 Thomas J. Schaefer. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16(2):185–225, 1978.
- 32 A.N. Siegel. *Combinatorial Game Theory*. Graduate Studies in Mathematics. American Mathematical Society, 2013.
- 33 Michael Soltys and Craig Wilson. On the complexity of computing winning strategies for finite poset games. *Theory Comput. Syst.*, 48:680–692, April 2011.
- 34 R. P. Sprague. Über mathematische Kampfspiele. *Tôhoku Mathematical Journal*, 41:438—444, 1935-36.
- 35 David Wolfe. Go endgames are PSPACE-hard. In Richard J. Nowakowski, editor, *More Games of No Chance*, volume 42 of *Mathematical Sciences Research Institute Publications*, pages 125–136. Cambridge University Press, 2002.

Grabbing Olives on Linear Pizzas and Pissaladières

Hungry Coatis¹ ✉

Université Côte d’Azur, Coati, I3S, CNRS, INRIA, Sophia Antipolis, France

Abstract

In this paper we revisit the problem entitled SHARING A PIZZA stated by P. Winkler by considering a new puzzle called SHARING A PISSALADIÈRE. The game is played by two polite coatis Alice and Bob who share a pissaladière (a $p \times q$ grid) which is divided into rectangular slices. Alice starts in a corner and then the coatis alternate removing a remaining slice adjacent to at most two other slices. On some slices there are precious olives of Nice and the aim of each coati is to grab the maximum number of olives. We first study the particular case of $1 \times n$ grid (i.e. a path) where the game is a graph grabbing game known as SHARING A LINEAR PIZZA. In that case each player can take only an end vertex of the remaining path. These problems are particular cases of a new class of games called d -degenerate games played on a graph with non negative weights assigned to the vertices with the rule that coatis alternatively take a vertex of degree at most d .

Our main results are the following. We give optimal strategies for paths (linear pizzas) with no two adjacent weighty vertices. We also give a recurrence formula to compute the gains which depend only on the parity of n and of the respective parities of weighty vertices with a complexity in $\mathcal{O}(h^2)$ where h denotes the number of parity changes in the weighty vertices. When the weights are only $\{0, 1\}$ we reduce the computation of the average number of olives collected by each player to a word counting problem. We solve SHARING A PISSALADIÈRE with $\{0, 1\}$ weights, when there is one olive or 2 olives. In that case Alice (resp. Bob) grabs almost all the olives if the number of vertices of the grid $n = p \times q$ is odd (resp. even). We prove that for a $2 \times q$ grid with a fixed number k of olives Bob grabs at least $\lceil \frac{k-1}{3} \rceil$ olives and almost always grabs all the k olives.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Grabbing game, degenerate graph, path, grid

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.12

Related Version *Full Version:* <https://hal.inria.fr/view/index/docid/3623938> [1]

1 Introduction

One of our motivations for this article came from the problem entitled SHARING A PIZZA stated by P. Winkler in the new edition of his famous book *Mathematical Puzzles* [14]. We copy here the statement.

Alice and Bob are preparing to share a circular pizza, divided by radial cuts into some arbitrary number of slices of various sizes. They will be using the “polite pizza protocol”: Alice picks any slice to start; thereafter, starting with Bob, they alternate taking slices but always from one side or the other of the gap. Thus after the first slice, there are just two choices at each turn until the last slice is taken (by Bob if the number of slices is even, otherwise by Alice). Is it possible for the pizza to have been cut in such a way that Bob has the advantage? in other words, so that with best play, Bob gets more than half the pizza?

In the solution page 286, P. Winkler wrote: *If the number of slices is even (as with most pizzas), one can apply the solution of the puzzle Coins in a Row (see problem in Chapter 7 [14] but also the first problem of the book in 2004 [12]). Alice can always get at least half the pizza. Indeed Alice can just number the slices 1, 2, etc. starting clockwise (say) from the*

¹ J.-C. Bermond, F. Havet, and M. Cosnard





■ **Figure 1** Two examples of pizzas where Bob can always get 5 of the 9 olives.

cut, and play so as to take all the even-numbered slices or all the odd, whichever is better for her. The argument fails if the number of slices is odd. But the odd case sounds even better for Alice since then she ends up with more slices. How can we get a handle on the odd case?

P. Winkler indicates that this puzzle was already devised by D. E. Brown in 1996, and has attracted a lot of attention, in part because of his conjecture [13] that Alice could always get at least $4/9$ of the pizza. Indeed he gave the surprising example of a 15-slice circular pizza with slice-sizes 0, 1, and 2 of which nothing can stop Bob from acquiring $5/9$. The conjecture has been proved independently by two groups [2, 6].

At this stage, P. Winkler rose an interesting question *Can a slice be of size zero?* and answered: *Mathematically, no problem; gastronomically, think of a slice of size ϵ .* In fact, in the figure of his book, he put pepperoni to indicate the slice sizes and he wrote *No matter how she plays, Alice can never get more than 4 of the 9 pepperoni chunks against a smart, hungry Bob.* We give here again the example with olives (instead of pepperoni). In both pizzas of Figure 1, there are 9 olives (placed in the same positions). The pizza on the left has 15 slices, three of them having two olives. The pizza on the right has 21 slices with at most one olive on each of them. It is obtained from the left pizza by splitting the slices with two olives into three parts, two with one olive and the middle one with none. One can check that on both pizzas Bob can always grab 5 olives.

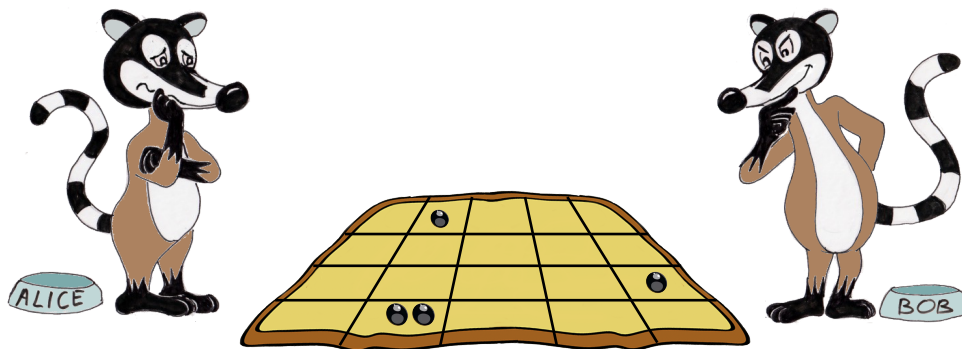
Another interesting question is : “ What is the right shape for a pizza ? ” Many people might answer round (circular). But according to Wikipedia² the traditional Sicilian pizza is typically rectangular, unlike the Neapolitan pizza which is circular. The roman “Pizza al taglio” (pizza by the slice) is baked in large rectangular trays, and generally divided in rectangular slices. Note also that the largest pizza³ ever baked is a linear pizza of length 1,853.88 meters.

Of higher importance to the authors is “pissaladière”. This marvelous dish is originally a specialty of the French city of Nice (under the name “pissaladeria”) and of Liguria (under the name “piscialandrea”). The classical pissaladière is presented as a rectangular grid where the slices are small rectangles (see Figure 2). Its traditional topping consists of caramelised (almost pureed) onions, anchovies and delicious small black olives of Nice, of which coatis are fond.

The original aim of our research was to tackle a new puzzle similar to Winkler’s one that we call SHARING A PISSALADIÈRE. The game is now played on a pissaladière, which is a rectangle, divided into rectangular slices by horizontal and vertical lines (knife cuts).

² https://en.wikipedia.org/wiki/Sicilian_pizza

³ <https://www.this-is-italy.com/world-biggest-pizza/>



■ **Figure 2** Two coatis trying to (politely) grab as many olives as possible on a pissaladière.

Olives are a precious ingredient and so there are only a small number k of olives; therefore many slices have no olives and some slices have one or more olives. The game is played by two coatis named Alice and Bob (see Figure 2). They both want to eat as many olives as possible, but they are civilized. The politeness imposes them to alternately pick a slice which is adjacent to at most two other slices. Hence, in the first move Alice can only pick one of the four corners of the pizza, and in the second move Bob can only pick one of the three remaining corners or one of the two slices adjacent to the corner removed by Alice. On Figure 2 you can see that the malicious Bob is content (and Alice disappointed). Indeed, with the disposal of the olives, Bob will grab all the 4 olives.

One particular interesting case is the $1 \times n$ grid or path of order n . In this case, the coati politeness imposes a stronger rule than the above, because otherwise any slice can be picked and the coati would rush to slices with olives. With the stronger rule, a coati can only pick one of the two slices at the ends of the (remaining) path. We will refer to this puzzle as SHARING A LINEAR PIZZA. Note that this problem appears in SHARING A PIZZA after the first move. Indeed, when Alice has picked the first slice, it remains a linear pizza. Therefore to solve an instance of SHARING A PIZZA, it suffices to solve n instances of SHARING A LINEAR PIZZA. This problem is equivalent to the puzzle COINS IN A ROW described in Winkler's books [12, 14] as follows: *On a table is a row of 50 coins, of various denominations. Alix picks a coin from one of the ends and puts it in her pocket; then Bert chooses a coin from one of the (remaining) ends, and the alternation continues until Bert pockets the last coin. Prove that Alix can play so as to guarantee at least as much money as Bert.* The main difference with SHARING A LINEAR PIZZA is that Alice is now Alix and Bob is now Bert. We note that P. Winkler mentioned the following in his book : *In fact, for Alix to play optimally, she needs to analyze all the possible situations that may later arise. This can be done by a technique called "dynamic programming".* More precisely, for an instance on the path of order n , $O(n^2)$ situations have to be considered, corresponding to the $O(n^2)$ subpaths, so the optimal strategies may be found in $O(n^2)$ time.

Various questions can be asked for these puzzles.

- (Q1) What is the minimum number of olives each coati can grab in function of the size of the pizza or pissaladière? This is the question asked by Winkler for the circular pizza.
- (Q2) For a given setting of k olives, can we give optimal strategies for the two coatis? In particular can we compute the number of olives each coati will grab (when they both play optimally) in a time polynomial in k (the number of olives) and independent of the size of the pizza?

- (Q3) What is the average number of olives grabbed by each coati if the k olives are uniformly distributed ?
- (Q4) What is the probability that a given coati (Alice or Bob) wins, that is grab at least half the olives ?

2 Statement of the problem and results

2.1 Graph-grabbing games

The three puzzles described above are particular cases of scoring games played on graphs (for scoring games, see for example the survey [7]). In particular, SHARING A PIZZA and SHARING A LINEAR PIZZA belongs to the class of *graph-grabbing* or *graph-sharing* games [3, 8, 9, 10]. In such games, two players Alice and Bob share a **connected graph with non-negative weights assigned to the vertices**. In this mathematical setting, the weights are real numbers, but in practical life the weights represent the number of olives on a slice and so are integers. The information is complete in the sense that both players know exactly the weights; in particular they know which slices have a zero weight (no olive). They alternately take the vertices one by one and collect their weights. The first turn is Alice's. There are some differences according to the rule taken to grab a vertex. In [3, 9] the authors consider two rules: the first one consists in removing a non cut vertex (i.e. the subgraph induced by the remaining vertices is connected during the whole game). The authors called this rule (R) (for "Remaining part connected"). The second rule, called (T) (for "Taken part connected"), imposes that the subgraph induced by the taken vertices is connected during the whole game. Hence SHARING A PIZZA is a graph-grabbing game on a cycle with any of the two rules (or both) because for a cycle (R) and (T) are identical; SHARING A LINEAR PIZZA is a graph-grabbing game on a path with rule (R). More generally, playing a graph-grabbing game on a tree with rule (R) imposes each player to take a leaf. For such games, Seacrest and Seacrest [11] showed that if a tree has an even number of vertices, Alice has a winning strategy (i.e. a strategy to obtain at least half of the total weight), thus answering a conjecture of Micek and Walczak [8]. They also conjectured that the same statement holds for every weighted connected bipartite graph with even order. Egawa et al. [4] proved that the conjecture holds for some particular bipartite graphs called $K_{m,n}$ -trees. As noted in [8], any odd tree on at least three vertices can be weighted so that Alice gets nothing (it suffices to put the whole weight at one non-leaf vertex). Eoh and Choi [5] gave general classes of graphs for which Alice has a winning strategy with rule (R) when the weights are 0 or 1. Cibulka et al. [3] showed, when playing with any of rules (T) and (R) or both, then for every $\epsilon > 0$ and for every $k \geq 1$, there is a k -connected graph G for which Bob has a strategy to obtain $(1 - \epsilon)$ of the total weight of the vertices. This contrasts with the original pizza game played on a cycle. They also show that the problem of deciding whether Alice has a winning strategy is PSPACE-complete if condition (R) or both conditions (T) and (R) are required. Micek and Walczak [9] showed the importance of parity in graph-grabbing games. For example, they proved that, with rule (T) on a tree with an odd number of vertices, Alice has a strategy to get at least $1/4$ of the weight.

2.2 d -degenerate games

Note that playing with rules (R) or (T) on a pissaladière (i.e. a (rectangular) grid) with at least two rows and two columns is everything but polite. Most of the time, the set of vertices that a player can remove is not very restricted; on her first move, Alice can always

remove the vertex she wants. Although this has been successfully practiced by barbarian soldiers like Attila or Alexandre the Great, civilized coatis need more restricted rules to share a pissaladière. Therefore we introduce the d -degenerate rule (D_d) yielding the d -**degenerate game**.

(D_d) : At each move, a player can only pick a d -**removable** vertex which is a vertex of degree at most d in the remaining graph.

Of course, with such a rule, all the vertices can be removed only if at each step there is a removable vertex. This is the case if and only if the graph is d -degenerate. Recall that a graph is d -**degenerate** if each of its subgraphs has a vertex of degree at most d . A graph G is d -degenerate if and only if it has a d -**degenerate ordering** that is an ordering (v_1, \dots, v_n) of its vertices such that v_i has degree at most d in $G\langle\{v_i, \dots, v_n\}\rangle$ for all $i \in [n]$.

Note that we could play the d -degenerate game on a graph which is not d -degenerate. Then the two coatis share a fraction of the graph until there is no d -removable vertex. But we shall not consider it in this paper, and shall only play the d -degenerate game on d -degenerate graphs.

The 1-degenerate connected graphs are the trees. In a tree, a 1-removable vertex is a leaf. Therefore, on trees, the 1-degenerate game is equivalent to the graph-grabbing game with rule (R). In particular, the 1-degenerate game on path is exactly SHARING A LINEAR PIZZA. Grids are a nice examples of 2-degenerate graphs and the 2-degenerate game on grids is exactly SHARING A PISSALADIÈRE.

2.3 Our results

When we started our study, our aim was to obtain results on SHARING A PISSALADIÈRE, which corresponds to the 2-degenerate game on a grid. We were interested in the case of integer-valued weight function, which corresponds to the case where an olive cannot be cut and is on a unique slice, and especially $\{0, 1\}$ -weight function, which corresponds to the case where there is at most one olive per slice. But we realized that questions (Q2) to (Q4) were not answered in the literature for SHARING A LINEAR PIZZA. Consequently, we first study this problem.

Consider the path with n vertices $P_n = (1, 2, \dots, n)$ together with a weight function $w : V(P_n) \rightarrow \mathbb{R}^+$. A **weighty** vertex is a vertex whose weight is nonzero. In that case question (Q1) is easily answered when n is even as noted by Winkler. Alice can use the even-odd strategy in which she removes either all the even vertices or all the odd ones, thus grabbing at least half of the weight (olives). However, the even-odd strategy is not necessarily optimal. For example, consider a path with $n = 12$ vertices and weights $w(2) = 1, w(5) = 2, w(7) = 3, w(10) = 4$ and all the other weights equal to zero. Playing the even-odd strategy Alice collects 5 olives, while we can show that she can get 6 olives (see Example in subsection3.1).

Therefore our aim in Section 3 is to describe optimal strategies on paths. We first consider **0-ordinary** weight function in which the end-vertices have weight zero and there are no two adjacent weighty vertices. In Subsection 3.1, we describe some strategies called Σ_0 and show that they are optimal (Theorem 4). It allows us to give a recursive formula to compute the gain of both players and to show that the weights collected by each player depend only on the parity of n and on the respective parities of the weighty vertices and not on the precise value of n nor of the values of the weights of the vertices. So, for 0-ordinary weight functions, one can compute the optimal gain in time $\mathcal{O}(k^2)$ where k is the number of weighty vertices. We then show that in fact, the complexity is in $\mathcal{O}(h^2)$ where h denotes the number of parity changes in the weighty vertices (Proposition 6). In Subsection 3.2, we extend strategy Σ_0 to

weight functions where the end-vertices may be weighty but two weighty vertices are still not adjacent. Then in Subsection 3.3, we revisit SHARING A PIZZA in the case where there is no adjacent weighty vertices: we show that Alice has an optimal strategy whose first move consist in grabbing a weighty vertex, and derive that the optimal gain can be computed in $\mathcal{O}(kh^2)$.

If we have consecutive weighty vertices, the situation is more complex as there are examples where a player has no interest in taking a weighty end-vertex. Consider the path on four vertices with $0 < w(1) < w(2)$ and $w(3) = w(4) = 0$. If at the first move Alice removes the weighty end-vertex 1, then she will gain only $w(1)$. But if she removes vertex 4, then whatever Bob plays she will grab vertex 2 and gains the weight $w(2)$.

In section 4, we show that for a d -degenerate game with $\{0, 1\}$ weight function, there is an optimal play where a coati grabs immediately an olive if possible (Lemma 8). This lemma applied for linear pizza enables us to delete two consecutive vertices of weight 1 giving one olive to each player (Proposition 9). Hence, for linear pizzas with a $\{0, 1\}$ weight function we completely answer question (Q2). Furthermore, in that case we show how to relate the computation of the gain of each player to counting on binary words. That allows us to give a way to compute asymptotically the average number of olives grabbed by each player and to answer questions (Q3) and (Q4) when the number k of olives is small. See Table 1.

In Section 5 we deal with our original problem SHARING A PISSALADIÈRE. We first consider the simple case where there is a single olive on the pissaladière: we prove that Alice collects the olive if and only if the pissaladière has an odd number of slices or if the olive is on a corner of the pissaladière. This allows us to completely answer questions (Q1) to (Q4) in this case (Corollary 13.) Then, we consider the game with k olives on a pissaladière with two rows of slices. We show in Corollary 15 that Alice might collect no olive while, in contrast, Bob can always collect $\lceil \frac{k-1}{3} \rceil$ olives (best possible). We also show in Proposition 16, that almost always Bob collects all the olives (and Alice none). Finally, we consider the case where there are two olives on different slices of the pissaladière. This case is more complex, and we do not answer (Q2). However we prove that if the number of slices is odd (resp. even), then Alice (resp. Bob) almost always grab the two olives (Theorem 18).

3 Sharing a linear pizza

In this section, we are given the path $P_n = (1, 2, \dots, n)$ of order n with a non-negative weight function w on the vertices. The weight of vertex i is denoted by $w(i)$. If the $w(i)$ are integers, we can consider vertices as slices of a linear pizza, the weight representing the number of olives on each slice.

We denote by $O = O(w)$ the set of k vertices $\{i_j \mid 1 \leq j \leq k\}$ which have a nonzero weight $w(i_j)$ called **weighty vertices** (slices with olives in the integral model). The other vertices have weight zero (no olive). The set O is **ordinary** if it contains no two adjacent vertices, that is if $|O \cap \{i, i+1\}| \leq 1$ for all $i \in [1, n-1]$. A set which is not ordinary is called **rare**. It is **s -ordinary** ($s = 0, 1, 2$) if it is ordinary and $|O \cap \{1, n\}| = s$. So s is the number of weighty end-vertices. The weight function w is **ordinary** (resp. **rare**, **s -ordinary**) if the set O is ordinary (resp. rare, s -ordinary).

Ordinary weight functions are useful because we can design for them optimal strategies to determine the final weight collected by each player and compute recursively this number in polynomial time (quadratic in k and not depending of n). Indeed after removing a weighty vertex, the remaining set of weighty vertices is still ordinary. Ordinary sets are important especially in the case of $\{0, 1\}$ -weight functions (or more generally integer-valued weight functions) because the proportion of ordinary sets over all possible sets O of k olives tends to 1 when n tends to $+\infty$. **In this section, we consider only ordinary weight functions.**

3.1 Optimal strategy for 0-ordinary weight functions

Strategies Σ_0 . At a given stage an end-vertex of the remaining path can be of three types: (1) “weighty” ; (2) “critical” if it has weight zero and is adjacent to a weighty vertex; (3) “safe” if it has weight zero and is not adjacent to a weighty vertex. In strategy Σ_0 , at any move if a player X can remove a weighty vertex, then X will do so and will gain the weight of this vertex. Otherwise, if the end-vertices have weight zero, two cases can appear. Either one end-vertex is safe and the player will remove it. Otherwise the two end-vertices are critical; the player is stacked and obliged to let in the next move the other player grab a weighty vertex; but the player can choose to remove the critical vertex which ensures the maximum gain at the end of the game. Then we apply recursively the strategy on the path P_n with a new weight function where the weight of the removed vertex is now 0 and so with a smaller set of weighty vertices.

At the beginning of the game, if the weight function is 0-ordinary, the end-vertices have weight 0 (that is $i_1 > 1$ and $i_k < n$). The set of safe vertices is $S = \{1 \leq i \leq i_1 - 2\} \cup \{i_k + 2 \leq i \leq n\}$. So, during the first $|S|$ moves, the strategy Σ_0 consists for each player to remove a safe vertex (the order in which the safe vertices are chosen has no importance). Then, at move $|S| + 1$, one player, denoted by \bar{X} , is obliged to take one of the two critical vertices ($i_1 - 1$ or $i_k + 1$). Then at move $|S| + 2$ the other player, denoted by X , will take the neighbor x of this vertex which is either i_1 or i_k . X is either Alice or Bob according the parity of $|S|$ which itself depends only on the parity of n and of the respective parities of i_1 and i_k . More precisely

- $X = A$ when $|S|$ is odd, that is when either $\{n$ is even and i_1 and i_k have the same parity}, or $\{n$ is odd and i_1 and i_k have different parity };
- $X = B$ when $|S|$ is even, that is when either $\{n$ is odd and i_1 and i_k have the same parity}, or $\{n$ is even and i_1 and i_k have different parity }.

Then we apply strategy Σ_0 on P_n with the weight of x reduced to zero. More precisely, let us denote for every $1 \leq \alpha \leq \beta \leq n$, by $w[\alpha, \beta]$ the weight function defined by $w[\alpha, \beta](i) = w(i)$ if $\alpha \leq i \leq \beta$, and $w[\alpha, \beta](i) = 0$ otherwise. We apply strategy Σ_0 on P_n with the new weight function $w[i_2, i_k]$ if $x = i_1$ and $w[i_1, i_{k-1}]$ if $x = i_k$. Note that the first $|S| + 2$ moves already done corresponds to successive removal of safe vertices in the strategy Σ_0 on P_n with this new weight function.

Example. Let us apply strategy Σ_0 on the example of the introduction: a path with $n = 12$ vertices and 4 weighty vertices $w(2) = 1, w(5) = 2, w(7) = 3, w(10) = 4$. At the first move vertex 12 is safe (and 1 is critical) and so Alice will remove 12. Then at the second move there are two critical vertices 1 and 11. Bob can choose which vertex he will remove and one can verify that the best choice is to remove 1. Then Alice grabs the weighty vertex 2. Then Bob removes the safe vertex 3. Now at move 5 Alice is stacked as there are two critical vertices 4 and 11. But she has the choice. If she does the greedy choice of removing the vertex 4 with the largest weight one can verify that she will grab at the end only 5 olives. But, if at move 5 Alice chooses the apparently bad vertex 11, Bob will be happy to grab at move 6 the 4 olives of vertex 10. At move 7 Alice removes the safe vertex 9. Then, whatever Bob does, Alice will grab at moves 9 and 11 the weighty vertices 5, 7 . In summary, Alice will get the olives of vertices 2, 5, 7 that is 6 olives as announced in the introduction.

The strategy Σ_0 is partly greedy in the sense that the player X who takes the first vertex x in O is uniquely determined. However, the other player \bar{X} can choose the vertex which maximizes his/her final gain or equivalently minimizes the gain of X . But as seen in the example this choice is not greedy.

The total weight collected by a player X with strategy Σ_0 on P_n with weight function w is denoted by $\sigma_0^X(n; w)$. The following lemma follows immediately from the definition. It gives a recursive formula to compute the gain of the players and shows that the weight collected by each player depends only on the parity of n and on the respective parities of the i_j and not on the precise value of n nor of the values of the i_j .

► **Lemma 1.** *Let P_n be a path with n vertices and w a 0-ordinary weight function with set $O = \{i_1, i_2, \dots, i_k\}$ of weighty vertices. Assume that the two players play according to strategy Σ_0 . Let $X = A$ if either $\{n$ is even and i_1 and i_k have the same parity}, or $\{n$ is odd and i_1 and i_k have different parity}; otherwise $X = B$. Then*

$$\begin{aligned}\sigma_0^X(n; w) &= \min \{w(i_1) + \sigma_0^X(n; w[i_2, i_k]), w(i_k) + \sigma_0^X(n; w[i_1, i_{k-1}])\}, \\ \sigma_0^{\bar{X}}(n; w) &= W - \sigma_0^X(n; w) = \max \{\sigma_0^{\bar{X}}(n; w[i_2, i_k]), \sigma_0^{\bar{X}}(n; w[i_1, i_{k-1}])\}.\end{aligned}$$

Our aim is now to prove that the strategies Σ_0 are optimal. To do so, we first prove that the strategies Σ_0 have the following property which is easy to prove for optimal strategies.

► **Lemma 2.** *Let P_n be a path with n vertices and w a 0-ordinary weight function. Then, for any player X : $\sigma_0^X(n; w) \geq \max \{\sigma_0^X(n; w[i_2, i_k]), \sigma_0^X(n; w[i_1, i_{k-1}])\}$.*

Proof. We proceed by induction on k , the result being trivial when $k = 1$. Suppose that the lemma is true until $k - 1$. Consider a game on P_n with the set of weighty vertices O of size k . By Lemma 1, two cases can appear for the player X .

- (i) Either $\sigma_0^X(n; w) = \max \{\sigma_0^X(n; w[i_2, i_k]), \sigma_0^X(n; w[i_1, i_{k-1}])\}$, or
- (ii) $\sigma_0^X(n; w) = \min \{w(i_1) + \sigma_0^X(n; w[i_2, i_k]), w(i_k) + \sigma_0^X(n; w[i_1, i_{k-1}])\}$.

In case (i), there is nothing to prove. Assume now that we are in case (ii). W.l.o.g. we may also assume that $w(i_1) + \sigma_0^X(n; w[i_2, i_k]) \leq w(i_k) + \sigma_0^X(n; w[i_1, i_{k-1}])$. Then $\sigma_0^X(n; w) = w(i_1) + \sigma_0^X(n; w[i_2, i_k])$. So it remains to prove the following claim.

▷ **Claim 3.** $\sigma_0^X(n; w[i_1, i_{k-1}]) \leq w(i_1) + \sigma_0^X(n; w[i_2, i_k])$.

Proof. By Lemma 1, $\sigma_0^X(n; w[i_1, i_{k-1}])$ is either $\max\{\sigma_0^X(n; w[i_2, i_{k-1}]), \sigma_0^X(n; w[i_1, i_{k-2}])\}$, or $\min\{w(i_1) + \sigma_0^X(n; w[i_2, i_{k-1}]), w_{k-1} + \sigma_0^X(n; w[i_1, i_{k-2}])\}$. Assume $\sigma_0^X(n; w[i_1, i_{k-1}]) = \min\{w(i_1) + \sigma_0^X(n; w[i_2, i_{k-1}]), w_{k-1} + \sigma_0^X(n; w[i_1, i_{k-2}])\}$. By the induction hypothesis, $\sigma_0^X(n; w[i_2, i_{k-1}]) \leq \sigma_0^X(n; w[i_2, i_k])$. So, $\sigma_0^X(n; w[i_1, i_{k-1}]) \leq w(i_1) + \sigma_0^X(n; w[i_2, i_{k-1}]) \leq w(i_1) + \sigma_0^X(n; w[i_2, i_k])$ proving the claim.

Assume $\sigma_0^X(n; w[i_1, i_{k-1}]) = \sigma_0^X(n; w[i_2, i_{k-1}])$. By the induction hypothesis, we have $\sigma_0^X(n; w[i_2, i_{k-1}]) \leq \sigma_0^X(n; w[i_2, i_k])$, so $\sigma_0^X(n; w[i_1, i_{k-1}]) \leq \sigma_0^X(n; w[i_2, i_k]) < w(i_1) + \sigma_0^X(n; w[i_2, i_k])$ proving the claim.

Henceforth, it remains to deal with the case where $\sigma_0^X(n; w[i_1, i_{k-1}]) = \sigma_0^X(n; w[i_1, i_{k-2}])$.

By Lemma 1, $\sigma_0^X(n; w[i_1, i_{k-2}])$ is either $\min\{w(i_1) + \sigma_0^X(n; w[i_2, i_{k-2}]), w(i_{k-2}) + \sigma_0^X(n; w[i_1, i_{k-3}])\}$, or $\max\{\sigma_0^X(n; w[i_2, i_{k-2}]), \sigma_0^X(n; w[i_1, i_{k-3}])\}$. As above, if it is either $\min\{w(i_1) + \sigma_0^X(n; w[i_2, i_{k-2}]), w(i_{k-2}) + \sigma_0^X(n; w[i_1, i_{k-3}])\}$ or $\sigma_0^X(n; w[i_2, i_{k-2}])$, we get the result using the induction hypothesis. Henceforth, we have to deal with the case where $\sigma_0^X(n; w[i_1, i_{k-1}]) = \sigma_0^X(n; w[i_1, i_{k-2}]) = \sigma_0^X(n; w[i_1, i_{k-3}])$.

And so on, by induction on $h = 1, \dots, k - 2$, either we get the result for some h , or we have $\sigma_0^X(n; w[i_1, i_{k-1}]) = \sigma_0^X(n; w[i_1, i_{k-1-h}])$ for every h . In particular, for $h = k - 1$, we get $\sigma_0^X(n; w[i_1, i_{k-1}]) = \sigma_0^X(n; w[i_1, i_1]) \leq w(i_1)$. So the claim holds. ◁

The claim finishes the proof of the lemma. ◀

► **Theorem 4.** *Strategies Σ_0 are optimal on P_n with 0-ordinary weight functions.*

Proof. The proof is by induction on the number k of weighty vertices. Suppose that the strategies Σ_0 are optimal for 0-ordinary weight functions having up to $k-1$ weighty vertices. Let us consider the game on P_n with a 0-ordinary weight function w with a set O of k weighty vertices.

By Lemma 1, with strategy Σ_0 , the gain of player X is $\sigma_0^X(n; w) = \min\{w(i_1) + \sigma_0^X(n; w[i_2, i_k]), w(i_k) + \sigma_0^X(n; w[i_1, i_{k-1}])\}$, and the gain of player \bar{X} is $\sigma_0^{\bar{X}}(n; w) = \max\{\sigma_0^{\bar{X}}(n; w[i_2, i_k]), \sigma_0^{\bar{X}}(n; w[i_1, i_{k-1}])\}$.

Consider an optimal strategy played by the two players. Let c be the first critical vertex removed by a player, and let x be its weighty neighbour. Either $c = i_1 - 1$ and $x = i_1$, or $c = i_k + 1$ and $x = i_k$.

Suppose for a contradiction that c is removed by X . In the following move, \bar{X} can remove x . Let $w' = w[i_2, i_k]$ if $x = i_1$ and $w' = w[i_1, i_{k-1}]$ if $x = i_k$. The vertices removed so far (including c and x) are safe for w' . Therefore it can be seen as the beginning of a strategy Σ_0 for w' which is optimal for w' by the induction hypothesis. Thus the gain of X using this strategy for w is at most its optimal gain for w' that is $\sigma_0^X(n; w[i_1, i_{k-1}])$ if $x = i_k$ and $\sigma_0^X(n; w[i_2, i_k])$ if $x = i_1$. In both cases, by Lemma 2, this gain is smaller than $\sigma_0^X(n; w)$, a contradiction.

Therefore c is removed by \bar{X} . In the following move X can remove x . Let $w' = w[i_2, i_k]$ if $x = i_1$ and $w' = w[i_1, i_{k-1}]$ if $x = i_k$. The vertices removed so far (including c and x) are safe for w' . Therefore it can be seen as the beginning of a strategy Σ_0 for w' which is optimal for w' by the induction hypothesis. Thus the gain of \bar{X} using this strategy for w is at most its optimal gain for w' which is $\sigma_0^{\bar{X}}(n; w[i_1, i_{k-1}])$ if $x = i_k$ and $\sigma_0^{\bar{X}}(n; w[i_2, i_k])$ if $x = i_1$. In both cases, it is no greater than $\sigma_0^{\bar{X}}(n; w)$, and it is strictly smaller if removing c does not maximize \bar{X} 's gain. So \bar{X} has no interest of playing differently from strategy Σ_0 and can remove safe vertices until it is forced to pick a critical vertex. Moreover, when he does so, he must remove a critical vertex that maximizes its gain (as in strategy Σ_0). On his side, X must also remove safe vertices as long as \bar{X} does because it cannot remove a critical vertex by the above paragraph. Therefore the strategy Σ_0 is optimal. ◀

Complexity. Since strategy Σ_0 is optimal, computing the gain of each player when they play optimally is computing $\sigma_0^A(n; w)$ and $\sigma_0^B(n; w)$. Using Lemma 1, one can do it in $\mathcal{O}(k^2)$ time. But we can do it faster. In fact, analyzing strategy σ_0 , we can prove that the players collect weighty vertices by blocks and not one by one. A **(parity) block** of olives in a 0-ordinary set of weighty vertices $O = \{i_1, \dots, i_k\}$ is a maximum set of successive weighty vertices having the same parity. The following lemma shows that that X collects either the rightmost block of olives or the leftmost one.

► **Lemma 5.** *Let w be a 0-ordinary weight function on P_n with set of weighty vertices $O = \{i_1, \dots, i_k\}$. Let $\{i_1, \dots, i_\alpha\}$ be its leftmost parity block of O and let $\{i_\beta, \dots, i_k\}$ be its rightmost parity block.*

Let $X = A$ if either $\{n$ is even and i_1 and i_k have the same parity}, or $\{n$ is odd and i_1 and i_k have different parity}; otherwise $X = B$. Then

$$\begin{aligned} \sigma_0^{\bar{X}}(n; w) &= \max \left\{ \sigma_0^{\bar{X}}(n; w[i_{\alpha+1}, i_k]), \sigma_0^{\bar{X}}(n; w[i_1, i_{\beta-1}]) \right\}, \quad \text{and so} \\ \sigma_0^X(n; w) &= \min \left\{ \sum_{j=1}^{\alpha} w(i_j) + \sigma_0^X(n; w[i_{\alpha+1}, i_k]), \sum_{j=\beta}^k w(i_j) + \sigma_0^X(n; w[i_1, i_{\beta-1}]) \right\}. \end{aligned}$$

Proof. By Lemma 1, we have $\sigma_0^{\overline{X}}(n; w) = \max \left\{ \sigma_0^{\overline{X}}(n; w[i_2, i_k]), \sigma_0^{\overline{X}}(n; w[i_1, i_{k-1}]) \right\}$.
 $\sigma_0^{\overline{X}}(n; w[i_2, i_k]) \geq \sigma_0^{\overline{X}}(n; w[i_{\alpha+1}, i_k])$ and $\sigma_0^{\overline{X}}(n; w[i_1, i_{k-1}]) \geq \sigma_0^{\overline{X}}(n; w[i_1, i_{\beta-1}])$ by induction.
Thus, $\sigma_0^{\overline{X}}(n; w) \geq \max \left\{ \sigma_0^{\overline{X}}(n; w[i_{\alpha+1}, i_k]), \sigma_0^{\overline{X}}(n; w[i_1, i_{\beta-1}]) \right\}$.

Let us now prove the opposite inequality by induction on k , the result holding when $k = 2$.
By the induction hypothesis $\sigma_0^{\overline{X}}(n; w[i_2, i_k]) = \max \left\{ \sigma_0^{\overline{X}}(n; w[i_{\alpha+1}, i_k]), \sigma_0^{\overline{X}}(n; w[i_2, i_{\beta-1}]) \right\}$
and by Lemma 2, $\sigma_0^{\overline{X}}(n; w[i_2, i_{\beta-1}]) \leq \sigma_0^{\overline{X}}(n; w[i_1, i_{\beta-1}])$.
Similarly by induction: $\sigma_0^{\overline{X}}(n; w[i_1, i_{k-1}]) = \max \left\{ \sigma_0^{\overline{X}}(n; w[i_{\alpha+1}, i_{k-1}]), \sigma_0^{\overline{X}}(n; w[i_1, i_{\beta-1}]) \right\}$
and by Lemma 2, $\sigma_0^{\overline{X}}(n; w[i_{\alpha+1}, i_{k-1}]) \leq \sigma_0^{\overline{X}}(n; w[i_{\alpha+1}, i_k])$. Therefore in both cases,
 $\sigma_0^{\overline{X}}(n; w[i_1, i_{k-1}]) \leq \max \left\{ \sigma_0^{\overline{X}}(n; w[i_{\alpha+1}, i_k]), \sigma_0^{\overline{X}}(n; w[i_1, i_{\beta-1}]) \right\}$. ◀

Using Lemma 5, we get the following proposition.

► **Proposition 6.** *Let P_n be a path with n vertices and w a 0-ordinary weight function. Then $\sigma_0^{\overline{X}}(n; w)$ and $\sigma_0^{\overline{X}}(n; w)$ can be computed in $O(h^2)$ arithmetic operations where h is the number of parity blocks of the set of weighty vertices (or equivalently the number of parity changes of the weighty vertices plus 1).*

3.2 Optimal strategies for 1- and 2-ordinary weight functions

Based on strategies Σ_0 for 0-ordinary weight functions, we now define the strategies Σ_1 and Σ_2 for 1-ordinary and 2-ordinary, respectively, weight functions.

Strategies Σ_1 . If the players have to play on the path P_n with a 1-ordinary weight function w , then the players do the following: At the first move A removes the weighty end-vertex; then the players play according to strategy Σ_0 on P_n with 0-ordinary weight function $w' = w[1, n-1] = w[i_1, i_{k-1}]$ if the removed vertex was n , or $w' = w[2, n] = w[i_2, i_k]$ if the removed vertex was 1.

Strategies Σ_2 . If the players have to play on the path P_n with a 2-ordinary weight function w , then the players do the following: At the first move, A removes the end-vertex with the largest weight and at the second move B removes the other end-vertex; Then the players play according to strategy Σ_0 on P_n with 0-ordinary weight function $w' = w[2, n-1] = w[i_2, i_{k-1}]$.

It can be proved that Strategies Σ_1 (resp. Σ_2) are optimal on P_n with 1-ordinary (resp. 2-ordinary) weight functions. See [1].

3.3 Back on Circular pizza

Here, we are given the cycle $C_n = (1, 2, \dots, n, 1)$ of order n with a non-negative weight function w on the vertices. As for paths, the weight function is **ordinary** if there are no adjacent weighty vertices. We show the following proposition (see[1] for its proof).

► **Proposition 7.** *Let C_n be a cycle with n vertices and w a 0-ordinary weight function with k weighty vertices.*

Alice has an optimal strategy whose first move consists in removing a weighty vertex. So the weight collected by Alice and Bob if they both play optimally can be computed in $O(kh^2)$ arithmetic operations where h is the number of parity blocks.

4 d -degenerate games with a $\{0, 1\}$ -weight function

A $\{0, 1\}$ -weight function w is completely determined by the set $O = \{v \mid w(v) = 1\}$. This set is then called the **olive set** and its elements the **olives**. Let G be a graph G on n vertices and an olive set $O \subseteq V(G)$. Coatis Alice and Bob are denoted by their initials A and B respectively. An **optimal play** is a sequence of moves made alternately by Alice and Bob when they play optimally. For $X \in \{A, B\}$, we denote by $\text{col}_d^X(G, O)$ the number of olives that X collects in an optimal play. Note that $\text{col}_d^A(G, O) + \text{col}_d^B(G, O) = |O|$. For any integer $k \in [n]$, we say that a coati **secures** s olives among k on G if he/she collects at least s olives for any set of k olives.

Question (Q1) consists in determining $\text{sec}_d^A(G, k)$ (resp. $\text{sec}_d^B(G, k)$) the maximum number of olives Alice (resp. Bob) secures.

Question (Q3) asks for the average number of olives $\overline{\text{col}}_d^X(G, k)$ coati X collects in the d -degenerate game over all possible k olive sets of G .

Finally, Question (Q4) asks for $\text{win}_d^A(G, k)$ (resp. $\text{win}_d^B(G, k)$) which is the probability that Alice (resp. Bob) wins, that is collects at least half the olives.

The following lemma shows that there is an optimal play where a coati grabs immediately an olive if possible. We denote by $R_d(G, O)$ the set of **removable olives** in the d -degenerate game on G , that are the vertices of O that have degree at most d in G .

► **Lemma 8.** *Let G be a d -degenerate graph and let O be a set of olives. If $R_d(G, O) = r > 0$, then for any order o_1, o_2, \dots, o_r of the r vertices of $R_d(G, O)$, there is an optimal play starting with o_1, o_2, \dots, o_r*

Proof. We prove the result by induction on the number of vertices of G , the result holding trivially if G has one vertex. We note that to prove the lemma it suffices to prove that there is an optimal play starting with o_1 . Indeed, by induction hypothesis there exist an optimal play in $(G, O) - \{o_1\}$ starting with o_2, \dots, o_r as these vertices are vertices of $O - o_1$ removable.

Let σ be an optimal play for (G, O) and suppose that σ starts with the vertex v_1 . If $v_1 = o_1$ there is nothing to prove. We distinguish two cases.

- $v_1 \in O$. So, let $v_1 = o_j$, with $2 \leq j \leq r$. By the induction hypothesis applied to $(G, O) - \{o_j\}$, there is an optimal play in $(G, O) - \{o_j\}$ starting with o_1 . So we have an optimal play in G, O $o_j, o_1, v_3, \dots, v_n$. But the play $\sigma' = (o_1, o_j, v_3, \dots, v_n)$ is also optimal as A and B get the same number of olives (the only difference is that now A grabs o_1 instead of o_j). Note that the proof will not be valid if the weighty vertices have different weights.
- $v_1 \notin O$. By the induction hypothesis, we have an optimal play in $(G, O) - \{v_1\}$ starting with o_1, o_2, \dots, o_r and so in (G, O) an optimal play $(v_1, o_1, \dots, o_r, v_{r+1}, v_{r+2}, \dots, v_n)$. The gain of A is therefore $\text{col}_d^A((G, O) - \{v_1\}) + G(A)$ where $G(A)$ is the gain of A in the optimal play $(v_{r+1}, v_{r+2}, \dots, v_n)$ in $(G, O) - \{v_1, o_1, \dots, o_r\}$ with A playing first (resp. second) if r is odd (resp. even).

Now consider the play where A plays first o_1 . By the induction hypothesis there is an optimal play in $(G, O) - \{o_1\}$ starting with o_2, \dots, o_r .

If $r \geq 2$, A can remove in move 3 the vertex v_1 and then A will gain $G(A)$ in the play in $(G, O) - \{o_1, v_1, o_2, o_3, \dots, o_r\}$ (with B starting). So we have a play $\sigma' = (o_1, o_2, v_1, o_3, \dots, o_r, v_{r+1}, v_{r+2}, \dots, v_n)$ where A gains $\lfloor r/2 \rfloor + G(A)$ which is the optimum and so σ' is an optimal play satisfying the lemma.

If $r = 1$ the gain of A is now 1 plus the gain $H(A)$ in an optimal play on $(G, O) - \{o_1\}$ with B playing first. To compute the gain $H(A)$ we will use the trick of computing the

12:12 Grabbing Olives on Linear Pizzas and Pissaladières

gain if we add an olive in v_1 . Let $H'(A)$ be the gain of A if we add an olive in v_1 that is in the game $(G, O \cup \{v_1\}) \setminus \{o_1\}$ with B playing first. As Alice can play on this game, the same strategy as if she was playing on $(G, O \setminus \{o_1\})$, we have $H(A) \geq H'(A) - 1$. By the induction hypothesis, in this game, there is an optimal play where B starts with v_1 and then consisting of the optimal play (v_2, \dots, v_n) and so the gain of A is exactly $H'(A) = G(A)$. Therefore $H(A) \geq G(A) - 1$. In summary there is a play σ' starting with o_1 , where A gains at least $G(A)$; that is exactly the value of an optimal play and so σ' is also an optimal play. \blacktriangleleft

4.1 Sharing a linear pizza with a $\{0, 1\}$ -weight function

The olive set $O = \{v \mid w(v) = 1\}$ is **rare** or α -**ordinary** for some $\alpha \in \{0, 1, 2\}$ if it corresponds to a rare or α -ordinary weight function. Lemma 8 implies the following proposition (see proof in [1]).

► **Proposition 9.** *Suppose that O contains two adjacent olives $\{i_j, i_j + 1\}$. Let $P'_{n-2} = P_n \setminus \{i_j, i_j + 1\}$ and $O' = O \setminus \{i_j, i_j + 1\}$. Then $\text{col}_1^A(P_n, O) = 1 + \text{col}_1^A(P'_{n-2}, O')$ and $\text{col}_1^B(P_n, O) = 1 + \text{col}_1^B(P'_{n-2}, O')$.*

This proposition enables us to deal with adjacent olives and so to answer Question (Q2) for linear pizza with a $\{0, 1\}$ -weight function. If there are h pairs of adjacent olives, we give h olives to each player and delete the corresponding vertices. We are left with a path of size $n - 2h$ and with an ordinary set of olives. Then we apply strategy Σ to compute the remaining gains of A and B .

Now we can give answers to questions (Q3) and (Q4). First we show that among the $\binom{n}{k}$ sets of k olives on P_n , the 0-ordinary sets prevail. Therefore, to obtain asymptotic formulas on $\overline{\text{col}}_1^A(P_n, k)$, (and so $\overline{\text{col}}_1^B(P_n, k) = k - \overline{\text{col}}_1^A(P_n, k)$), $\text{win}_1^A(P_n, k)$, $\text{win}_1^B(P_n, k)$, for k fixed, we can only consider 0-ordinary sets. Recall that when n is even $\text{win}_1^A(P_n, k) = 1$ by Winkler's result. In that case we associate to a 0-ordinary set $O = \{i_1, \dots, i_k\}$ of k olives a **parity word** $W_O = \gamma_1 \gamma_2 \cdots \gamma_k$ where $\gamma_j = 1$ if i_j is odd, and $\gamma_j = 0$ otherwise. We then show that the problem is reduced to computations of words. All the details are given in [1]. Table 1 summarizes the obtained values when $k \leq 6$ and suggests Conjecture 10.

■ **Table 1** Asymptotic value of $\overline{\text{col}}_1^A(P_n, k)$ and $\text{win}_1^A(P_n, k)$ for $1 \leq k \leq 6$.

	k	1	2	3	4	5	6
$\overline{\text{col}}_1^A(P_n, k)$	n even	1	$3/2 + \mathcal{O}(\frac{1}{n})$	$9/4 + \mathcal{O}(\frac{1}{n})$	$23/8 + \mathcal{O}(\frac{1}{n})$	$7/2 + \mathcal{O}(\frac{1}{n})$	$131/32 + \mathcal{O}(\frac{1}{n})$
	n odd	$\mathcal{O}(\frac{1}{n})$	$1/2 + \mathcal{O}(\frac{1}{n})$	$3/4 + \mathcal{O}(\frac{1}{n})$	$9/8 + \mathcal{O}(\frac{1}{n})$	$3/2 + \mathcal{O}(\frac{1}{n})$	$61/32 + \mathcal{O}(\frac{1}{n})$
$\text{win}_1^A(P_n, k)$	n even	1	1	1	1	1	1
	n odd	$2/n$	$1/2 + \mathcal{O}(\frac{1}{n})$	$\mathcal{O}(\frac{1}{n})$	$1/4 + \mathcal{O}(\frac{1}{n})$	$\mathcal{O}(\frac{1}{n})$	$3/16 + \mathcal{O}(\frac{1}{n})$

► **Conjecture 10.** *Let $C_k = \lim_{n \rightarrow \infty, n \text{ even}} \overline{\text{col}}_1^A(P_n, k)$ and $D_k = \lim_{n \rightarrow \infty, n \text{ odd}} \text{win}_1^A(P_n, k)$. Then*

$$\frac{C_k}{k} \rightarrow \frac{1}{2} \text{ when } k \rightarrow \infty \text{ and } D_{2p} \rightarrow 0 \text{ when } p \rightarrow \infty.$$

5 Sharing a pissaladière

In this section, we study SHARING A PISSALADIÈRE, the 2-degenerate game on rectangular grids $G_{p \times q}$ with p rows and q columns. We always consider $p \leq q$ and $p > 1$. Indeed if $p = 1$ the grid is reduced to a path considered in the preceding sections with the stronger

rule of picking only vertices of degree 1. Vertices are denoted (a, b) where a is the index of the row and b the index of the column. A vertex of $G_{p \times q}$ is a **corner vertex** if it is in $\{(1, 1), (p, 1), (1, q), (p, q)\}$. Otherwise it is a **central vertex**.

At a given stage in the game, a vertex is **critical** if it is removable and its removal leaves an olive removable. This happens when the olive has three remaining neighbours, in which case its removable neighbours are critical. A vertex is **safe** if it is removable, not an olive and not critical.

The strategy for a coati is to take a safe vertex and when possible to force the other coati to take a critical vertex.

5.1 One olive

We first consider the case where the pissaladière contains only one olive. Here again parity is important. Alice will collect the olive if pq is odd or if it is in a corner and Bob will collect a central olive if pq is even. The following lemma is more general since it deals with grid subgraphs and will be useful for the case of 2 olives.

► **Lemma 11.** *Let $O = \{o\}$ be a set of one olive in a subgraph H of the grid (not necessarily connected) on $n(H)$ vertices. Then $\text{col}_2^A(H, O) = 1$ if and only if $n(H)$ is odd or $d_H(o) \leq 2$.*

Proof. If $d_H(o) \leq 2$, then Alice collects the olive at her first move. Henceforth, we may assume $d_H(o) \geq 3$.

Assume $n(H)$ is odd. Alice's strategy is the following. She removes a safe vertex until Bob removes a critical vertex. Then the olive becomes removable and Alice grabs it. Let us prove that Alice can always take a safe vertex. Assume that it is Alice's turn to play and no critical vertex has been removed. Since $n(H)$ is odd, the remaining graph has an odd number of vertices and contains at least o and three of its neighbours and hence it has at least 5 vertices. Therefore, at least one of the vertices is neither critical nor o . By symmetry, we may assume that there is such a vertex $x = (a', b')$ in the northwest part from $o = (a, b)$ that is a vertex with $a' \leq a$ and $b' \leq b$. Moreover, we may consider x such that there is no vertex in the northwest part from it. Then x is removable, and thus safe concluding the proof. If $n(H)$ is even, Bob can play the same strategy as Alice's above, and so collects the olive. ◀

A particular case of the previous lemma is when the subgraph is the grid itself.

► **Proposition 12.** *Let o be the single olive in $G_{p \times q}$. Then $\text{col}_2^A(G_{p \times q}, o) = 1$ if and only if pq is odd or o is a corner.*

With Proposition 12, we can answer questions (Q1) to (Q4) when there is a single olive on the pissaladière. Note that for any $X \in \{A, B\}$, $\text{win}_2^X(G_{p \times q}, 1) = \overline{\text{col}}_2^X(G_{p \times q}, 1)$

► **Corollary 13.** *(i) if pq is odd then $\text{sec}_2^A(G_{p \times q}, 1) = 1$, and $\text{sec}_2^A(G_{p \times q}, 1) = 0$, else (ii) $\text{sec}_2^B(G_{p \times q}, 1) = 0$.*

(iii) If pq is odd then $\overline{\text{col}}_2^A(G_{p \times q}, 1) = 1$ and $\overline{\text{col}}_2^B(G_{p \times q}, 1) = 0$, else (iii) $\overline{\text{col}}_2^A(G_{p \times q}, 1) = \frac{4}{pq}$ and $\overline{\text{col}}_2^B(G_{p \times q}, 1) = 1 - \frac{4}{pq}$.

5.2 Pissaladière of height 2

In this subsection, we consider SHARING A PISSALADIÈRE when the pissaladière has two rows. The two coatis plays the 2-degenerate game on the grid $G_{2 \times q}$. We shall prove that $\text{sec}_2^A(G_{2 \times q}, k) = 0$ and $\text{sec}_2^B(G_{2 \times q}, k) = \lceil \frac{k-1}{3} \rceil$ (Corollary 15) and $\overline{\text{col}}_2^A(G_{2 \times q}, k)$ tends to 0 when $q \rightarrow \infty$ and k is fixed (Proposition 16).

12:14 Grabbing Olives on Linear Pizzas and Pissaladières

Let O be a set of olives in $G_{2 \times q}$. An **olive component** of O in G is a component of $G_{2 \times q}(O)$. The set of olive components of size ≥ 2 will be denoted $\mathcal{C}'(O)$. An olive in a component of size 1 is said to be **lonesome**; said otherwise an olive is lonesome if none of its neighbours is in O . Two lonesome olives are **related** if they have two common neighbours. The **lonesome graph** is the graph whose vertices are the lonesome olives and in which there is an edge between two lonesome olives if and only if they are related. A **wave** is a connected component in the lonesome graph. We denote by $\mathcal{W}(O)$ the set of waves. If there is a lonesome corner olive in O which is in an odd wave, then we set $\epsilon(O) = 1$, otherwise we set $\epsilon(O) = 0$.

► **Lemma 14.** *Let O be a set of olives in $G_{2 \times q}$. Then*

$$\text{col}_2^B(G_{2 \times q}, O) \geq \sum_{W \in \mathcal{W}(O)} \left\lceil \frac{|W|}{2} \right\rceil - \epsilon(O) + \sum_{C \in \mathcal{C}'(O)} \left\lfloor \frac{|C|}{2} \right\rfloor.$$

Proof. Let v be a vertex. We denote by $L(v)$ (resp. $R(v)$) the set of vertices that are left (resp. right) to v and not adjacent to v :

$$L(a, b) = \{(a', b') \mid b' < b\} \setminus \{(a, b-1)\} \quad \text{and} \quad R(a, b) = \{(a', b') \mid b' > b\} \setminus \{(a, b+1)\}.$$

Note that if v is a central vertex then $L(v)$ and $R(v)$ are both odd.

Bob's strategy is the following.

1. If an olive is removable, then Bob collects it.
2. If there is no removable olive, then Bob takes a removable vertex not adjacent to a remaining lonesome olive (as we will see such a vertex always exists).

Let us show that when there is no removable olive, Bob has always the possibility to take a removable vertex not adjacent to a lonesome olive and so 2. can be applied. That is clear if there is no lonesome olive. If a lonesome olive remains, then it is a central one. Let o_ℓ be the leftmost (that is the one with smallest second coordinate) remaining lonesome olive and o_r the rightmost (that is the one with largest second coordinate) remaining lonesome olive. Observe that all removable vertices are in $L(o_\ell) \cup R(o_r)$ which cardinal is even. Then Bob removes a vertex in this set. Doing so Alice will not be able to collect a lonesome olive in her next move. The only way for Alice to collect a lonesome central olive is to remove a vertex adjacent to two related lonesome olives, to let Bob collect one of them by 1., and to collect the other. However Alice may collect a lonesome olive in a corner on her first move. Therefore Bob collects at least $\sum_{W \in \mathcal{W}(O)} \lceil |W|/2 \rceil - \epsilon(O)$ lonesome olives.

Moreover, for any olive component C of size greater than 1, Bob collects at least $\lfloor |C|/2 \rfloor$ olives in C . Indeed each time Alice collects an olive in the component if there is a remaining olive Bob also collects another one. ◀

► **Corollary 15.** *Let k be a positive integer.*

- (i) *If $q \geq 2k + 1$, then $\text{sec}_2^A(G_{2 \times q}, k) = 0$.*
- (ii) *If $q \geq k + 1$, then $\text{sec}_2^B(G_{2 \times q}, k) = \lceil \frac{k-1}{3} \rceil$.*

Proof. (i) Let $O = \{(1, b) \mid 2 \leq b \leq 2k + 1 \text{ and } b \text{ even}\}$. Then O is made of k lonesome central olives which are not related to each other. Hence, by Lemma 14, $\text{col}_2^B(G_{2 \times q}, O) \geq |\mathcal{C}^1(O)| = |\mathcal{C}(O)| = k$.

(ii) Using Lemma 14, one can show that $\text{sec}_2^B(G_{2 \times q}, k) \geq \lceil \frac{k-1}{3} \rceil$. Let us now prove $\text{sec}_2^B(G_{2 \times q}, k) \leq \lceil \frac{k-1}{3} \rceil$. Since sec_2^B is non-decreasing it suffices to prove this statement when $k \equiv 1 \pmod 3$, say $k = 3h + 1$ for some integer h .

Let $O_i = \{(1, 3i), (2, 3i), (2, 3i + 1)\}$ for all $i \in [h]$, and let $O = \{(1, 1)\} \cup \bigcup_{i=1}^h O_i$.

Set $L_i = \{(a, b) \mid b < 3i - 1\}$ and $R_i = \{(a, b) \mid b > 3i + 2\} \cup \{(1, 3i + 2)\}$.

Let us describe an optimal strategy for Alice. She first removes the olive $(1, 1)$. Then at each move she does the following. If it is possible to collect an O_i , then she collects the first olive. By lemma 8 Bob takes the second olive, then Alice takes the third.

Otherwise let ℓ (resp. r) be the smallest (resp. largest) integer i such that O_i has not been removed. Then Alice removes a vertex of $L_\ell \cup R_r$. This is always possible because all removed vertices are in this set and this set has an odd number of vertices. Doing so Bob will not be able to collect an olive at his next move. (In particular, it will not collect the first olive of any O_i).

Applying this strategy Alice collects the first and third olives of all O_i . Therefore Alice collects two olives per O_i . Since she also collects the olive $(1, 1)$ in her first move, she collects $2h + 1$ olives. Consequently Bob collects $h = \frac{k-1}{3}$ olives. ◀

► **Proposition 16.** *Let be fixed positive integer k .*

$$\begin{aligned} \overline{\text{col}}_2^A(G_{2 \times q}, k) &= \mathcal{O}(1/q) \text{ and } \overline{\text{col}}_2^B(G_{2 \times q}, k) = k + \mathcal{O}(1/q); \\ \text{win}_2^A(G_{2 \times q}, k) &= \mathcal{O}(1/q), \text{ and } \text{win}_2^B(G_{2 \times q}, k) = 1 + \mathcal{O}(1/q). \end{aligned}$$

Proof. We distinguish two kinds of olive sets: *ordinary sets* in which the k olives are central lonesome and not related, and the ones that are not ordinary that we call *rare sets*. For an ordinary set O , by Lemma 14, $\text{col}_2^B(G_{2 \times q}, O) = k$ and so $\text{col}_2^A(G_{2 \times q}, O) = 0$.

A usual combinatorial computation gives that the number of rare sets is $\mathcal{O}(q^{k-1})$. The result follows. ◀

5.3 Two olives on a pissaladière

In this subsection, the coatis are given a pissaladière $G_{p \times q}$ with two olives thrown uniformly at random on two different slices. Our aim is to estimate the number of olives that Alice (or Bob) will collect when playing the 2-degenerate game. Unsurprisingly, the parity of the number $n = p \times q$ of slices plays an important role. Therefore, we define **Content** (C) to be Alice (resp. Bob) and **Defeated** (D) to be Bob (resp. Alice) if n is odd (resp. even).

The following lemma shows that Content always collects at least one olive.

► **Lemma 17.** *For any set of two olives O , Content collects at least one olive.*

Proof. Let O be a set of two olives.

If O contains two corners, then Content can remove one of them on its first move, so $\text{col}_2^C(G_{p \times q}, O) \geq 1$.

Assume now that one of the two olives, say o_1 , is not a corner. Playing its optimal strategy for $\{o_1\}$, Content collects at least $\text{col}_2^C(G_{p \times q}, \{o_1\})$ olives. Thus $\text{col}_2^C(G_{p \times q}, O) \geq \text{col}_2^C(G_{p \times q}, \{o_1\}) = 1$, by Lemma 11. ◀

We shall prove that Content almost always grabs the two olives.

► **Theorem 18.** $\overline{\text{col}}_2^C(G_{p \times q}, 2) = 2 + \mathcal{O}(1/n)$.

For the proof we will use the following notation. A vertex $v = (a, b)$ has (at most) four neighbours: its **north neighbour** $\mathbb{N}(v) = (a - 1, b)$, its **south neighbour** $\mathbb{S}(v) = (a + 1, b)$, its **west neighbour** $\mathbb{W}(v) = (a, b - 1)$ and its **east neighbour** $\mathbb{E}(v) = (a, b + 1)$.

The (open) **neighbourhood** of a vertex v , denoted by $N_G(v)$, in a graph G is the set of vertices adjacent to it. The **closed neighbourhood** of a vertex v is $N_G[v] = N_G(v) \cup \{v\}$. When the graph G is clear from the context, we often drop the superscript and use $N(v)$ and $N[v]$ instead of $N_G(v)$ and $N_G[v]$ respectively.

12:16 Grabbing Olives on Linear Pizzas and Pissaladières

Let $S = V \setminus (N[o_1] \cup N[o_2])$. Let S^* be the set of vertices of S that can be removed without removing any vertex of $N[o_1] \cup N[o_2]$.

For a matching (set of independent edges) M , we denote by $V(M)$ the set the vertices incident to edges of this matching, and for every vertex v in $V(M)$ we denote by $M(v)$ the vertex matched to v by M (that is such that $\{v, M(v)\} \in M$).

At a given stage in the game, a vertex is **critical** if it is removable and its removal leaves an olive removable. This happens when the olive has three remaining neighbours, in which case its removable neighbours are critical. A vertex is **safe** if it is removable, not a olive and not critical. A non-olive vertex which is not removable is called **locked**. Observe that a locked vertex may become removable and thus either critical or safe. A removable vertex which is one of the four neighbours of an olive goes from safe to critical when another neighbour of the olive is removed. It might go from critical to safe when its neighbouring olive is removed. Note there always exists a safe vertex in S^* unless all vertices of S^* have been removed.

Let $O = \{o_1, o_2\}$ be a set of two olives in $G_{p \times q}$. We set $o_1 = (a_1, b_1)$ and $o_2 = (a_2, b_2)$ and **we always assume that o_1 is smaller than o_2 in the lexicographical order, that is either $a_1 < a_2$, or $a_1 = a_2$ and $b_1 < b_2$** . In other words, either o_1 is north of o_2 or o_1 and o_2 are in the same latitude (row) and o_1 is west of o_2 .

For each $i \in [2]$, we abbreviate $N(o_i)$, $S(o_i)$, $W(o_i)$, $E(o_i)$ in, respectively, N_i, S_i, W_i, E_i . Furthermore, for every $X, Y \in \{N, S, W, E\}$, we also abbreviate $X(Y_i)$ into XY_i , and so on.

We now split the sets of two olives in two categories: rare and ordinary.

A set of two olives is **rare** if it is of one of the following types:

- Type A: At least one olive is a corner.
- Type B: The two olives are at distance at most 4, that is $|a_2 - a_1| + |b_2 - b_1| \leq 4$.
- Type C : The two olives are on a same row along the two opposite sides, that is $o_1 = (a, 1)$ and $o_2 = (a, q)$ with $1 < a < p$. Or the two olives are on a same column along the two opposite sides, that is $o_1 = (1, b)$ and $o_2 = (p, b)$ with $1 < b < q$.

The sets that are not rare are called **ordinary**.

We shall now prove that Content collects the two olives if the set is ordinary. As the number of ordinary sets of two olives is obviously $\binom{n}{2} + \mathcal{O}(n)$, this directly implies Theorem 18. Note that for many rare sets, Content also collects the two olives. Due to lack of space and because it is sufficient to prove Theorem 18, we only prove it here for ordinary sets. For more details, see [1].

► **Theorem 19.** *Let O be an ordinary set of two olives in $G_{p \times q}$. Then Content collects the two olives.*

The strategy for Content is to take a safe vertex and when possible to force Defeated to take a critical vertex. The proof of this theorem is in two parts; in one we prove that Content grabs the first olive, and in the other we prove that Content grabs the second olive. As it is the easiest, we begin with this later part.

► **Lemma 20.** *Let O be a set of two olives at distance at least 3 in $G_{p \times q}$. Then Content collects the second olive.*

Proof. Let X be the coati that collected the first olive and \bar{X} its opponent. By Lemma 8, we may assume that X collected this olive as soon as possible, and so right after \bar{X} removed a critical vertex adjacent to it. Hence, after the removal of the first olive, we are left with a subgraph H of the grid in which the second olive is not removable (for otherwise it would

have been possible to remove it earlier since the two olives have no common neighbours). If X is Defeated (resp. Content), then H has an odd (resp. even) number of vertices, and so by Lemma 11 with Content (resp. Defeated) in the role of Alice, Content collects the second olive. \blacktriangleleft

► **Lemma 21.** *Let O be an ordinary set of two olives in $G_{p \times q}$. Then Content collects the first olive.*

Proof. Set $V = V(G_{p \times q})$ and $O = (o_1, o_2)$ with $o_1 = (a_1, b_1)$ and $o_2 = (a_2, b_2)$. Recall that we may assume, as stated above, that $a_1 < a_2$, or $a_1 = a_2$ and $b_1 < b_2$.

Since O is not of Type A, it contains no corner. Therefore a coati collects the first olive if and only its opponent is the first to remove a critical vertex. Therefore the strategy of both coatis is to force its opponent to be the first to remove a critical vertex. Therefore we may assume that **both coatis remove a safe vertex until it is impossible. Let R be the set of vertices removed until a coati is blocked, that is there is no more safe vertex to remove. We shall prove that $|R| \equiv pq \pmod{2}$, which implies that Content collects the first olive.** Recall that as long as there remains at least one vertex of S^* in the graph, one of them is safe and the coatis are not blocked. So $S^* \subseteq R$.

In the figures, we put the two olives and their neighbours in general position. Note that if o_1 (resp. o_2) has degree 3 – that is on the border –, then one of $\mathbb{N}_1, \mathbb{W}_1, \mathbb{S}_1$ (resp. $\mathbb{N}_2, \mathbb{E}_2, \mathbb{S}_2$) does not exist and the three neighbours of o_1 (resp. o_2) are critical. Recall that there is no olive in a corner (otherwise it will be a rare set of Type 1). We indicate with a $*$ the vertices which are in S^* . We put nothing for vertices locked at the beginning and recall that they might at some stage become removable and thus either critical and safe.

By symmetry, we can suppose furthermore that $b_2 \geq b_1$ and that $a_2 - a_1 \leq b_2 - b_1$. We distinguish various cases. We consider them in decreasing order of the possible values of $a_2 - a_1$ and for each value of $a_2 - a_1$ in decreasing order of the values of $b_2 - b_1$.

▷ **Claim 22.** If $a_2 - a_1 \geq 3$ (and so $b_2 - b_1 \geq 3$), then Content collects the first olive.

Proof. One can check that $S^* = S$. (See Table 2.)

■ **Table 2** Two olives with $a_2 - a_1 = 3$ and $b_2 - b_1 = 3$. Vertices in S^* are marked with $*$.

*	*	*	*	*	*	*	*
*	*	\mathbb{N}_1	*	*	*	*	*
*	\mathbb{W}_1	o_1	\mathbb{E}_1	*	*	*	*
*	*	\mathbb{S}_1	*	*	*	*	*
*	*	*	*	*	\mathbb{N}_2	*	*
*	*	*	*	\mathbb{W}_2	o_2	\mathbb{E}_2	*
*	*	*	*	*	\mathbb{S}_2	*	*
*	*	*	*	*	*	*	*

If one olive has four neighbours R contains also a neighbour of this olive. Thus $|R| = pq - 8$, and so Content collects the first olive. \blacktriangleleft

▷ **Claim 23.** If $a_2 - a_1 = 2$ and $b_2 - b_1 \geq 4$, then Content collects the first olive.

Proof. The proof is identical to Claim 22 as again $S^* = S$. \blacktriangleleft

▷ **Claim 24.** If $a_2 - a_1 = 2$ and $b_2 - b_1 = 3$, then Content collects the first olive.

12:18 Grabbing Olives on Linear Pizzas and Pissaladières

Proof. One can check that $S^* = S \setminus \{\mathbb{S}\mathbb{E}_1, \mathbb{N}\mathbb{W}_2\}$. (See Table 3). We distinguish two cases:

Assume first that either $\{o_1$ is of degree 3 or R contains \mathbb{W}_1 or $\mathbb{N}_1\}$ and either $\{o_2$ is of degree 3 or R contains \mathbb{E}_2 or $\mathbb{S}_2\}$. Then $\mathbb{S}\mathbb{E}_1$ and $\mathbb{N}\mathbb{W}_2$ remain locked, that is do not belong to R . Thus $|R| = pq - 10$, and so Content collects the first olive.

Otherwise R contains a vertex in $\{\mathbb{S}_1, \mathbb{E}_1, \mathbb{N}_2, \mathbb{W}_2\}$. Then R contains necessarily $\mathbb{S}\mathbb{E}_1$ and $\mathbb{N}\mathbb{W}_2$. Thus $|R| = pq - 8$, and so Content collects the first olive. \triangleleft

■ **Table 3** Two olives with $a_2 - a_1 = 2$ and $b_2 - b_1 = 3$. Vertices in S^* are marked with $*$.

*	*	*	*	*	*	*	*
*	*	\mathbb{N}_1	*	*	*	*	*
*	\mathbb{W}_1	o_1	\mathbb{E}_1	*	*	*	*
*	*	\mathbb{S}_1			\mathbb{N}_2	*	*
*	*	*	*	\mathbb{W}_2	o_2	\mathbb{E}_2	*
*	*	*	*	*	\mathbb{S}_2	*	*
*	*	*	*	*	*	*	*

▷ **Claim 25.** If $a_2 - a_1 = 1$ and $b_2 - b_1 \geq 4$, then Content collects the first olive.

Proof. The proof is similar to Claim 24. Here $S^* = S \setminus L$, where $L = \{(a_1, b) \mid b_1 + 1 < b < b_2\} \cup \{(a_1 + 1, b) \mid b_1 < b < b_2 - 1\}$. If we set $d = b_2 - b_1$ ($d \geq 4$), then $|L| = 2(d - 2)$. We have the same two cases as in the preceding claim. In the first case $|R| = pq - 2d - 6$, and in the second case $|R| = pq - 8$. In both cases, Content collects the first olive. \triangleleft

▷ **Claim 26.** If $a_1 = a_2$ and $b_2 - b_1 \geq 5$, then Content collects the first olive.

Proof. Set $d = b_2 - b_1$. We have $d \geq 3$. Moreover, since O is not of Type C, $d \leq q - 2$.

If $a_1 > 1$, let $L_{\mathbb{N}} = \{(a_1 - 1, b) \mid b_1 < b < b_2\}$, in which case $|L_{\mathbb{N}}| = d - 1$ and if $a_1 = 1$, $L_{\mathbb{N}} = \emptyset$. If $a_1 < q$, let $L_{\mathbb{S}} = \{(a_1 + 1, b) \mid b_1 < b < b_2\}$, in which case $|L_{\mathbb{S}}| = d - 1$ and if $a_1 = q$, $L_{\mathbb{S}} = \emptyset$. Let $L_{\text{eq}} = \{(a_1, b) \mid b_1 + 1 < b < b_2 - 1\}$; then $|L_{\text{eq}}| = d - 3$. Let $L = L_{\mathbb{N}} \cup L_{\text{eq}} \cup L_{\mathbb{S}}$. One can check that $S^* = S \setminus L$. (See Table 4).

■ **Table 4** Two olives with $a_2 = a_1$ and $b_2 - b_1 = 5$. Vertices in S^* are marked with $*$.

*	*	*	*	*	*	*	*	*	*
*	*	\mathbb{N}_1					\mathbb{N}_2	*	*
*	\mathbb{W}_1	o_1	\mathbb{E}_1			\mathbb{W}_2	o_2	\mathbb{E}_2	*
*	*	\mathbb{S}_1					\mathbb{S}_2	*	*
*	*	*	*	*	*	*	*	*	*

If $a_1 = 1$ or $a_1 = q$, then $b_1 \neq 1$ and $b_q \neq q$ because O is not of Type 1. The two olives have only three neighbours, so $R = S^*$ and $|R| = pq - 2d - 4 \equiv pq \pmod{2}$. Thus Content collects the first olive. In what follows **we suppose that** $1 < a_1 < q$ and so $|L| = 3d - 5$.

Observe the following facts:

- If \mathbb{N}_1 or \mathbb{N}_2 is in R , then $L_{\mathbb{N}} \subset R$. Otherwise $L_{\mathbb{N}} \cap R = \emptyset$.
- If \mathbb{S}_1 or \mathbb{S}_2 is in R , then $L_{\mathbb{S}} \subset R$. Otherwise $L_{\mathbb{S}} \cap R = \emptyset$.
- $R \cap \{\mathbb{E}_1, \mathbb{W}_2\} = \emptyset$.
- If $\{\mathbb{N}_1, \mathbb{S}_2\} \subset R$ or $\{\mathbb{S}_1, \mathbb{N}_2\} \subset R$, then $L_{\text{eq}} \subseteq R$. Otherwise $L_{\text{eq}} \cap R = \emptyset$.

If R contains at least one vertex in $\{\mathbb{N}_1, \mathbb{N}_2, \mathbb{S}_1, \mathbb{S}_2\}$, then $|R| \equiv pq \pmod{2}$ so Content collects the first olive. Indeed suppose for example that R contains \mathbb{N}_1 , then either it contains \mathbb{S}_2 and $|R| = pq - 8$, otherwise $|R| = pq - 2d - 4$ (using the above facts). If $R \cap \{\mathbb{N}_1, \mathbb{N}_2, \mathbb{S}_1, \mathbb{S}_2\} = \emptyset$, then all the vertices of $L_{\mathbb{N}} \cup L_{\text{eq}} \cup L_{\mathbb{S}}$ remain locked. Thus $|R| = pq - 3d - 3$.

If d is odd then $|R| \equiv pq \pmod{2}$ in both cases, so Content collects the first olive.

If d is even, Content has to be careful. He must remove safe vertices so that $R \cap \{\mathbb{N}_1, \mathbb{N}_2, \mathbb{S}_1, \mathbb{S}_2\} \neq \emptyset$. Since O is not of Type 5, then either $b_1 > 1$ or $b_2 < q$. By symmetry, we may assume $b_1 > 1$. we now describe a strategy for Content such that R does not contain \mathbb{W}_1 .

Let M be the matching $\{(i, b_1 - 1), (i, b_1)\}$. Observe that M matches \mathbb{W}_1 with o_1 , NW_1 with \mathbb{N}_1 , and SW_1 with \mathbb{S}_1 . Content applies the following procedure until either s/he collects an olive, or a vertex in $\{\mathbb{N}_1, \mathbb{S}_1, \mathbb{N}_2, \mathbb{S}_2\}$ is removed.

(R) If Defeated just removed a vertex in $V(M)$, then Content removes the vertex to which it is matched by M . Otherwise, it removes a safe vertex not in $V(M)$.

Let us prove that this strategy is valid, i.e. that rule can be applied. Assume Defeated removed a vertex v in $V(M)$, then $M(v)$ has not been removed earlier. Furthermore, if v is , then one of its neighbours w in $\{\mathbb{N}(v), \mathbb{S}(v)\}$ has been removed before and so $M(w)$ has been removed and therefore $M(v)$ becomes removable and Content can remove it. Note also that $R \setminus V(M) = V(G) \setminus \{V(M) \cup L \cup \{\mathbb{E}_1, \mathbb{W}_2, \mathbb{N}_2, o_2, \mathbb{S}_2\}\}$ (\mathbb{N}_2 and \mathbb{S}_2 cannot have been removed otherwise the procedure stopped). Therefore, $|R \setminus V(M)| = pq - 2p - 3d - 10 \equiv pq \pmod{2}$ (as d is even) and so when Content plays it remains at least one removable vertex in $R \setminus V(M)$ and the second part of Rule (R) can be applied.

Finally, suppose for a contradiction that \mathbb{W}_1 is removed during the procedure. Before being removed one of the two vertices NW_1 or SW_1 has been removed (because o_1 is not removed since the procedure did not stop). But those vertices are matched with \mathbb{N}_1 and \mathbb{S}_1 respectively. Thus, by Rule (R), one of \mathbb{N}_1 and \mathbb{S}_1 has been removed before and the procedure stopped, a contradiction.

Thus, applying the above procedure, one vertex in $\{\mathbb{N}_1, \mathbb{N}_2, \mathbb{S}_1, \mathbb{S}_2\}$ is removed. Hence, as noted before $|R| \equiv pq \pmod{2}$ and Content collects the first olive. \triangleleft

Since O is not of type B, the only possible cases are the ones considered in the above claims. \blacktriangleleft

6 Conclusion

In this paper, we studied some particular cases of graph-grabbing games. We made an extensive analysis of SHARING A LINEAR PIZZA, including optimal strategies and complexity results. However there are still many open problems : study the case of adjacent weighted vertices for general weights and prove the conjecture that for $\{0, 1\}$ weights each player grabs asymptotically half of the olives.

We also introduced d -generate games for which we obtained preliminary results and performed a more extensive study for the grids case that we called Pissaladière. We proved that for two olives A (resp. B) almost always grabs the two olives if the size is odd (resp. even). We conjecture that the same result holds for k olives.

► **Conjecture 27.** *Let O be an ordinary set of k olives in $G_{p \times q}$. If $n = pq$ is odd (resp. even), then A (resp. B) grabs the k olives. If n is odd $\overline{\text{col}}_2^A(G_{p \times q}, k) = k + \mathcal{O}(1/n)$. If n is even $\overline{\text{col}}_2^B(G_{p \times q}, 2) = k + \mathcal{O}(1/n)$*

Finally, many problems remain open for other d -generate games.

References

- 1 Jean-Claude Bermond, Michel Cosnard, and Frédéric Havet. Grabbing olives on linear pizzas and pissaladières. Working paper, March 2022. URL: <https://hal.inria.fr/hal-03623938>.
- 2 Josef Cibulka, Jan Kyncl, Viola Mészáros, Rudolf Stolar, and Pavel Valtr. Solution of Peter Winkler’s pizza problem. In Jiri Fiala, Jan Kratochvíl, and Mirka Miller, editors, *International Workshop on Combinatorial Algorithms - IWOCA*, volume 5874 of *Lecture Notes in Computer Science*, pages 356–367. Springer, 2009. doi:10.1007/978-3-642-10217-2_35.
- 3 Josef Cibulka, Jan Kyncl, Viola Mészáros, Rudolf Stolar, and Pavel Valtr. Graph sharing games: Complexity and connectivity. *Theoretical Computer Science*, 494:49–62, 2013. doi:10.1016/j.tcs.2012.12.029.
- 4 Yoshimi Egawa, Hikoe Enomoto, and Naoki Matsumoto. The graph grabbing game on $K_{m,n}$ -trees. *Discrete Mathematics*, 341(6):1555–1560, 2018. doi:10.1016/j.disc.2018.02.023.
- 5 Soogang Eoh and Jihoon Choi. The graph grabbing game on 0,1-weighted graphs. *Results in Applied Mathematics*, 3:100028, 2019. doi:10.1016/j.rinam.2019.100028.
- 6 Kolja Knauer, Piotr Micek, and Torsten Ueckerdt. How to eat 4/9 of a pizza. *Discrete Mathematics*, 311(16):1635–1645, 2011. doi:10.1016/j.disc.2011.03.015.
- 7 Urban Larsson, Richard J. Nowakowski, and Carlos Pereira dos Santos. *Games of No Chance 5*, volume 70 of *MSRI Publications*, chapter 3 Scoring games: the state of the play, pages 89–111. Cambridge University Press, 2017.
- 8 Piotr Micek and Bartosz Walczak. A graph-grabbing game. *Combinatorics, Probability and Computing*, 20(4):623–629, 2011. doi:10.1017/S0963548311000071.
- 9 Piotr Micek and Bartosz Walczak. Parity in graph sharing games. *Discrete Mathematics*, 312(10):1788–1795, 2012. doi:10.1016/j.disc.2012.01.037.
- 10 Moshe Rosenfeld. A gold grabbing game, <http://garden.irmacs.sfu.ca/category/rosenfeld>, 2009.
- 11 Deborah E. Seacrest and Tyler Seacrest. Grabbing the gold. *Discrete Mathematics*, 312(10):1804–1806, 2012. doi:10.1016/j.disc.2012.01.010.
- 12 Peter Winkler. *Mathematical Puzzles: a connoisseur’s collection*. CRC Press, 2004.
- 13 Peter Winkler. Problem posed at Building Bridges, a conference in honour of 60th birthday of L. Lovász, Budapest, 2008.
- 14 Peter Winkler. *Mathematical Puzzles*. A K Peters CRC Press, 2020.

How Fast Can We Play Tetris Greedily with Rectangular Pieces?

Justin Dallant  

Université libre de Bruxelles, Belgium

John Iacono  

Université libre de Bruxelles, Belgium

Abstract

Consider a variant of Tetris played on a board of width w and infinite height, where the pieces are axis-aligned rectangles of arbitrary integer dimensions, the pieces can only be moved before letting them drop, and a row does not disappear once it is full. Suppose we want to follow a greedy strategy: let each rectangle fall where it will end up the lowest given the current state of the board. To do so, we want a data structure which can always suggest a greedy move. In other words, we want a data structure which maintains a set of $O(n)$ rectangles, supports queries which return where to drop the rectangle, and updates which insert a rectangle dropped at a certain position and return the height of the highest point in the updated set of rectangles. We show via a reduction from the Multiphase problem [Pătraşcu, 2010] that on a board of width $w = \Theta(n)$, if the OMv conjecture [Henzinger et al., 2015] is true, then both operations cannot be supported in time $O(n^{1/2-\epsilon})$ simultaneously. The reduction also implies polynomial bounds from the 3-SUM conjecture and the APSP conjecture. On the other hand, we show that there is a data structure supporting both operations in $O(n^{1/2} \log^{3/2} n)$ time on boards of width $n^{O(1)}$, matching the lower bound up to an $n^{o(1)}$ factor.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases Tetris, Fine-grained complexity, Dynamic data structures, Axis-aligned rectangles

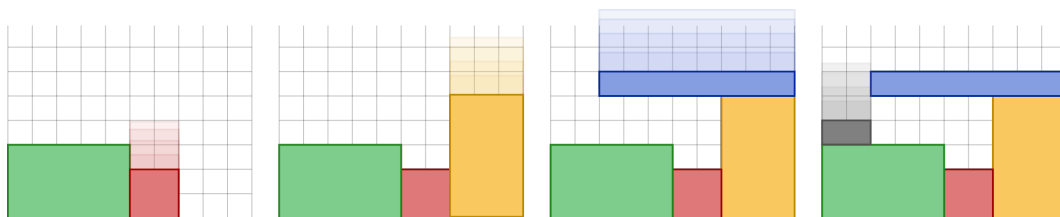
Digital Object Identifier 10.4230/LIPIcs.FUN.2022.13


Related Version *Full Version:* <https://arxiv.org/abs/2202.10771>

Funding *Justin Dallant:* Supported by the French Community of Belgium via the funding of a FRIA grant.

John Iacono: Supported by the Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001 1.

Acknowledgements We thank Nemaou for involuntarily providing us with the initial inspiration for this work.



 **Figure 1** Illustration of a few turns of the game.



© Justin Dallant and John Iacono;
licensed under Creative Commons License CC-BY 4.0
11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 13; pp. 13:1–13:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Consider a game played on a board of a certain width w , which has a wall on the bottom, left and right side but extends on the top as far as we wish. The game goes like this: at each turn, we are given a rectangle of arbitrary integer height and arbitrary integer width (as long as it fits on the board), and we let it drop on the other rectangles (or the bottom of the board) from above starting at the horizontal position of our liking. This is not unlike the basic rule of the famous game Tetris, but instead of having tetrominoes (pieces made of four orthogonally connected unit squares), we have rectangular pieces made of any number of unit squares. See Figure 1 for an example of a possible execution of the game.

Now, our goal as the rectangle droppers is to have the rectangles be as low as possible (in Tetris, once a certain height is reached, it means game over!). There are of course various clever strategies and heuristics which could be adopted here. But assume without loss of plausibility that we (the authors) are greedy, and as greedy players we want to adopt the following strategy: at every turn, we let the rectangle drop at a horizontal position which will minimize its vertical position, with no care for the future rectangles we might be given. The question we seek to answer is: how long do we need to think in each turn when adopting this strategy? Here we assume, with a not-so-insignificant loss of plausibility, that we are as powerful as a word-RAM machine. In particular, all conjectures, cited results and claims in this paper are for the word-RAM model with words of length $\Theta(\log n)$, although some carry over to other popular models. All integers appearing in any problem or procedure in this paper have a binary representation of $O(\log n)$ bits.

Let us formalize this problem with the following definition.

► **Definition 1.** *A Rectangle Dropping Data Structure (RDDS) maintains a set of $O(n)$ independent¹ axis-aligned rectangles in the plane with integer coordinates, lying on or above the x -axis and between the vertical lines $x = 0$ and $x = w$, and allows the following:*

- *(Preprocessing) Initialize an empty RDDS containing no rectangle.*
- *(Update) Given an axis-aligned rectangle R and a non-negative integer x_d , drop R with left border at x -coordinate x_d (here we assume that R and x_d are such that R will lie between the lines $x = 0$ and $x = w$).*
- *(Query) Given an axis-aligned rectangle R , return the position where R must be dropped to end up as low as possible (or one such position if it is not unique) as well as the height of the highest point in the set of rectangles which would result from that move.*

Note that an RDDS is actually a bit more powerful than what we need to follow the greedy strategy, as this data structure does not force us to play greedily at each turn, but provides a way to suggest a greedy move at every turn. As no one likes to be forced to do anything (even when the thing in question is what we would have done of our own volition), we will study this variant of the problem. Our results could however be adapted to the case where we can only perform updates corresponding to a greedy strategy.

We would like to give some bounds on the time needed to perform the operations of an RDDS. Because in practice it seems hard to prove good lower bounds on the time needed to solve computational problems, one popular approach (dating back to the seminal works of Cook [26] and Karp [54]) has been to show the hardness of some problem assuming the conjectured hardness of some other “key problem” (this constitutes what is called a conditional lower bound). This approach, when dealing with specific polynomial bounds (in

¹ We say two rectangles are independent when their interiors do not intersect.

contrast to showing the NP-hardness of a problem for example) is part of a field of study known today as fine-grained complexity (see e.g., introductory surveys by Bringmann [20] and V. V. Williams [94]). Some of these key problems are the 3-SUM problem, All-Pairs-Shortest-Paths (APSP), Boolean Matrix Multiplication (BMM), Triangle Detection in a graph, Boolean Satisfiability (SAT) and the Orthogonal Vectors problem (2OV).

The use of such approaches for data structures solving dynamic problems was pioneered by Pătraşcu [78]. In particular, he introduced the so-called Multiphase problem and showed that it can be fine-grained reduced to many dynamic problems. He also showed that assuming the famous 3-SUM problem is hard, there are polynomial lower bounds to how fast the Multiphase problem can be solved, which in turn implies polynomial lower bounds for the dynamic problems which it reduces to. There has since been much work on conditional lower bounds for dynamic problems [1, 2, 3, 5, 6, 7, 15, 16, 17, 24, 31, 43, 44, 53, 57, 58, 77, 90, 95]. We mention some of the expansions on Pătraşcu’s work relevant to this paper in Section 2.1. In a previous paper by the authors [32], Pătraşcu’s work and these expansions were exploited to obtain conditional polynomial lower bounds for a variety of dynamic geometric problems. The lower bounds obtained here were inspired by this previous work.

After reviewing the Tetris literature in Section 1.1, and the definitions of the hardness conjectures in Section 2, we reduce the Multiphase problem to the conception of an efficient RDDS in Section 3, implying that under the OMv conjecture [43], any RDDS requires essentially $n^{1/2}$ time per operation. This reduction also implies (weaker, but still polynomial assuming an empty RDDS can be initialized quickly) lower bounds from the 3-SUM conjecture and the APSP conjecture (see Section 2.1 for definitions of these hardness conjectures). We also give in Section 4 a data structure matching the lower bound obtained from the OMv conjecture (up to an $n^{o(1)}$ factor).

1.1 Related work

Tetris, created in 1984 by Alexey Leonidovich Pajitnov of the Dorodnitsyn Computing Centre of the Soviet Academy of Science, was one of the greatest inventions of the USSR, one that was widely embraced by the citizens of both the Warsaw Pact and NATO as well as non-aligned countries. In the computer science literature, the allure of Tetris has been irrepressible, with works using Tetris found in virtually every subdiscipline.

One might be tempted to begin reviewing the literature with the paper whose title is the one single word “Tetris” [96]. However, one will quickly realize that the Tetris that they speak of is not the game so beloved by humanity, but rather “Tetris is an Asynchronous Byzantine Fault Tolerance consensus algorithm designed for next generation high-throughput permission and permissionless blockchain.”

As for the game itself, the most research in the computer science community, unsurprisingly, is about getting computers to play Tetris. There, Tetris has been used as a research tool more often than any other video game [75] and has been subjected to just about every machine learning and artificial intelligence paradigm [4]. However, success is far from guaranteed, as *The Unsuitability of Supervised Backpropagation Networks for Tetris* shows [60].

Researchers have discovered that in addition to computers, humans can also play Tetris. They have studied how humans interact as they play Tetris and how this relates to primate socialization in general, observing that “baboon social intelligence contrasts both with that of Tetris players and human infants, because both have the ability to provoke epistemic effects” [27]. Other human angles have been studied as well, including studying styles of play [18, 55], humanity’s suboptimal choices [80], eye movement while playing [51, 61, 62, 85], and how champion Tetris players play, including the relationship between Hick’s Law and

13:4 How Fast Can We Play Tetris Greedily with Rectangular Pieces?

expertise [64], and social exertion [67]. Tetris has been used as well to develop a model of cognition [91]. Additionally, a “Brain Computer Interface (BCI) game that is inspired [by] the Tetris game” was created where they perform experiments with children with attention deficit and hyperactivity disorder (ADHD) [76] (see also [59, 74]).

Several kinds of researchers have worked on helping humans to play Tetris better [68] and addressed the all-important work on the influence of the tempo of the music one listens to while playing Tetris on the score [47, 48, 49]. Other researchers worked to study human performance in Tetris, including the elements that can be used to predict whether a human will become a Tetris master [8, 41, 63, 81, 82]. The issue of dynamically adjusting the difficulty of the game has also been the subject of study [9, 10, 65, 66, 84].

One great step in the development of Tetris was the creation of a mobile version of “the stable and reliable game based on the mature game algorithm” [92]. Also of note was *Pop-out Tetris* [56] where they applied the “Fish Tank Virtual Reality technique to a commodity tablet” as well as an embedded system created for Tetris [72]. The long history of Tetris and its many implementations has led one group to study these implementations over time, deeming them “worthy of historical analysis” [52].

From an education perspective, Tetris was used by [25] to help students learn Chinese proverbs by putting Chinese characters on each Tetris block. Within mainstream computer science education, Tetris is a common subject of programming projects [73].

Tetris has also spawned its own fan fiction, which has been studied: “In [one work] there are also racial and gender complications, the story ending on a humorous note with an absurd twist, as in a Samuel Beckett play, when a message appears over the head of the disillusioned blocks: “Play again?”. It is obvious that in these stories the game of Tetris is an allegory for human life and behavior” [79]. From the point of view of a narrative, “An experiment was made with the game Tetris, showing that a control system designed around evaluation of player’s tension [makes] it possible to obtain an execution trace [close] to a narrative one” [33].

Following the immense success of cannabis legalization, Tetris has followed suit with Enhanced Tetris Legalization [28, 29, 30, 69].

A surprisingly active field has been the ability of a Tetris player to hide messages via gameplay [70, 71, 86, 87].

On the more theoretical side, it has been shown how to view the game through the lens of Krohn-Rhodes theory [50], how to use mixed integer programming to pack Tetris-like pieces in 3D [37], how to create Tetris boards with a given pattern [45], and how to generate polyominoes [38]. In [13], they “propose a straight Coward algorithm to transform a Tetris-like game into its corresponding automaton.”

Closer to the topic of this paper is the algorithmic work relating to the complexity of playing Tetris. The starting point for this consists of results about the hardness of approximating various Tetris objectives (e.g. number of rows cleared, number of moves before reaching a certain height), even with foreknowledge of the entire sequence of pieces [19, 35, 36]. This work was generalized to show NP-completeness for k -ominoes with $k \geq 4$, and surprisingly also show that it is NP-complete to clear the board even for $k = 2$ when rotations are not allowed [34]. These reductions were further tightened in [11, 12] to when the board is small and in [89] for the variant where you can rotate and then move down, but when no rotations are allowed after the block has begun to fall. The undecidability of whether a set of sequences of Tetris moves described by a regular language contains a sequence that results in an empty board has been proven [46]. While this all sounds very negative, a more positive outlook can be found in *Why Most Decisions Are Easy in Tetris* [83].

Unrelated to Tetris, the problem studied is a form of dynamic *reverse range query* [88]. In a standard range query one has a collection of objects which are preprocessed such that given a range, some question about the objects that intersect the range can be answered. The literature on range queries is immense, with variants depending on the type of objects, the type of ranges, and the specific queries (see, for example, the chapter on range queries in [40]). Classically, what is interesting is that for points and queries that ask how many points intersect the range, the answer is polylog for orthogonal queries and polynomial for non-orthogonal queries, assuming reasonable space. In a reverse range query, we are looking for the ranges that would give a certain result, and to provide the extreme such range under some measure; if there is only a single such reverse range query without parameters, this is interesting as a data structure only when the point set is changing dynamically.

For example, the classic problem of finding the smallest circle enclosing a set of points in the plane can be viewed as such a reverse range query, where one is searching for a circular range query that returns n points, and among all such queries is looking for the smallest one; in the dynamic case the best known data structure takes polylogarithmic time, even when deletions are allowed [22]. On the other hand, the best known data structure for maintaining the largest empty circle in a set of points (with center constrained to be inside a fixed triangle) takes time $\tilde{O}(n^{7/8})$ [21] (or $\tilde{O}(n^{11/12})$ when deletions are also allowed [22]).

For the case of finding (not necessarily axis-aligned) rectangular ranges enclosing all points, the best known data structure takes time $\tilde{O}(n^{1/2})$ to find the minimum-perimeter range and $\tilde{O}(n^{5/6})$ for the minimum-area range (also [21]). No conditional lower bounds are known for these problems, which remains a major open problem. In our case we are looking for a three-sided upwards-facing query that returns zero objects,² and among all such queries to find the one that is as low as possible.

2 A few hardness conjectures

Let us quickly go over some of the key conjectures in fine-grained complexity which will be relevant here. The first, and arguably one of the earliest in the field (with the seminal work of Gajentaan and Overmars [39]), is the 3-SUM conjecture.

► **Definition 2** (3-SUM conjecture). *The following problem (3-SUM) requires $n^{2-o(1)}$ expected time to solve: given a set of n integers, decide if three of them sum up to 0.*

The 3-SUM problem can be easily solved in $O(n^2)$ time, and with a lot of effort this can be slightly improved, both for integer and real inputs [14, 23, 42].

The second problem we will mention is All-Pairs-Shortest-Path (APSP).

► **Definition 3** (APSP conjecture). *The following problem (APSP) requires $n^{3-o(1)}$ expected time to solve: given a weighted directed graph G with no negative cycles, compute the distance between every pair of vertices in G .*

Here, the fastest known algorithm runs in slightly subcubic time [93]. Both problems (3-SUM and APSP) are related in many ways (see e.g., [95]). One particular way in which they are related is that they both reduce in a fine-grained manner to the so-called Exact Triangle problem. In particular, if the following conjecture is false, then both the 3-SUM conjecture and the APSP conjecture are false.

² Although we have presented our results as a board containing rectangles, our upper and lower bounds also hold for points.

► **Conjecture 4** (Exact Triangle conjecture). *The following problem (Exact Triangle) requires $n^{3-o(1)}$ expected time to solve: given a weighted graph G and a target weight T , determine if there is a triangle in G whose edge weights sum to T .*

The following problem and the corresponding conjecture were introduced by Henzinger et al. [43] as a way to unify most known conditional lower bounds for dynamic problems under a single conjecture, and sometimes even strengthen the known bounds.

► **Definition 5** (Online Boolean Matrix-Vector Multiplication (OMv)). *We are given an $n \times n$ boolean matrix M . We can preprocess this matrix, after which we are given a sequence of n boolean column-vectors of size n denoted by v_1, \dots, v_n , one by one. After seeing each vector v_i , we must output the product Mv_i before seeing v_{i+1} .*

► **Conjecture 6** (OMv conjecture). *Solving OMv requires $n^{3-o(1)}$ expected time.*

2.1 Pătraşcu's Multiphase Problem

The following problem was introduced by Pătraşcu [78] as a means to prove polynomial lower bounds for many dynamic problems.

► **Definition 7** (Multiphase Problem). *In the Multiphase Problem, we are given a family $\mathcal{F} = \{F_1, \dots, F_k\}$ of k non-empty subsets of $\{1, 2, \dots, m\}$, such that every element appears in at least one of the subsets. Let $n = m \cdot k$. We want to design a data structure D which maintains a set of $O(n)$ objects and allows for the following:*

- (Step 1) First, we read \mathcal{F} and are allowed $O(t_1 \cdot n)$ expected time to initialize D .
- (Step 2) Then, we receive a subset $J \subset \{1, 2, \dots, m\}$ and are allowed $O(t_2 \cdot m)$ expected time to modify D .
- (Step 3) Finally, we are given an index $1 \leq i \leq k$ and after $O(t_3)$ expected time we decide if $J \cap F_i \neq \emptyset$.

Pătraşcu conjectured that there exist constants $\gamma > 1$ and $\delta > 0$ such that if $k = \Theta(m^\gamma)$ then the Multiphase problem requires $t_1 + t_2 + t_3 \geq n^\delta$. While this is still unknown, he showed that this is true if we assume the 3-SUM conjecture. Specifically, he showed the following (taking $k = \Theta(m^{5/2})$).

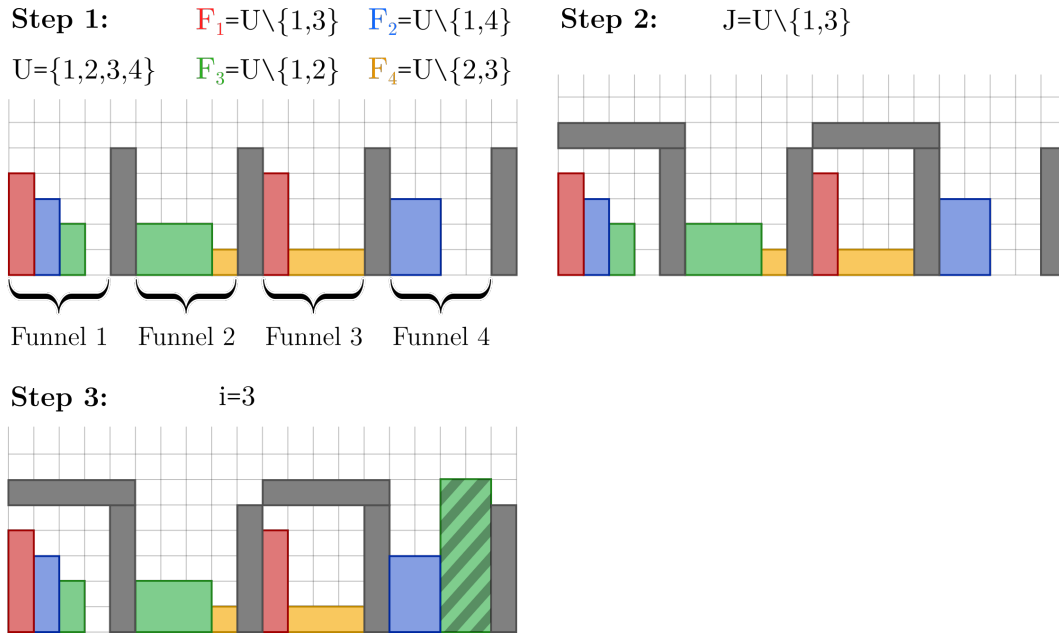
► **Theorem 8** ([78]). *Under the 3-SUM conjecture, the Multiphase Problem requires $t_1 + t_2 + t_3 \geq n^{1/7-o(1)}$.*³

The techniques of Pătraşcu also allow to have different trade-offs in the lower bounds for t_1 , t_2 and t_3 . His results have been expanded upon in different ways. On the one hand, Kopelowitz et al.[58] have tightened the reduction from 3-SUM to Set Disjointness (which is an intermediate problem in the reduction from 3-SUM to the Multiphase problem). Their results in particular imply the following:

► **Theorem 9** ([58]). *Under the 3-SUM conjecture, for any $0 < \gamma < 1$, the Multiphase Problem requires*

$$t_1 \cdot n + t_2 \cdot n + t_3 \cdot n^{\frac{1+\gamma}{3-2\gamma}} \geq n^{\frac{2}{3-2\gamma}-o(1)}.$$

³ Note that here we have expressed the bound in terms of $n = k \cdot m$ whereas Pătraşcu originally expressed his bound in terms of what we call m in this paper.



■ **Figure 2** Illustration of the reduction from the Multiphase problem to RDDS. Notice that in Step 3, the striped green rectangle can only be dropped to lie at a height lower than 2 in a funnel which is not blocked and where there is no other green rectangle. Such a funnel corresponds to an element of U which is both in J and in F_3 .

On the other hand, the hardness of the Multiphase Problem has been shown from other hardness conjectures. Vassilevska Williams and Xu [95] have fine-grained reduced Set Disjointness to the Exact Triangle problem. Their result implies in particular that in the previous theorem, one can replace the 3-SUM conjecture by the Exact Triangle conjecture, giving the following.

► **Theorem 10** ([95]). *Under the Exact Triangle conjecture (and in particular under either the 3-SUM conjecture or the APSP conjecture), for any $0 < \gamma < 1$, the Multiphase problem requires*

$$t_1 \cdot n + t_2 \cdot n + t_3 \cdot n^{\frac{1+\gamma}{3-2\gamma}} \geq n^{\frac{2}{3-2\gamma}-o(1)}.$$

Better bounds can be obtained if we assume the OMv conjecture instead. Indeed, the work of Henzinger et al. implies the following.

► **Theorem 11** ([43]). *Under the OMv conjecture, for any $0 < \gamma < 1$, if t_1 is at most polynomial in n then the Multiphase Problem requires*

$$t_2 \cdot n^{1-\gamma} + t_3 \cdot n^\gamma \geq n^{1-o(1)}.$$

3 Reduction from the Multiphase Problem

► **Theorem 12.** *If there exists an RDDS which maintains a set of $O(n)$ for boards of width $w = \Theta(n)$ with expected preprocessing time t_p , expected update time t_u and expected query time t_q , then we can solve the Multiphase Problem with $t_1 = t_p/n + t_u$, $t_2 = t_u$ and $t_3 = t_q$.*

13:8 How Fast Can We Play Tetris Greedily with Rectangular Pieces?

Proof. Suppose we have such an RDDS and consider an instance \mathcal{F} of the Multiphase problem.

We perform Step 1 as follows (see Figure 2). Given a family $\mathcal{F} = \{F_1, F_2, \dots, F_k\}$ of subsets of $\{1, 2, \dots, m\}$, we set the width of the board to $m \cdot (k + 1) = \Theta(n)$. For every $1 \leq j \leq m$, drop a rectangle of height $k + 1$ and width 1 with left border at $x = j(k + 1) - 1$. This defines m regions of width k on the board (between these rectangles and the borders), which we associate with the m elements of the base set. Now for j going from 1 to m we create a sort of “funnel” structure in region j as follows: for i going from 1 to k , if $j \notin F_i$, drop a rectangle of height $k + 1 - i$ such that its right border is at $x = i + (j - 1)(k + 1)$, its bottom border is at $y = 0$ and its left border lies as far left as possible. All in all, we have inserted $n = O(k \cdot m)$ rectangles in $O(t_p + n \cdot t_u)$ expected time.

When given $J \subset \{1, 2, \dots, m\}$ in Step 2, drop a rectangle of width $k + 1$ and height 1 with left border at $x = (j - 1)(k + 1)$ for every $1 \leq j \leq m$ such that $j \notin J$. This has the effect of “blocking” the regions which are not relevant in the next step, and costs $O(m \cdot t_u)$ expected time.

Finally, when given an index i in Step 3, query the RDDS with a rectangle of width $k + 1 - i$ and height $k + 2$. It is easy to check that if there is some $j \in J \cap F_i$, then the bottom of such a rectangle can drop in the funnel corresponding to j in such a way that its bottom lies at or below $y = k - i$ (so its top lies at or below $y = 2k + 2 - i$). On the other hand, if $J \cap F_i = \emptyset$, then the funnel of every region which was not blocked in the previous is too narrow at height $y = k + 1 - i$ for the rectangle to fall through. Thus, we can solve the Multiphase Problem with a single query in Step 3, costing $O(t_q)$ expected time. ◀

This then implies the following:

► **Theorem 13.** *Under the Exact Triangle conjecture (and in particular under either the 3-SUM conjecture or the APSP conjecture), for any $0 < \gamma < 1$, an RDDS for boards of width $w = \Theta(n)$ requires*

$$t_p + t_u \cdot n + t_q \cdot n^{\frac{1+\gamma}{3-2\gamma}} \geq n^{\frac{2}{3-2\gamma}-o(1)}.$$

In particular, for $\gamma = 2/3$, we have

$$t_p/n + t_u + t_q \geq n^{1/5-o(1)}.$$

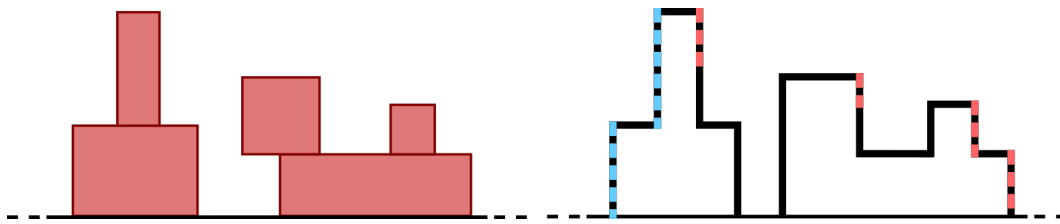
Moreover, if t_p is at most polynomial in n , then under the OMv conjecture, for any $0 < \gamma < 1$ we have

$$t_u \cdot n^{1-\gamma} + t_q \cdot n^\gamma \geq n^{1-o(1)}.$$

In particular, for $\gamma = 1/2$, we have

$$t_u + t_q \geq n^{1/2-o(1)}.$$

Note that, as mentioned in the introduction of the paper, with some care this reduction could be adapted to the case where a query is always followed by an update dropping the rectangle at the returned position, and no other updates are allowed (i.e. we are forced to always follow the suggested greedy strategy). This can be done by creating “notches” of different widths and heights at the top of the rectangles delimiting the different funnels, so that in Step 2 the funnel over which a rectangle is dropped can be chosen by specifying the width of the rectangle. In particular we could get the same bound from the OMv conjecture



■ **Figure 3** Illustration of the skyline, left staircase (blue dotted line) and right staircase (red dotted line) of a set of rectangles.

(exploiting the fact that Theorem 11 already holds for the Multiphase problem where $k = m$). It could also be adapted to the case where we are allowed to rotate the rectangles before dropping them, by stretching all the rectangles enough horizontally so that rotating them is never advantageous (here we would need to consider boards of larger than linear width). As rectangles are a special case of polyominoes, all these reductions hold also for arbitrary polyominoes of size $O(n)$.

4 A near optimal data structure

The previous section shows that if we want to follow the greedy strategy, then we cannot always make a move very quickly. While this at least gives us a pretty good excuse for our slow plays, let us now focus on the positive side and prove that we can match this lower bound up to an $n^{o(1)}$ factor.

► **Definition 14.** *Given a finite set S of axis-aligned rectangles in the plane lying on or above x -axis, the skyline of S is the union of the top, left and right boundaries of the region B obtained as follows:*

- extend the bottom of every rectangle in S so that its bottom lies on the x -axis;
- let B be the union of all these newly-obtained rectangles.

The left staircase of S consists of the parts of the skyline of S visible from the point at $(-\infty, 0)$. Similarly, the right staircase of S consists of the parts of the skyline of S visible from the point at $(+\infty, 0)$.

Notice that starting with a set of n independent axis-aligned rectangles, its skyline is a set of curves consisting of at most $4n - 1$ axis-aligned segments, while the left and right staircases each consist of at most n vertical segments. Moreover, the skyline contains all the information needed to know how far we can drop a rectangle of a certain width.

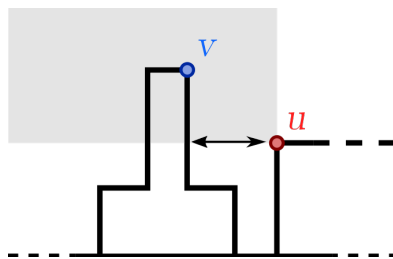
To get an RDDS with $n^{1/2}$ update and query time, up to a polylogarithmic factor, we will maintain the skyline, left and right staircases of our set of rectangles, broken into at most $O(n^{1/2})$ chunks each consisting of a sequence of roughly $n^{1/2}$ segments appearing consecutively on the skyline.

To start, it is an easy exercise to show the following via a simple plane sweep.

► **Lemma 15.** *Given a set of n independent axis-aligned rectangles, one can construct its skyline, left and right staircase in $O(n \log n)$ time (where we store the segments appearing in each in order of appearance left to right in three distinct arrays).*

Now, instead of looking for how low we can drop a rectangle of a given width, let us instead turn this question on its head and ask how wide of a rectangle we can drop if we want it to lie at or below a certain height h . If this is impossible, let's say this maximum width is

13:10 How Fast Can We Play Tetris Greedily with Rectangular Pieces?



■ **Figure 4** The width of the largest rectangle whose right border touches u is the horizontal distance from u to the horizontally nearest vertex in the shaded region (which in this case is v).

0. Remember that we are still constrained by two vertical lines between which our rectangle needs to lie (otherwise, we could have our rectangle be as wide as we wish and simply drop it next to the existing rectangles). Two special cases arise: when the widest rectangle which can be dropped to height h or below touches the left vertical line or when this rectangle touches the right vertical line. The maximum width of a rectangle corresponding to these cases can be determined in $O(\log n)$ time given the left and right staircase of the existing rectangles, simply by binary searching through both and finding where they meet the horizontal line $y = h$. What about the general case? Here we have the following.

► **Lemma 16.** *Given the skyline of a set of n rectangles, we can construct a data structure in $O(n \log n)$ with which given some height h , we can return the width and position of the widest rectangle which can be dropped at or below height h in $O(\log n)$ time.*

Proof. We will precompute the answer for all heights h which correspond to some vertex in the skyline.

We start by computing for every height h , the widest rectangle which can be dropped so that its right-side touches a vertex at height h on the skyline. We do this by traversing the skyline from left to right and maintaining the answer for the visited vertices in some data-structure allowing for $O(\log n)$ lookup and updates (some flavor of self-balancing binary search tree for example). When we reach some vertex u at height h , we want to quickly determine how wide we can make a rectangle whose lower-right corner touches u without it intersecting the interior of the skyline. Notice that this is the difference in x -coordinates between u and the horizontally nearest neighbor to the left of u which is strictly above u (see Figure 4). If we store the previously visited vertices in a self-balancing binary search tree sorted by y -coordinate and store in every node of that tree the rightmost vertex in the corresponding subtree, we can get this information in $O(\log n)$ time. If this width is larger than the current largest width stored for height h , update it. Overall this procedure can be carried out in $O(n \log n)$ time.

Similarly by traversing the skyline from right to left we compute for every height h , the widest rectangle which can be dropped so that its left-side touches a vertex at height h on the skyline.

Then, for every height h corresponding to some vertex, the widest rectangle that can drop at or below height h is the maximum width of a rectangle touching a vertex at height h' for all $h' \leq h$. We can compute this for all relevant h , as well as the corresponding location where a rectangle should be dropped, in linear time given the information we have already computed. We store this information in a binary search tree ordered by h . Now given any height (not necessarily corresponding to a vertex of the skyline), we can determine in $O(\log n)$ how wide a rectangle can be dropped at or below that height (and where to drop it) by looking in the binary search tree for the largest stored height no larger than h . ◀

We are now ready to break the skyline into chunks to make a data structure which supports insertions.

► **Theorem 17.** *There is a data structure which maintains a set of $O(n)$ independent axis-aligned rectangles in the plane, lying on or above the x and between the vertical lines $x = 0$ and $x = w$, with $O(n \log n)$ construction time and which allows the following operations:*

- (Update) *Given an axis-aligned rectangle R and a number x_d , drop R at x -coordinate x_d (here we assume that R and x_d are such that R will lie between the lines $x = 0$ and $x = w$).*
- (Query) *Given a height h return the widest rectangle which can be dropped to lie at height h , together with the position at which it needs to be dropped.*

Updates can be supported in $O(n^{1/2} \log^{3/2} n)$ time and queries in $O(n^{1/2} \log^{1/2} n)$ time.

Proof. To build our data structure, we start by computing the skyline of the set of rectangles in $O(n \log n)$ time. We add to the skyline long vertical segments corresponding to the lines $x = 0$ and $x = w$. Then we break the skyline into $O((n/\log n)^{1/2})$ contiguous pieces, each consisting of a skyline having at most $2(n \log n)^{1/2}$ vertices. We also make sure that the number of vertices in any two consecutive chunks of skyline sums up to at least $(n \log n)^{1/2}$. For each of those, we build the data structure of Lemma 16, the left and right staircase and compute the maximum height of a vertex. We store these data structures in a pointer list L , in left to right order. This costs a total time of $O((n/\log n)^{1/2} \cdot (n \log n)^{1/2} \log n) = O(n \log n)$.

Now, given a query height h , we proceed as follows. We query each of the small data structures in L to find the maximum width of a rectangle which can be dropped at or below height h over all of them. This costs $O((n/\log n)^{1/2} \log n) = O((n \log n)^{1/2})$ time. We are left to deal with rectangles which could span across multiple of these skyline chunks. Here we traverse L with two pointers p_1 and p_2 in a “sliding window” fashion:

- We start with p_1 at the head of the list and p_2 its successor. While both have not reached the end of the list, we go through the following steps.
- If the maximum height of a vertex in the skyline chunk corresponding to p_2 is at or below h and p_2 has not yet reached the tail of the list, we move p_2 to its successor in L .
- Otherwise, we find the widest rectangle which can be dropped at or below height h and overlaps both skylines corresponding to p_1 and p_2 partially by binary searching through the right staircase corresponding to p_1 and the left staircase of p_2 . Once this is done, we move p_1 over to p_2 and move p_2 to its successor (if p_2 has not yet reached the tail of the list).

By keeping track of the maximum width and the position it was encountered at throughout this procedure, they can be reported in $O((n/\log n)^{1/2} \log n) = O(n^{1/2} \log^{1/2} n)$ time.

Now, given a rectangle R to drop at a given position, we first find all chunks of skyline in L which will be completely covered by R as well as the pointers p_1 and p_2 to the elements in L corresponding to the skylines which overlap with the left and right border of R . By binary searching through the corresponding staircases and finding the maximum height of a vertex in the chunks which are covered by R we can find out at which height it will end up. Notice that the skylines which are covered by R are no longer relevant at this point, so we can ignore the corresponding data structures in L by moving the successor pointer of p_1 so that it points to p_2 . We now rebuild the data structure p_1 points to, including the rectangle R , as well as the one p_2 points to, excluding all parts of the skyline covered by R . If this causes any of the two skylines to have more than $2(n \log n)^{1/2}$ vertices, we can simply divide it in approximate halves and rebuild the two corresponding data structures from scratch in $O((n \log n)^{1/2} \cdot \log n)$ time each. If two consecutive skylines have fewer than $(n \log n)^{1/2}$ vertices together, merge

13:12 How Fast Can We Play Tetris Greedily with Rectangular Pieces?

them and rebuild the corresponding data structure from scratch in $O((n \log n)^{1/2} \cdot \log n)$ time. These divisions and merges can happen at most twice for a given update, and ensure that we always have $O((n/\log n)^{1/2})$ small data structures storing $O((n \log n)^{1/2})$ vertices each. The total cost of an update is then $O((n/\log n)^{1/2} + (n \log n)^{1/2} \cdot \log n) = O(n^{1/2} \log^{3/2} n)$. ◀

Given such a data structure, we can solve our original question. We keep track of all the $O(n)$ possible heights a rectangle could be dropped at, and when given a rectangle of a certain width, we do a binary search through all possible heights to find the lowest height which will accommodate a rectangle of that width. This increases the query time by a factor of $O(\log n)$, giving the following:

► **Theorem 18.** *There is an RDDS with $O(n \log n)$ construction time which supports queries and updates in $O(n^{1/2} \log^{3/2} n)$ time.*

This result applies even in the case where the RDDS is not necessarily initialized with an empty set of rectangles. Considering the lower bound from the OMv conjecture given in the previous section, this is likely close to optimal (although we have not made a big effort to optimize the polylogarithmic terms in the runtime). If one were to face an opponent playing the game with a sub-polynomial speed advantage over this data structure, we venture to guess this could be mitigated by an appropriate amount of distraction and foul play.

References

- 1 Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 477–486. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.58.
- 2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.
- 3 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018. doi:10.1137/15M1050987.
- 4 Simón Algorta and Özgür Simsek. The game of Tetris in machine learning. *CoRR*, abs/1905.01652, 2019. arXiv:1905.01652.
- 5 Josh Alman, Matthias Mních, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 41:1–41:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.41.
- 6 Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 114–125. Springer, 2014. doi:10.1007/978-3-662-43948-7_10.
- 7 Amihood Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B. Riva Shalom. Mind the gap! - online dictionary matching with one gap. *Algorithmica*, 81(6):2123–2157, 2019. doi:10.1007/s00453-018-0526-2.
- 8 Diana Sofía Lora Ariza. El clustering de jugadores de Tetris. In David Camacho, Marco Antonio Gómez-Martín, and Pedro Antonio González-Calero, editors, *Proceedings 2st Congreso de la Sociedad Española para las Ciencias del Videojuego, Barcelona, Spain, June 24, 2015*,

- volume 1394 of *CEUR Workshop Proceedings*, pages 36–45. CEUR-WS.org, 2015. URL: http://ceur-ws.org/Vol-1394/paper_4.pdf.
- 9 Diana Sofía Lora Ariza, Antonio A. Sánchez-Ruiz, and Pedro A. González-Calero. Time series and case-based reasoning for an intelligent Tetris game. In David W. Aha and Jean Lieber, editors, *Case-Based Reasoning Research and Development - 25th International Conference, ICCBR 2017, Trondheim, Norway, June 26-28, 2017, Proceedings*, volume 10339 of *Lecture Notes in Computer Science*, pages 185–199. Springer, 2017. doi:10.1007/978-3-319-61030-6_13.
 - 10 Diana Sofía Lora Ariza, Antonio A. Sánchez-Ruiz, and Pedro A. González-Calero. Towards finding flow in Tetris. In Kerstin Bach and Cindy Marling, editors, *Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings*, volume 11680 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2019. doi:10.1007/978-3-030-29249-2_18.
 - 11 Sualeh Asif, Michael J. Coulombe, Erik D. Demaine, Martin L. Demaine, Adam Hesterberg, Jayson Lynch, and Mihir Singhal. Tetris is np-hard even with $O(1)$ rows or columns. *J. Inf. Process.*, 28:942–958, 2020. doi:10.2197/ipsjip.28.942.
 - 12 Sualeh Asif, Michael J. Coulombe, Erik D. Demaine, Martin L. Demaine, Adam Hesterberg, Jayson Lynch, and Mihir Singhal. Tetris is np-hard even with $O(1)$ rows or columns. *CoRR*, abs/2009.14336, 2020. arXiv:2009.14336.
 - 13 Davide Baccherini and Donatella Merlini. Combinatorial analysis of Tetris-like games. *Discret. Math.*, 308(18):4165–4176, 2008. doi:10.1016/j.disc.2007.08.009.
 - 14 Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3sum. *Algorithmica*, 50(4):584–596, 2008. doi:10.1007/s00453-007-9036-3.
 - 15 Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS in undirected graphs: breaking the $o(m)$ barrier. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 730–739. SIAM, 2016. doi:10.1137/1.9781611974331.ch52.
 - 16 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 303–318. ACM, 2017. doi:10.1145/3034786.3034789.
 - 17 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs under updates and in the presence of integrity constraints. In Benny Kimelfeld and Yael Amsterdamer, editors, *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, volume 98 of *LIPICs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICDT.2018.8.
 - 18 Jacquelyn H. Berry and Wayne D. Gray. HOT: higher order Tetris, experts’ subgoals and activities. In Ashok K. Goel, Colleen M. Seifert, and Christian Freksa, editors, *Proceedings of the 41th Annual Meeting of the Cognitive Science Society, CogSci 2019: Creativity + Cognition + Computation, Montreal, Canada, July 24-27, 2019*, page 3409. cognitivesciencesociety.org, 2019. URL: <https://mindmodeling.org/cogsci2019/papers/0717/index.html>.
 - 19 Ron Breukelaar, Erik D. Demaine, Susan Hohenberger, Hendrik Jan Hoogetboom, Walter A. Kosters, and David Liben-Nowell. Tetris is hard, even to approximate. *Int. J. Comput. Geom. Appl.*, 14(1-2):41–68, 2004. doi:10.1142/S0218195904001354.
 - 20 Karl Bringmann. Fine-grained complexity theory (tutorial). In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPICs*, pages 4:1–4:7. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.STACS.2019.4.
 - 21 Timothy M. Chan. Semi-online maintenance of geometric optima and measures. *SIAM J. Comput.*, 32(3):700–716, 2003. doi:10.1137/S0097539702404389.

13:14 How Fast Can We Play Tetris Greedily with Rectangular Pieces?

- 22 Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. *Discret. Comput. Geom.*, 64(4):1235–1252, 2020. doi:10.1007/s00454-020-00229-5.
- 23 Timothy M. Chan. More logarithmic-factor speedups for 3sum, (median, +)-convolution, and some geometric 3sum-hard problems. *ACM Trans. Algorithms*, 16(1):7:1–7:23, 2020. doi:10.1145/3363541.
- 24 Lijie Chen, Erik D. Demaine, Yuzhou Gu, Virginia Vassilevska Williams, Yinzhan Xu, and Yuancheng Yu. Nearly optimal separation between partially and fully retroactive data structures. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018, June 18-20, 2018, Malmö, Sweden*, volume 101 of *LIPICs*, pages 33:1–33:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.SWAT.2018.33.
- 25 Tsung Teng Chen. Chitetris: A Chinese proverb learning game utilizing Tetris game plot. In Gautam Biswas, Diane Carr, Yam San Chee, and Wu-Yuin Hwang, editors, *2010 Third IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning, DIGITEL 2010, Kaohsiung, Taiwan, April 12-16, 2010*, pages 194–197. IEEE, 2010. doi:10.1109/DIGITEL.2010.34.
- 26 Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.
- 27 Stephen J. Cowley and Karl F. MacDorman. What baboons, babies and Tetris players tell us about interaction: a biosocial view of norm-based social learning. *Connect. Sci.*, 18(4):363–378, 2006. doi:10.1080/09540090600879703.
- 28 Antonios N. Dadaliaris, Evangelia Nerantzaki, Maria G. Koziri, Panagiotis Oikonomou, Thanasis Loukopoulos, and Georgios I. Stamoulis. Performance evaluation of Tetris-based legalization heuristics. In *Proceedings of the 20th Pan-Hellenic Conference on Informatics, Patras, Greece, November 10-12, 2016*, page 60. ACM, 2016. doi:10.1145/3003733.3003778.
- 29 Antonios N. Dadaliaris, Evangelia Nerantzaki, Panagiotis Oikonomou, Yannis Hatzaras, Angeliki-Olympia Troumpoulou, Ioannis Arvanitakis, and Georgios I. Stamoulis. Enhanced Tetris legalization. In *Proceedings of the SouthEast European Design Automation, Computer Engineering, Computer Networks and Social Media Conference, SEEDA-CECNM 2016, Kastoria, Greece, September 25-27, 2016*, pages 32–35. ACM, 2016. doi:10.1145/2984393.2984410.
- 30 Antonios N. Dadaliaris, Panagiotis Oikonomou, Maria G. Koziri, Evangelia Nerantzaki, Yannis Hatzaras, Dimitrios Garyfallou, Thanasis Loukopoulos, and Georgios I. Stamoulis. Heuristics to augment the performance of Tetris legalization: Making a fast but inferior method competitive. *J. Low Power Electron.*, 13(2):220–230, 2017. doi:10.1166/jolpe.2017.1483.
- 31 Søren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 48:1–48:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.48.
- 32 Justin Dallant and John Iacono. Conditional lower bounds for dynamic geometric measure problems, 2021. arXiv:2112.10095.
- 33 Guylain Delmas, Ronan Champagnat, and Michel Augeraud. Plot control for emergent narrative: A case study on Tetris. *Int. J. Intell. Games Simul.*, 5(1), 2008. URL: http://www.scit.wlv.ac.uk/OJS_IJIGS/index.php/IJIGS/article/view/56.
- 34 Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Adam Hesterberg, Andrea Lincoln, Jayson Lynch, and Yun William Yu. Total Tetris: Tetris with monominoes, dominoes, trominoes, pentominoes, . *J. Inf. Process.*, 25:515–527, 2017. doi:10.2197/ipsjjip.25.515.
- 35 Erik D. Demaine, Susan Hohenberger, and David Liben-Nowell. Tetris is hard, even to approximate. *CoRR*, cs.CC/0210020, 2002. URL: <https://arxiv.org/abs/cs/0210020>.

- 36 Erik D. Demaine, Susan Hohenberger, and David Liben-Nowell. Tetris is hard, even to approximate. In Tandy J. Warnow and Binhai Zhu, editors, *Computing and Combinatorics, 9th Annual International Conference, COCOON 2003, Big Sky, MT, USA, July 25-28, 2003, Proceedings*, volume 2697 of *Lecture Notes in Computer Science*, pages 351–363. Springer, 2003. doi:10.1007/3-540-45071-8_36.
- 37 Giorgio Fasano. A MIP approach for some practical packing problems: Balancing constraints and tetris-like items. *4OR*, 2(2):161–174, 2004. doi:10.1007/s10288-004-0037-7.
- 38 Enrico Formenti and Paolo Massazza. From Tetris to polyominoes generation. *Electron. Notes Discret. Math.*, 59:79–98, 2017. doi:10.1016/j.endm.2017.05.007.
- 39 Anka Gajentaan and Mark H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2.
- 40 Jacob E. Goodman and Joseph O’Rourke, editors. *Handbook of Discrete and Computational Geometry, Second Edition*. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035315.
- 41 Wayne D. Gray, Ray S. Perez, John K. Lindstedt, Anna Skinner, Robin Johnson, Richard E. Mayer, Deanne Adams, and Robert K. Atkinson. Tetris as research paradigm: An approach to studying complex cognitive skills. In Paul Bello, Marcello Guarini, Marjorie McShane, and Brian Scassellati, editors, *Proceedings of the 36th Annual Meeting of the Cognitive Science Society, CogSci 2014, Quebec City, Canada, July 23-26, 2014*. cognitivesciencesociety.org, 2014. URL: <https://mindmodeling.org/cogsci2014/papers/019/>.
- 42 Allan Grönlund and Seth Pettie. Threesomes, degenerates, and love triangles. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 621–630. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.72.
- 43 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.
- 44 Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional hardness for sensitivity problems. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPICs*, pages 26:1–26:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ITCS.2017.26.
- 45 Hendrik Jan Hoogeboom and Walter A. Kosters. How to construct Tetris configurations. *Int. J. Intell. Games Simul.*, 3(2):97–105, 2004. URL: http://www.scit.wlv.ac.uk/OJS_IJIGS/index.php/IJIGS/article/view/95.
- 46 Hendrik Jan Hoogeboom and Walter A. Kosters. Tetris and decidability. *Inf. Process. Lett.*, 89(6):267–272, 2004. doi:10.1016/j.ipl.2003.12.006.
- 47 Aline Hufschmitt, Stéphane Cardon, and Éric Jacopin. Can musical tempo makes Tetris game harder? In *IEEE Conference on Games, CoG 2020, Osaka, Japan, August 24-27, 2020*, pages 608–611. IEEE, 2020. doi:10.1109/CoG47356.2020.9231593.
- 48 Aline Hufschmitt, Stéphane Cardon, and Éric Jacopin. Manipulating player performance via music tempo in Tetris. In Pejman Mirza-Babaei, Victoria McArthur, Vero Vanden Abeele, and Max Birk, editors, *CHI PLAY ’20: The Annual Symposium on Computer-Human Interaction in Play, Virtual Event, Canada, November 2-4, 2020 - Companion Volume / Extended Abstracts*, pages 146–152. ACM, 2020. doi:10.1145/3383668.3419934.
- 49 Aline Hufschmitt, Stéphane Cardon, and Éric Jacopin. Dynamic manipulation of player performance with music tempo in Tetris. In Tracy Hammond, Katrien Verbert, Dennis Parra, Bart P. Knijnenburg, John O’Donovan, and Paul Teale, editors, *IUI ’21: 26th International Conference on Intelligent User Interfaces, College Station, TX, USA, April 13-17, 2021*, pages 290–296. ACM, 2021. doi:10.1145/3397481.3450684.

13:16 How Fast Can We Play Tetris Greedily with Rectangular Pieces?

- 50 Peter C. Jentsch and Chrystopher L. Nehaniv. Exploring Tetris as a transformation semigroup. *CoRR*, abs/2004.09022, 2020. [arXiv:2004.09022](https://arxiv.org/abs/2004.09022).
- 51 Patrick Jermann, Marc-Antoine Nüssli, and Weifeng Li. Using dual eye-tracking to unveil coordination and expertise in collaborative Tetris. In Tom McEwan and Lachlan McKinnon, editors, *Proceedings of the 2010 British Computer Society Conference on Human-Computer Interaction, BCS-HCI 2010, Dundee, United Kingdom, 6-10 September 2010*, pages 36–44. ACM, 2010. URL: <http://dl.acm.org/citation.cfm?id=2146309>.
- 52 Will Jordan. Morphology of the tetromino-stacking game: The design evolution of Tetris [abstract]. In Tanya Krzywinska, Helen W. Kennedy, and Barry Atkins, editors, *Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory, DiGRA 2009, London, UK, September 1-4, 2009*. Digital Games Research Association, 2009. URL: <http://www.digra.org/digital-library/publications/morphology-of-the-tetromino-stacking-game-the-design-evolution-of-tetris-abstract/>.
- 53 Adam Karczmarz and Jakub Lacki. Fast and simple connectivity in graph timelines. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, volume 9214 of *Lecture Notes in Computer Science*, pages 458–469. Springer, 2015. doi:10.1007/978-3-319-21840-3_38.
- 54 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 55 Stephen A. Keating, Wan-Chen Lee, Travis W. Windleharth, and Lee Jin Ha. The style of Tetris is...possibly Tetris?: Creative professionals’ description of video game visual styles. In Tung Bui, editor, *50th Hawaii International Conference on System Sciences, HICSS 2017, Hilton Waikoloa Village, Hawaii, USA, January 4-7, 2017*, pages 1–10. ScholarSpace / AIS Electronic Library (AISeL), 2017. URL: <http://hdl.handle.net/10125/41402>.
- 56 Sirisilp Kongsilp, Hathaichanok Chawmungkrung, and Takashi Komuro. Pop-out Tetris: An implementation of a tablet FTVR game. In *15th International Joint Conference on Computer Science and Software Engineering, JCSSE 2018, Nakhonpathom, Thailand, July 11-13, 2018*, pages 1–5. IEEE, 2018. doi:10.1109/JCSSE.2018.8457377.
- 57 Tsvi Kopelowitz and Robert Krauthgamer. Color-distance oracles and snippets. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, volume 54 of *LIPICs*, pages 24:1–24:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.CPM.2016.24.
- 58 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1272–1287. SIAM, 2016. doi:10.1137/1.9781611974331.ch89.
- 59 Laurens R. Krol, Sarah-Christin Freytag, and Thorsten O. Zander. Meyendtris: a hands-free, multimodal tetris clone using eye tracking and passive BCI for intuitive neuroadaptive gaming. In Edward Lank, Alessandro Vinciarelli, Eve E. Hoggan, Sriram Subramanian, and Stephen A. Brewster, editors, *Proceedings of the 19th ACM International Conference on Multimodal Interaction, ICMI 2017, Glasgow, United Kingdom, November 13 - 17, 2017*, pages 433–437. ACM, 2017. doi:10.1145/3136755.3136805.
- 60 Ian J. Lewis and Sebastian L. Beswick. Generalisation over details: The unsuitability of supervised backpropagation networks for Tetris. *Adv. Artif. Neural Syst.*, 2015:157983:1–157983:8, 2015. doi:10.1155/2015/157983.
- 61 Weifeng Li, Marc-Antoine Nüssli, and Patrick Jermann. Gaze quality assisted automatic recognition of social contexts in collaborative Tetris. In Wen Gao, Chin-Hui Lee, Jie Yang, Xilin

- Chen, Maxine Eskénazi, and Zhengyou Zhang, editors, *Proceedings of the 12th International Conference on Multimodal Interfaces / 7. International Workshop on Machine Learning for Multimodal Interaction, ICMI-MLMI 2010, Beijing, China, November 8-12, 2010*, pages 8:1–8:8. ACM, 2010. doi:10.1145/1891903.1891914.
- 62 Weifeng Li, Marc-Antoine Nüssli, and Patrick Jermann. Exploring personal aspects using eye-tracking modality in Tetris-playing. In *IEEE 13th International Workshop on Multimedia Signal Processing (MMSP 2011), Hangzhou, China, October 17-19, 2011*, pages 1–4. IEEE, 2011. doi:10.1109/MMSP.2011.6093841.
- 63 John K. Lindstedt and Wayne D. Gray. Extreme expertise: Exploring expert behavior in Tetris. In Markus Knauff, Michael Pauen, Natalie Sebanz, and Ipke Wachsmuth, editors, *Proceedings of the 35th Annual Meeting of the Cognitive Science Society, CogSci 2013, Berlin, Germany, July 31 - August 3, 2013*. cognitivesciencesociety.org, 2013. URL: <https://mindmodeling.org/cogsci2013/papers/0183/index.html>.
- 64 John K. Lindstedt and Wayne D. Gray. The "cognitive speed-bump": How world champion Tetris players trade milliseconds for seconds. In Stephanie Denison, Michael Mack, Yang Xu, and Blair C. Armstrong, editors, *Proceedings of the 42th Annual Meeting of the Cognitive Science Society - Developing a Mind: Learning in Humans, Animals, and Machines, CogSci 2020, virtual, July 29 - August 1, 2020*. cognitivesciencesociety.org, 2020. URL: <https://cogsci.mindmodeling.org/2020/papers/0020/index.html>.
- 65 Diana Lora, Antonio A. Sánchez-Ruiz, and Pedro A. González-Calero. Difficulty adjustment in Tetris with time series. In David Camacho, Marco Antonio Gómez-Martín, and Pedro Antonio González-Calero, editors, *Proceedings of the 3rd Congreso de la Sociedad Española para las Ciencias del Videjuego, Barcelona, Spain, June 29, 2016*, volume 1682 of *CEUR Workshop Proceedings*, pages 89–100. CEUR-WS.org, 2016. URL: http://ceur-ws.org/Vol-1682/CoSeCiVi16_paper_10.pdf.
- 66 Diana Lora, Antonio A. Sánchez-Ruiz, Pedro A. González-Calero, and Marco Antonio Gómez-Martín. Dynamic difficulty adjustment in Tetris. In Zdravko Markov and Ingrid Russell, editors, *Proceedings of the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2016, Key Largo, Florida, USA, May 16-18, 2016*, pages 335–339. AAAI Press, 2016. URL: <http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS16/paper/view/12824>.
- 67 Danica Mast and Sanne de Vries. Cooperative Tetris: The influence of social exertion gaming on game experience and social presence. In Ronald Poppe, John-Jules Ch. Meyer, Remco C. Veltkamp, and Mehdi Dastani, editors, *Intelligent Technologies for Interactive Entertainment - 8th International Conference, INTETAIN 2016, Utrecht, The Netherlands, June 28-30, 2016, Revised Selected Papers*, volume 178 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 115–123. Springer, 2016. doi:10.1007/978-3-319-49616-0_11.
- 68 Taishi Oikawa, Chu-Hsuan Hsueh, and Kokolo Ikeda. Improving human players' t-spin skills in Tetris with procedural problem generation. In Tristan Cazenave, H. Jaap van den Herik, Abdallah Saffidine, and I-Chen Wu, editors, *Advances in Computer Games - 16th International Conference, ACG 2019, Macao, China, August 11-13, 2019, Revised Selected Papers*, volume 12516 of *Lecture Notes in Computer Science*, pages 41–52. Springer, 2019. doi:10.1007/978-3-030-65883-0_4.
- 69 Panagiotis Oikonomou, Antonios N. Dadaliaris, Thanasis Loukopoulos, Athanasios Kakarountas, and Georgios I. Stamoulis. A Tetris-based legalization heuristic for standard cell placement with obstacles. In *7th International Conference on Modern Circuits and Systems Technologies, MOCAS 2018, Thessaloniki, Greece, May 7-9, 2018*, pages 1–4. IEEE, 2018. doi:10.1109/MOCAS.2018.8376559.
- 70 Zhan-He Ou and Ling-Hwei Chen. Hiding data in Tetris. In *International Conference on Machine Learning and Cybernetics, ICMLC 2011, Guilin, China, July 10-13, 2011, Proceedings*, pages 61–67. IEEE, 2011. doi:10.1109/ICMLC.2011.6016684.

13:18 How Fast Can We Play Tetris Greedily with Rectangular Pieces?


- 71 Zhan-He Ou and Ling-Hwei Chen. A steganographic method based on tetris games. *Inf. Sci.*, 276:343–353, 2014. doi:10.1016/j.ins.2013.12.024.
- 72 Jing Pang, Manuel Pravin, and Robert Tanihaha. Microblaze soft core based FPGA embedded system design of Tetris game. In Hamid R. Arabnia and Ashu M. G. Solo, editors, *Proceedings of the 2010 International Conference on Embedded Systems & Applications, ESA 2010, July 12-15, 2010, Las Vegas Nevada, USA*, pages 79–85. CSREA Press, 2010.
- 73 Nick Parlante. Nifty assignments: tetris on the brain. *ACM SIGCSE Bull.*, 33(4):25–27, 2001. doi:10.1145/572139.572162.
- 74 Georgios Patsis, Hichem Sahli, Werner Verhelst, and Olga De Troyer. Evaluation of attention levels in a Tetris game using a brain computer interface. In Sandra Carberry, Stephan Weibelzahl, Alessandro Micarelli, and Giovanni Semeraro, editors, *User Modeling, Adaptation, and Personalization - 21th International Conference, UMAP 2013, Rome, Italy, June 10-14, 2013, Proceedings*, volume 7899 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2013. doi:10.1007/978-3-642-38844-6_11.
- 75 Colin Pinnell. Computer games for learning: An evidence-based approach (author: Richard e. mayer). *J. Educ. Technol. Soc.*, 18(4):523–524, 2015. URL: https://www.j-ets.net/ETS/journals/18_4/40.pdf.
- 76 Gabriel Pires, Mario Torres, Nuno Casaleiro, Urbano Nunes, and Miguel Castelo-Branco. Playing Tetris with non-invasive BCI. In *2011 IEEE 1st International Conference on Serious Games and Applications for Health, SeGAH 2011, Braga, Portugal, November 16-18, 2011*, pages 1–6. IEEE Computer Society, 2011. doi:10.1109/SeGAH.2011.6165454.
- 77 Maximilian Probst. On the complexity of the (approximate) nearest colored node problem. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPICs*, pages 68:1–68:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ESA.2018.68.
- 78 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing, STOC '10*, pages 603–610, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1806689.1806772.
- 79 Jana Rambusch, Tarja Susi, Stefan Ekman, and Ulf Wilhelmsson. A literary excursion into the hidden (fan) fictional worlds of Tetris, starcraft, and dreamfall. In Tanya Krzywinska, Helen W. Kennedy, and Barry Atkins, editors, *Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory, DiGRA 2009, London, UK, September 1-4, 2009*. Digital Games Research Association, 2009. URL: <http://www.digra.org/digital-library/publications/a-literary-excursion-into-the-hidden-fan-fictional-worlds-of-tetris-starcraft-and-dreamfall/>.
- 80 Catherine Sibert and Wayne D. Gray. When experts err: Using Tetris models to detect true errors from deliberate sub-optimal choices. In Ashok K. Goel, Colleen M. Seifert, and Christian Freksa, editors, *Proceedings of the 41th Annual Meeting of the Cognitive Science Society, CogSci 2019: Creativity + Cognition + Computation, Montreal, Canada, July 24-27, 2019*, page 3572. cognitivesciencesociety.org, 2019. URL: <https://mindmodeling.org/cogsci2019/papers/0880/index.html>.
- 81 Catherine Sibert, Wayne D. Gray, and John K. Lindstedt. Tetris-: Exploring human performance via cross entropy reinforcement learning models. In David C. Noelle, Rick Dale, Anne S. Warlaumont, Jeff Yoshimi, Teenie Matlock, Carolyn D. Jennings, and Paul P. Maglio, editors, *Proceedings of the 37th Annual Meeting of the Cognitive Science Society, CogSci 2015, Pasadena, California, USA, July 22-25, 2015*. cognitivesciencesociety.org, 2015. URL: <https://mindmodeling.org/cogsci2015/papers/0377/index.html>.
- 82 Catherine Sibert, John K. Lindstedt, and Wayne D. Gray. Tetris: Exploring human strategies via cross entropy reinforcement learning models. In Paul Bello, Marcello Guarini, Marjorie McShane, and Brian Scassellati, editors, *Proceedings of the 36th Annual Meeting of the Cognitive*

- Science Society, CogSci 2014, Quebec City, Canada, July 23-26, 2014*. *cognitivesciencesociety.org*, 2014. URL: <https://mindmodeling.org/cogsci2014/papers/743/>.
- 83 Özgür Simsek, Simón Algorta, and Amit Kothiyal. Why most decisions are easy in Tetris - and perhaps in other sequential decision problems, as well. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1757–1765. JMLR.org, 2016. URL: <http://proceedings.mlr.press/v48/simsek16.html>.
- 84 Katta Spiel, Sven Bertel, and Fares Kayali. "Not another Z piece!": Adaptive difficulty in TETRIS. In Gloria Mark, Susan R. Fussell, Cliff Lampe, M. C. Schraefel, Juan Pablo Hourcade, Caroline Appert, and Daniel Wigdor, editors, *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, pages 5126–5131*. ACM, 2017. doi:10.1145/3025453.3025721.
- 85 Katta Spiel, Sven Bertel, and Fares Kayali. Adapting gameplay to eye movements - an exploration with TETRIS. In Joan Arnedo, Lennart E. Nacke, Vero Vanden Abeele, and Z O. Toups, editors, *Extended Abstracts of the Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts, Barcelona, Spain, October 22-25, 2019*, pages 687–695. ACM, 2019. doi:10.1145/3341215.3356267.
- 86 Guo-Dong Su, Chin-Chen Chang, Chia-Chen Lin, and Zhi-Qiang Yao. Secure high capacity tetris-based scheme for data hiding. *IET Image Process.*, 14(17):4633–4645, 2020. doi:10.1049/iet-ipr.2019.1694.
- 87 Guo-Dong Su, Chin-Chen Chang, Chia-Chen Lin, and Zhi-Qiang Yao. Erratum: Secure high capacity tetris-based scheme for data hiding. *IET Image Process.*, 15(12):3020, 2021. doi:10.1049/ipr2.12291.
- 88 Tadao Takaoka. The reverse problem of range query. *Electron. Notes Theor. Comput. Sci.*, 78:281–292, 2003. doi:10.1016/S1571-0661(04)81018-1.
- 89 Oscar Temprano. Complexity of a Tetris variant. *CoRR*, abs/1506.07204, 2015. arXiv:1506.07204.
- 90 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 456–480. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00036.
- 91 Vladislav Daniel Veksler and Wayne D. Gray. State definition in the Tetris task: Designing a hybrid model of cognition. In *Proceedings of the International Conference on Cognitive Modeling, ICCM 2004, Pittsburgh, Pennsylvania, USA, July 30 - August 1, 2004*, pages 394–395, 2004. URL: <http://www.lrdc.pitt.edu/schunn/ICCM2004/proceedings/abstracts/Vekser.pdf>.
- 92 Jianping Wang, Jun Chen, and Xiao-Min Li. Design a mobile Tetris game based on J2ME platform. In Chunfeng Liu, Jincai Chang, and Aimin Yang, editors, *Information Computing and Applications - Second International Conference, ICICA 2011, Qinhuangdao, China, October 28-31, 2011. Proceedings, Part II*, volume 244 of *Communications in Computer and Information Science*, pages 387–393. Springer, 2011. doi:10.1007/978-3-642-27452-7_53.
- 93 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.
- 94 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians (ICM 2018)*, pages 3447–3487. WORLD SCIENTIFIC, 2019. doi:10.1142/9789813272880_0188.
- 95 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and APSP. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 786–797. IEEE, 2020. doi:10.1109/FOCS46700.2020.00078.
- 96 Jiajun Xu and Sam Huang. Tetris. *CoRR*, abs/1811.08614, 2018. arXiv:1811.08614.




The Synchronization Game on Subclasses of Automata

Henning Fernau   

Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Carolina Haase 

Informatik, Hochschule Trier, Germany

Stefan Hoffmann   

Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Abstract

The notion of synchronization of finite automata is connected to one of the long-standing open problems in combinatorial automata theory, which is Černý’s Conjecture. In this paper, we focus on so-called synchronization games. We will discuss how to present synchronization questions in a playful way. This leads us to study related complexity questions on certain classes of finite automata. More precisely, we consider weakly acyclic, commutative and k -simple idempotent automata. We encounter a number of complexity classes, ranging from L up to PSPACE.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Complexity classes

Keywords and phrases Synchronization of finite automata, computational complexity

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.14

1 Introduction

“Gamification” is a catchword that describes, in a broader sense, how to teach or to explain possibly complicated concepts via games. In a narrower sense, we are discussing *educational games*. It is tempting to use this idea to explain ideas from combinatorics or complexity theory. For instance, one can play The Hamiltonian Circuit on one’s smartphone¹. As a predecessor, clearly Hamilton’s Icosian Game is to be mentioned². This has been done (not necessarily with didactical motivations) when combining ideas from game theory with graph-theoretic models of real-world phenomena, with *vertex cover games* [15] serving as one example. An alternative (but complementing) route is to analyze the combinatorial or computational complexity nature of games, which has been one of the driving themes of the FUN conference series; a certain overview on this ever-growing area can be found in [9, 16].

One drawback of many notions from combinatorics, in particular from graph theory, is that they have a rather static nature. This makes it rather difficult to define meaningful games based on these notions. The picture changes if dynamics enters the scene. One of the easiest ways to induce dynamics into graphs is when interpreting edge-labeled directed graphs as finite automata, because reading a word by a finite automaton infuses action into the otherwise static graph objects.

¹ See <https://apps.apple.com/us/app/the-hamiltonian-circuit/id1484345208>. In fact, there are also game variations of the Hamiltonian Circuit (“Maker-Breaker” games, see [24]) that are close to the ideas followed in this paper concerning combinatorial automata theory.

² See <https://www.puzzlemuseum.com/month/picm02/200207icosian.htm>, this has also interesting connections to quaternions, as explained in [3].



This then allows to explain important notions like synchronization, which is linked to an intriguing long-standing open combinatorial question in automata theory, namely, whether or not each synchronizable deterministic finite-state automaton possesses a so-called synchronizing word of a length quadratic in its number of states. This is (basically) also known as Černý’s Conjecture.³ A 2-person game called *synchronization game*, where Alice plays against Bob, was introduced in [13]. This game was indeed motivated by a gamification approach to software testing; see [4]. Fominikh, Martyugin and Volkov showed that whether or not Alice or Bob have a winning strategy is indeed a computationally hard question in general. In this paper, we look into this question, restricted to several classes of automata.⁴

Coming back to the idea of gamification, this game (and its analysis) can be seen as a playful doorway into a number of concepts that are prominent in Computer Science:

- graphs and automata are the most obvious concepts, as this is how a typical game would look like, although one could also think of designs that look more like classical board-games (see below) and hence obfuscate this possibly “too abstract” presentation by graphs;
- combinatorial questions and combinatorial thinking are at the heart of many mathematical reasonings in Computer Science and are also at the source of this game;
- computational complexity is another area that is touched and about which a player might get a feel, as the “game complexity” is somehow related to this concept.

The paper is structured as follows. In Section 2, we briefly describe the necessary background for the technical parts of the paper. We also shortly discuss the class of weakly acyclic finite automata as a simple example of a subclass of finite automata and also to illustrate the questions considered throughout the paper. In Section 3, we return to the question of board designs for automata games in general, which also motivates our major technical theme of this paper, which is to look at synchronization and synchronization games from the perspective of their complexity on special classes of automata. In Section 4, we give an overview over the situation in terms of complexity for general deterministic finite automata. Already here, a certain span of complexity classes from NL to PSPACE shows up. Then, in Section 5, we consider commutative automata. The span of complexity classes now contains L, P, NP, and Π_2^P . Again another class of automata is considered in Section 6: k -simple idempotent ones. Here, as a main result, we also show a full understanding about when Alice can win the synchronization game. In Section 7, we briefly discuss monotonic automata that are different from the previously considered classes in terms of complexity insofar as synchronizability is different from Alice having a winning strategy on a given automaton. In the last section, we discuss several other aspects, including the question of level design for synchronization games.

2 Preliminaries

An *alphabet* is a finite, non-empty set whose elements are called *letters*. As usual, Σ^* denotes the set of all words over the alphabet Σ , including the *empty word* ε . Let $u \in \Sigma^*$ be a word and $a \in \Sigma$. By $|u|_a$ we denote the number of times the symbol a occurs in u . Let

³ As it is often the case with famous mathematical conjectures, this question has quite some history. We are not diving into this here, but only mention that a few years ago, a special issue of a journal was dedicated to this question, see [38]; there, also English translations of the first papers of this topic can be found, which were written in Slovak and in German back in the 1960s.

⁴ It should be noted that two different games have been suggested in the literature in connection with synchronization: a 1-person game [2] and a stochastic 2-person game [20], but we only discuss the synchronization game introduced in [13] in this paper.

$w = w_1w_2 \cdots w_n$ be a word consisting of n subsequent letters. Then, n is also called the *length* of w . In this paper, a *deterministic finite automaton*, or DFA for short, is specified as $\mathcal{A} = (Q, \Sigma, \delta)$, where Σ is the input alphabet, Q is the state alphabet, and $\delta : Q \times \Sigma \rightarrow Q$ is the (total) transition function;⁵ δ is then extended to $\delta : Q \times \Sigma^* \rightarrow Q$ by $\delta(q, \varepsilon) = q$ and $\delta(q, ua) = \delta(\delta(q, u), a)$ for $a \in \Sigma$. Likewise, δ is also extended to $\delta : 2^Q \times \Sigma^* \rightarrow 2^Q$. A word $w \in \Sigma^*$ is called *synchronizing* for \mathcal{A} if there exists a *synchronizing state* q_{sync} so that, for each $q \in Q$, $\delta(q, w) = q_{\text{sync}}$. A DFA is called *synchronizing* if it admits a synchronizing word. A state σ is called a *sink state* if, for all input letters a , $\delta(\sigma, a) = \sigma$. Whether or not a DFA is synchronizing can be checked by considering a restricted variation of the power-set automaton. For a DFA $\mathcal{A} = (Q, \Sigma, \delta)$, its *pair automaton* is defined as $\mathcal{P}_2(\mathcal{A}) = (Q_{\mathcal{P}_2}, \Sigma, \delta_{\mathcal{P}_2})$ with state set $Q_{\mathcal{P}_2} = \{\{s, t\} \mid s, t \in Q \wedge s \neq t\} \cup \{\perp\} = \binom{Q}{2} \cup \{\perp\}$ and transition function

$$\delta_{\mathcal{P}_2}(\{s, t\}, a) = \begin{cases} \{\delta(s, a), \delta(t, a)\} & \text{if } \delta(s, a) \neq \delta(t, a) \\ \perp & \text{if } \delta(s, a) = \delta(t, a) \end{cases}$$

Moreover, $\delta_{\mathcal{P}_2}(\perp, a) = \perp$. In other words, \perp is a sink state that is entered via symbol a whenever the two states s, t of the original automaton are synchronized when reading a . Apart from \perp , a pair automaton can be viewed as a sub-automaton of the well-known powerset automaton. The mentioned check is based on the following result that is well-known in the area of synchronizing automata, compare the surveys [32, 37]:

► **Lemma 2.1.** $\mathcal{A} = (Q, \Sigma, \delta)$ is synchronizing if and only if for each unordered pair of states $\{s, t\} \in \binom{Q}{2}$, there exists a word $w \in \Sigma^*$ such that $\delta_{\mathcal{P}_2}(\{s, t\}, w) = \perp$.

States $\{s, t\} \in Q_{\mathcal{P}_2}$ with $\delta_{\mathcal{P}_2}(\{s, t\}, a) = \perp$ are therefore also called *synchronizing pairs*.

Next, we describe a related 2-persons game, the *synchronization game*. There are two players, Alice (Synchronizer) and Bob (Desynchronizer), whose moves alternate. They play on a DFA $\mathcal{A} = (Q, \Sigma, \delta)$. Alice who plays first wants to synchronize \mathcal{A} , while Bob aims to prevent synchronization or, if synchronization is unavoidable, to delay it as long as possible. More formally, at the beginning, all states are *active*, or, more pictorially, all states hold coins. Alice starts by playing the first letter w_1 . After her move, only the states in $Q_1 = \delta(Q, w_1)$ hold coins. Then, Bob plays the second letter w_2 , so that then, the states in $Q_2 = \delta(Q, w_1w_2) = \delta(Q_1, w_2)$ hold coins, etc. Alice wins if there is some t such that $|Q_t| = 1$. Then, $w_1w_2 \cdots w_t$ is a synchronizing word, and q_t with $\{q_t\} = Q_t$ is the corresponding synchronizing state. Provided that both players play optimally, the outcome of such a game depends only on \mathcal{A} . But who can enforce a win, and if so, how does a winning strategy look like? This game was introduced in [13], where also several complexity results have been established, see Section 4. The following result is of particular importance when analyzing winning strategies. It clearly bears some similarities with Lemma 2.1.

► **Lemma 2.2** ([13]). *Alice has a winning strategy in the synchronization game on a DFA iff she has a winning strategy in every position in which only two states of the DFA hold coins.*

Therefore, we could alternatively start a synchronization game by having Bob choose two states on which he places one coin each. Alice would win if she can synchronize these two states, although Bob will try to prevent this from happening. In particular, we could say that Bob has won when he manages to have two coins on the same two states twice when he

⁵ We do not need to speak about initial or final states in this paper. Sometimes, this variant of automata is also called semi-automata.

is about to move. In this paper, we are looking for further conditions under which one can tell if Alice or Bob will win the synchronization game. Therefore, we will look into special cases of the synchronization game that can be described by special classes of finite automata. Therefore, we are introducing DFAs with several special properties in the following.

Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a DFA. Extending the definition of simple idempotent letters given in [29], we call a letter $a \in \Sigma$ *k-simple idempotent* if $|\delta(Q, a)| = |Q| - k$ and $\delta(q, a) = q$ for all $q \in \delta(Q, a)$. We say that \mathcal{A} is a *cyclic automaton with a k-simple idempotent* over a binary alphabet Σ if there exists a *k-simple idempotent* letter $a \in \Sigma$ and $b \in \Sigma$ permutes all states cyclically. States are numbered in clockwise order in relation to b . We say that two states $q, p \in Q$ are neighbors if $\delta(q, b) = p$. The *b-distance* of two states q, p is the number k of b -transitions between them, that is $k = \min\{\ell \in \mathbb{N} \mid \delta(q, b^\ell) = p \vee \delta(p, b^\ell) = q\}$.

We call $\mathcal{A} = (Q, \Sigma, \delta)$ *connected* if the undirected simple graph $G = (V, E)$ with vertex set $V = Q$ and edge set $E = \{\{p, q\} \mid p \neq q \wedge \exists a \in \Sigma : \delta(p, a) = q \vee \delta(q, a) = p\}$ is connected (note that this is different from the way connectedness is defined for automata in [5]).

Let $\mathcal{A} = (Q, \Sigma, \delta)$ and $p, q \in Q$. We say that the state q is *reachable* from p , written $p \lesssim q$, if there exists a word $u \in \Sigma^*$ such that $q = \delta(p, u)$. Note that this relation yields a quasiorder on the states and it induces a partial order on the equivalence classes where two states $p, q \in Q$ are equivalent if $p \lesssim q$ and $q \lesssim p$. The equivalence classes are precisely the strongly connected components in the automaton graph. A DFA is called *weakly acyclic* if each \lesssim -equivalence class contains only one element. Hence, the quasi-order \lesssim is in fact a partial order. This is why these automata are sometimes called *partially ordered*. In general, a DFA with a sink state σ is synchronizing if and only if the sink state is reachable from each other state; in this case, the sink state is also the synchronizing state.

We consider weakly acyclic automata as a warm-up for this paper. Synchronization problems on this class of DFA have been investigated in [18, 30]. The following result shows that the question whether or not Alice has a winning strategy is particularly easy for this type of DFA. Compare this with Theorem 5.2, which gives the same for commutative automata; we will refer to this theorem here also in the proof, as this theorem serves rather as an appetizer for the technical parts of this paper.

► **Theorem 2.3.** *For a weakly acyclic automaton $\mathcal{A} = (Q, \Sigma, \delta)$, the following are equivalent:*

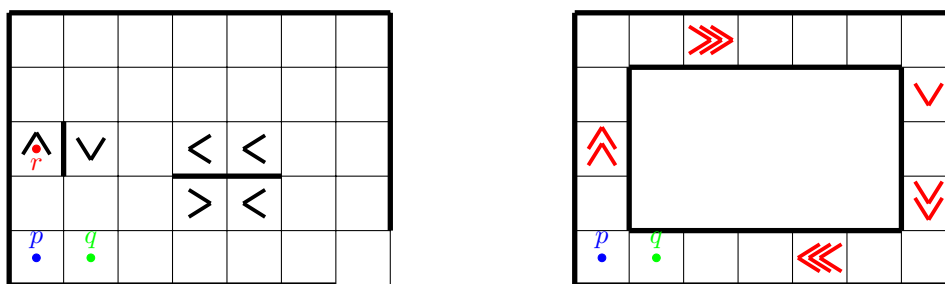
1. \mathcal{A} has a unique sink state,
2. \mathcal{A} is synchronizing,
3. Alice has a winning strategy on \mathcal{A} .

Proof. (1) implies (2): As every maximal state is a sink state, by uniqueness, there exists a unique maximal state (with respect to \lesssim). By definition of the reachability relation, there exists a word mapping every state to this unique maximal sink state. Then the automaton is synchronizing with the same argument as in the proof of Theorem 5.2.

(2) implies (3): Let $\mathcal{A} = (Q, \Sigma, \delta)$ be synchronizing and weakly acyclic. By definition, there is a partial ordering on the state set $Q = \{q_1, \dots, q_n\}$ such that $q_i \lesssim q_j$ implies $i \leq j$. Then, q_n is a sink state and hence the synchronizing state. Also, there is a letter a_i and some $j_i > i$ with $\delta(q_i, a_i) = q_{j_i}$ for $i = 1, \dots, n-1$. Therefore, Alice can win after she made at most $n-1$ moves, as she can shorten the distance to the sink state in each move.

(3) implies (1): If Alice has a winning strategy, there exists a synchronizing word u . Every maximal state q is a sink state, so there exists at least one. However, if \mathcal{A} is synchronizing, then only one sink state can exist, as one cannot reach a sink state from another one. ◀

Hence, checking if Alice has a winning strategy is the same as checking if the DFA is synchronizing, which is in NL, while both complexities differ in general, see Theorem 4.1.

(1) A board design with $\delta(p, e) = q$ and $\delta(q, w) = p$.

(2) A racing track design

■ **Figure 1** Designing boards for the synchronization game: two different proposals.

3 On Board Designs

How can we present a synchronization game in a compact and attractive fashion? This is the question we like to discuss in this section. Clearly, every finite automaton can be presented by a transition table or by its automaton graph. However, we claim that more traditional designs like those known from traditional board games like chess or like games with racing tracks that are also popular children’s games may be more appealing than the ones usually applied in automata theory. Notice that we cannot represent every automaton this way, but notice that the proposed board designs do relate to the classes of automata studied throughout this paper.

Referring to Example (1) in Figure 1, we can interpret such a board design as a finite automaton over the input alphabet $\{e, n, s, w\}$ (with the letters denoting the east, north, south and west directions of movement) as follows:

- Each cell represents a state of the automaton.
- The input letters let us move through the board as expected: e moves one step “east”, i.e., to the right, etc.
- If the move would hit a wall, indicated by a thicker drawn line, then the direction indicated by the arrow in the current state should be “executed.” Therefore, in our picture, $\delta(r, wsss) = p = \delta(r, esss) = \delta(r, nsss)$.
- If a wall is hit but no arrow is drawn, then the direction is inverted, i.e., if one bumps against the wall by moving east, a west move is executed, as $\delta(p, w) = q$ and $\delta(p, sn) = r$.⁶
- An exception is the cell in the right lower corner: here the open walls indicate that this is a “way out.” More formally, there is one more state to be reached this way (by moving either south or east), a sink state σ , which must be synchronizing. Hence, $\delta(q, eeeee) = \sigma$.
- Notice that apart from the exceptions that have been worked out above, the automaton behaves as a commutative automaton, e.g., $\delta(q, nw) = \delta(q, wn)$. This is also partially true for our rules at the walls, e.g., $\delta(p, se) = \delta(p, es) = \delta(q, n)$. However, $\delta(p, sn) = \delta(p, nn) \neq \delta(p, ns) = p$. But without the walls and in particular the “special walls” built in the middle of the board, the game might appear to be a bit boring.
- Also notice that board designs often (but not necessarily) lead to automata whose graphs are planar. Also such automata have been previously studied in our setting; see [25].

⁶ An alternative might be to get stuck when bumping against a wall; this would correspond to considering incomplete deterministic automata. Then, we arrive at different synchronization problems, and we like to avoid discussions in this direction in this paper.

14:6 The Synchronization Game on Subclasses of Automata

The goal of the game is as in the general case and can also be played with just two pieces (or coins) on the board. The two players move both of them at the same time according to one of the letters a, e, n, w . While player one will try to move both coins on the same cell of the board, player two will try to prevent this from happening. To make the game more interesting, it is suitable to introduce the additional rule that it is prohibited to enter the same configuration twice.

Such a board design is quite compact: The example above denotes a 36-state DFA, whose formal transition function is clearly less intuitive than this board presentation. Also when comparing it to a classical automaton graph representation, the suggested presentation has an edge, mainly because of the implicit transition arcs and implicit arc labels. Additionally, if one tries to draw an automaton graph as small as the board can be drawn without losing readability, the letters that need to be written on the arcs will be hard to decipher.

One can also use such a board to define a single-player game as follows. First, specify three positions p, q, r on the board (an example is shown in Figure 1 on the left side, but in general you can think of special tasks that a player draws from a pile of tasks at the beginning of the game), plus a number n . The question to solve is to find a sequence of movements, i.e., a word x of length at most n over the alphabet $\{e, n, s, w\}$, such that for the transition function δ that can be associated to the board, we find that $\delta(p, x) = \delta(q, x) = r$. With the positions shown in Figure 1, it is not possible to find any such word. This is due to the fact that one can show that, assuming that the two pieces stay on the board, the Manhattan distance between the two pieces on the board (ignoring walls) will always be an odd number. In general, this specific form of a synchronizability question could be interesting as a single-player game, although it is polynomial-time solvable (without the length restriction), mainly because of facts: (a) as discussed above, boards can be used as quite compact representations of DFA; (b) although words that synchronize two states are of quadratic size only, see Lemma 2.1, this might mean that the player might have to look for a synchronizing word of a length like 100 even for the small board displayed in Figure 1. This makes this question quite challenging for a human player.

A second example is shown in Figure 1 (2). Here, the underlying automaton is a cyclic one over a binary alphabet with a k -simple idempotent. The b -cycle of such an automaton is interpreted as kind of a racing track, where each state of the automaton corresponds to a square. There are two coins in the game. Players can only move both of them at the same time and in clockwise order. The goal for player one is for both coins to end up on the same square, while the opponent tries to prevent exactly that. Resembling a game of tag. In each round players can choose between either moving both coins forward by one or, if they are positioned on a field with one or more red arrows, move the number of steps indicated by the number of arrows on the square. In this case, if there are no arrows on a square, a coin positioned on this square does not move. Obviously, this is equivalent to the synchronization game by Lemma 2.2.

Apart from being more similar to board games, this perspective on finite automata is also motivated by robot navigation problems; see [26]. Conceptually, it is interesting to note that this particular paper talks about homing sequences, a notion quite akin to that of synchronizing words, cf. the exposition of Sandberg [32]. Of course, one could think of different puzzles played on such a board: the traditional SHORT SYNCHRO problem would lead to a one-person game, while the synchronization game itself is a two-persons game. Further variations (or possibly levels) can be designed by introducing further special “effects” (or board cell symbols). For instance, one could introduce cells where the directions are rather interpreted as knight moves, etc.

4 General Complexity Considerations

The concept of synchronization has been studied quite intensively also in the light of complexity theory. In the following, we assume acquaintance with the basic complexity concepts behind on side of the reader. Our main questions are the following ones:

- SYNCHRO: Given a DFA \mathcal{A} , is \mathcal{A} synchronizing?
- SHORT SYNCHRO: Given a DFA \mathcal{A} and an integer $k \geq 0$, does \mathcal{A} possess a synchronizing word of length at most k ?
- SYNCHROGAME: Given a DFA \mathcal{A} , does Alice possess a winning strategy on \mathcal{A} ?
- SHORT SYNCHROGAME: Given a DFA \mathcal{A} and an integer $k \geq 0$, can Alice enforce a win on \mathcal{A} in at most k moves?

It is possible to explain four important classical complexity classes with these problems.

► **Theorem 4.1.** *Under logspace-reductions, we can state the following:*

- SYNCHRO is NL-complete.
- SHORT SYNCHRO is NP-complete.
- SYNCHROGAME is P-complete.
- SHORT SYNCHROGAME is PSPACE-complete.

In the following proof, we reduce from the P-complete AND/OR GRAPH ACCESSIBILITY PROBLEM, or AGAP for short, that was first introduced in [19]; we follow the presentation of Sudborough [35] as a pebble game, as it makes the connection to SYNCHROGAME more transparent. The input is a directed acyclic graph $G = (V, E)$ and a labelling function $f : V \rightarrow \{\vee, \wedge\}$ such that there are no edges in between \vee -labeled and no edges in between \wedge -labeled vertices.⁷ The rules of this pebble game are simple: (1) one can always place a pebble on a node which has no outgoing edges, and (2a) one can place a pebble on node x , when $f(x) = \wedge$ and all nodes to which an edge is directed from node x contain pebbles, and (2b) one can place a pebble on node x , when $f(x) = \vee$ and at least one node to which an edge is directed from node x contains a pebble. The question is whether a pebble can be placed on one of the nodes of indegree zero by playing the pebble game. It is possible to restrict one's attention to directed acyclic graphs that have exactly one vertex t of outdegree zero and one vertex s of indegree zero. Then, we also say that t is *reachable* from s in G . Notice that then conditions (1) and (2a) mean the same if $f(t) = \wedge$. Moreover, the labelled directed acyclic graph G can be transformed into a labelled directed acyclic bipartite graph G' that has outdegree at most two by replacing a larger number of outgoing edges by a tree. Notice that AGAP is basically equivalent to the non-emptiness problem of alternating finite automata introduced in [7, 8]

Proof. The NL-completeness of SYNCHRO is somewhat folklore, it is based on the similarity of the condition of Lemma 2.1 to reachability in graphs. We also refer to [17, Theorem 5]. The NP-completeness of SHORT SYNCHRO was shown at least three times independently from each other in the literature, with nearly identical proofs; see [27, 10, 14] in chronological order. The PSPACE-completeness of SHORT SYNCHROGAME was proved in [13].

We are now proving that SYNCHROGAME is P-complete, a result that nicely complements earlier findings, because it shows two effects that can be observed already from the previous list of results: (a) the “game variant” of the synchronization problem is harder than the classical variant; (b) the length-bounded variant is harder than the unbounded variation.

⁷ The classical exposition does not require this bipartiteness condition, but it is not hard to see that bipartiteness can be enforced with a logspace machine.

The following reduction is inspired by that of [17, Theorem 5]. We start from a labelled directed acyclic bipartite graph $G = (V, E)$ that has outdegree at most two, with one vertex t of outdegree zero, labeled like $f(t) = \wedge$, and one vertex s of indegree zero. We construct a DFA $\mathcal{A} = (Q, \Sigma, \delta)$ such that Alice can win on \mathcal{A} if and only if t is reachable from s in G . Let $\Sigma = \{a, b, c\}$. We can assume that s is an \wedge -vertex (otherwise, put a new vertex before s that is a \wedge -vertex and has only one transition to s). Set $Q = V \cup \{s'\}$, where $s' \notin V$. Now, for each vertex, if two edges go out, label one with a and the other one with b , and if only a single edge emanates, label this one with a and b , i.e., add two transitions, and if no edge leaves, add two self-loops for a and b . Furthermore, set $\delta(s', a) = \delta(s', b) = \delta(s', c) = s$. For every other vertex $q \in Q \setminus \{s'\}$, set $\delta(q, c) = t$ if q corresponds to a \wedge -vertex (where Bob has a choice later) and $\delta(q, c) = s$ if q corresponds to a \vee -vertex. By the choice of c , if it is Alice's turn (where she lands on a \vee -vertex) she never chooses c (except at the beginning to set everything to either s or t), because this would “destroy” her progress, i.e., reset to her starting position where only s and t are “active” states. If it is Bob's turn (on the \wedge -vertices) he also never chooses c (or at least we do not have to consider those moves in the following), as this would instantly map everything to t and Alice wins.

Recall that due to Lemma 2.2, we only need to discuss what happens if there are only two coins left in the game. In fact, this led to the discussion of a variation of the pair automaton in [13, Theorem 4] that keeps (additionally) track of whether it is Alice's turn or Bob's turn. On this automaton, a marking algorithm is presented that works exactly like the pebble game described above. The discussion in the previous paragraph shows that we need not consider this whole pair automaton for the synchronization game that we constructed, only the state pairs reachable from $\{s, t\}$ matter. However, now it can be observed that the automaton \mathcal{A} described above is isomorphic to the pair automaton of \mathcal{A} , restricted to state pairs reachable from $\{s, t\}$. The reasoning of [13, Theorem 4] hence shows that Alice has a winning strategy on \mathcal{A} if and only if t is reachable from s in G . ◀

► **Corollary 4.2.** *On weakly acyclic DFAs, SYNCHRO and SYNCHROGAME are NL-complete.*

Proof. By the previous theorem, checking if Alice has a winning strategy is the same as checking if the automaton is synchronizing, which is in NL. Hardness can be shown by a reduction similar as the one used in Theorem 4.1 to show P-completeness, but using the ordinary acyclic graph reachability (GAP), which is NL-complete. ◀

5 Commutative Automata

The DFA $\mathcal{A} = (Q, \Sigma, \delta)$ is *commutative*, if for each $q \in Q$ and $a, b \in \Sigma$, we have $\delta(q, ab) = \delta(q, ba)$. Every unary DFA, i.e., if $|\Sigma| = 1$, is commutative. See Figure 2 for two examples.

5.1 Combinatorial and Structural Results

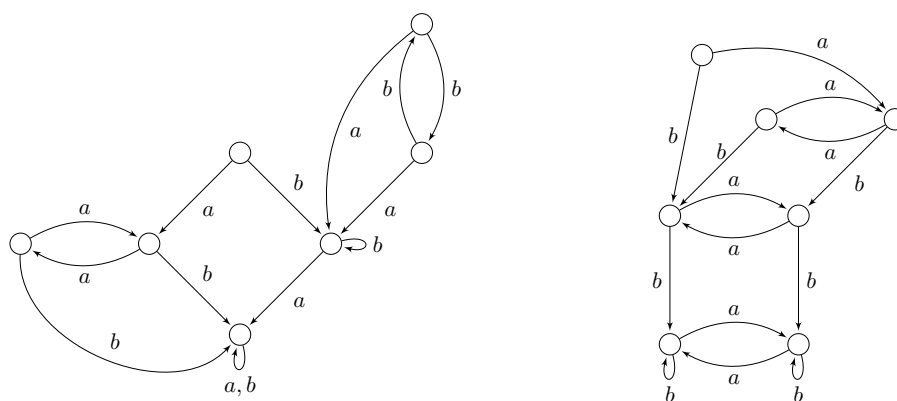
It is easy to construct connected automata with several sink states. However, for a commutative automaton, we can only have at most one sink state.

► **Lemma 5.1.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a commutative DFA. If the DFA \mathcal{A} is connected and has a sink state, then this sink state is unique and reachable from every other state.*

With Lemma 5.1, we can prove the following for synchronizing commutative DFAs.

► **Theorem 5.2.** *For a commutative automaton \mathcal{A} , the following are equivalent:*

1. \mathcal{A} is connected and contains a sink state (which must be unique),
2. \mathcal{A} is synchronizing (and the synchronizing state is the unique sink state),
3. Alice has a winning strategy when playing on \mathcal{A} .



■ **Figure 2** Left: A commutative synchronizing automaton with synchronizing word aab . Right: A commutative automaton that is not synchronizing.

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a commutative automaton.

(1) implies (2): Assume \mathcal{A} is connected and let s_f be the sink state. By Lemma 5.1, it is unique and reachable from every other state. Write $Q = \{q_0, q_1, \dots, q_n\}$. Then, for each number $i \in \{0, 1, \dots, n\}$ there exists $u_i \in \Sigma^*$ such that $\delta(q_i, u_i) = s_f$. Hence, the word $u = u_0 u_1 \dots u_n$ synchronizes \mathcal{A} . Note that this simple arguments gives a quadratic upper bound for a synchronizing word. However, for commutative DFA the better bound $n - 1$ is known [28]. That the synchronizing state is a unique sink state was shown in [12, Lemma 20].

(2) implies (3): Let $u = u_1 \dots u_n \in \Sigma^*$ be a synchronizing word with $u_i \in \Sigma$ for each $i \in \{1, \dots, n\}$. Then, if Alice plays u_i in her i -th move, she will win after at most n moves. For if Bob chooses the letter b_1, \dots, b_{n-1} after the i -th move of Alice, we have, by commutativity,

$$\delta(Q, u_1 b_1 u_2 b_2 \dots u_{n-1} b_{n-1} u_n) = \delta(Q, u_1 u_2 \dots u_n b_1 b_2 \dots b_{n-1}) = \delta(\delta(Q, u), b_1 b_2 \dots b_{n-1})$$

and the latter set is a singleton set, as $\delta(Q, u)$ is a singleton set.

(3) implies (1): Similar to Theorem 2.3. ◀

In Theorem 5.2 it was shown that if \mathcal{A} is synchronizing, then Alice has a winning strategy by using a synchronizing word directly as a winning strategy. Combined with the fact that a shortest synchronizing word in a commutative n -state automaton has length $n - 1$, first proven in [28] and reproven by a combinatorial analysis in [12], we find the next bound for the shortest number of moves that Alice has to perform in a winning strategy.

► **Proposition 5.3.** *Let \mathcal{A} be a commutative automaton with n states on which Alice can win. Then she can win performing at most $n - 1$ moves and this bound is best possible.*

5.2 Complexity-Theoretical Results

With Theorem 5.2, we can show that testing synchronizability of commutative automata is L-complete. Furthermore, contrary to Theorem 4.1, the “gamified” variant has the same complexity on commutative automata, i.e., is L-complete.

► **Theorem 5.4.** *For commutative automata \mathcal{A} , the problems SYNCHRO and SYNCHROGAME are L-complete (even for a fixed unary alphabet).*

14:10 The Synchronization Game on Subclasses of Automata

In [11, Theorem 3] a reduction from HITTINGSET to a DFA is given that, in fact, yields a commutative automaton (with an unbounded alphabet) and shows that this problem is NP-hard on this class. Hence, SHORT SYNCHRO is NP-complete on commutative automata.

Given $\mathcal{A} = (Q, \Sigma, \delta)$, a *permutational letter* $a \in \Sigma$ is a letter such that $q \mapsto \delta(q, a)$ is a permutation, i.e., a bijective mapping. The problems are SHORT SYNCHROCOMMPERMCHAR and SHORT SYNCHROGAMECOMMPERMCHAR are defined as SHORT SYNCHRO and SHORT SYNCHROGAME but with the input restricted to commutative automata having at least one permutational letter. This letter can be used by Bob to “delay” an eventual win of Alice on a synchronizing commutative automaton, as mapping at least two states to a single state might help Alice. Note that commutativity is crucial here, as then we can move Bob’s choices to the beginning, and they do not interfere with the moves from Alice, i.e., do not make the word longer (as is possible in the non-commutative setting by evading states that might get synchronized, i.e., mapped to a single state, by Alice). With this idea, we can show the next theorem.

► **Theorem 5.5.** *The following problems are equivalent under logspace-reductions:*

1. SHORT SYNCHRO on commutative automata,
2. SHORT SYNCHROCOMMPERMCHAR,
3. SHORT SYNCHROGAMECOMMPERMCHAR.

and so they are all NP-complete (with the remarks preceding this statement).

Proof. The first two problems are obviously reducible to each other: The second is a restriction of the first, and if we have an instance of the first, we can add an “idle”-letter, i.e., a letter a such that $\delta(q, a) = q$ for each state q and this letter never appears in a shortest synchronizing word and it is a permutational letter.

Now for the equivalence of the latter two problems. Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a commutative automaton with permutational letter $b \in \Sigma$. Then \mathcal{A} has a synchronizing word of length $\leq k$ if and only if Alice can win in at most k moves on \mathcal{A} . Suppose Alice can win in at most k moves. In particular, she can win in case Bob always chooses the letter b . More specifically, consider a game with k rounds where Bob always chooses the letter b and Alice wins, i.e., a game

$$a_1 b a_2 b \cdots a_{k-1} b a_k.$$

Observe that we have $\delta(Q, b^k) = Q$, as b is a permutational letter. Hence, we have $\delta(Q, a_1 b a_2 b \cdots a_{k-1} b a_k) = \delta(Q, b^{k-1} a_1 a_2 \cdots a_{k-1} a_k) = \delta(Q, a_1 a_2 \cdots a_{k-1} a_k)$ and so the word $a_1 a_2 \cdots a_{k-1} a_k$ synchronizes \mathcal{A} .

Conversely, if $u \in \Sigma^*$ is synchronizing with $u = u_1 \cdots u_k$, $u_i \in \Sigma$, then Alice simply has to choose the letter u_i at her i -th move. This works because when the sequence of moves is $u_1 b_1 u_2 b_2 \cdots b_n u_n$ after Alice has following this strategy, where the b_i indicate Bob’s moves, then, by commutativity, $\delta(Q, u_1 b_1 u_2 b_2 \cdots b_n u_n) = \delta(Q, b_1 b_2 \cdots b_n u_1 u_2 \cdots u_n) \subseteq \delta(Q, u_1 u_2 \cdots u_n)$ and the latter set is a singleton set as u is a synchronizing word. So Alice has won. ◀

As SHORT SYNCHROCOMMPERMCHAR is a special case of SHORT SYNCHROGAME on commutative automata, we get the next corollary to Theorem 5.5.

► **Corollary 5.6.** *The problem SHORT SYNCHROGAME is NP-hard on commutative automata.*

For general commutative input automata, we do not know if this problem is also in NP. The best we can show here is that the problem is in \prod_2^P , a complexity class from the second level of the polynomial-time hierarchy (see [34]). This poses the natural (open) question if SHORT SYNCHROGAME is complete for any well-known class between NP and \prod_2^P .

► **Theorem 5.7.** *SHORT SYNCHROGAME is in Π_2^P for commutative input automata.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a commutative automaton and $k \geq 0$. We claim that Alice has a winning strategy within at most k moves if and only if, for every word u with $|u| \leq k - 1$, there exists a word v with $|v| = k$ such that

$$|\delta(Q, uv)| = 1.$$

First, suppose Alice has a winning strategy of length at most k and let $u \in \Sigma^*$ with $u = u_1 \cdots u_{k-1}$ and $u_i \in \Sigma$. Then, there exists $v = v_1 \cdots v_k$ such that, when Bob plays u , Alice reacts to u_i with v_{i+1} , and

$$|\delta(Q, v_1 u_1 v_2 \cdots u_{k-1} v_k)| = 1.$$

By commutativity, $\delta(Q, uv) = \delta(Q, v_1 u_1 v_2 \cdots u_{k-1} v_k)$.

Conversely, consider an arbitrary game, played till the i -th round with $i \leq k$, having a sequence of moves $a_1 b_1 a_2 b_2 \cdots b_{i-2} a_{i-1} b_{i-1}$. Then there exists a letter $a_i \in \Sigma$ such that after having played for k rounds, the word

$$a_1 b_1 a_2 b_2 \cdots a_{k-1} b_{k-1} a_k$$

synchronizes \mathcal{A} , as for $u = b_1 b_2 \cdots b_{k-1}$ there exists a word $v = a_1 a_2 \cdots a_k$ such that $|\delta(Q, uv)| = 1$ and by commutativity $\delta(Q, a_1 b_1 a_2 b_2 \cdots a_{k-1} b_{k-1} a_k) = \delta(Q, uv)$.

Note that we can assume $|u| = k - 1$ by commutativity.

Lastly, given a word and an automaton, it can be checked in logarithmic space [17, Theorem 4] if this word synchronizes a given automaton. Hence, the formula

$$\forall u_1 \cdots u_{k-1} \in \Sigma \exists v_1, \dots, v_k \in \Sigma : |\delta(Q, u_1 \cdots u_{k-1} v_1 \cdots v_k)| = 1$$

can be checked by an alternating non-deterministic polynomial-time Turing machine that starts in a universal state, guesses the sequence u_1, \dots, u_{k-1} and then switches to an existential state, from which it guesses the remaining words and finally checks if the guessed word synchronizes \mathcal{A} . This shows containment in Π_2^P . ◀

6 Cyclic Automata with a k -Simple Idempotent over a Binary Alphabet

Recall the definition of cyclic DFAs with a k -simple idempotent over a binary alphabet $\Sigma = \{a, b\}$: There is one k -simple idempotent letter $a \in \Sigma$, while the letter $b \in \Sigma$ permutes all states cyclically. Notice that there is always a race-track design for cyclic DFAs with a k -simple idempotent as introduced in Section 3.

6.1 Combinatorial and Structural Results

The idempotency immediately leads to our first result. For simplicity, we call synchronizing pairs $\{q, q'\}, \{p, p'\} \in Q_{\mathcal{P}_2}$ with $\delta_{\mathcal{P}_2}(\{q, q'\}, b) = \{p, p'\}$ a *double synchronizing pair*.

► **Theorem 6.1.** *Let \mathcal{A} be a cyclic DFA with a k -simple idempotent. Bob has a winning strategy for \mathcal{A} if there is no double synchronizing pair in its pair automaton $\mathcal{P}_2(\mathcal{A})$.*

Proof. As a consequence of Lemma 2.2 it is sufficient to consider only the last two coins left in the synchronization game. On the respective pair automaton, this means that there is one coin left which Bob wants to keep from reaching \perp . Whenever it is Bob's turn

14:12 The Synchronization Game on Subclasses of Automata

on a non-synchronizing pair $\{s, t\}$ where choosing b would lead into a synchronizing pair, choosing a will lead to a non-synchronizing pair. Otherwise, \mathcal{A} would not be cyclic with a k -simple idempotent. Furthermore, whenever it is Bob's turn on a synchronizing pair, choosing b will lead to a non-synchronizing state. Bob can therefore always prevent that Alice's turn starts in a synchronizing pair, keeping her from winning. ◀

Since there can only ever be one synchronizing pair in the pair automaton of a 1-simple idempotent automaton, the next corollary follows immediately.

► **Corollary 6.2.** *Bob has a winning strategy on every cyclic DFA with a 1-simple idempotent.*

To get a better feel for this newly introduced class of DFAs, we continue by looking at cyclic DFAs with a 2-simple idempotent before getting back to the general case. The proof of the following theorem (which is based on some case distinction) leads to a nice result, as it allows us to determine who has a winning strategy on a cyclic DFA with a 2-simple idempotent by simply looking at the DFA in question.

► **Theorem 6.3.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a cyclic DFA with a 2-simple idempotent with $\Sigma = \{a, b\}$ and a single b -cycle. Alice has a winning strategy on the synchronizing game on \mathcal{A} if and only if $\delta(q_{i+1 \bmod n}, a) = q_i$ and $\delta(q_{i+2 \bmod n}, a) = q_i$ for any $i \in \{0, 1, \dots, n-1\}$ and $n \geq 4$.*

► **Corollary 6.4.** *Let \mathcal{A} be a cyclic automaton with a k -simple idempotent. Bob has a winning strategy for \mathcal{A} if there is no double synchronizing pair in its pair automaton $\mathcal{P}_2(\mathcal{A})$ which is reachable from every state $q \in Q_{\mathcal{P}_2}$.*

By a similar argument as in the proof of Theorem 6.1, Bob can prevent Alice from leaving a cycle in the pair automaton. A cycle can consist of both a - and b -transitions. To leave a cycle, Alice needs two states $\{s, t\}, \{s', t'\} \in \mathcal{P}_2(\mathcal{A})$ that both have an a -transition leaving the cycle. Bob can also not stop her from leaving if there exists a synchronizing pair preceded or followed by a state with an a -transition leaving the cycle. Otherwise, Alice could synchronize.

► **Corollary 6.5.** *Let \mathcal{A} be a cyclic DFA with a k -simple idempotent. Bob has a winning strategy in the synchronization game on \mathcal{A} if there is a cycle in its pair automaton that does not include a double synchronizing pair and which Bob can prevent Alice from leaving.*

Combining the previous results, we conclude with the following theorem.

► **Theorem 6.6.** *Alice has a winning strategy in the synchronization game on a cyclic DFA \mathcal{A} with a k -simple idempotent if and only if*

1. *There is at least one double synchronizing pair in its pair automaton $\mathcal{P}_2(\mathcal{A})$ that is reachable from every state $\{q, q'\} \in \mathcal{P}_2(\mathcal{A})$,*
2. *There is no cycle in $\mathcal{P}_2(\mathcal{A})$ that does not contain a double synchronizing pair and which Bob can prevent Alice from leaving.*

Proof. If there is no double synchronizing pair in the pair automaton that is reachable from every $\{q, q'\} \in \mathcal{P}_2(\mathcal{A})$ or there is a cycle without a double synchronizing pair which Bob can prevent Alice from leaving, Bob wins by Theorem 6.1 and Corollaries 6.4 and 6.5.

If there is a double synchronizing pair that is reachable from every $\{q, q'\} \in \mathcal{P}_2(\mathcal{A})$, Bob needs an a -transition either skipping or leading away from the double synchronizing pair to keep Alice from synchronizing. Since there is a path from every state in the pair automaton to the synchronizing pair in question, this always creates a cycle. If Alice cannot leave this

cycle and there is no other synchronizing pair within this cycle, then Bob wins. If there is a synchronizing pair that Bob cannot avoid, then Alice wins. If there is a synchronizing pair which Bob can avoid, then the argument repeats itself. ◀

6.2 Complexity-Theoretical Results

A *k-simple idempotent DFA* is a DFA such that every letter either is a *k-simple idempotent letter* or permutes the states (called a *permutational letter* in Section 5). Next, we show that for *k-simple idempotent automata* with an unbounded alphabet, the problem SHORT SYNCHRO is NP-complete. However, we do not know if the problem remains hard for a fixed alphabet, or for cyclic automata with a *k-simple idempotent* over a binary alphabet. More precisely, we prove the following result.

► **Theorem 6.7.** *SHORT SYNCHRO is NP-complete for the class of k -simple idempotent DFAs, for each $k \geq 3$, even if no letter induces a permutation.*

In [22, Proposition 6.1], it was already shown that for 1-simple idempotent automata SHORT SYNCHRO is NP-complete. By adding "dummy" states ending at the synchronizing state in the construction from [22], it can be adapted to *k-simple idempotent automata*. However, note that the used reduction from SAT uses an unbounded number of permutations (corresponding to variables) and an unbounded number of 1-simple idempotent letters (corresponding to the clauses). Contrary, our reduction, by using a special variant of SAT, uses no permutations at all (and adding permutations can be achieved easily) and only *k-simple idempotent letters*. As we were concerned with bounding the number of letters in the previous subsection, our new reduction that is presented in the long version of the paper is of interest in this context.

7 Monotonic Automata

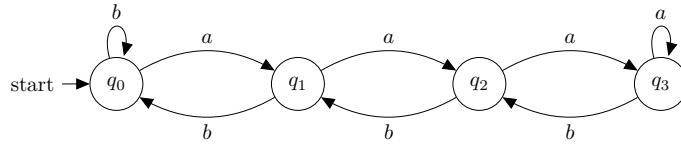
An automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is *monotonic* if there exists a linear ordering of the states $Q = \{q_0, q_1, \dots, q_{n-1}\}$ such that if $q_i \leq q_j$, i.e., $i \leq j$, then $\delta(q_i, x) \leq \delta(q_j, x)$ for each $x \in \Sigma$.

Monotonic automata were introduced by Ananichev & Volkov [1] (the monotonic automata as introduced earlier by Eppstein [10] are more general, as they are only required to respect a cyclic order and were called *oriented automata* by Ananichev & Volkov). In [31], further problems related to (subset) synchronization problems on monotonic automata were investigated. An open problem from [1] was investigated in [33]. Checking if a given DFA is monotonic is NP-complete [36]. Note that monotonic DFAs also appeared in the context of games previously in [21], where they describe winning conditions on certain infinite games.

The problem SHORT SYNCHRO is solvable in polynomial time on monotonic automata. This follows from the more general result that this problem is solvable in polynomial time on oriented automata as shown in [10] (recall the remark above that what Eppstein calls monotonic was later renamed to oriented). However, we do not know the precise computational complexity for SHORT SYNCHROGAME on monotonic DFAs.

Note that if $u \in \Sigma^*$ and $q \in Q$, then either $q \leq \delta(q, u)$ or $\delta(q, u) \leq q$. The first case implies $q \leq \delta(q, u) \leq \delta(q, u^2) \leq \dots$ and so, if $q = \delta(q, u^i)$ for some $i > 0$, then $q = \delta(q, u)$ and similarly in the second case. So, no word can permute a subset of states non-trivially and every monotonic automaton is counter-free in the sense of [23].

The DFA in Figure 3 is synchronizable by the word *aaa*. Yet, Alice has no winning strategy on this automaton. For if $\{q_0, q_1\} \subseteq \delta(Q, u)$ and it is Bob's turn, he can choose *a*, and if $\{q_2, q_3\} \subseteq \delta(Q, u)$, he can choose *b*. Doing so, when it is Alice's turn, we have $\{q_1, q_2\} \subseteq \delta(Q, ux)$ and in the next turn Bob is faced with one of the two situations outlined above. As $\{q_1, q_2\} \subseteq Q$, we can conclude inductively that Alice cannot synchronize this DFA.



■ **Figure 3** A synchronizable monotonic automaton for which Alice has no winning strategy.

In [25], the authors outlined a modeling of synchronization games by *accessibility games* as used in software testing [4]. An accessibility game is a tuple (G, v, A) where G is a finite directed graph, v a vertex and A a subset of vertices. At the start a token is placed on v and then the two players Alice and Bob successively move the token around among neighboring vertices and the goal of Alice is to move the token to a vertex in A , whereas Bob tries to prevent exactly this. Now, in [25] it was shown that every accessibility game can be modelled as a synchronization game started from a single subset containing two states (in [25], the authors generalize the synchronization game to arbitrary starting set instead of the whole state set, but this generalization is also implicit in the pair criterion from Lemma 2.2) and this conversion can be done in polynomial time. The authors also give a conversion of a synchronization game into a accessibility game. However, this conversion involves all subsets of the starting set (which, in our context, is always the full set of states Q) and is not doable in polynomial time. Next, we show that the synchronization game for every monotonic automaton can be converted into an equivalent accessibility game in polynomial time.

► **Proposition 7.1.** *For a given monotonic automaton, we can construct in polynomial time an accessibility game equivalent to the synchronization game (even for the generalization that the starting set is not the full state set).*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a monotonic automaton with linear ordering of the states given by a numbering $Q = \{q_0, q_1, \dots, q_{n-1}\}$. Then, as for each $q_i \in Q$, $i \in \{0, 1, \dots, n-1\}$, and $u \in \Sigma^*$ we have $\delta(q_0, u) \leq \delta(q_i, u) \leq \delta(q_{n-1}, u)$, we find that $|\delta(Q, u)| = 1$ if and only if $\delta(q_0, u) = \delta(q_{n-1}, u)$. Construct the graph G with vertex set $\{\{q, q'\} \mid q, q' \in Q\}$ and edge set $\{(\{q, q'\}, \{q'', q'''\}) \mid \exists x \in \Sigma : \delta(\{q, q'\}, x) = \{q'', q'''\}\}$. Then Alice has a winning strategy in the synchronization game on \mathcal{A} if and only if she has a winning strategy in the accessibility game on $(G, \{q_0, q_{n-1}\}, \{\{q\} \mid q \in Q\})$.

Lastly, note that the argument works with any subset $S \subseteq Q$ as a starting set, but instead of q_0, q_{n-1} we use the minimal state and the maximal state in S . ◀

8 Discussions

We have studied conditions that allow us to tell if Alice or Bob have a winning strategy in a *synchronization game*. Concludingly, we discuss some of the consequences that our studies may have on the design of an implementation of a synchronization game. In our eyes, the consequences are mostly concerning the design of different levels for this game. Namely, assume that the game is played between two persons (one of them could be replaced by a computer). Then, it seems to make a difference which DFAs are chosen to be synchronized. Hence, we propose to choose DFAs from the classes that we studied here in order to design lower levels of the game. In particular, commutative DFAs seem to lead to quite simple games. Also, they can be nicely visualized, as explained in Section 3. Then, with growing experience, automata can be proposed that are more complicated, not only in terms of their number of states, but also concerning the classes of automata that they belong to. Another

possibility for level design comes with the aspect of time, here best measured in terms of an upper bound on the length of the synchronizing word: if less time is permitted, then Alice has a harder time to synchronize the DFA. Notice that the corresponding basic decision problem (given a DFA \mathcal{A} and an integer k , is there a synchronizing word of length at most k ?) is NP-complete even for seemingly easy classes of DFAs; we refer here to [27, 10, 22, 6], in chronological order. This step-bounded game variant is proven to be even PSPACE-complete in [13].

There is one further aspect concerning this level design: one could view the task of Alice in particular as that of extending a “current word” to some synchronizing word by appending letters. As mentioned above, the question if a word can be extended in such a way at all is a particular case of an *extension problem* as studied in [12]. As in general extension problems in this context depend on the chosen ordering on the words, we could increase the level of difficulty in the synchronization game by changing the partial order. For instance, instead of extending a word to the right (corresponding to a prefix ordering), one could also extend it to the left (i.e., using a suffix ordering), or to both sides (corresponding to an infix ordering) or anywhere in the word (corresponding to a subsequence ordering). One might even give different rules for Alice or Bob. This also opens room for further studies concerning winning strategies for these game variations.

References

- 1 Dimitry S. Ananichev and Mikhail V. Volkov. Synchronizing monotonic automata. *Theoretical Computer Science*, 327(3):225–239, 2004.
- 2 Dimitry S. Ananichev, Mikhail V. Volkov, and Yu. I. Zaks. Synchronizing automata with a letter of deficiency 2. *Theoretical Computer Science*, 376(1-2):30–41, 2007.
- 3 Janet H. Barnett. Early writings on graph theory: Hamiltonian circuits and the icosian game. In Brian Hopkins, editor, *Resources for Teaching Discrete Mathematics: Classroom Projects, History Modules, and Articles*, volume 74 of *MAA Notes*, pages 217–224. Mathematical Association of America, 2009.
- 4 Andreas Blass, Yuri Gurevich, Lev Nachmanson, and Margus Veanes. Play to test. In Wolfgang Grieskamp and Carsten Weise, editors, *Formal Approaches to Software Testing, 5th International Workshop, FATES*, volume 3997 of *LNCS*, pages 32–46. Springer, 2005.
- 5 Stojan Bogdanović, Balázs Imreh, Miroslav Cirić, and Tatjana Petković. Directable automata and their generalizations: A survey. *Novi Sad J. Math.*, 29(2), 1999.
- 6 Jens Bruchertseifer and Henning Fernau. Synchronizing series-parallel deterministic automata with loops and related problems. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 55(7):1–24, 2021.
- 7 Janusz A. Brzozowski and Ernst L. Leiss. On equations for regular languages, finite automata, and sequential networks. *Theoretical Computer Science*, 10:19–35, 1980. doi:10.1016/0304-3975(80)90069-9.
- 8 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.
- 9 Erik D. Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In Jirí Sgall, Ales Pultr, and Petr Kolman, editors, *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS*, volume 2136 of *LNCS*, pages 18–32. Springer, 2001.
- 10 David Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510, 1990.
- 11 Henning Fernau, Pinar Heggernes, and Yngve Villanger. A multi-parameter analysis of hard problems on deterministic finite automata. *Journal of Computer and System Sciences*, 81(4):747–765, 2015.

- 12 Henning Fernau and Stefan Hoffmann. Extensions to minimal synchronizing words. *Journal of Automata, Languages and Combinatorics*, 24(2-4):287–307, 2019.
- 13 Fedor M. Fominykh, Pavel V. Martyugin, and Mikhail V. Volkov. P(1)aying for synchronization. *International Journal of Foundations of Computer Science*, 24(6):765–780, 2013.
- 14 Pavel Goralčík and Václav Koubek. Rank problems for composite transformations. *International Journal of Algebra and Computation*, 5(3):309–316, 1995.
- 15 Vasily V. Gusev. The vertex cover game: Application to transport networks. *Omega*, 97:102102, 2020.
- 16 Robert A. Hearn and Erik D. Demaine. *Games, puzzles and computation*. A K Peters, 2009.
- 17 Stefan Hoffmann. Constrained synchronization and commutativity. *Theoretical Computer Science*, 890:147–170, 2021.
- 18 Stefan Hoffmann. Constrained synchronization and subset synchronization problems for weakly acyclic automata. In Nelma Moreira and Rogério Reis, editors, *Developments in Language Theory - 25th International Conference, DLT*, volume 12811 of *LNCS*, pages 204–216. Springer, 2021.
- 19 Neil Immerman. Length of predicate calculus formulas as a new complexity measure. In *20th Annual Symposium on Foundations of Computer Science, FOCS*, pages 337–347. IEEE Computer Society, 1979.
- 20 Raphael M. Jungers. The synchronizing probability function of an automaton. *SIAM Journal of Discrete Mathematics*, 26:177–192, 2012.
- 21 Eryk Kopczynski. Half-positional determinacy of infinite games. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP, Proceedings, Part II*, volume 4052 of *LNCS*, pages 336–347. Springer, 2006.
- 22 Pavel Martyugin. Complexity of problems concerning reset words for some partial cases of automata. *Acta Cybernetica*, 19(2):517–536, 2009.
- 23 Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. Research Monograph No. 65)*. The MIT Press, 1971.
- 24 Mirjana Mikalački and Miloš Stojaković. Fast strategies in biased maker-breaker games. *Discrete Mathematics & Theoretical Computer Science*, 20(2), 2018.
- 25 Juan Andres Montoya and Christian Nolasco. Some remarks on synchronization, games and planar automata. *Electronic Notes in Theoretical Computer Science*, 339:85–97, 2018. The XLII Latin American Computing Conference.
- 26 Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103:299–347, 1993.
- 27 Igor Rystsov. On minimizing the length of synchronizing words for finite automata. In *Theory of Designing of Computing Systems*, pages 75–82. Institute of Cybernetics of Ukrainian Acad. Sci., 1980. (in Russian).
- 28 Igor Rystsov. Reset words for commutative and solvable automata. *Theoretical Computer Science*, 172(1-2):273–279, 1997.
- 29 Igor Rystsov. Estimation of the length of reset words for automata with simple idempotents. *Cybernetics and Systems Analysis*, 36(3):339–344, 2000.
- 30 Andrew Ryzhikov. Synchronization problems in automata without non-trivial cycles. *Theoretical Computer Science*, 787:77–88, 2019. Implementation and Application of Automata (CIAA 2017).
- 31 Andrew Ryzhikov and Anton Shemyakov. Subset synchronization in monotonic automata. *Fundamenta Informaticae*, 162(2-3):205–221, 2018.
- 32 Sven Sandberg. Homing and synchronizing sequences. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems, Advanced Lectures [The volume is the outcome of a research seminar that was held in Schloss Dagstuhl in January 2004]*, volume 3472 of *LNCS*, pages 5–33. Springer, 2004.

- 33 Tamara Shcherbak. The interval rank of monotonic automata. In Jacques Farré, Igor Litovsky, and Sylvain Schmitz, editors, *Implementation and Application of Automata, 10th International Conference, CIAA*, volume 3845 of *LNCS*, pages 273–281. Springer, 2005.
- 34 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.
- 35 Ivan Hal Sudborough. The complexity of path problems in graphs and path systems of bounded bandwidth. In Hartmut Noltemeier, editor, *Graphtheoretic Concepts in Computer Science, Proceedings of the International Workshop WG '80*, volume 100 of *LNCS*, pages 293–305. Springer, 1981.
- 36 Marek Szykula. Checking whether an automaton is monotonic is NP-complete. In Frank Drewes, editor, *Implementation and Application of Automata - 20th International Conference, CIAA*, volume 9223 of *LNCS*, pages 279–291. Springer, 2015.
- 37 Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA*, volume 5196 of *LNCS*, pages 11–27. Springer, 2008.
- 38 Mikhail V. Volkov. Preface: Special issue on the Černý conjecture. *Journal of Automata, Languages and Combinatorics*, 24(2-4):119–121, 2019.

Making Life More Confusing for Firefighters

Samuel D. Hand  

School of Computing Science, University of Glasgow, UK

Jessica Enright  

School of Computing Science, University of Glasgow, UK

Kitty Meeks  

School of Computing Science, University of Glasgow, UK

Abstract

It is well known that fighting a fire is a hard task. The FIREFIGHTER problem asks how to optimally deploy firefighters to defend the vertices of a graph from a fire. This problem is NP-Complete on all but a few classes of graphs. Thankfully, firefighters do not have to work alone, and are often aided by the efforts of good natured civilians who slow the spread of a fire by maintaining firebreaks when they are able. We will show that this help, although well-intentioned, unfortunately makes the optimal deployment of firefighters an even harder problem. To model this scenario we introduce the TEMPORAL FIREFIGHTER problem, an extension of FIREFIGHTER to temporal graphs. We show that TEMPORAL FIREFIGHTER is also NP-Complete, and remains so on all but one of the underlying classes of graphs on which FIREFIGHTER is known to have polynomial time solutions. This motivates us to explore making use of the temporal structure of the graph in our search for tractability, and we conclude by presenting an FPT algorithm for TEMPORAL FIREFIGHTER with respect to the temporal graph parameter vertex-interval-membership-width.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Temporal graphs, Spreading processes, Parameterised complexity

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.15

Related Version *Previous Version:* <https://arxiv.org/abs/2202.12599> [13]

Funding *Samuel D. Hand:* Supported by an EPSRC doctoral training account.

Jessica Enright: Supported by EPSRC grant EP/T004878/1.

Kitty Meeks: Supported by EPSRC grant EP/T004878/1.

1 Introduction

Imagine a fire breaks out on an island within an archipelago. The fire service always have one on duty firefighter, who is quickly deployed to protect one of the other islands. The islands within the archipelago are connected by bridges, which the fire now spreads along to all unprotected islands that it neighbours. By now, the fire service have called in another firefighter, who is again deployed to protect an island, and the process repeats. The question of determining how many islands can be saved from the fire in such a scenario is formalised by the FIREFIGHTER problem, which models the spread of the fire over the vertices of a graph [14].

As noted by Fomin et al. firefighting is a tough job [11]. Specifically FIREFIGHTER is NP-Complete on arbitrary graphs, although progress has been made on identifying graph classes for which it can be solved in polynomial time [9, 12]. In particular these are: interval graphs, permutation graphs, P_k -free graphs for $k > 5$, split graphs, cographs, and graphs of maximum degree three providing the root is of degree two. Additionally, both the parameterised complexity and the approximability of the problem have been considered [1, 3, 7, 8]. For a more general review of known results about FIREFIGHTER, see the work by Finbow and MacGillivray [10].



© Samuel D. Hand, Jessica Enright, and Kitty Meeks;
licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 15; pp. 15:1–15:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

15:2 Making Life More Confusing for Firefighters

Thankfully for the firefighters, they don't have to do all the work themselves. When they can spare the time, islanders will help to maintain firebreaks at the bridges, thus delaying the spread of the fire. Whilst this help is of course much appreciated, it unfortunately makes the problem of choosing how to optimally deploy firefighters all the more confusing, even when it is known ahead of time when the islanders will be available to maintain the firebreaks. In this paper we explore how to simplify this decision making, both by making use of the layout of the archipelago, and the availability of the islanders. In order to do this we introduce **TEMPORAL FIREFIGHTER**, an extension of **FIREFIGHTER** to a variety of graph in which the edges of an underlying graph are assigned times at which they are active. We refer to this variant of graph as a temporal graph. Existing algorithmic work on temporal graphs has explored how they change the notions of both paths and connectivity [2, 4, 5, 15, 17, 19]. For a survey of algorithmic work on temporal graphs see Michai [18], and for a more multidisciplinary overview see the work by Holme and Saramäki [16].

We begin in Section 2 by giving a formal definition **TEMPORAL FIREFIGHTER**, before exploring how extending **FIREFIGHTER** in this way affects the spread of the fire and the complexity of the associated decision problem. We find that for every class \mathcal{C} of graphs for which **FIREFIGHTER** is NP-Complete, **TEMPORAL FIREFIGHTER** is NP-Complete on the class of temporal graphs with the graphs of \mathcal{C} underlying graphs. This motivates a search for tractable cases of **TEMPORAL FIREFIGHTER** in two directions. Firstly, in Section 3, we explore its complexity when the underlying graph class is restricted, finding that it remains NP-Complete on all but one of the underlying graph classes for which **FIREFIGHTER** is tractable. More promisingly, in Section 4, we investigate restricting the temporal structure, and give an algorithm that is FPT with respect to the temporal graph parameter vertex-interval-membership-width.

2 Preliminaries

Formally, the **FIREFIGHTER** problem asks how many vertices it is possible to prevent from burning on a connected, undirected, loop-free, rooted graph in the following discrete time process:

1. At time $t = 0$, the root is labeled as burning.
2. At all times $t \geq 1$, a chosen vertex is labeled as defended, and the fire then spreads to all undefended vertices adjacent to the fire.
3. This process ends once the fire can no longer spread.

A vertex v is valid to defend on timestep i if and only if v is not burning or already defended on timestep i . We refer to a sequence of such valid defences for **FIREFIGHTER** as a strategy.

► **Definition 1 (A Strategy)**. *A strategy is a sequence of vertices v_1, v_2, \dots, v_ℓ , such that each v_i is a valid defence on timestep i .*

We say a vertex is saved if it is not burning once the process ends. The decision problem then asks how many vertices can be saved on a given graph:

FIREFIGHTER

Input: A rooted graph (G, r) and an integer k .

Output: Does there exist a strategy that saves at least k vertices on G when the fire starts at vertex r ?

We now extend FIREFIGHTER to temporal graphs, using the definition of temporal graph first introduced by Kempe et al. [17].

► **Definition 2 (A Temporal Graph).** A pair (G, λ) where G is the underlying static graph (V, E) and $\lambda : E \rightarrow 2^{\mathbb{N}}$ is the time-labeling function, assigning to each edge a set of timesteps at which it is active.

The lifetime Λ of a temporal graph refers to the final time at which any edge is active:

► **Definition 3 (Lifetime).** The lifetime Λ of a temporal graph (G, λ) is the maximum time on any edge. $\Lambda = \max\{\max \lambda(e) : e \in E(G)\}$.

Temporal graphs introduce a new notion of adjacency. We say that two vertices are temporally adjacent on a given timestep if there is an edge between them active on that timestep.

► **Definition 4 (Temporal Adjacency).** Two adjacent vertices v_1 and v_2 in the temporal graph (G, λ) are temporally adjacent at time t if $t \in \lambda(v_1, v_2)$.

In TEMPORAL FIREFIGHTER, just as in FIREFIGHTER, the fire begins burning at a root vertex r , and on each timestep a single vertex can be defended before the fire spreads. Unlike FIREFIGHTER the fire does not spread to all adjacent vertices, but only to vertices to which it is temporally adjacent.

We define a strategy for TEMPORAL FIREFIGHTER exactly as in Definition 1 for FIREFIGHTER, and the decision problem is then defined analogously to FIREFIGHTER:

TEMPORAL FIREFIGHTER

Input: A rooted temporal graph $((G, \lambda), r)$ and an integer k .

Output: Does there exist a strategy that saves at least k vertices on (G, λ) when the fire starts at vertex r ?

In some ways modifying FIREFIGHTER to take place on a temporal graph actually makes the job of the firefighters easier. Assigning times to the edges of a static graph only serves to limit the spread of the fire. In particular, the fire in TEMPORAL FIREFIGHTER can only spread along temporally admissible paths, these being a subset of the paths in the underlying static graph.

► **Definition 5 (Temporally Admissible).** A temporally admissible path on the temporal graph (G, λ) is a path on G with edges e_1, \dots, e_ℓ , such that there is a strictly increasing sequence of times t_1, \dots, t_ℓ with $t_i \in \lambda(e_i)$ for every i .

Furthermore, assigning times to the edges can only slow the rate at which the fire spreads down a path; the fire is still limited to spreading at a rate of at most one vertex per timestep along that path. We refer to the earliest time at which the fire can burn fully along the length of a path from the root as the arrival time of the path.

► **Definition 6 (Arrival Time).** The arrival time of a temporally admissible path containing edges e_1, \dots, e_ℓ on the temporal graph (G, λ) is the minimum t_ℓ such that t_ℓ is the end of a strictly increasing sequence of times t_1, \dots, t_ℓ with $t_i \in \lambda(e_i)$ for every i .

As a result, if the same defences are made, the fire cannot reach anywhere in TEMPORAL FIREFIGHTER on a rooted temporal graph $((G, \lambda), r)$ that it would not be able to reach in FIREFIGHTER on the underlying static graph (G, r) . This gives us the following observation.

15:4 Making Life More Confusing for Firefighters

► **Observation 7.** *Any strategy $S = v_1, \dots, v_\ell$ for FIREFIGHTER on a rooted graph (G, r) is also a valid strategy for TEMPORAL FIREFIGHTER on any rooted temporal graph $((G, \lambda), r)$. Furthermore any vertex saved by S in FIREFIGHTER must also be saved by S in TEMPORAL FIREFIGHTER.*

However, the decision problem remains just as hard, as we can assign times in a rooted temporal graph $((G, \lambda), r)$ such that TEMPORAL FIREFIGHTER simulates FIREFIGHTER for any rooted graph (G, r) . This is achieved by setting $\lambda(e) = \{1, \dots, |V(G)| - 1\}$ for every edge e . By time $|V(G)| - 1$ every vertex would have been defended, so the process must be over. Thus, for the entirety of the time during which the fire can spread, every edge is active, just as in FIREFIGHTER. In this respect we can view FIREFIGHTER to be a special case of TEMPORAL FIREFIGHTER.

Note that TEMPORAL FIREFIGHTER is in NP, as a strategy acts as a certificate that can be checked in polynomial time by simulating TEMPORAL FIREFIGHTER. We then have the following observation, as the above method for simulating FIREFIGHTER preserves the underlying graph class.

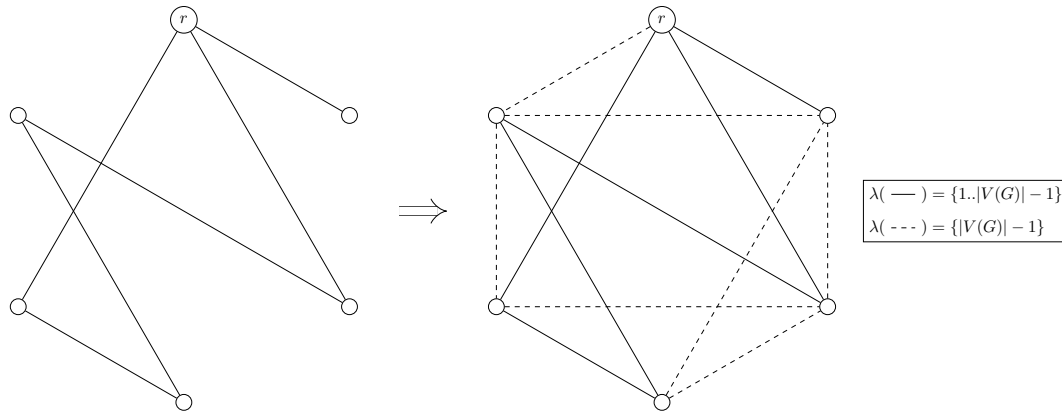
► **Observation 8.** *For every class \mathcal{C} of graphs for which FIREFIGHTER is NP-Complete, TEMPORAL FIREFIGHTER is NP-Complete on the class of temporal graphs with the graphs of \mathcal{C} as the underlying graphs.*

3 Restricting the Underlying Graph

As we have seen that TEMPORAL FIREFIGHTER is NP-Complete on any class of temporal graphs $\{(G, \lambda) : G \in \mathcal{C}\}$ where \mathcal{C} is a class of graphs for which FIREFIGHTER is NP-Complete, we now determine its complexity on underlying graph classes for which FIREFIGHTER is known to be solvable in polynomial time. These are: interval graphs, permutation graphs, P_k -free graphs for $k > 5$, split graphs, cographs, and graphs of maximum degree three providing the root is of degree two[9, 12]. We prove that TEMPORAL FIREFIGHTER is NP-Complete for all of these classes except the last, for which we find there is a polytime solution. Additionally we establish that it is NP-Complete for AT-free graphs, a class for which the complexity of FIREFIGHTER has not been determined. This lack of tractable cases of TEMPORAL FIREFIGHTER when restricting the underlying graph motivates restricting the temporal structure, which we explore in Section 4, finding this to be a fruitful route to tractability.

All of these hardness results follow from the fact that TEMPORAL FIREFIGHTER is hard when the underlying graph is a clique. This can be shown by reduction from FIREFIGHTER by assigning times to the edges in a static graph G so that they will be active at all times up until $|V(G)| - 1$, at which point the fire can certainly no longer spread. We then add further edges to make the graph a clique, and have them only active from time $|V(G)| - 1$ onwards such that they will not affect the spread of the fire. TEMPORAL FIREFIGHTER on such a clique will then simulate FIREFIGHTER on G . A sketch of this construction can be seen in Figure 1.

We in fact prove the stronger result that TEMPORAL FIREFIGHTER is hard on cliques of n vertices with lifetime of less than $n^{\frac{1}{c}}$ for any positive integer constant c . This reduction operates by adding $n^c - n$ vertices to a static graph, and assigning times in such a way that they will all burn immediately, without affecting the spread of the fire over the existing graph. All defences then take place on a clique constructed in the same manner as that described above.



■ **Figure 1** An example of the reduction for TEMPORAL FIREFIGHTER on cliques.

We first show that we can add edges to a temporal graph in such a way that they do not affect which vertices can be saved.

► **Lemma 9.** *Suppose there is a strategy $S = v_1, \dots, v_\ell$ for TEMPORAL FIREFIGHTER on the rooted temporal graph $((V, E), \lambda), r$ that saves k vertices. Let F be any set of additional edges not in E , and $\lambda' : E \cup F \rightarrow \mathbb{N}$ be a labelling function with $\lambda'|_E = \lambda$ and $\min(\lambda'(f)) \geq |V| - 1$ for all $f \in F$. Let S' be the strategy consisting of all the defences in S followed by defending every remaining undefended vertex in an arbitrary order. S' will then save k vertices in TEMPORAL FIREFIGHTER on $((V, E \cup F), \lambda'), r$.*

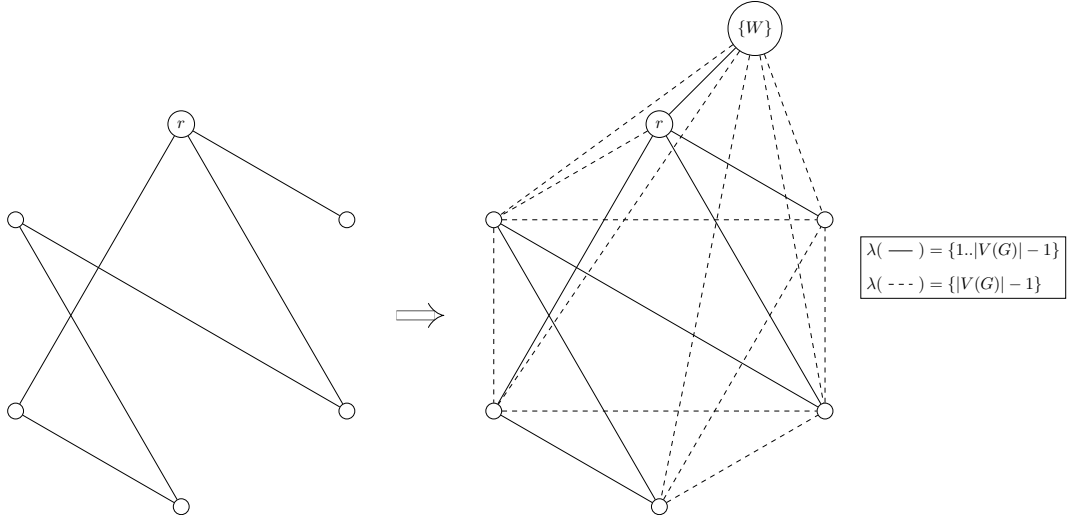
Proof. For each timestep $t \leq \ell$ consider any vertex v that does not burn by the end of timestep t when the first t defences from S are played on $((V, E), \lambda), r$. We will show by induction on t that this vertex does not burn when the first t defences from S are played on $((V, E \cup F), \lambda'), r$. See that in particular this allows us to inductively assume that the defences are valid on $((V, E \cup F), \lambda'), r$, as in particular the inductive hypothesis will imply that for any $t' \leq t$, a defence v'_t will not burn before the end of timestep $t' - 1$ on $((V, E \cup F), \lambda'), r$.

If $t = 0$ then the only vertex to have burnt in both graphs is r . Otherwise, consider all the paths from r to v in $((V, E), \lambda), r$. As v does not burn, each of these paths either contains a defended vertex or has an arrival time greater than t . Now consider the paths from r to v in $((V, E \cup F), \lambda'), r$. For each of these paths, either it is one of the aforementioned paths from $((V, E), \lambda), r$, or contains an edge from F and thus has an arrival time of at least $|V| - 1 > t$. In either case, the fire cannot have burnt along the path to v , and thus v does not burn.

Finally note that there is time to make the extra defences in S' before the fire spreads down the additional edges in F , as all vertices in the graph must be defended by the time these edges are active. ◀

We are now ready to give the reduction. This result allows us to determine that TEMPORAL FIREFIGHTER is NP-Complete on the class of temporal graphs $\{((G, \lambda), r) : (G, r) \in \mathcal{C}\}$ for all but one of the classes \mathcal{C} for which it is known that FIREFIGHTER has a polynomial time solution.

► **Theorem 10.** *For any constant $c \in \mathbb{N}$, TEMPORAL FIREFIGHTER is NP-Complete when restricted to temporal graphs whose underlying graph is a clique and whose lifetime is at most $n^{\frac{1}{c}}$ where n is the number of vertices in the graph.*



■ **Figure 2** An example of the reduction for TEMPORAL FIREFIGHTER on cliques with bounded lifetime. The vertex marked $\{W\}$ represents the set W containing $|V(G)|^c - |V(G)|$ vertices.

Proof. We give a reduction from FIREFIGHTER; given an instance $((G, r), k)$ of FIREFIGHTER and a constant c we construct an instance $((G', \lambda), r, k)$ of TEMPORAL FIREFIGHTER which is a yes-instance if and only if $((G, r), k)$ is a yes-instance of FIREFIGHTER.

Letting $\ell = |V(G)|$, we now construct an instance $((G', \lambda), r, k)$ of TEMPORAL FIREFIGHTER with ℓ^c vertices as follows. Let W be a set of $\ell^c - \ell$ vertices not in G . Then let G' be the graph (V', E') , with $V' = V(G) \cup W$, and E' containing edges connecting every pair of distinct vertices in V' , making the graph a clique, as shown in Figure 2. We then define λ as follows:

$$\lambda(e) = \begin{cases} \{1, 2, \dots, \ell - 2\} & \text{if } e \in E(G) \\ & \text{or } e = rv \text{ and } v \in W \\ \{\ell - 1\} & \text{otherwise.} \end{cases}$$

Note that the lifetime of this instance is $\ell - 1$, which is less than $|V(G')|^{\frac{1}{c}} = \ell$, as required.

We now show that if $((G, r), k)$ is a yes-instance of FIREFIGHTER then $((G', \lambda), r, k)$ is a yes-instance of TEMPORAL FIREFIGHTER. If $((G, r), k)$ is a yes-instance then there is a strategy S for FIREFIGHTER on (G, r) that saves at least k vertices. We claim that we can save at least k vertices in TEMPORAL FIREFIGHTER on $((G', \lambda), r)$ by first playing the defences from S , and then defending in arbitrary order the remaining unburnt vertices in $V(G') \setminus W$.

If we play the defences from S , then every vertex in W burns on the first timestep, and we are left with only the vertices in $V(G)$ to defend. Any path from a vertex in W to a vertex in $V(G)$ other than those that go via the root has an arrival time of at least $\ell - 1$, and we have at most $\ell - 2$ vertices left to defend after W burns, so the process must have ended before the fire spreads from W into $V(G)$. We can then consider only the spread of the fire over the subgraph of $((G', \lambda), r)$ induced by $V(G)$, and this is the temporal graph $((V(G), E(G) \cup F), \lambda')$ where F contains edges connecting every pair of vertices in $V(G)$ not connected by edges in $E(G)$, and λ' is defined as follows.

$$\lambda'(e) = \begin{cases} \{1, 2, \dots, \ell - 2\} & \text{if } e \in E(G) \\ \{\ell - 1\} & \text{otherwise} \end{cases}$$

We know from Observation 7 that the strategy S will save at least k vertices in any temporal graph with G as the underlying graph, and then from Lemma 9 we know that it is possible to save at least k vertices on $((V(G), E(G) \cup F), \lambda')$, and thus $((G', \lambda), r, k)$ is a yes-instance.

To show the converse, we first argue that if it is possible to save k vertices in TEMPORAL FIREFIGHTER on $((G', \lambda), r)$ then in particular it is possible to do this without defending any vertices in W . It is only possible to defend a vertex in W on the first timestep, as every undefended vertex in W will burn on timestep 1. As G is connected, there must be at least one vertex v in $V(G)$ on $((G', \lambda), r)$ that is connected to r by an edge active at times $\{1, 2, \dots, \ell - 2\}$, and so v will burn on the first timestep if a vertex in W is defended. Thus, defending v instead of a vertex in W on the first timestep saves at least as many vertices.

Next we observe that in any strategy that does not defend a vertex in W , the fire stops spreading by timestep $\ell - 1$, as every vertex in W burns instantly on timestep 1, and by timestep $\ell - 1$ it must be the case that every vertex in $V(G)$ is burnt or defended.

It follows that if $((G', \lambda), r, k)$ is a yes-instance then there is a strategy for TEMPORAL FIREFIGHTER on $((G', \lambda), r)$ that saves at least k vertices, and does not defend any vertices in W .

We now see that this same strategy is valid for FIREFIGHTER on (G, r) . Firstly, it does not defend any vertices in W . Secondly, if we consider any defence v_i in the strategy, then we can see that the paths from the root r to v_i in (G, r) are all also present in $((G', \lambda), r)$ and have arrival times equal to their length. If v_i does not burn in TEMPORAL FIREFIGHTER on $((G', \lambda), r)$ then every such path is either defended, or has length greater than i . Thus, if we inductively assume the first $i - 1$ defences from S are valid on (G, r) , we can see that when these defences are played v_i cannot burn in FIREFIGHTER on (G, r) , as every undefended path from r to v_i must have length greater than i .

Furthermore if a vertex v does not burn in TEMPORAL FIREFIGHTER on $((G', \lambda), r)$ then it must not burn in FIREFIGHTER on (G, r) , as the temporally admissible paths between r and v in (G', λ) are a superset of the paths between r and v in G . If v does not burn in TEMPORAL FIREFIGHTER then each of these paths either contains a defended vertex, or has an arrival time of $\ell - 1$, and is not present in G . Therefore it is possible to save at least k vertices on (G, r) , and $((G, r), k)$ is also a yes-instance. ◀

As a result we can deduce that TEMPORAL FIREFIGHTER is NP-Complete on several clique containing classes for which FIREFIGHTER is in P. For the same reason, we can determine that TEMPORAL FIREFIGHTER is NP-Complete on AT-free graphs, a class for which the complexity of FIREFIGHTER is still an open problem.

► **Corollary 11.** *TEMPORAL FIREFIGHTER is NP-Complete on split graphs, unit interval graphs, cographs, P_k -free graphs for $k > 2$, and AT-free graphs.*

We have seen, TEMPORAL FIREFIGHTER is hard on several graph classes for which FIREFIGHTER is easy. However there is one non-trivial class for which both FIREFIGHTER and TEMPORAL FIREFIGHTER are easy, that being the class of graphs of maximum degree three, with a root of degree at most two.

A proof that FIREFIGHTER is easy on this class is given by Finbow et al. [9]. This proof works due to the fact that it is always optimal to restrict the fire to spreading down only one path on such a graph. An algorithm need only find the shortest path at which the fire can be contained at the end, and then defend accordingly. Exactly the same can be done for TEMPORAL FIREFIGHTER, the only difference being in calculating where the fire can be contained – sometimes the active times of the edges allow the fire to be contained at a vertex in TEMPORAL FIREFIGHTER where it could not be contained in FIREFIGHTER.

We present a strategy S for TEMPORAL FIREFIGHTER on a temporal graph $((G, \lambda), r)$ of maximum degree three where r is of degree two, and show that this strategy is optimal and computable in polynomial time. This strategy and proof only requires slight modifications from that given by Finbow et al. for FIREFIGHTER [9].

Throughout, for any two vertices v and u in a temporal graph (G, λ) let $\text{dist}(v, u)$ be the number of edges on the shortest path between v and u in the underlying graph G .

After defining the strategy S we show that no strategy that does not always defend next to the fire can outperform S , and thus that there always exists an optimal strategy which only defends next to the fire. Such a strategy, due to the degree restriction, limits the fire to spreading along a single path. An optimal strategy then finds the shortest of such paths to a vertex at which the spread of the fire can be stopped. We observe that this can be done at any vertex u where there are one or less incident edges not on the path and active on timestep $\text{dist}(r, u) + 1$. Stated otherwise, as soon as the fire reaches a vertex at which the temporal nature of the graph delays its spread, it is possible to contain it, and in an optimal strategy the fire will spread along the path to such a vertex at a rate of one vertex per timestep, just as in FIREFIGHTER. We then show that strategy S is exactly this strategy, and then that the number of vertices saved by such an optimal strategy can be computed in polynomial time, as required.

We begin by defining three sets that will be used in the strategy: V_0 , V_1 , and V_c .

V_0 and V_1 are the sets of all vertices u that at time $\text{dist}(r, u) + 1$ are temporally adjacent, respectively, to 0 and 1 vertices not on the shortest underlying path between r and u . V_c is the set of all vertices that lie on a cycle and are not in V_0 or V_1 . Additionally for any vertex u , $C(u)$ denotes the length of the shortest cycle containing u .

Strategy S operates by first finding a vertex $u \in V_0 \cup V_1 \cup V_c$ that minimizes the function $f(u)$, defined below.

$$f(u) = \begin{cases} \text{dist}(r, u) + 1 & \text{if } u \in V_0 \cup V_1 \\ \text{dist}(r, u) + C(u) - 1 & \text{if } u \in V_c \end{cases}$$

If $u \in V_0 \cup V_1$, then let P be the shortest path from r to u on the underlying graph G . As u minimizes f , this path will always be temporally admissible and have an arrival time equal to its length.

If the path was not temporally admissible, or did not have an arrival time equal to its length, then there would be a vertex v on the path and closer to r than u which would not be temporally adjacent to the next vertex on the path. Thus $v \in V_0 \cup V_1$, and $f(v) < f(u)$.

The strategy is then to always defend the vertex adjacent to the fire that does not lie on P , up until turn $f(u)$. On turn $f(u)$ a non-burning neighbour of u should be defended, prioritising a temporally adjacent neighbour if one exists. If there is a further non-burning neighbour of u , this should be defended on turn $f(u) + 1$. Once the fire stops spreading, the burnt vertices will be all those on P , meaning that in total $f(u)$ vertices will be burnt.

Otherwise, if $u \in V_c$, then let C be the shortest cycle containing u , and P the shortest path from r to u .

Note that as $f(u)$ is minimal, it must be the case that P is either of length 0, or does not contain any edges of C . If it did, then there would be a vertex v on P and C with a neighbour on P but not on C . As v lies on the shortest path between r and u , it is necessarily closer to r than u . Additionally, as v lies on the same cycle as u , we would have that $C(v) \leq C(u)$, and thus $f(v) \leq f(u)$ which is impossible.

The strategy is then to always defend the vertex adjacent to the fire that does not lie on P , up until turn $\text{dist}(r, u) + 1$. On turn $\text{dist}(r, u) + 1$, one of the two non-burning vertices on C adjacent to the fire should be defended. On each following turn, the vertex adjacent to

the fire but not on C should be defended. Once the fire stops spreading, the burnt vertices will be all those on P and all those on C except one, meaning once again $f(u)$ vertices are burnt in total.

We now show that there is an optimal strategy that always defends next to the fire. The argument here is equivalent to that for FIREFIGHTER [9], but uses comparison to our newly defined strategy S to show optimality.

► **Lemma 12.** *Given a rooted temporal graph $((G, \lambda), r)$ of maximum degree 3 and with a root of degree 2, there is an optimum strategy that always defends next to the fire.*

Proof. Assume there is some counterexample minimal in number of vertices, that is a graph $((G, \lambda), r)$ with no optimal strategy that always defends next to the fire. Let x_1 and x_2 be the two neighbours of r . If there is an optimal strategy in which the first vertex defended is a neighbour of r , say x_1 without loss of generality, then $((G - \{r, x_1\}, \lambda), x_2)$ is a smaller counterexample – a contradiction.

Let T be some optimal strategy for TEMPORAL FIREFIGHTER on $((G, \lambda), r)$, and let u be the closest vertex to r defended in T . This cannot be a neighbour of r , and thus $\text{dist}(r, u) \geq 2$.

If no neighbours of u are burning once the fire stops spreading, then there are no temporally admissible paths between r and u . In this case T wastes a defence defending u , a vertex that will never burn, and is thus non-optimal, a contradiction.

If only one neighbour of u is burning once the fire stops spreading, then defending this neighbour instead of u saves one more vertex, and thus T is once again non-optimal.

If once the fire stops spreading two neighbours of u are burning, then u lies on a cycle that is completely burnt except for u . In this case we argue that the strategy S must save at least as many vertices. Strategy S finds a vertex v that minimizes $f(v)$, and always causes $f(v)$ vertices to burn, saving the rest. If T performs better than S , then there is a vertex $w \in V_c$ lying on the same cycle as u where the entire path between r and w has burnt, as well as the entire cycle except for u , thus meaning that $f(w) < f(v)$ vertices burn. This is impossible – $f(v)$ is a minimum. As S always defends next to the fire, its optimality contradicts the assumption. ◀

We now show that strategy S is an optimal strategy, and thus that TEMPORAL FIREFIGHTER is in P for temporal graphs of maximum degree 3 with roots of degree 2.

► **Theorem 13.** *TEMPORAL FIREFIGHTER can be solved in polynomial time on a rooted temporal graph $((G, \lambda), r)$ of maximum degree 3 with a root of degree at most 2.*

Proof. First we note that if strategy S is played on the graph $((G, \lambda), r)$ then $\min\{f(u) \mid u \in V_0 \cup V_1 \cup V_c\}$ vertices will burn, and furthermore this value can be computed in polynomial time. We now show that strategy S is optimal, and thus TEMPORAL FIREFIGHTER is in P for temporal graphs of maximum degree 3 with roots of degree 2.

By Lemma 12 there is an optimal strategy T in which each vertex defended is next to the fire, thus restricting the fire to spreading down a single path. Let w be the final vertex to burn, at the end of this path.

Due to the degree restriction there are at most two vertices adjacent to w that do not lie on the path from r down which the fire burnt to reach w . There are two ways in which the fire can stop spreading at w . In the first case both of these vertices are defended after the fire reaches w , and in the second at least one of these vertices has already been defended before the fire reaches w , and any undefended neighbours are defended afterwards.

15:10 Making Life More Confusing for Firefighters

In the first case, we must have that $w \in V_0 \cup V_1$, as there must have been time to make these defences after the fire reached w . Furthermore at least $\text{dist}(r, w) + 1$ vertices must have burnt, and strategy S performs at least as well, and is therefore optimal.

Otherwise, in the second case, w must lie on a cycle that is fully burnt except for one vertex, as the already defended vertex must be adjacent to some burning vertex, and therefore adjacent to a vertex that lies on the burnt path from r to w , as these are the only vertices that burn. Let v be the first vertex on C to have burnt. A path from r to v must be fully burnt, as is all of C except for one vertex. Thus $f(v)$ vertices have burnt in total. As strategy S finds a vertex u that minimises $f(u)$, and allows $f(u)$ vertices to burn, it must be the case that $f(u) = f(v)$, and thus strategy S is optimal. ◀

4 Restricting the Temporal Structure

As we have seen, our firefighters are going to have a hard time deciding on an optimal deployment strategy regardless of the layout of the archipelago. Our analysis of the complexity of TEMPORAL FIREFIGHTER when restricting the underlying graph class shows that for most known graph classes \mathcal{C} where FIREFIGHTER is polytime solvable, TEMPORAL FIREFIGHTER is NP-Complete on the class of temporal graphs $\{(G, \lambda) : G \in \mathcal{C}\}$. This naturally leads us to consider whether the firefighters might be able to make use of some structure in the availability of the islanders, rather than the static layout of the islands. We now discuss the tractability of TEMPORAL FIREFIGHTER when restricting the temporal structure of the graph.

We show that TEMPORAL FIREFIGHTER is fixed parameter tractable when parameterised by vertex-interval-membership-width. Intuitively, bounding this parameter limits how active the graph can be on any given timestep.

Vertex-interval-membership-width, along with the vertex interval membership sequence, was defined by Bumpus and Meeks [6]. Begin by letting $\text{mintime}(v)$ denote the minimum timestep upon which an incident edge of v is active for all vertices v . Define maxtime equivalently for the maximum timestep.

► **Definition 14** (Vertex Interval Membership Width). *The vertex interval membership sequence of a temporal graph (G, λ) is the sequence $(F_t)_{t \in [\Lambda]}$ of vertex-subsets of G where $F_t = \{v \in V(G) : \text{mintime}(v) \leq t \leq \text{maxtime}(v)\}$ and Λ is the lifetime of (G, λ) .*

The vertex-interval-membership-width of a temporal graph (G, λ) is then the integer $\omega = \max_{t \in [\Lambda]} |F_t|$.

Note that a vertex v can only be in a sequence of consecutive members of the interval membership sequence. That is, it is impossible for there to be a vertex v and times s, t and u such that $s < t < u$ where v is in F_s and F_u but not in F_t . Furthermore, the vertex interval membership sequence of a graph can be computed in polynomial time [6], and thus so can the vertex-interval-membership-width.

We find that TEMPORAL FIREFIGHTER is FPT when parameterised by vertex-interval-membership-width. To simplify our analysis when showing this, we actually use the related problem TEMPORAL FIREFIGHTER RESERVE.

TEMPORAL FIREFIGHTER RESERVE is the temporal extension of the FIREFIGHTER RESERVE problem described by Fomin et al. [12]. In TEMPORAL FIREFIGHTER RESERVE, it is not required to make a defensive move every timestep. Rather, each timestep a budget is incremented by 1, and it is then possible to defend any number of vertices less than or equal to the budget simultaneously, subtracting from the budget appropriately.

Just as in the static case, allowing the defence to build up a reserve in this manner does not affect the number of vertices than can be saved. In fact, the proof works identically to that for the static case as given by Fomin et al. [12].

► **Lemma 15.** *It is possible to save at least k vertices in TEMPORAL FIREFIGHTER RESERVE on $((G, \lambda), r)$ if and only if it is possible to save at least k vertices in TEMPORAL FIREFIGHTER RESERVE on $((G, \lambda), r)$.*

Proof. Given a temporal graph $((G, \lambda), r)$, assume there is a strategy for TEMPORAL FIREFIGHTER RESERVE that saves k vertices. Any strategy for TEMPORAL FIREFIGHTER is a valid strategy for TEMPORAL FIREFIGHTER RESERVE, and thus it is also possible to save k vertices in TEMPORAL FIREFIGHTER RESERVE by playing the same strategy.

Now assume that there is a strategy that saves k vertices in TEMPORAL FIREFIGHTER RESERVE on $((G, \lambda), r)$. We can transform this strategy into a valid strategy for TEMPORAL FIREFIGHTER that saves the same number of vertices as follows: if at any timestep t the strategy defends $d > 1$ vertices, there must have been $d - 1$ timesteps at some point prior to this where no defences were made. By making exactly one of these d defences on each of these $d - 1$ prior timesteps, and timestep t itself, we produce a valid strategy for TEMPORAL FIREFIGHTER. Modifying the strategy in this manner creates a valid strategy, as if defending vertex v is valid on timestep t , it must also be valid at any timestep less than t . Finally, as the exact same defences occur, only at an earlier time, the modified strategy must also save at least k vertices. ◀

Additionally, we note that in TEMPORAL FIREFIGHTER RESERVE there is always an optimal strategy that only defends temporally adjacent to the fire, as any defence can be delayed until the turn upon which the defended vertex would burn. More generally, there is always an optimal strategy which defends only vertices at time i if they have an incident edge active at time i . From now on when we refer to strategies for TEMPORAL FIREFIGHTER RESERVE, we assume that they all have this property.

We now give an algorithm for TEMPORAL FIREFIGHTER RESERVE that iterates over the vertex interval membership sequence of the input graph, and show that it is an FPT-algorithm with respect to vertex-interval-membership-width.

The algorithm takes as input a rooted temporal graph $((G, \lambda), r)$, and an integer k , and determines if it is possible to save k vertices in temporal firefighter reserve played on the graph. For any edge set A , let $V(A)$ be the set of vertices with an incident edge in A . The algorithm then operates by recursively computing a sequence of sets $L_i \in \mathcal{P}(F_i) \times \mathcal{P}(F_i) \times \{1, 2, \dots, \Lambda\} \times \{1, 2, \dots, n\}$ for each F_i in the vertex interval membership sequence of the input graph.

An element of L_i is a 4-tuple (D, B, g, c) where D is a set of defended vertices in F_i , B is a set of burnt vertices in F_i , g is the budget that will be available on timestep $i + 1$, and c is the total count of vertices that have burnt at time i .

To determine the spread of the fire it is only necessary to keep track of the vertices that have burnt or been defended in F_i , as if a vertex is not in F_i all its incident edges must either only be active before time i , or after time i . If the former is the case, then the fire cannot spread from or to it after time i , meaning that whether it is burning or defended does not affect the spread of the fire after this point. If the latter is the case then the vertex cannot be burning, as the fire cannot have reached it yet, and as we only defend vertices with incident edges active at time i , it cannot be defended either.

Additionally, it is possible to compute these defended and burning sets recursively from only a previous entry in the sequence, as a vertex v can only be in a sequence of consecutive F_i s.

15:12 Making Life More Confusing for Firefighters

The problem can then be answered by checking if there is any entry $(D, B, g, c) \in L_\Lambda$ where Λ is the lifetime of the graph, such that $|V(G)| - c \geq k$.

We recursively compute the sequence L_i , beginning by initialising $L_0 = (\emptyset, \{r\}, 1, 1)$. We let E_i be the set of edges active at time i , and $N_i(S)$ the set of all vertices temporally adjacent at time i to the vertices in S , for any set $S \subseteq V(G)$. Note that $V(E_i) \subseteq F_i$, as if a vertex has an incident edge active at time i , it is certainly in F_i .

We then require that, for any set $A \subseteq V(E_i) \setminus (B \cup D)$ containing vertices to be defended on timestep i , $(D', B', g - |A| + 1, c') \in \mathcal{P}(F_i) \times \mathcal{P}(F_i) \times \{1, 2, \dots, \Lambda\} \times \{1, 2, \dots, n\}$ is in L_i if and only if there is a tuple (D, B, g, c) in L_{i-1} , such that:

- (1) $D' = (D \cap F_i) \cup A$
- (2) $B' = (B \cap F_i) \cup N_i(B) \setminus D'$
- (3) $g - |A| + 1 > 0$
- (4) $c' = c + |N_i(B) \setminus (B \cup D')|$

That is, given sets of burning and defended vertices in F_{i-1} , we consider all the possible defences on vertices with incident edges active at time i , and create sets of burning and defended vertices in F_i appropriately.

Condition 1 ensures that the defended set contains only vertices with incident edges in F_i , and contains the set of new defences A .

Condition 2 specifies that the burning set contains only vertices with incident edges in F_i , and that all non-defended vertices temporally adjacent to the fire burn.

Condition 3 ensures that the budget is correct. The budget available on timestep $i + 2$ will be $g - d + 1$ if on timestep $i + 1$ the budget is g , and d vertices are to be defended, as the budget decreases by the number of defences made, but increases by 1 per timestep.

Condition 4 counts the number of newly burnt vertices, ensuring that there only exists an entry with burnt vertex count c if there is a corresponding strategy on which c vertices burn by timestep i .

We now show that computing these sets correctly answers the TEMPORAL FIREFIGHTER problem. That is that entries exist in the sequence if and only if there is a corresponding strategy, and thus it is possible to check if k vertices can be saved in temporal firefighter on the rooted temporal graph $((G, \lambda), r)$.

► **Theorem 16.** *Given a temporal graph $((G, \lambda), r)$ there is an entry $(D, B, g, c) \in L_i$ if and only if there exists a strategy for TEMPORAL FIREFIGHTER RESERVE on $((G, \lambda), r)$ such that D and B correspond to the vertices in F_i that are defended and burnt respectively by timestep i , g is the budget that will be available on timestep $i + 1$, and c is the total number of vertices that burn by timestep i .*

Proof. We proceed by induction on i . After timestep 0, only one vertex (the root) has burnt, and $L_0 = \{(\emptyset, \{r\}, 1, 1)\}$. Now suppose that the result holds at timestep $i - 1$.

We now show that if $(D', B', g - d + 1, c') \in L_i$ then there is a corresponding strategy. For $(D', B', g - d + 1, c')$ to be in L_i there must be an entry $(D, B, g, c) \in L_{i-1}$ and set of vertices $A \subseteq V(E_i) \setminus (B \cup D)$ with $d = |A|$ such that conditions 1 through 4 hold. By our induction hypothesis there is a corresponding strategy S_{i-1} for this entry such that D and B are the vertices in F_{i-1} that are defended and burnt respectively by timestep $i - 1$, g is the budget available at the end of timestep $i - 1$, and c is the total number of vertices burnt by timestep $i - 1$. If we take S_{i-1} and extend it by defending the set of vertices A on timestep i , then we obtain a strategy S_i that we claim corresponds to $(D', B', g - d + 1, c')$.

First see that by the definition of A , all the defences it contains are valid, as A contains only vertices in $V(E_i) \subseteq F_i$ that have not either already burnt or been defended.

The vertices that are newly defended on timestep i in S_i are only those in A . Thus the vertices that are defended in F_i on timestep i in S_i are those that were already defended and are also in F_i , that is $D \cap F_i$, and those that are newly defended. Thus by condition **1** the vertices in F_i that are defended by timestep i in S_i are those in D' .

The vertices that then burn on timestep i in S_i are all those temporally adjacent to the fire and not defended. Additionally, any vertex from which the fire spreads on timestep i must be in F_i , as it must have an incident edge active at time i . For the same reason, any defended vertex that the fire would otherwise burn on timestep i must also be in F_i . Thus, $N_i(B) \setminus D'$ is the set of vertices that newly burn. Therefore the vertices that have burnt in F_i on timestep i in S_i are those that had already burnt and are also in F_i , that is $B \cap F_i$, and those that newly burn: $N_i(B) \setminus D'$. Thus by condition **2** the vertices in F_i that have burnt by timestep i in S_i are those in B' .

The budget available on timestep $i + 1$ in S_i is the budget available on timestep i incremented by 1, with $|A|$, the number of defences made on timestep i , subtracted. Thus by condition **3** the budget available at timestep $i + 1$ in S_i is $g - d + 1$.

The set of vertices that newly burn after timestep i in S_i is all those temporally adjacent to the fire and not defended or already burning, so the number of such vertices is $|N_i(B) \setminus (B \cup D')|$. The total number of vertices to have burnt after timestep i in S_i is then $c + |N_i(B) \setminus (B \cup D')|$, thus by condition **4** the total number of vertices to have burnt is c' .

We now show the converse: that if there is a strategy S such that after time i the sets of vertices that have been defended and burnt are D_S and B_S respectively, g' is the available budget, and c' is the total number of vertices to have burnt, then there is a corresponding entry $(D', B', g', c') \in L_i$, such that $D' = D_S \cap F_i$ and $B' = B_S \cap F_i$.

Consider the state at timestep $i - 1$ if strategy S is played. By our induction hypothesis there is a corresponding entry $(D, B, g, c) \in L_{i-1}$ where D is the set of vertices in F_{i-1} that are defended at time $i - 1$, B is the set of vertices in F_{i-1} that are burnt at time $i - 1$, g is the budget that will be available at time i , and c is the total number of vertices to have burnt at time $i - 1$.

Let A be the vertices defended at time i in strategy S . As we consider only strategies that only defend vertices at time i with incident edges at time i , and A is a valid defence, we have that $A \subseteq V(E_i) \setminus (B \cup D)$.

By our induction hypothesis D is the set of vertices in F_{i-1} that are defended by time $i - 1$, so the set of vertices in F_i defended by time i is $D_S \cap F_i = (D \cap F_i) \cup A$.

Again by the induction hypothesis B is the set of vertices in F_{i-1} that are burnt by time $i - 1$. The only vertices from which the fire can spread on timestep i are those that have an incident edge active at time i , and thus are in F_i . For the same reason the only defended vertices that would otherwise burn on timestep i are in F_i . Therefore the vertices in F_i burnt by time i are $B_S \cap F_i = (B \cap F_i) \cup N_i(B) \setminus D_S = (B \cap F_i) \cup N_i(B) \setminus (D \cup A)$.

Finally, the budget available at time i is g , and so the budget at time $i + 1$ is $g' = g - |A| + 1$, and the number of vertices burnt after time $i - 1$ is c , so the number of vertices burnt after time i is $c' = c + |N_i(B) \setminus (B \cup D_S)| = c + |N_i(B) \setminus (B \cup (D \cap F_i) \cup A)|$.

Thus we see that, given (D, B, g, c) is an entry in L_{i-1} , we have that $(D_S \cap F_i, B_S \cap F_i, g', c')$ satisfies conditions **1-4**, and thus is an entry in L_i . ◀

We now determine the runtime of computing all sets L_i , thus showing that TEMPORAL FIREFIGHTER is FPT when parameterised by vertex-interval-membership-width.

► **Theorem 17.** *It is possible to solve TEMPORAL FIREFIGHTER in time $O(8^\omega \omega \Lambda^3)$ for a rooted temporal graph $((G, \lambda), r)$ where Λ is the lifetime of the graph, and ω is the vertex-interval-membership-width.*

Proof. TEMPORAL FIREFIGHTER RESERVE, and therefore TEMPORAL FIREFIGHTER can be answered by computing all sets L_i . Thus it suffices to show that each of these sets can be computed in the required time.

To compute L_i every possible defence on a set of vertices with incident edges active at time i must be considered for every entry in L_{i-1} .

First observe that the total number of burnt vertices on any given timestep i is at most $\sum_{j=1}^i |V(E_j)| = O(\omega\Lambda)$, as on each timestep j only vertices in $V(E_j)$ can burn, and $|V(E_j)| \leq 2|E_j| \leq 2\omega$, and for any timestep i we have that $i \leq \Lambda$.

Now see that for any i , we have that $|L_i| = O(4^\omega \omega \Lambda^2)$ as $L_i \subseteq \mathcal{P}(F_i) \times \mathcal{P}(F_i) \times \{1, \dots, \Lambda\} \times \{1, \dots, \omega\Lambda\}$, and $|\mathcal{P}(F_i)| \times |\mathcal{P}(F_i)| = 2^{2\omega} = 4^\omega$.

Furthermore, on each timestep i we only consider defending vertices in $V(E_i)$, and $|V(E_i)| \leq \omega$. Thus for each timestep there are at most 2^ω defences to consider.

As described, for each timestep i in the lifetime Λ of the graph, it is necessary to compute every possible set of defences for every entry in L_i . The overall complexity is therefore $O(4^\omega \omega \Lambda^2 \times 2^\omega \times \Lambda) = O(8^\omega \omega \Lambda^3)$ as required. ◀

5 Conclusion

In this paper we introduced the TEMPORAL FIREFIGHTER problem, an extension of FIREFIGHTER to temporal graphs. We found that this problem is, like FIREFIGHTER, NP-Complete on arbitrary graphs, and in particular is NP-Complete on any underlying graph class for which FIREFIGHTER is NP-Complete. In order to try and identify places where TEMPORAL FIREFIGHTER is tractable, we began by determining its complexity on several underlying graph classes for which it is known FIREFIGHTER can be solved in polynomial time.

Despite finding that TEMPORAL FIREFIGHTER is NP-Complete on all but one underlying graph class that we considered, we were able to find a promising avenue for tractability in restricting the temporal structure of the graph, and found that the problem is FPT when parameterised by the temporal graph parameter vertex-interval-membership-width.


An interesting direction for future work on the problem would be to further investigate such temporal parameters. A natural first goal would be to determine the complexity of TEMPORAL FIREFIGHTER when the maximum number of edges active each timestep is bounded. Furthermore, it would be worthwhile to determine the complexity of TEMPORAL FIREFIGHTER when parameterised by interval membership width, a relative of vertex-interval-membership-width also introduced by Bumpus and Meeks [6]. This measure can be arbitrarily larger than the vertex-interval-membership-width.

References

- 1 Elliot Anshelevich, Deeparnab Chakrabarty, Ameya Hate, and Chaitanya Swamy. Approximability of the firefighter problem - computing cuts over time. *Algorithmica*, 62(1-2):520–536, 2012. doi:10.1007/s00453-010-9469-y.
- 2 Kyriakos Axiotis and Dimitris Fotakis. On the size and the approximability of minimum temporally connected subgraphs. *CoRR*, abs/1602.06411, 2016. arXiv:1602.06411.
- 3 Cristina Bazgan, Morgan Chopin, Marek Cygan, Michael R. Fellows, Fedor V. Fomin, and Erik Jan van Leeuwen. Parameterized complexity of firefighting. *J. Comput. Syst. Sci.*, 80(7):1285–1297, 2014. doi:10.1016/j.jcss.2014.03.001.
- 4 Sandeep Bhadra and Afonso Ferreira. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In Samuel Pierre, Michel Barbeau, and Evangelos Kranakis, editors, *Ad-Hoc, Mobile, and Wireless Networks, Second International Conference, ADHOC-NOW 2003 Montreal, Canada, October 8-10, 2003*,

- Proceedings*, volume 2865 of *Lecture Notes in Computer Science*, pages 259–270. Springer, 2003. doi:10.1007/978-3-540-39611-6_23.
- 5 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.*, 14(2):267–285, 2003. doi:10.1142/S0129054103001728.
 - 6 Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. *CoRR*, abs/2103.05387, 2021. arXiv:2103.05387.
 - 7 Leizhen Cai, Elad Verbin, and Lin Yang. Firefighting on trees: $(1-1/e)$ -approximation, fixed parameter tractability and a subexponential algorithm. In Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, editors, *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, volume 5369 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2008. doi:10.1007/978-3-540-92182-0_25.
 - 8 Janka Chlebíková and Morgan Chopin. The firefighter problem: Further steps in understanding its complexity. *Theor. Comput. Sci.*, 676:42–51, 2017. doi:10.1016/j.tcs.2017.03.004.
 - 9 Stephen Finbow, Andrew D. King, Gary MacGillivray, and Romeo Rizzi. The firefighter problem for graphs of maximum degree three. *Discret. Math.*, 307(16):2094–2105, 2007. doi:10.1016/j.disc.2005.12.053.
 - 10 Stephen Finbow and Gary MacGillivray. The firefighter problem: a survey of results, directions and questions. *Australas. J Comb.*, 43:57–78, 2009. URL: http://ajc.maths.uq.edu.au/pdf/43/ajc_v43_p057.pdf.
 - 11 Fedor V. Fomin, Pinar Heggernes, and Erik Jan van Leeuwen. Making life easier for firefighters. In Evangelos Kranakis, Danny Krizanc, and Flaminia L. Luccio, editors, *Fun with Algorithms - 6th International Conference, FUN 2012, Venice, Italy, June 4-6, 2012. Proceedings*, volume 7288 of *Lecture Notes in Computer Science*, pages 177–188. Springer, 2012. doi:10.1007/978-3-642-30347-0_19.
 - 12 Fedor V. Fomin, Pinar Heggernes, and Erik Jan van Leeuwen. The firefighter problem on graph classes. *Theor. Comput. Sci.*, 613:38–50, 2016. doi:10.1016/j.tcs.2015.11.024.
 - 13 Samuel Hand, Jessica Enright, and Kitty Meeks. Making life more confusing for firefighters, 2022. arXiv:2202.12599.
 - 14 Bert Hartnell. Firefighter! an application of domination. In *the 24th Manitoba Conference on Combinatorial Mathematics and Computing, University of Minitoba, Winnipeg, Cadada, 1995*, 1995.
 - 15 Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the bron-kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Soc. Netw. Anal. Min.*, 7(1):35:1–35:16, 2017. doi:10.1007/s13278-017-0455-0.
 - 16 Petter Holme and Jari Saramäki. Temporal networks. *CoRR*, abs/1108.1780, 2011. arXiv:1108.1780.
 - 17 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *J. Comput. Syst. Sci.*, 64(4):820–842, 2002. doi:10.1006/jcss.2002.1829.
 - 18 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.
 - 19 Huanhuan Wu, James Cheng, Yiping Ke, Silu Huang, Yuzhen Huang, and Hejun Wu. Efficient algorithms for temporal path computation. *IEEE Trans. Knowl. Data Eng.*, 28(11):2927–2942, 2016. doi:10.1109/TKDE.2016.2594065.

Sorting Balls and Water: Equivalence and Computational Complexity

Takehiro Ito ✉ 

Graduate School of Information Sciences, Tohoku University, Japan

Jun Kawahara ✉ 

Kyoto University, Japan

Shin-ichi Minato ✉ 


Kyoto University, Japan

Yota Otachi ✉ 

Nagoya University, Japan

Toshiki Saitoh ✉ 

Kyushu Institute of Technology, Japan

Akira Suzuki ✉ 

Graduate School of Information Sciences, Tohoku University, Japan

Ryuhei Uehara ✉ 


Japan Advanced Institute of Science and Technology, Ishikawa, Japan

Takeaki Uno ✉

National Institute of Informatics, Tokyo, Japan

Katsuhisa Yamanaka ✉ 

Iwate University, Japan

Ryo Yoshinaka ✉ 

Graduate School of Information Sciences, Tohoku University, Japan

Abstract

Various forms of sorting problems have been studied over the years. Recently, two kinds of sorting puzzle apps are popularized. In these puzzles, we are given a set of bins filled with colored units, balls or water, and some empty bins. These puzzles allow us to move colored units from a bin to another when the colors involved match in some way or the target bin is empty. The goal of these puzzles is to sort all the color units in order. We investigate computational complexities of these puzzles. We first show that these two puzzles are essentially the same from the viewpoint of solvability. That is, an instance is sortable by ball-moves if and only if it is sortable by water-moves. We also show that every yes-instance has a solution of polynomial length, which implies that these puzzles belong to NP. We then show that these puzzles are NP-complete. For some special cases, we give polynomial-time algorithms. We finally consider the number of empty bins sufficient for making all instances solvable and give non-trivial upper and lower bounds in terms of the number of filled bins and the capacity of bins.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases Ball sort puzzle, recreational mathematics, sorting pairs in bins, water sort puzzle

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.16

Related Version *Previous Version:* <https://arxiv.org/abs/2202.09495>

Funding This research is partially supported by JSPS Kakenhi Grant Number JP18H04091, Japan. *Takehiro Ito:* Partially supported by JSPS KAKENHI Grant Numbers JP19K11814 and JP20H05793, Japan.

Jun Kawahara: JSPS KAKENHI Grant Number JP20H05794, Japan.

Yota Otachi: JSPS KAKENHI Grant Numbers JP18K11168, JP18K11169, JP20H05793, JP21K11752.

Toshiki Saitoh: JSPS KAKENHI Grant Number JP19K12098 and JP21H05857, Japan.

Akira Suzuki: Partially supported by JSPS KAKENHI Grant Numbers JP20K11666 and JP20H05794, Japan.

Ryuhei Uehara: JSPS KAKENHI Grant Numbers JP20H05961, JP20H05964, JP20K11673.

Katsuhisa Yamanaka: Partially supported by JSPS KAKENHI Grant Numbers 19K11812, Japan.



© Takehiro Ito, Jun Kawahara, Shin-ichi Minato, Yota Otachi, Toshiki Saitoh, Akira Suzuki, Ryuhei Uehara, Takeaki Uno, Katsuhisa Yamanaka, and Ryo Yoshinaka; licensed under Creative Commons License CC-BY 4.0

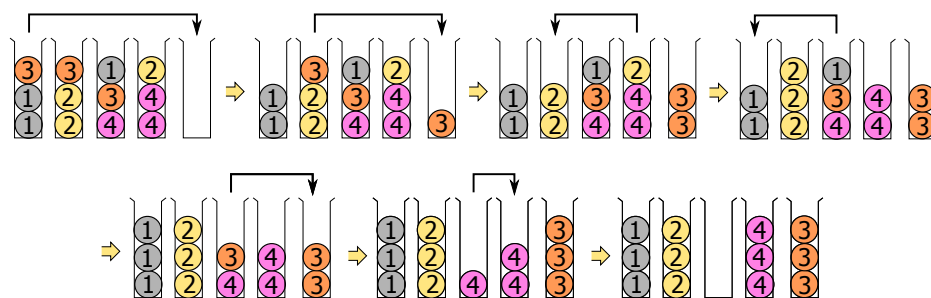
11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 16; pp. 16:1–16:17

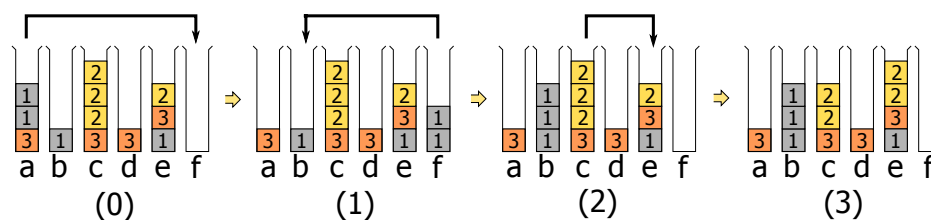
Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** An example of the ball sort puzzle.



■ **Figure 2** Rules of the water sort puzzle. From the initial configuration (0), we obtain configuration (1) by moving two units of water of label 1 from a to f, configuration (2) by moving two units of water of label 1 from a to b, and configuration (3) by moving one unit of water of label 2 from c to e.

1 Introduction

The *ball sort puzzle* and the *water sort puzzle* are popularized recently via smartphone apps.¹ Both puzzles involve bins filled with some colored units (balls or water), and the goal is to somehow sort them. The most significant feature of these puzzles is that each bin works as a stack. That is, the items in a bin have to follow the “last-in first-out” (LIFO) rule.

In the ball sort puzzle, we are given hn colored balls in n bins of capacity h and k additional empty bins. For a given (unsorted) initial configuration, the goal of this puzzle is to arrange the balls in order; that is, to make each bin either empty or full with balls of the same color. (The ordering of bins does not matter in this puzzle.) The rule of this puzzle is simple: (0) Each bin works like a stack, that is, we can pick up the top ball in the bin. (1) We can move the top ball of a bin to the top of another bin if the second bin is empty or it is not full and its top ball before the move and the moved ball have the same color. An example with $h = 3$, $n = 4$, and $k = 1$ is given in Figure 1.

The water sort puzzle is similar to the ball sort puzzle. Each ball is replaced by colored water of a unit volume in the water sort puzzle. In the water sort puzzle, the rules (0) and (1) are the same as the ball sort puzzle except one liquid property: Colored water units are merged when they have the same color and they are consecutive in a bin. When we pick up a source bin and move the top water unit(s) to a target bin, the quantity of the colored water on the top of the bin to be moved varies according to the following conditions (Figure 2). If the target bin has enough margin, all the water of the same color moves to the target bin. On the other hand, a part of the water of the same color moves up to the limit of the target bin if the target bin does not have enough margin.

¹ These sort puzzles are released by several different developers. As far as the authors checked, it seems that the first one is released by IEC Global Pty Ltd in January, 2020.

In this paper, we investigate computational complexities of the ball and water sort puzzles. We are given $n + k$ bins of capacity h . In an initial configuration, n bins are full (i.e., filled with h units) and k bins are empty, where each unit has a color in the color set C . Our task is to fill n bins monochromatically, that is, we need to fill each of them with h units of the same color. We define **BSP** and **WSP** as the problems of deciding whether a given initial configuration can be reconfigured to a sorted configuration by a sequence of ball moves and water moves, respectively. We assume that instances are encoded in a reasonable way, which takes $\Theta(nh \log |C| + \log k)$ bits. Without loss of generality, we assume that each color $c \in C$ occurs exactly hj times for some positive integer j in any instance since otherwise the instance is a trivial no-instance. (This implies that $|C|$ is at most n .)

1.1 Our results

We first prove that the problems **BSP** and **WSP** are actually equivalent. By their definitions, a yes-instance of **WSP** is a yes-instance of **BSP** as well. Thus our technical contribution here is to prove the opposite direction. As a byproduct, we show that a yes-instance of the problems admits a reconfiguration sequence of length polynomial in $n + h$ as a yes-certificate. This implies that the problems belong to NP. We also show that **BSP** and **WSP** are solvable in time $O(h^n)$.

We next show that **BSP** and **WSP** are indeed NP-complete. By slightly modifying this proof, we also show that even for some kind of *trivial* yes-instances of **WSP**, it is NP-complete to find a shortest reconfiguration to a sorted configuration.

We show that if the capacity h is 2 and the number of colors is n , then **BSP** and **WSP** are polynomial-time solvable. In this case, we present an algorithm that finds shortest reconfiguration sequences for yes-instances.

We also consider the following question: how many empty bins do we need to ensure that all initial configurations can reach a sorted configuration? Observe that **BSP** and **WSP** are trivial if $k \geq hn$. It is not difficult to see that $k \geq n$ is also sufficient by using an idea based on bucket sort. By improving this idea, we show that $k \geq \lceil \frac{h-1}{h} n \rceil$ is sufficient for all instances. We also show that some instances need $k \geq \lfloor \frac{19}{64} \min\{n, h\} \rfloor$ empty bins.

1.2 Related studies

Various forms of sorting problems have been studied over the years. For example, sorting by reversals is well investigated in the contexts of sorting network [6, Sect. 5.2.2], pancake sort [2], and ladder lotteries [8]. There is another extension of sorting problem from the viewpoint of recreational mathematics defined as follows. For each $i \in \{1, \dots, n\}$, there are h balls labeled i . These hn balls are given in n bins in which each bin is of capacity h . In one step, we are allowed to swap any pair of balls. Then how many swaps do we need to sort the balls? This sorting problem was first posed by Peter Winker for the case $h = 2$ in 2004 [7], and an almost optimal algorithm for that case was given by Ito et al. in 2012 [4].

The ball/water sorting puzzles are interesting also from the viewpoints of not only just puzzles but also combinatorial reconfiguration. Recently, *reconfiguration problems* attracted attention in theoretical computer science (see, e.g., [5]). These problems arise when we need to find a step-by-step transformation between two feasible solutions of a problem such that all intermediate results are also feasible and each step abides by a fixed reconfiguration rule, that is, an adjacency relation defined on feasible solutions of the original problem. In this sense, the ball/water sort puzzles can be seen as typical implementations of the framework of reconfiguration problems, while their reconfiguration rules are non-standard.

In most reconfiguration problems, representative reconfiguration rules are *reversible*; that is, if a feasible solution A can be reconfigured to another feasible solution B , then B can be reconfigured to A as well (see e.g., the puzzles in [3]). In the ball sort puzzle, we can reverse the last move if we have two or more balls of the same color at the top of the source bin, or there is only one ball in the source bin. Otherwise, we cannot reverse the last move. That is, some moves are reversible while some moves are one-way in these puzzles. (In Figure 1, only the last move is reversible.) In the water sort puzzle, basically, we cannot separate units of water of the same color once we merge them, however, there are some exceptional cases. An example is given in Figure 2; the move from configuration (2) to configuration (3) is reversible, but the other moves are not.

2 Equivalence of balls and water

This section presents fundamental properties of BSP and WSP, from which we will conclude that

- a configuration is a yes-instance of BSP if and only if it is a yes-instance of WSP, and
- BSP and WSP both belong to NP.

2.1 Notation used in this section

Let B and C be finite sets of *bins* and *colors*, respectively. The set of sequences of colors of length at most h is denoted by $C^{\leq h}$. A sequence of $c \in C$ of length l is called *monochrome* and denoted by c^l . The empty sequence is denoted as ε , which is, in our terminology, also called monochrome. A *configuration* is a map $S: B \rightarrow C^{\leq h}$. An instance of BSP and WSP is a configuration S such that $|S(b)| \in \{h, 0\}$ for all $b \in B$. A bin b is *sorted* under S if $S(b) \in \{\varepsilon, c^h\}$ for some $c \in C$. A configuration S is *sorted* or *goal* if all bins are sorted under S . Sometimes we identify $b \in B$ with $S(b)$ when S is clear from the context.

The i th element of a sequence α is denoted by $\alpha[i]$ for $1 \leq i \leq |\alpha|$. The *top color* of a color sequence α is $\text{TC}(\alpha) = \alpha[|\alpha|]$ if α is not empty. In case α is empty, $\text{TC}(\alpha)$ is defined to be ε . A *border* in α is a non-negative integer i such that either $1 \leq i < |\alpha|$ and $\alpha[i] \neq \alpha[i+1]$ or $i = 0$, and the set of borders of α is denoted by $\mathcal{D}(\alpha)$. We count non-trivial borders under S as $D(S) = \sum_{b \in B} |\mathcal{D}(S(b)) \setminus \{0\}|$. For example, in Figure 2, the values of D are 4, 3, 3, 3 in (0), (1), (2), (3), respectively.

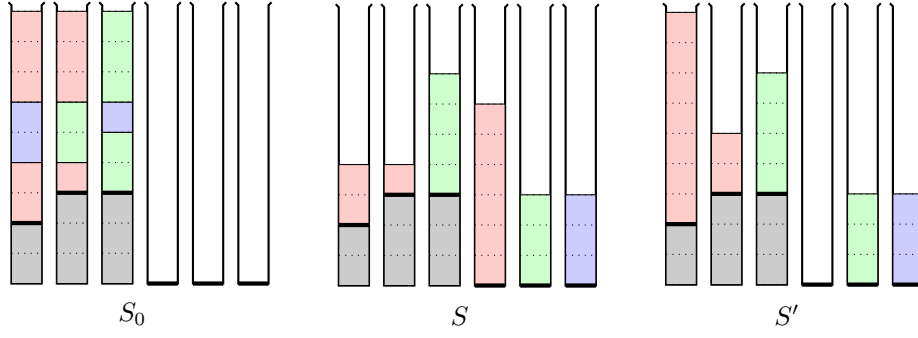
We now turn to define a move of the sort puzzles with the terminologies introduced above. Intuitively, we pick up two bins b_1 and b_2 from B , and pour the top item(s) from b_1 to b_2 if b_2 is empty or the top item of b_2 has the same color. The quantity of the items are different in the cases of balls and water. In the ball case, a unit is moved if possible. On the other hand, two or more units of water of the same color move at once until all units are moved or b_2 becomes full. We define the move more precisely below.

A *ball-move* can modify a configuration S into S' , denoted as $S \rightarrow S'$, if there are $b_1, b_2 \in B$ and $c \in C$ such that

- $S(b_1) = S'(b_1) \cdot c$, $\text{TC}(S(b_2)) \in \{c, \varepsilon\}$, $S'(b_2) = S(b_2) \cdot c$, and
- $S(b) = S'(b)$ for all $b \in B \setminus \{b_1, b_2\}$.

A *water-move* can modify S into S' , denoted as $S \Rightarrow S'$, if there are $m \geq 1$, $b_1, b_2 \in B$ and $c \in C$ such that

- $S(b_1) = S'(b_1) \cdot c^m$, $\text{TC}(S(b_2)) \in \{c, \varepsilon\}$, and $S'(b_2) = S(b_2) \cdot c^m$,
- either $\text{TC}(S'(b_1)) \neq c$ or $|S'(b_2)| = h$, and
- $S(b) = S'(b)$ for all $b \in B \setminus \{b_1, b_2\}$.



■ **Figure 3** The configuration S can be obtained from the initial configuration S_0 and can be modified into the tight one S' . The top-borders TB_S of S are shown with thick lines. This figure does not care the units below those borders. All of S_0 , S , and S' have $\mathcal{F}_{\text{red}}(\text{TB}_S) = 9$ red units, $\mathcal{F}_{\text{green}}(\text{TB}_S) = 7$ green units, and $\mathcal{F}_{\text{blue}}(\text{TB}_S) = 3$ blue units above the borders in total. The colors red, green, and blue require $\mathcal{M}_{\text{red}}(\text{TB}_S) = 0$, $\mathcal{M}_{\text{green}}(\text{TB}_S) = 1$, and $\mathcal{M}_{\text{blue}}(\text{TB}_S) = 1$ monochrome bins, respectively, which coincide with the number of the monochrome bins of respective colors in S' .

The reflexive and transitive closures of \rightarrow and \Rightarrow are denoted as \rightarrow^* and \Rightarrow^* , respectively. Clearly any single water-move can be simulated by some number of ball-moves. Thus we have $\Rightarrow^* \subseteq \rightarrow^*$.

2.2 Fundamental Theorems of BSP and WSP

Hereafter we fix an arbitrary initial configuration S_0 . Some notions introduced below are relative to S_0 , though it may not be explicit. The *top-border table* of a configuration S is a map $\text{TB}_S: B \rightarrow \{0, 1, \dots, h-1\}$ defined as $\text{TB}_S(b) = \max \mathcal{D}(S(b))$.

The main object of this section is to observe that top-border tables play an essential role in analyzing the solvability of an instance. Note that $\text{TB}_S(b) = 0$ if and only if b is monochrome. Once we have reached a configuration with $\text{TB}_S(b) = 0$ for all $b \in B$, we can achieve a goal configuration by trivial moves. Figure 3 illustrates some notions introduced in this section.

By the definition of moves, one can easily observe the following properties.

► **Observation 1.** *If $S \rightarrow^* T$, the following holds for all bins $b \in B$:*

1. *Moves monotonically remove borders from top to bottom:*

$$\mathcal{D}(T(b)) = \{i \in \mathcal{D}(S(b)) \mid i \leq \text{TB}_T(b)\};$$

2. *The contents below the top borders of T do not change:*

$$T(b)[i] = S(b)[i] \text{ for all } i \leq \text{TB}_T(b);$$

3. *If b is not monochrome in T , the colors of all the units above the border in T are the color of the unit just above that border in S :*

$$\text{TC}(T(b)) = S(b)[\text{TB}_T(b) + 1] \text{ if } \text{TB}_T(b) \neq 0.$$

We first give a necessary condition for $S_0 \rightarrow^* S$.

Recall that throughout the game play, the total number of units of each color does not change, i.e., if $S_0 \rightarrow^* S$, it holds for every color $c \in C$,

$$\sum_{b \in B} |\{i \mid 1 \leq i \leq h \text{ and } S_0(b)[i] = c\}| = \sum_{b \in B} |\{i \mid 1 \leq i \leq |S(b)| \text{ and } S(b)[i] = c\}|.$$

16:6 Sorting Balls and Water: Equivalence and Computational Complexity

Since the units under the top border of each bin have not been moved (Observation 1-2), the total numbers of units of each color c above the borders of TB_S coincide in S_0 and S . We count those units of color c as

$$\mathcal{F}_c(\text{TB}_S) = \sum_{b \in B} |\{i \mid \text{TB}_S(b) < i \leq h \text{ and } S_0(b)[i] = c\}|.$$

Those units may have been moved, but units of color c can go only to empty bins or bins whose top color is c in S . Let us partition the bins into $|C| + 1$ groups with respect to S , where

- $B_\varepsilon(\text{TB}_S) = \{b \in B \mid \text{TB}_S(b) = 0\}$: monochrome bins,
- $B_c(\text{TB}_S) = \{b \in B \setminus B_\varepsilon \mid S_0(b)[\text{TB}_S(b) + 1] = c\}$: non-monochrome bins with the top color c .

Note that if $b \notin B_\varepsilon(\text{TB}_S)$, the top color of b in S is $\text{TC}(S(b)) = S(b)[\text{TB}_S(b) + 1] = S_0(b)[\text{TB}_S(b) + 1]$. Since each bin $b \in B_c$ may have at most $h - \text{TB}_S(b)$ units of color c above the border $\text{TB}_S(b)$, there are

$$\mathcal{G}_c(\text{TB}_S) = \sum_{b \in B_c} (h - \text{TB}_S(b))$$

units of color c can be on the top layer of non-monochrome bins. Thus, we must have at least

$$\mathcal{M}_c(\text{TB}_S) = \max \left\{ 0, \left\lceil \frac{\mathcal{F}_c(\text{TB}_S) - \mathcal{G}_c(\text{TB}_S)}{h} \right\rceil \right\}$$

monochrome bins in B_ε which have some units of color c . Therefore, it is necessary that

$$\sum_{c \in C} \mathcal{M}_c(\text{TB}_S) \leq |B_\varepsilon(\text{TB}_S)|. \quad (1)$$

We remark that the values of the functions introduced above, namely \mathcal{F}_c , B_c , \mathcal{M}_c , are all determined by the top-border table TB_S . Those values coincide for S and S' if $\text{TB}_S = \text{TB}_{S'}$. Let us say that S is *consistent with S_0* if and only if TB_S satisfies Eq. (1). Moreover, S is *tight* if S has exactly $\mathcal{M}_c(\text{TB}_S)$ monochrome bins with at least one unit of every color $c \in C$.

► **Lemma 2.** *Suppose S is consistent with S_0 . There is a tight configuration S' such that $\text{TB}_S = \text{TB}_{S'}$ and $S \Rightarrow^* S'$.*

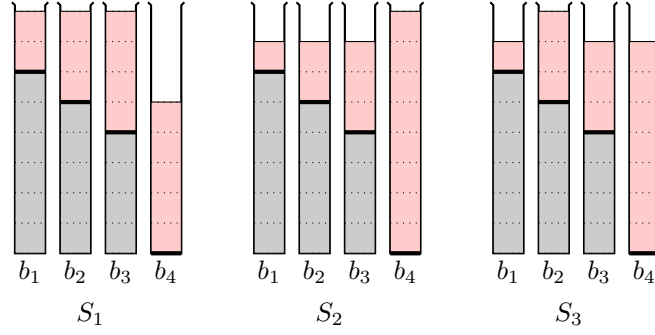
Proof. Recall that it is necessary that S has at least $\mathcal{M}_c(\text{TB}_S)$ monochrome bins of each color c . If S has more than that, one can move all the units of any of the monochrome bins of color c to other non-empty bins. The definition of \mathcal{M}_c guarantees that we can repeat this until we have just $\mathcal{M}_c(\text{TB}_S)$ monochrome bins of color c . ◀

Of course TB_S being consistent with S_0 does not imply $S_0 \rightarrow^* S$. Suppose $S_0 \rightarrow^* S \rightarrow S'$ with $\text{TB}_S \neq \text{TB}_{S'}$, where S' has one less border than S by moving a unit above the top border of some bin b to some other bins. Note that $S(b)$ is not monochrome. During the move, the bin b can keep units below its top border only. Therefore, the number of necessary monochrome bins of each color c for this move is

$$\mathcal{M}_c^b(\text{TB}_S) = \max \left\{ 0, \left\lceil \frac{\mathcal{F}_c(\text{TB}_S) - (\mathcal{G}_c(\text{TB}_S) - (h - \text{TB}_S(b)))}{h} \right\rceil \right\}$$

and it is necessary that

$$\sum_{c \in C} \mathcal{M}_c^b(\text{TB}_S) \leq |B_\varepsilon(\text{TB}_S)|.$$



■ **Figure 4** All configurations S_1 , S_2 and S_3 are tight and have the same top-border table $\tau = \text{TB}_{S_1} = \text{TB}_{S_2} = \text{TB}_{S_3}$, where $\tau(b_1) = 7$, $\tau(b_2) = 5$, and $\tau(b_3) = 4$. In those configurations, one can move the units above the top border of any of b_1 or b_2 , but it is impossible for b_3 . This solely depends on the top-border table τ and the total number $\mathcal{F}_{\text{red}}(\tau) = 16$ of red units above the top borders. It is independent of how those red units are distributed over the bins in the current configuration.

Those conditions depend on top-border configuration of S . For two top-border tables τ and τ' and a bin $b \in B$, we write $\tau \stackrel{b}{\Rightarrow} \tau'$ if

- $\tau(a) = \tau'(a)$ for all $a \in B \setminus \{b\}$,
- $\tau(b) > 0$ and $\tau'(b) = \max\{i \in \mathcal{D}(S_0(b)) \mid 0 \leq i < \tau(b)\}$,
- $\sum_{c \in C} \mathcal{M}_c^b(\tau) \leq |B_\varepsilon(\tau)|$.

We write $\tau \Rightarrow \tau'$ if $\tau \stackrel{b}{\Rightarrow} \tau'$ for some $b \in B$. Figure 4 may help intuitive understanding the above argument.

► **Theorem 3.** *Suppose a configuration S is consistent with S_0 . Then, there is a configuration T such that $S \Rightarrow^* T$ if and only if $\text{TB}_S \Rightarrow^* \text{TB}_T$, where \Rightarrow^* is the reflexive and transitive closure of \Rightarrow .*

Proof. Suppose $S \Rightarrow T$ and $\text{TB}_S \neq \text{TB}_T$. Then TB_S and TB_T must satisfy the condition for $\text{TB}_S \Rightarrow \text{TB}_T$.

Suppose $\text{TB}_S \Rightarrow \text{TB}_T$. We assume without loss of generality that S is tight by Lemma 2. The condition $\sum_{c \in C} \mathcal{M}_c^b(\tau) \leq |B_\varepsilon(\tau)|$ implies that units of color $c = S_0(b)[\text{TB}_S(b) + 1]$ in concern can be distributed to bins other than b . If $\mathcal{M}_c^b(\text{TB}_S) = \mathcal{M}_c(\text{TB}_S)$, one can move the units in b above $\text{TB}_T(b)$ to other bins whose top color is c . If $\mathcal{M}_c^b(\text{TB}_S) > \mathcal{M}_c(\text{TB}_S)$, one can move those units to an empty bin, which must be available in S , since S is tight. ◀

► **Corollary 4.** *An initial configuration S_0 is a yes-instance of BSP if and only if it is a yes-instance of WSP.*

Proof. Since any water-move can be simulated by some ball-moves, it is enough to show the converse. We claim that, for any configurations S and S' such that $\text{TB}_S = \text{TB}_{S'}$, if $S \rightarrow T$, then there is T' such that $S' \Rightarrow^* T'$ and $\text{TB}_T = \text{TB}_{T'}$.

By Theorem 3, either $\text{TB}_S = \text{TB}_T$ or $\text{TB}_S \Rightarrow \text{TB}_T$. If $\text{TB}_S = \text{TB}_T$, then $T' = S'$ proves the claim. Otherwise, the claim immediately follows from Theorem 3. Thus, if $S_0 \rightarrow^* T$ and T is a goal configuration, then one can have $S_0 \Rightarrow^* T'$ for some T' with $\text{TB}_T = \text{TB}_{T'}$. By modifying T' tight by Lemma 2, we get a goal configuration. ◀

If $\tau \stackrel{b}{\Rightarrow} \tau'$, from the values $\mathcal{F}_c(\tau)$ and $\mathcal{G}_c(\tau)$, one can compute $\mathcal{F}_c(\tau')$ and $\mathcal{G}_c(\tau')$ in constant time, by preprocessing S_0 , using the following equations:

$$\mathcal{F}_c(\tau') = \begin{cases} \mathcal{F}_c(\tau) + (\tau(b) - \tau'(b)) & \text{if } S_0(b)[\tau'(b) + 1] = c, \\ \mathcal{F}_c(\tau) & \text{otherwise,} \end{cases}$$

$$\mathcal{G}_c(\tau') = \begin{cases} \mathcal{G}_c(\tau) - (h - \tau(b)) & \text{if } S_0(b)[\tau(b) + 1] = c, \\ \mathcal{G}_c(\tau) + (h - \tau'(b)) & \text{if } S_0(b)[\tau'(b) + 1] = c \text{ and } \tau'(b) > 0, \\ \mathcal{G}_c(\tau) & \text{otherwise.} \end{cases}$$

► **Corollary 5.** *BSP and WSP belong to NP.*

Proof. By Theorem 3, S_0 is a yes-instance if and only if there is a bin sequence (b_1, \dots, b_m) with $m = D(S_0)$ ($= \sum_{b \in B} |\mathcal{D}(S_0(b)) \setminus \{0\}|$) that admits a top-border table sequence (τ_0, \dots, τ_m) such that $\tau_0 = \text{TB}_{S_0}$ and $\tau_{i-1} \stackrel{b_i}{\Rightarrow} \tau_i$ for all i . Each τ_i is uniquely determined by τ_{i-1} and b_i and one can verify $\tau_{i-1} \stackrel{b_i}{\Rightarrow} \tau_i$ in constant time, by maintaining the values of \mathcal{F}_c and \mathcal{G}_c (or just $\mathcal{F}_c - \mathcal{G}_c$). ◀

A solution for an instance is essentially an order of borders of the instance to remove.

► **Corollary 6.** *BSP and WSP can be solved in $O(h^n)$ time.*

Proof. There are at most $\prod_{b \in B} |\mathcal{D}(S_0(b))| \leq h^n$ distinct top-border tables. ◀

► **Corollary 7.** *For every yes-instance of BSP, there exists a sequence of ball-moves to a goal configuration of length at most $(2h - 1)hn$.*

Proof. In BSP, $h - 1$ ball-moves are enough to eliminate a border from a tight configuration, if there is a way to eliminate the border. To make the obtained configuration tight takes at most h moves, since the obtained configuration may have at most one monochrome bin to empty when the previous configuration is tight. Since there can be at most $(h - 1)n$ borders to eliminate, every yes-instance has a solution consisting of at most $(2h - 1)hn$ ball-moves. ◀

► **Corollary 8.** *For every yes-instance of WSP, there exists a sequence of water-moves to a goal configuration of length at most $2(h - 1)nl$ where $l = \min\{h, n\}$.*

Proof. In WSP, $l = \min\{h, n\}$ water-moves are enough to eliminate a border from a tight configuration, if there is a way to eliminate the border. To make the obtained configuration tight, it takes again l moves. To see this, observe that the obtained configuration may have at most one monochrome bin to empty as in the ball case. If one water-move does not empty the source monochrome bin, it must make the target bin full. Since one cannot make more than n full bins, to empty a monochrome bin takes at most l water-moves. Therefore, every yes-instance has a solution consisting of at most $2(h - 1)nl$ water-moves. ◀

3 NP-completeness

In this section, we show that BSP and WSP are NP-complete even with two colors. By Corollaries 4 and 5, it suffices to show that WSP with two colors is NP-hard. By slightly modifying the proof, we also show that given a trivial yes-instance of WSP, it is NP-complete to find a shortest sequence of water-moves to a sorted configuration, where an instance is trivial in the sense that it contains many (say hn) empty bins.

► **Theorem 9.** *BSP and WSP are NP-complete even with two colors.*

Proof. As mentioned above, it suffices to show that WSP with two colors is NP-hard. We prove it by a reduction from the following problem 3-PARTITION, which is known to be NP-complete even if B is bounded from above by some polynomial in m [1].

3-PARTITION

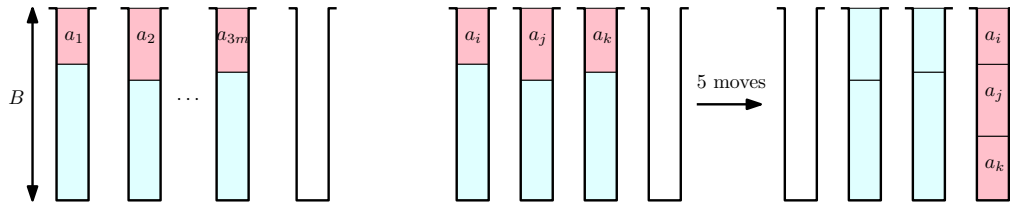
Input: Positive integers $a_1, a_2, a_3, \dots, a_{3m}$ such that $\sum_{i=1}^{3m} a_i = mB$ for some positive integer B and $B/4 < a_i < B/2$ for $1 \leq i \leq 3m$.

Question: Is there a partition of $\{1, 2, \dots, 3m\}$ into m subsets A_1, A_2, \dots, A_m such that $\sum_{i \in A_j} a_i = B$ for $1 \leq j \leq m$?

In the reduction, we use two colors red and blue. A non-empty bin is *red* (*blue*) if it contains red (blue, resp.) units only. A non-empty bin is *red-blue* if its top units are red and the other units are blue. From an instance $\langle a_1, \dots, a_{3m} \rangle$ of 3-PARTITION, we define an instance S of WSP as follows (see Figure 5):

- the capacity of each bin is B ;
- for each $i \in [3m]$, it contains a full red-blue bin b_i with a_i red units and $B - a_i$ blue units;
- it contains one empty bin.

We show that S is a yes-instance of WSP if and only if $\langle a_1, \dots, a_{3m} \rangle$ is a yes-instance of 3-PARTITION.



■ **Figure 5** The reduction from 3-PARTITION to WSP.

To show the if direction, assume that $\langle a_1, \dots, a_{3m} \rangle$ is a yes-instance of 3-PARTITION. We can construct a sequence of water-moves from S to a sorted configuration as follows. Let A_1, \dots, A_m be a partition of $\{1, \dots, 3m\}$ such that $\sum_{i \in A_j} a_i = B$ for $1 \leq j \leq m$. Using one empty bin, we can sort $b_{j_1}, b_{j_2}, b_{j_3}$ with $A_j = \{j_1, j_2, j_3\}$ as follows (see Figure 5). We first move all red units to the empty bin. Since the number of red units is $a_{j_1} + a_{j_2} + a_{j_3} = B$, the bin becomes full. Now b_{j_1}, b_{j_2} , and b_{j_3} are blue bins containing $(B - a_{j_1}) + (B - a_{j_2}) + (B - a_{j_3}) = 2B$ blue units in total. We move the units in b_{j_3} to b_{j_1} and b_{j_2} . After that, b_{j_1} and b_{j_2} become full and b_{j_3} is empty. Using this new empty bin, we can continue and sort all bins.

To show the only-if direction, we prove the following slightly modified statement. (What we need is the case of $\rho = \beta = 0$.)

Let ρ and β be non-negative integers, and S_0 be the instance of WSP obtained from S by adding ρ full red bins and β full blue bins. If S_0 is a yes-instance of the decision problem, then $\langle a_1, \dots, a_{3m} \rangle$ is a yes-instance of 3-PARTITION.

Assume that there is a sorting sequence S_0, \dots, S_ℓ , where S_ℓ is a sorted configuration. We use induction on m . (We need the dummy bins for the induction step.) The case of $m = 1$ is trivial since all instances of 3-PARTITION with $m = 1$ are yes-instances. Assume that $m \geq 2$ and that the statement holds for strictly smaller instances of 3-PARTITION.

Observe that S_0 contains $3m + \rho + \beta + 1$ bins and $(3m + \rho + \beta)B$ units of water ($(m + \rho)B$ red units and $(2m + \beta)B$ blue units). Since the capacity of bins is B , every S_i contains at most one empty bin, and if there is one, then the other bins are full. Observe also that each bin in each S_i is either red, blue, red-blue, or empty.

16:10 Sorting Balls and Water: Equivalence and Computational Complexity

We say that a move *opens* a bin b_i if the move makes b_i red units free for the first time. Observe that each red unit in b_1, \dots, b_{3m} has to move at least once since otherwise non-monochromatic bins will remain. Thus, all b_1, \dots, b_{3m} will eventually be opened. Let $b_p, b_q,$ and b_r be the first three bins opened in the sorting sequence in this order.

Assume that b_r is opened by the move from S_{j-1} to S_j . Let $\alpha = a_p + a_q + a_r$.

▷ **Claim 10.** S_j satisfies the following conditions.

1. There are $\rho + 1$ red bins, and they contain at least $\rho B + \alpha$ units.
2. There are $\beta + 3$ blue bins, and they contain $(\beta + 3)B - \alpha$ units.
3. There are $3m - 3$ red-blue bins, and they contain $(2m - 3)B + \alpha$ blue units and at most $mB - \alpha$ red units.
4. There is no empty bin. (This follows for free from the other conditions.)

Proof. Observe that we cannot create a new red-blue bin by any sequence of moves. Thus the number of red-blue bins is $3m - 3$. Since each red-blue bin b_i contains exactly $B - a_i$ blue units and at most a_i red units, the red-blue bins contain exactly $(2m - 3)B + \alpha$ blue units and at most $mB - \alpha$ red units in total.

The blue bins contain exactly $(\beta + 3)B - \alpha$ units. Since b_r contains exactly $B - a_r$ units, the other blue bins contain $(\beta + 2)B - (a_p + a_q)$ units, and thus the number of blue bins is at least $\lceil ((\beta + 2)B - (a_p + a_q))/B \rceil + 1 \geq \beta + 3$, where the inequality holds as $a_p + a_q < B$.

Since the red bins contain at least $\rho B + \alpha$ units, the number of red bins is at least $\rho + \lceil \alpha/B \rceil \geq \rho + 1$. Recall that the total number of bins is $3m + \rho + \beta + 1$. Thus, the number of red bins is exactly $\rho + 1$ and the number of blue bins is exactly $\beta + 3$. ◁

▷ **Claim 11.** $\alpha = B$.

Proof. In S_j , the $\rho + 1$ red bins contain at least $\rho B + \alpha$ units. This implies that $\alpha \leq B$. Suppose to the contrary that $\alpha < B$. We show that under this assumption, the conditions in Claim 10 cannot be violated by any sequence of moves. This contradicts that the final state S_ℓ does not contain any red-blue bin.

Let T be a configuration satisfying the conditions in Claim 10 and T' be a configuration obtained from T by one move. It suffices to show that T' still satisfies the conditions in Claim 10. Assume that the move is from a bin b to another bin b' .

First consider the case where we moved blue units. In this case, b and b' have to be blue bins. Hence, to show that all properties are satisfied, it suffices to show that b is not empty after the move. Indeed, if b becomes empty after the move, then $\beta + 2$ blue bins contain $(\beta + 3)B - \alpha > (\beta + 2)B$ units in total. This contradicts the capacity of bins.

Next consider the case where we moved red units. The bins b and b' are red or red-blue. If b becomes empty (when it was red) or blue (when it was red-blue), then the total number of red bins and red-blue bins becomes $3m + \rho - 3$ but they still contain $(3m + \rho - 3)B + \alpha$ units ($(m + \rho)B$ red and $(2m - 3)B + \alpha$ blue). This contradicts the capacity of bins. Thus, we know that the type of b does not change by the move. If the move is either from red to red, from red-blue to red-blue, or from red-blue to red, then the all conditions are satisfied. Assume that the move is from red to red-blue and that the red-blue bins contain more than $mB - \alpha$ red units after the move. Now the $3m - 3$ red-blue bins contain more than $(2m - 3)B + \alpha + mB - \alpha = (3m - 3)B$ units. This again contradicts the capacity of bins. ◁

By the claims above and the capacity of bins, S_j satisfies the following conditions.

1. There are $\rho + 1$ full red bins.
2. There are $\beta + 3$ blue bins, and they contain $(\beta + 2)B$ units.

3. There are $3m - 3$ full red-blue bins, and they contain $(2m - 2)B$ blue units and $(m - 1)B$ red units.
4. There is no empty bin.

Observe that each red-blue bin b_i in S_j is not opened so far, and thus contains $B - a_i$ blue units (and a_i red units as it is full).

Let us take a look at the sorting sequence from S_j to S_ℓ . Since there is no empty bin and all bins containing red units are full in S_j , we need to move blue units in blue bins first. Unless we make an empty bin, the situation does not change. Let S_h be the first configuration after S_j that contains an empty bin. By the capacity of bins and the discussion so far, S_h satisfies the following conditions.

1. There are $\rho + 1$ full red bins.
2. There are $\beta + 2$ full blue bins.
3. For each $i \in \{1, \dots, 3m\} \setminus \{p, q, r\}$, the bin b_i is a full red-blue bin that contains a_i red units and $B - a_i$ blue units.
4. There is one empty bin.

Without loss of generality, assume that $\{p, q, r\} = \{3m - 2, 3m - 1, 3m\}$. Let S' be the instance of WSP obtained from $a_1, \dots, a_{3(m-1)}$ by the reduction above, and S'_0 be the one obtained from S' by adding $\rho + 1$ full red bins and $\beta + 2$ full blue bins. Observe that S'_0 can be obtained from S_h by renaming the bins. Thus the sorting sequence from S_h to S_ℓ can be applied to S'_0 by appropriately renaming the bins. Therefore, we can apply the induction hypothesis to S'_0 and thus $a_1, \dots, a_{3(m-1)}$ is a yes-instance of 3-PARTITION. Since $\alpha = a_{3m-2} + a_{3m-1} + a_{3m} = B$, the instance a_1, \dots, a_{3m} is also a yes-instance of 3-PARTITION. ◀

Corollaries 7 and 8 and Theorem 9 imply that given an integer t and a configuration S , it is NP-complete to decide whether there is a sequence of length at most t from S to a sorted configuration under both settings in BSP and WSP. We observe a slightly stronger result for WSP.

► **Corollary 12.** *Given an integer t and an instance S of WSP, it is NP-complete to decide whether there is a sorting sequence for S with length at most t even if S is guaranteed to be a yes-instance.*

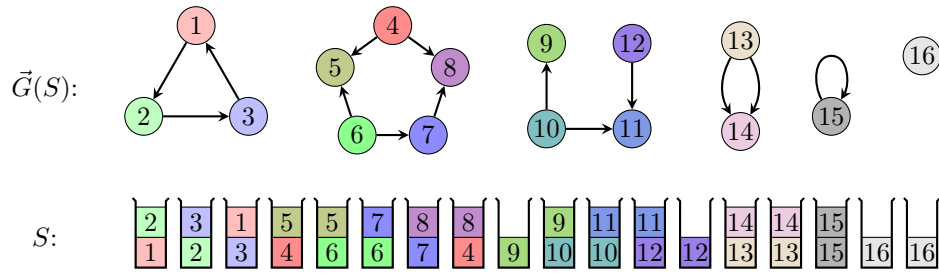
Proof. From an instance $\langle a_1, \dots, a_{3m} \rangle$ of 3-PARTITION, we first construct an instance of WSP as described in the proof of Theorem 9, and then add a sufficient number of empty bins to guarantee that the resultant instance is a yes-instance. This is always possible with a polynomial number of empty bins as we see in Section 5.² Let S denote the constructed instance. We set $t = 5m$.

The proof of Theorem 9 implies that if $\langle a_1, \dots, a_{3m} \rangle$ is a yes-instance, then S admits a sorting sequence of length $5m$. (See Figure 5.)

Conversely, assume that S admits a sorting sequence of length $5m$. Recall that each of the $3m$ red-blue bins in S contains a_i red units at the top and $B - a_i$ blue units at the bottom for some i . Since each full blue bin in the final configuration contains units from at least two original bins, we need at least one move for it. Thus we need at least $2m$ moves to make $2m$ full blue bins. This implies that we have at most $3m$ moves that involve red units. Actually, the number of such moves is exactly $3m$ since each red unit has to move at least

² It is not difficult to show that this instance only needs a constant number of bins. In fact, two empty bins are enough to perform a greedy algorithm for sorting.

16:12 Sorting Balls and Water: Equivalence and Computational Complexity



■ **Figure 6** Configuration S and its graph representation $\vec{G}(S)$.

once. Since $B/4 < a_i < B/2$ for all i , each of the m full red bins in the final configuration contains units from at least three original bins, and thus it needs at least three moves. If some red bin contains units from more than three original bins, then it needs at least four moves. This contradicts the assumption that we have at most $3m$ moves for red units. Thus we can conclude that each red bin in the final configuration contains units from exactly three original bins. This gives a solution to the instance of 3-PARTITION as the capacity of the bins is B . ◀

4 Polynomial-time algorithms when $h = 2$ and $|C| = n$

In this section, we focus on the special case of $h = 2$ and $|C| = n$. In popular apps, it is often the case that h is a small constant and $|C| = n$. The case $h = 2$ is the first nontrivial case in this setting. Under this setting, every ball-move is a water-move and vice versa, except for moving (a) unit(s) from a bin with two units of the same color to an empty bin, which is a vacuous move. Therefore, in this section we do not distinguish water-moves and ball-moves and simply call them moves. We prove that in this setting, all instances with $k \geq 2$ are yes-instances. We also show that we can find a shortest sorting sequence in $O(n)$ time (if any exists).

We say that a bin of capacity 2 is a *full bin* if it contains two units, and a *half bin* if it contains one unit.

► **Theorem 13.** *If $h = 2$, $|C| = n$, and $k \geq 2$, then all instances of WSP are yes-instances. Moreover, a shortest sorting sequence can be found in $O(n)$ time.*

Proof. For the first claim of the theorem, it is enough to consider the case $k = 2$. Under a configuration S , we say a color $c \in C$ is *sorted* if the two units of color c are in the same bin. For a configuration S , we define a directed multigraph $\vec{G}(S) = (V, A)$ as follows (see Figure 6). The vertex set V is the color set C . We add one directed edge from c to c' for each full bin that contains a unit of color c at the bottom and a unit of color c' at the top. That is, A consists of $S(b)$ for all full bins b . (We may add self-loops here.) For the directed multigraph $\vec{G}(S)$, we denote its underlying multigraph by $G(S) = (V, E)$, where E is a multiset obtained from A by ignoring the directions. Since $|C| = n$, each color appears twice in S . When S consists of full bins, $G(S)$ is a 2-regular graph with n edges, which means that $G(S)$ is a set of cycles. We call a self-loop a *trivial cycle*. If S also contains half bins, then $G(S)$ is a disjoint union of cycles and paths.

Let p_S denote the number of nontrivial directed cycles in $\vec{G}(S) = (V, A)$, q_S the number of vertices of indegree 2 in $\vec{G}(S) = (V, A)$, and r_S the number of vertices with no self-loop, i.e., the number of unsorted colors. We prove that if S has either two empty bins or one empty and two half bins, then a shortest sorting sequence has length $p_S + q_S + r_S$. Clearly the initial configuration, which has two empty bins, satisfies the condition.

We first prove that there exists a sorting sequence of length $p_S + q_S + r_S$. We use an induction on $p_S + q_S + r_S$. When $p_S + q_S + r_S = 0$, all colors are sorted and thus S is a goal. Now we turn to the inductive step, which consists of four cases.

- (Case 1)** Suppose S has no half bins and $q_S = 0$. Note that when S has no half bins, it has two empty bins. Then one can move the top unit of an arbitrary bin to an empty bin. This eliminates a directed cycle in the graph. Then we have one empty and two half bins. The claim follows from the induction hypothesis.
- (Case 2)** Suppose S has no half bins and there is a vertex c whose indegree is 2. By moving the two units of c to an empty bin, we use two moves, while decreasing both q_S and r_S . Then we have one empty and two half bins. The claim follows from the induction hypothesis.
- (Case 3)** Suppose S has two half bins one of which has a color of outdegree 0. Then, the other unit of that color is not below another unit in some bin. By moving that unit on top of the half bin, we can sort the color by one move. This decreases r_S by one, and does not change q_S . After this move, still we have two half bins. The claim follows from the induction hypothesis.
- (Case 4)** Otherwise, S has two half bins b and b' and both colors in the half bins have outdegree 1. Let c_0 be the color in b and (c_0, c_1, \dots, c_m) the sequence of vertices such that $(c_{i-1}, c_i) \in A$ for all i where c_m has outdegree 0. By the assumption, c_m is not the color of the unit of b' . Therefore, c_m has indegree 2, i.e., both units of c_m appear as the top units of full bins. We sort the color c_m using the empty bin. It takes two moves and decreases both r_S and q_S by one. We then sort c_{m-1}, \dots, c_0 in this order by m moves, which decreases r_S by m , where we require no additional empty bins and finally the bin b will be empty. We have one empty and two half bins. The claim follows from the induction hypothesis.

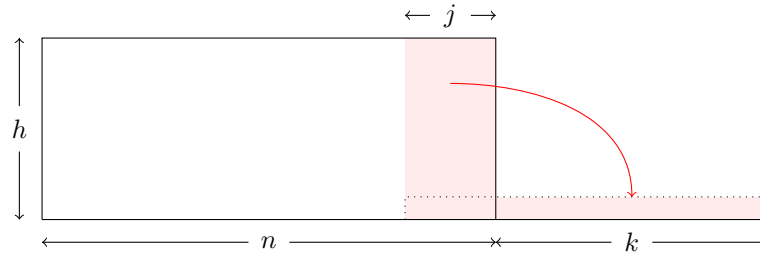
We next prove that there exists no sorting sequence of length less than $p_S + q_S + r_S$ with regardless of the number k of empty bins. Note that if all colors are sorted, $p_S + q_S + r_S = 0$. We show that any move decreases the potential by at most one.

- (Case 1)** If the color of the moved unit belongs to a directed cycle, this reduces p_S by one but not q_S . The other unit of the same color has a unit of another color on its top. Therefore this cannot reduce r_S .
- (Case 2)** If the color of the moved unit has indegree 2, this reduces q_S by one but none of p_S or r_S .
- (Case 3)** Otherwise, any other kind of moves cannot reduce p_S or q_S . Clearly one move cannot reduce r_S by two.

Since $p_S + q_S + r_S = O(n)$, we can sort any instance in $O(n)$ moves. Each feasible move can be found in $O(1)$ time, which completes the proof. ◀

► **Theorem 14.** *If $h = 2$, $|C| = n$, and $k = 1$, then WSP can be solved in $O(n)$ time. For a yes-instance, a shortest sorting sequence can be found in $O(n)$ time.*

Proof. For a configuration S , we again use the directed multigraph $\vec{G}(S) = (V, A)$ and its underlying multigraph by $G(S) = (V, E)$ defined in the proof of Theorem 13. Let S_0 be a given instance of WSP with $h = 2$, $k = 1$, and $|C| = n$. To simplify, we assume that S_0 has no full bin that contains two units of the same color without loss of generality. (Hereafter, when we have a full bin that contains two units of the same color, we remove it from the configuration and never touch. In terms of $\vec{G}(S)$, $\vec{G}(S)$ has no self-loop, and once $\vec{G}(S)$



■ **Figure 7** Refinement of the naive bucket sort.

produces a vertex with a self-loop, we remove it from \vec{G} .) Observe that if a configuration S consists of n' full bins and one empty bin, then $G(S)$ is a 2-regular multigraph with n' edges. That is, $G(S)$ is a set of cycles of size at least 2 since we remove vertices with self-loops.

Now we focus on a cycle C' in G . When C' is a directed cycle in \vec{G} (as C_3 in Figure 6), it is easy to remove the vertices in C' from G by using one empty bin. When C' contains exactly one vertex v of outdegree 2 in \vec{G} , C' contains another vertex u of indegree 2. In this case, we first move two units of color u to the empty bin, and we can sort the other bins using two half bins. Lastly, we can get two units of color v together, and obtain an empty bin again. Now we consider the case that C' contains (at least) two vertices v and v' of outdegree 2 in \vec{G} (as C_5 in Figure 6). In this case, we can observe that we eventually get stuck with two half bins of colors v and v' . That is, S is a no-instance if and only if $\vec{G}(S)$ contains at least one cycle C' that has at least two vertices of outdegree 2 in $\vec{G}(S)$.

In summary, we can conclude that the original instance S_0 can be sorted if and only if every cycle in $G(S_0)$ contains at most one vertex of outdegree 2 in $\vec{G}(S_0)$. The construction of $\vec{G}(S_0)$ and $G(S_0)$ can be done in $O(n)$ time, and this condition can be checked in $O(n)$ time. Moreover, it is easy to observe that sorting items in each cycle of length n' in $G(S_0)$ requires $n' + 1$ moves, and $n' + 1$ moves are sufficient. Therefore, the optimal sorting requires $n + \ell$ moves, where ℓ is the number of cycles in $G(S_0)$, which can be computed in $O(n)$ time. ◀

By the proofs of Theorems 13 and 14, we have the following corollary:

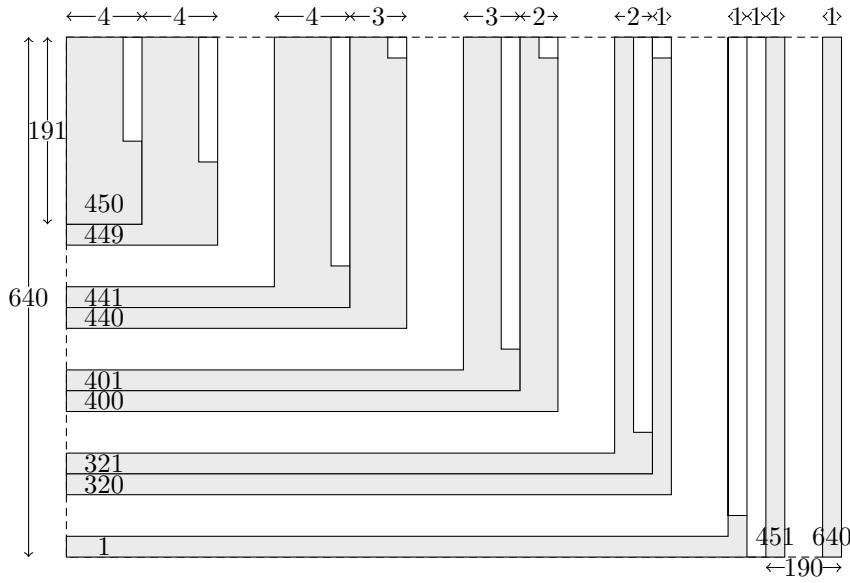
► **Corollary 15.** *If $h = 2$ and $|C| = n$, any yes-instance of WSP has a solution of length $O(n)$.*

5 The number of sufficient bins

In this section, we consider the minimum number $k(n, h)$ for n and h such that all instances of WSP with n full bins of capacity h with $k(n, h)$ empty bins are yes-instances. We can easily see that such a number exists and that $k(n, h) \leq n$ as follows. Let S be an instance with n full and k empty bins of capacity h such that $k \geq n$. For each color c used in S , let j_c be the positive integer such that S contains $j_c \cdot h$ units of color c . We assign j_c empty bins to each color c . Since $\sum_c j_c = n$, we can easily “bucket sort” S by moving water units in the n bins to empty bins following the assignment of the empty bins to the colors. On the other hand, as shown in Theorem 14, we have no-instances when $k = 1$ even if $h = 2$ (see Section 6 for a larger no-instance).

In the following, we improve the upper bound of $k(n, h)$ and show a lower bound.

► **Theorem 16.** $k(n, h) \leq \lceil \frac{h-1}{h} n \rceil$.



■ **Figure 8** When the first unit of color 1 has been moved, where $n = 640$ and $k = 190$.

Proof. It suffices to show the lemma for the case where $n = |C|$. We refine the bucket-sort base algorithm described above. See Figure 7. Suppose that we have at least $k = \lceil \frac{h-1}{h}n \rceil$ empty bins. Then, using those empty bins, one can make $j = \lfloor \frac{n}{h} \rfloor$ other bins monochrome, since $k \geq (h-1)j$. Now we have $k+j = n$ monochrome bins. When some of those bins have the same color, we merge them. Then we can dedicate different n bins to pairwise different colors, and we can use them to sort the remaining bins by the bucket sort. ◀

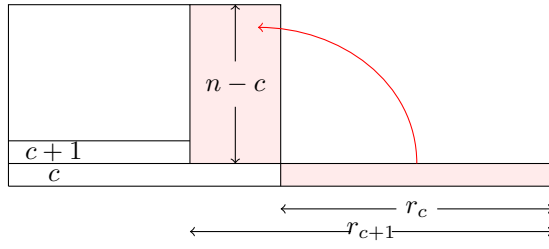
We note that $\lceil \frac{h-1}{h}n \rceil = n$ when $h > n$, which coincides with the naive bucket sort algorithm.

► **Theorem 17.** $k(n, h) \geq \lceil \frac{19}{64} \min\{n, h\} \rceil$.

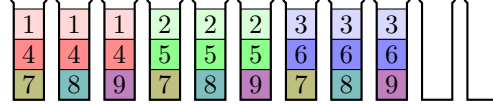
Proof. We show a configuration of $h = n$ bins where every bin has the same contents $(1, \dots, n)$. In the case where $n > h$, one can add $n - h$ extra bins whose contents are already sorted. In the case where $n < h$, one can arbitrarily pad the non-empty bins of the above configuration with $h - n$ extra units of each color at the bottom.

Suppose that it is solvable with k empty bins. We consider the very first moment when a unit of color 1 has been moved. Figure 8 shows an example configuration with $n = 640$ and $k = 190$. Note that any initially empty bin will always be monochrome. Let U be the set of colors c such that at least one unit of the color c occupies a bin which was initially empty, where $|U| \leq k$, and r_c be the number of units of color c which have been removed from the initial location. Particularly $r_1 = 1$. Then we have $r_{c+1} \geq r_c$ for all colors $c \geq 1$, since a unit of color c can be removed only after all the units above have been removed. Moreover, if $c \notin U$, units of color c which can be put only on units of the same color. That is, from both the source and target bins involved in the move, the unit of color $c+1$ must have been removed. Each of the target bins accepts at most $(n-c)$ units of water color c on top of the unit initially located there. Therefore, for $c \notin U$, $r_c \leq (r_{c+1} - r_c)(n-c)$ (see Figure 9), i.e.,

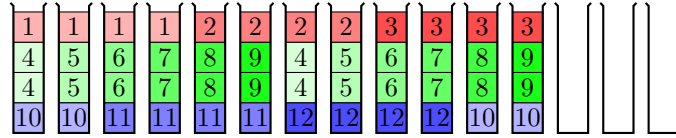
$$r_{c+1} - r_c \geq \left\lceil \frac{r_c}{n-c} \right\rceil. \tag{2}$$



■ **Figure 9** If $c \notin U$, $r_c \leq (r_{c+1} - r_c)(n - c)$.



(a)



(b)

■ **Figure 10** No-instances for (a) $h = 3, k = 2, n = 9$ and (b) $h = 4, k = 3, n = 12$.

We will give a lower bound of the size of U that admits a sequence of integers (r_1, \dots, r_n) with $1 \leq r_1 \leq \dots \leq r_n \leq n$ that satisfies Eq. (2) for $c \notin U$. Suppose that (r_1, \dots, r_n) satisfies the condition with $c \in U$ and $c_{c+1} \notin U$. Then, one can easily see that $U' = U \setminus \{c\} \cup \{c+1\}$ admits a sequence $(r_1, \dots, r_{c-1}, r'_c, r_{c+1}, r_n)$ with $r'_c = r_{c+1}$ that satisfies Eq. (2). Therefore, we may and will assume that $U = \{n - k + 1, \dots, n\}$.

For $c \leq n - k$, $r_1 = 1$ and $r_{c+1} - r_c \geq 1$ implies $r_c \geq c$. Hence, for $n/2 < c \leq n - k$,

$$r_{c+1} - r_c \geq \left\lceil \frac{r_c}{n - c} \right\rceil > \frac{n/2}{n - n/2} = 1$$

implies $r_c \geq 2c - n/2$. Hence, for $(5/8)n < c \leq n - k$,

$$r_{c+1} - r_c \geq \left\lceil \frac{r_c}{n - c} \right\rceil > \frac{(5/4)n - n/2}{n - (5/8)n} = 2$$

implies $r_c \geq 3c - (9/8)n$. Hence, for $(11/16)n < c \leq n - k$,

$$r_{c+1} - r_c \geq \left\lceil \frac{r_c}{n - c} \right\rceil > \frac{(33/16)n - (9/8)n}{n - (11/16)n} = 3$$

implies $r_c \geq 4c - (29/16)n$. On the other hand, $r_c \leq n$. That is, $c \leq n - k$ implies $c \leq (45/64)n$, i.e., $k \geq (19/64)n$. ◀

6 Concluding remarks

In this paper, we investigate the problems of solvability of ball and water sort puzzles. We show that both problems are equivalent and NP-complete. We also show that even for trivial instances of the water sort puzzle, it is NP-complete to find a shortest sorting sequence.

As discussed in Section 5, any instance can be sorted when k is large enough. Especially, as discussed in Section 4, $k = 2$ is enough for $h = 2$, while we have a no-instance for $k = 1$ and $h = 2$. On the other hand, there exist no-instances with $k = 2$ for $h = 3$ and $k = 3$ for $h = 4$ (Figure 10). We constructed them by hand and verified that they are no-instances by using a computer program. It is interesting to give the boundary of k for a given h , especially, $h = 3$, that allows us to sort any input. Does k depend on both h and n , or is it independent from n ?

Recently, a Japanese puzzle maker produces a commercial product of the ball sort puzzle.³ They extend the ball sort puzzle by introducing three different aspects as follows. (1) It contains three different capacities of bins. That is, it allows to use different size of bins. (2) The balls are not only colored, but also numbered. In some problems, the puzzle asks us to not only arrange the balls of the same color into a bin, but also they should be in order in each bin. In some other problems, the puzzle asks us to arrange the balls so that the sums of the numbers in every non-empty bins are equal. (3) It contains transparent balls. That is, these transparent balls are just used for arranging balls, and their final positions do not matter in the final state. Those extensions seems to be reasonable future work.

References

- 1 Michael R. Garey and David S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman, 1979.
- 2 William H. Gates and Christos H. Papadimitriou. Bounds for sorting by prefix reversal. *Discret. Math.*, 27(1):47–57, 1979. doi:10.1016/0012-365X(79)90068-2.
- 3 R. A. Hearn and E. D. Demaine. *Games, Puzzles, and Computation*. A K Peters Ltd., 2009.
- 4 Hiro Ito, Junichi Teruyama, and Yuichi Yoshida. An almost optimal algorithm for Winkler’s sorting pairs in bins. *Progress in Informatics*, 9:3–7, 2012.
- 5 Takehiro Ito, Erik D. Demaine, Nicholas J.A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412:1054–1065, 2011.
- 6 Donald E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley Publishing Company, 2nd edition, 1998.
- 7 Peter Winkler. *Mathematical puzzles: A Connoisseur’s collection*, volume 143, pages 149–151. A K Peters, 2004.
- 8 Katsuhisa Yamanaka, Shin-ichi Nakano, Yasuko Matsui, Ryuhei Uehara, and Kento Nakada. Efficient Enumeration of All Ladder Lotteries and Its Application. *Theoretical Computer Science*, 411:1714–1722, 2010.

³ <https://www.hanayamatoys.co.jp/product/products/product-cat/puzzle/>

Skiing Is Easy, Gymnastics Is Hard: Complexity of Routine Construction in Olympic Sports

James Koppel  

Electrical Engineering and Computer Science, MIT, Cambridge, MA, USA

Yun William Yu   

Computer and Mathematical Sciences, University of Toronto at Scarborough, Canada

Department of Mathematics, University of Toronto, Canada

Abstract

Some Olympic sports, like the marathon, are purely feats of human athleticism. But in others such as gymnastics, athletes channel their athleticism into a routine of skills. In these disciplines, designing the highest-scoring routine can be a challenging problem, because the routines are judged via a combination of artistic merit, which is largely subjective, and technical difficulty, which comes with complicated but objective scoring rules. Notably, since the 2006 Code of Points, FIG (International Gymnastics Federation) has sought to make gymnastics scoring more objective by encoding more of the score in those objective technical side of scoring, and in this paper, we show how that push is reflected in the computational complexity of routine optimization.

Here, we analyze the purely-technical component of the scoring rules of routines in 17 different events across 5 Olympic sports. We identify four attributes that classify the common rules found in scoring functions, and, for each combination of attributes, prove hardness results or provide algorithms for designing the highest-scoring routine according to the objective technical component of the scoring functions. Ultimately, we discover that optimal routine construction for events in artistic, rhythmic, and trampoline gymnastics is NP-hard, while optimal routine construction for all other sports is in P .

2012 ACM Subject Classification Theory of computation; Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases complexity, games, sports

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.17

Funding *Yun William Yu*: This work is supported by faculty startup funds from UTSC.

Acknowledgements The authors thank Aviv Adler and Quanquan Liu for comments on earlier drafts of this paper, and thank the members of the MIT Gymnastics team for fostering our love of gymnastics.

1 Introduction: Olympic Sports and Complexity

There has been much work on determining the complexity classes of various logic puzzles and video games [13, 17, 11, 12], as players must strategize and solve constraints to win. Sensibly, there has been little work on the complexity of physical sports, where the challenges are purely athletic. The winning strategy of the 100-meter dash is very simple: run fast.

But in other sports, winning is more complicated. We turn our attention to judged routine-based sports such as figure skating, freestyle skiing, and gymnastics, where athletes demonstrate a string of moves (or “skills”) called a “routine,” where the routine must meet prescribed constraints (e.g.: at most 7 jumps in figure skating), and the routine’s score is different from the sum of its parts (e.g.: a “connection bonus” for chaining together two difficult moves in some gymnastics events). Many an athlete has sat by the sidelines trying to think of the highest-scoring routine they can do. For it is hard. NP-hard?



© James Koppel and Yun William Yu;
licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

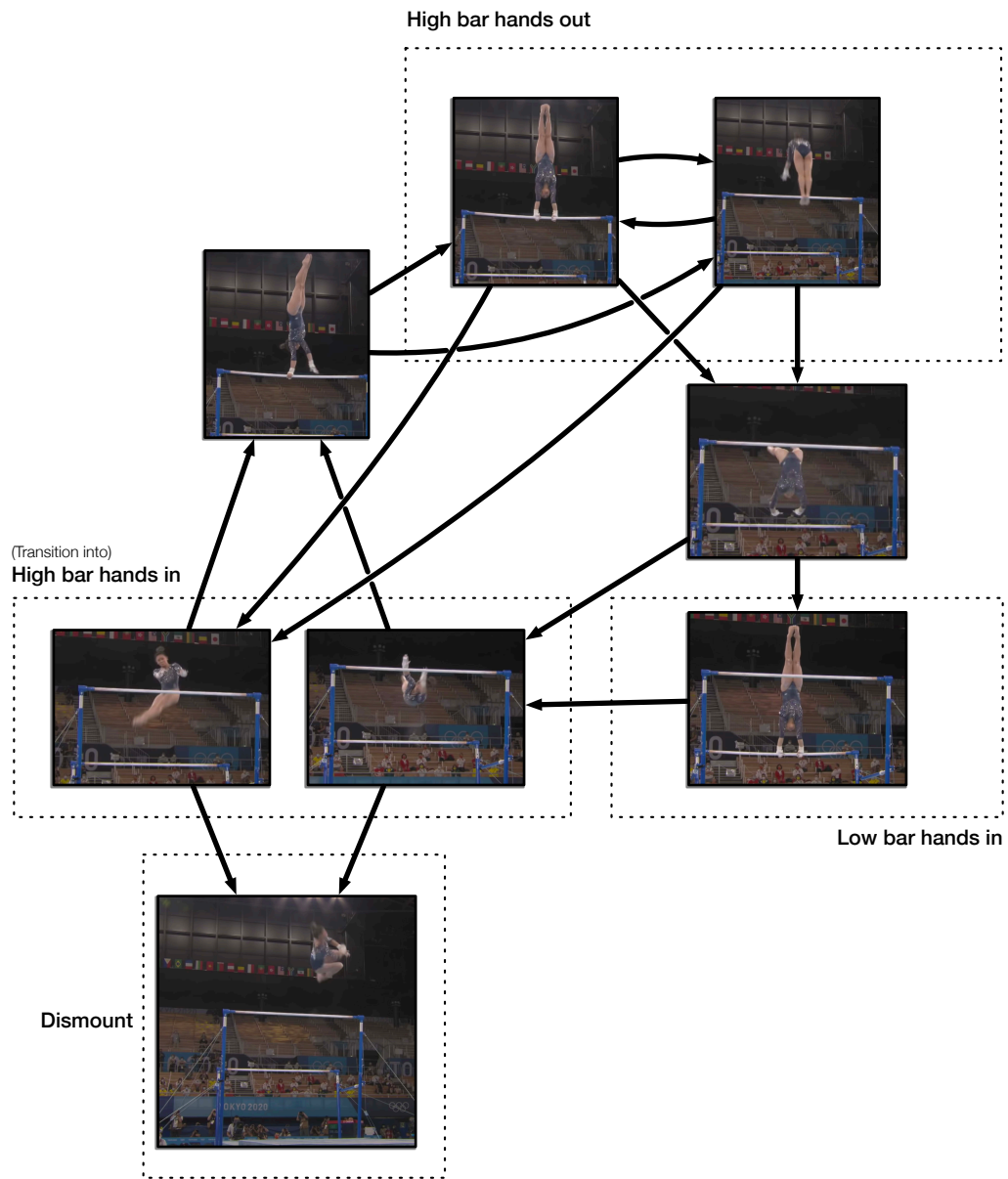
Editors: Pierre Fraigniaud and Yushi Uno; Article No. 17; pp. 17:1–17:20

Leibniz International Proceedings in Informatics

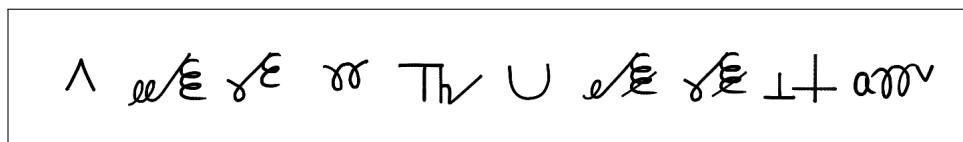


LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

17:2 Routine Construction Complexity



■ **Figure 1** Graph of possible transitions between skills, constructed from skills seen in Suni Lee's uneven bars routine at the Tokyo Olympics. Source: [19]



■ **Figure 2** Example Men's Floor routine in gymnastics judging notation; the skills form a linear string (Source and explanation: [2]).

In this paper, we consider a range of routine-based judged Olympic sports and events. We find that the scoring functions are generally made up of four different types of rules, and that the NP-hardness of constructing the optimal routine within a scoring function depends on which types of rules are present.

2 Background

In a judged routine-based sport, the athlete chooses some sequence of skills, which taken together form a “routine”. A scoring function then maps that sequence of skills to the “start value” of the routine, which is the maximum number of points achievable if the athlete performs the routine correctly with perfect form. Judges are employed to determine both (1) whether or not the athlete actually performed the skills of the routine and (2) what deductions to apply for imperfect form. The final score given is the start value of the routine that was performed, minus any deductions. At the end of the day, the athlete (or team of athletes) with the highest score wins.

In some sports such as figure skating and rhythmic gymnastics, routines are judged both on their technical difficulty and artistic dance merits. Typically, the final score will be a sum of a technical difficulty score and an “artistic presentation” score. As the latter is inherently subjective, we ignore all artistic presentation scores, and focus purely on technical difficulty and execution scores. However, do note that sports that place a higher emphasis on artistic presentation need not have as complicated technical scoring rules.

In the real world, there are two important optimization tasks in routine construction: (1) maximizing the expected score or (2) maximizing the probability of achieving some fixed score. The first task’s relevance is obvious, as higher scores are more likely to win. There are also situations where achieving a fixed score is more useful, e.g.: in Olympic women’s artistic gymnastics, an athlete who has accumulated a large lead on vault, beam, and uneven bars may only need a low score on floor to guarantee an overall win.

In this paper we consider a slightly simplified version of the routine construction where each athlete is able to reliably always perform any skill in their repertoire with a fixed deduction. More precisely, there is a Code of Points which provides the set of allowable skills and the scoring function, and the athlete knows the deductions they will incur for imperfect form every time they perform a skill. With this simplification, the complexity of routine construction depends on the scoring function.

Olympic sports have rules in the 100s of pages, and we shall see that we cannot capture every aspect of the sport in our mathematical models. Our work is limited to the objective technical difficulty scores, yet the choices of elements in a routine interact with the subjective artistic scores in sports that have them. Nonetheless, we do create a model that can express nearly all the intricacies of scoring, and most seeming-exceptions can still be encoded into the inputs using tricks similar to ones seen in some of the proofs.

2.1 Generalizing Sports

At the Olympics, each routine-based sport has a finite set of scorable moves and a fixed limit on routine length, so that optimizing a routine in a given year is trivially a finite problem. But the computational difficulty grows over time. In gymnastics, new skills can be invented at high-level competitions, which are then named after the first athlete to successfully perform that skill in competition – for example, on Balance Beam, the Biles is a double-twisting double-tucked salto backwards dismount, named after Simone Biles, who first performed the skill at the 2019 U.S. National Gymnastics Championships. It is theoretically possible to

17:4 Routine Construction Complexity

#	Executed Elements	Info Base Value	GOE
1	4Lz	11.50	4.76
2	4F	11.00	2.04
3	4T	9.50	3.39
4	3A	8.00	2.97
5	CCSp4	3.20	1.23
6	StSq4	3.90	1.67
7	4T+3T	15.07 X	3.39
8	3Lz+3T	11.11 X	2.11
9	3F+1Eu+3S	11.11 X	0.30
10	ChSq1	3.00	2.14
11	FCCoSp4	3.50	1.35
12	CCoSp4	3.50	1.50
		94.39	

■ **Figure 3** Portion of scoring card from Nathan Chen’s Free Skating routine at the 2019 World Championships. Explanation given at [1]

perform a gymnastics routine entirely consisting of skills not previously recognized. More broadly, for all sports under study, every few years their byzantine rulebooks are rewritten, and with each rewrite the roster of skills expands.

For each sport, it is thus natural to generalize to **arbitrary sets of skills** and with **new scoring rules similar to ones that already exist**. The hardness proofs in this paper will imply that e.g.: a SAT formula can be encoded into the scoring rules of a future version of the sport combined with the abilities of a particular athlete facing them. When the kinds of constraints found in scoring rules can encode NP-hard problems, it provides evidence of the practical difficulty faced by an athlete trying to optimize their routine in the finite problem posed by today’s sports in a fashion analogous to how the gadgets found in hardness proofs of puzzle games give evidence of the cognitive challenge in solving levels that actually exist.

With this generalization in mind, we now investigate more formally the families of scoring functions found in various Olympic sports.

3 String Scoring Problems

In a routine, an athlete demonstrates a string of skills. These routines can be naturally modeled as a literal string, built from an “alphabet” Σ consisting of the skills they can physically perform that will be recognized by the judges. In real life, this set is a literal alphabet, as each sport defines a set of symbols notating each skill; Figs. 2 and 3 give example routines for gymnastics floor and figure skating in actual judging notation. The rules for scoring then induce a *scoring function* $f : \Sigma^+ \rightarrow \mathbb{R}$ mapping a string to a score. The goal is now to construct the highest scoring string that an athlete can perform.

In the remainder of this section, we present several families of scoring functions corresponding to the different types of scoring rules of different Olympic sports. We first start by introducing a simple family of scoring functions: assign a point value and deduction to each skill, and then sum the value of each skill in a routine, minus the deductions. We refer to this as the basic *compositional* scoring function, because the overall score is composed of the individual scores and deductions combined.

The scoring functions used in the Olympic sports we analyze in this paper all build on this foundation, but add complexities such as penalties for repeated moves or bonuses for difficult sequences. We also consider extra constraints on the set of allowed strings: some sequences of moves may be physically impossible (e.g.: doing a handstand on the high bar after already having dismounted it), or at least barred by the rules. We still call these modified scoring functions compositional, because although more complicated, they still depend on the point values and deductions of individual skills.

■ **Table 1** Summary table of NP-hardness of different pairs of rule types, as proven in this manuscript. We consider four different types of rules: *Non-hierarchical* ANTI-REPETITION, *Hierarchical* ANTI-REPETITION, CONNECTION, and INCOMPLETEGRAPH. Note that the ELEMENTGROUP penalty we describe is subsumed by the ANTI-REPETITION rules. (Lemma 2) While some Olympic sports only have one type of rule, many include multiple types of rules. Note that we only show pairs, because all triples of rule types are NP-hard.

	<i>Non-hierarchical</i> ANTI-REPETITION	<i>Hierarchical</i> ANTI-REPETITION	CONNECTION	INCOMPLETE GRAPH
<i>Non-hierarchical</i> ANTI-REPETITION	NP-hard Thm 6	NP-hard Thm 6	NP-hard Thm 6	NP-hard Thm 6
<i>Hierarchical</i> ANTI-REPETITION		In P Thm 5	NP-hard Cor 10	NP-hard Thm 9
CONNECTION			In P Thm 7	In P Cor 11
INCOMPLETE GRAPH				In P Cor 8

Within variants of compositional approaches, the modifications different types of rules make to a simple summation greatly affect the computational complexity. In this paper, we consider four different types of rules commonly found in Olympic scoring codes of points, which will form part of the input to the string scoring problem.

1. ANTI-REPETITION: Are identical or similar skills allowed to be repeated for more points? If so, how many times? Note that deductions are always repeatable.
2. ELEMENTGROUP: Are skills partitioned into non-overlapping classes and routines penalized for not having a representative from each class?
3. CONNECTIONS: Are contiguous pairs of particular skills worth more or fewer points than those skills separately?
4. INCOMPLETEGRAPH: Are certain orderings of skills entirely disallowed (or physically impossible), disqualifying the entire routine?

Note that these are just a subset of the types of rules an actual Olympic sport has, but this taxonomy captures most of the technical non-subjective rules used.

How the rules are applied also makes a big difference to the hardness. We will find that the ANTI-REPETITION and ELEMENTGROUP rules are almost equivalent, but the differentiating factor for NP-hardness is whether or not groups of similar skills are hierarchically arranged. We more formally define each rule type and give a few examples of Olympic sports with each of those classes of rules in the remainder of this section, but for the impatient, the full hardness results can be found in Table 1 and the full classification in Table 2.

Before going on though, let's formally define the problem:

- Let Σ be the set of possible skills, and $n = |\Sigma|$, the number of skills.
- Let $m = |S|$ be the maximum length of an allowed routine $S \in \Sigma^+$.
- Let q be the total number of scoring rules (of any type, defined in the next section).
- Let $z = n + m + q$, the size of the input.

When we speak of poly-time algorithms, we want it to be polynomial in the input size z . Note that in our definition of input size, we are effectively measuring the length of the routine m in unary. This is because we are considering the optimization variant of the scoring problem, where the goal is to determine the highest scoring routine subject to these constraints; without including the output size m , it would be easy to come up with a degenerate case where $n = 1$ and $q = 0$ but m is arbitrarily large.

17:6 Routine Construction Complexity

■ **Table 2** Olympic events categorized by rule types. Note that we consider *Non-hierarchical* to imply *Hierarchical*, because there is always a subset of the *Non-hierarchical* rules that is *Hierarchical*. See Sec. 6 for discussion of individual sports.

Sport/event	<i>Hierarchical</i> ANTI- REPETITION	<i>Non-hierarchical</i> ANTI- REPETITION	CONNECTION	INCOMPLETE GRAPH	Complexity
Skiing (4 events)	N	N	N	Y	P
Figure skating (Free Skate, Single and Pairs)	Y	N	N	N	P
Rhythmic gymnastics (Individual and Team)	Y	N	(?)	Y	NP-hard
Trampoline	Y	N	N	Y	NP-hard
MAG Floor	Y	N	Y	Y	NP-hard
Pommel Horse	Y	Y	N	Y	NP-hard
Rings	Y	N	N	Y	NP-hard
Parallel bars	Y	N	Y	Y	NP-hard
High Bar	Y	N	Y	Y	NP-hard
WAG Floor	Y	Y	Y	Y	NP-hard
Balance Beam	Y	N	Y	Y	NP-hard
Uneven bars	Y	Y	Y	Y	NP-hard

Moving on to scoring and rules,

- Let $M \in \mathbb{Z}^+$ be the maximum possible score for any individual skill, bonus, or penalty. For the purposes of our setup M should be polynomial in z . Note that in some of our reductions, we further increase M by a polynomial factor.
- Let $p : \Sigma \rightarrow [M]$ be the point value of a skill.
- Let $d : \Sigma \rightarrow [M]$ be the deduction the athlete will receive when performing a skill.

The other rules described below will be encoded as modifications to a summed compositional score. Furthermore, note that in all of the codes of points, all of the non-integer scores are rational, so we simply scale them up to be integers.

In practice, although we include m , n , and q in the input size, the length of the routine m generally tends to be a small fixed constant. Instead, the natural generalization of Olympic scoring is in the number of skills n and the number of rules q because (1) each athlete adds more skills to their portfolio with training, and (2) new skills are invented and added to the code of points over time – indeed, many gymnastics skills are named after the first athlete to successfully perform that skill in a recognized high-level competition.

3.1 Compositional scoring with no additional rules

Without additional rules, the unmodified compositional score for a routine $S = s_1 \dots s_m$ is

$$f_{\text{BASIC}}(S) = \sum_{i=1}^m (p(s_i) - d(s_i)). \quad (1)$$

We were unable to find examples of basic compositional scoring with no additional rules in Olympic sports. This is entirely unsurprising, because as we will see later in Theorem 4, the optimal routine is not only trivially computable, but also entirely uninteresting – it is just the highest scoring move repeated over and over.. We present it here not because it is used, but simply because it is the base upon which all of the other scoring functions are built.

<p>4. Special repetitions:</p> <p>a) Repeated elements (same Code Identification Number) cannot contribute to the “D” score. On Rings, this rule is extended so that a maximum of 1 final strength position in each EG may be recognized for difficulty. Thus, for example only two cross type elements (regular, L cross, or V cross) or support scale type elements (regular or straddled) are permitted in an exercise for difficulty value (one in Group II and one in Group III).</p> <p>b) A maximum of 2 Guczoghy type elements can be present in the exercise.</p>	<p>7. Special repetitions:</p> <p>a) A maximum of 2 strength elements (including strength handstands) may be performed in an exercise for content value. Elements I.1 to I.48 are considered to be strength elements, except:</p> <ul style="list-style-type: none"> - Element I.19 - Handstand (2 sec). - Element I.31 - ½ or 1/1 turn in handstand or to handstand. <p>b) A maximum of 2 circle, flair or Russian elements may be performed in an exercise for difficulty value.</p>
--	---

(a)

(b)

■ **Figure 4** Special repetition rules for rings (a) and men’s floor (b) in artistic gymnastics [5].

3.2 Anti-Repetition rules

Many sports design their rules to force competitors to construct more diverse routines via ANTI-REPETITION rules. A gymnast performing on the high bar may do a giant swing all around the bar many times to transition into various other skills. However, only one of these “giants” is itself scored as a skill (though the gymnast may still incur deductions for bad form). Sometimes these restrictions also apply to similar skills, such as only allowing two scored strength elements (such as handstands, v-sits, or support levers) in men’s floor, as seen in Fig. 4.

Formally define an ANTI-REPETITION rule as a pair (ρ, k) , where $k \in \mathbb{N}$ denotes the maximum number of skills in a subset $\rho \subseteq \Sigma$ that can be recognized for points. Given a routine $S = s_1 \dots s_m$, we define a length- m bitstring $R = r_1 \dots, r_m$ specifying whether or not each skill is recognized for points, where R has to satisfy all of the ANTI-REPETITION rules. Then the modified compositional score for a routine is

$$f_{\text{ANTI-REPETITION}}(S) = \max_{\text{valid } R} \sum_{i=1}^m (p(s_i) \cdot r_i - d(s_i)). \quad (2)$$

When all skills are recognized, such as when there are no ANTI-REPETITION rules, R is all 1’s and this scoring function is equivalent to f_{BASIC} from Equation 1.

In plain language, all performed skills swept up in the ANTI-REPETITION rules will be worth 0 points, but still count for any deductions; however, if there are multiple satisfying bitstrings, the highest scoring one will be chosen. As an aside, how the ρ_j sets in the ANTI-REPETITION rules overlap will be a salient factor in our analysis later.

3.3 Element group penalty

Another way to encourage routine diversity is to partition skills into some number of non-overlapping *element groups*, and then penalize not having skills from every element group. For example, on men’s floor, gymnasts will be penalized if they lack any of the three major disjoint element groups (non-acrobatic, acrobatic forwards, acrobatic backwards). Element group requirements are also present in other sports and can be more complicated; see Fig. 5.

More formally, an ELEMENTGROUP rule is a pair (ρ, p_ρ) , where $\rho \subseteq \Sigma$ is the set of skills within an element group, and p_ρ is the point value associated with that element group. Given a set of q_{eg} non-overlapping ELEMENTGROUP penalties (ρ_j, p_{ρ_j}) , and an indicator variable I_j which is 1 if $s_i \in \rho_j$ for some $i \in [m]$, we get a negative additive modification to the basic compositional score $-\sum_j (1 - I_j) p_{\rho_j}$. Of course, penalizing for lacking an element group is equivalent to giving a bonus for having an element group, so we can instead use a positive additive modification $\sum_k I_j p_{\rho_j}$, which is a bit easier to parse. This modification can of course

17:8 Routine Construction Complexity

Difficulty Components	
Difficulty of Body (DB) Highest 9 counted	Difficulty of Apparatus (DA) Minimum 1 Maximum 20 (in performance order)
Special Requirement	
Difficulty of Body Groups: Jump/Leaps \wedge Minimum 1 Balances \top Minimum 1 Rotations \circlearrowleft Minimum 1	Full body waves (W) Minimum 2
Dynamic Elements with Rotation - R Maximum 5 (in performance order)	

11.3 Composition Requirements (CR) – D-Panel 2.00

1. Flight element from HB to LB award 0.50
2. Flight element on the same bar award 0.50
3. Different grips (*not cast, MT or DMT*) award 0.50
4. Non-flight element with min. 360° turn (*not MT*) award 0.50

(a)
(b)

■ **Figure 5** Element group rules for rhythmic gymnastics [7] and uneven bars [6].

<p>d) Backward swings to handstand that simply reverse direction and swing back down in the reverse direction are deducted (composition errors) each time with 0.30 points. Specific examples of such layaways are:</p> <ul style="list-style-type: none"> • Following a kip cast or back uprise to handstand - layaway to giant swing bwd., Stalder, free hip, ½ turn to el-grip, etc. • Following a backward uprise to handstand and hop to overgrip - swing forward to giant swing bwd., Stalder, free hip, etc. <p>(Also other angle deductions also need to be applied for missing the handstand position).</p> <p>e) Any flight element with salto over the bar requires a giant swing afterward or -0.3 (E-jury deduction).</p>	<p>b) Special rule: Elements to one bar in cross support have the same value as done to two bars, except they increase by one value more when connected to Healy type elements (each Healy element also increases by one value) hold is allowed in the one bar handstand.</p>
--	---

(a)
(b)

■ **Figure 6** Skill-specific connection bonuses in high-bar and parallel-bars [5].

be simply added to either f_{BASIC} or $f_{\text{ANTI-REPETITION}}$ to get

$$f_{\text{ELEMENTGROUP}} = \sum_{i=1}^m (p(s_i) - d(s_i)) + \sum_{j=1}^{q_{eg}} I_j p_{\rho_j} \quad (3)$$

and

$$f_{\text{ANTI-REPETITION+ELEMENTGROUP}} = \max_{\text{valid } R} \sum_{i=1}^m (p(s_i) \cdot r_i - d(s_i)) + \sum_{j=1}^{q_{eg}} I_j p_{\rho_j} \quad (4)$$

3.4 Connections

The previous two classes of rules did not depend at all on the linear ordering of a string of skills in a routine. However, some events, like the balance beam, have “connection bonuses”, whereby performing a specific set of skills in a row is worth more than each skill would individually. For example, a layout step-out smoothly transitioning into a back flip with piked legs is worth 0.1 more points than the sum of the two individual skills’ values¹. Because of these connection bonuses, the highest scoring routine may have repeated elements, even when these repeated elements themselves are not scored. There can also be negative connection

¹ Although fractional points are possible, all scores are rational in Olympic codes of points, so we simply rescale to integers in our model.

bonuses, such as the 0.1 point penalty for “illogical connections” in rhythmic gymnastics, or simply to model deductions for expected errors when an athlete attempts two difficult moves in sequence. While connection bonuses most commonly apply to all pairs in a broad class of skills, Fig. 6 gives examples of positive and negative connection bonuses being awarded for more specific combinations of skills.

More formally, a CONNECTION rule can be modelled as a triple (s_1, s_2, c_{12}) , where s_1 and s_2 are the two consecutive skills, and c_{12} is the amount of bonus points to be given. Alternately, it can also be thought of as a sparse asymmetric $n \times n$ matrix C , where c_{s_1, s_2} is the connection bonus between skills s_1 and s_2 . Given such a matrix C , CONNECTION bonuses amount to an additive modification to the compositional score $\sum_{i=1}^{m-1} c_{s_i, s_{i+1}}$. Thus,

$$f_{\text{CONNECTION}}(S) = \sum_{i=1}^m (p(s_i) - d(s_i)) + \sum_{i=1}^{m-1} c_{s_i, s_{i+1}}. \quad (5)$$

Of course, it is straight-forward to construct the variant $f_{\text{ANTI-REPETITION+CONNECTION}}$, or the variant $f_{\text{ELEMENTGROUP+CONNECTION}}$, or $f_{\text{ANTI-REPETITION+ELEMENTGROUP+CONNECTION}}$ by combining this with Equations 2, 3, and 4

3.5 Incomplete Graphs

An alternative view of connection scores is to cast the set of moves Σ as a complete directed graph with skills as nodes and connection bonuses as edge weights, where a routine is a path through the graph and its score is the sum of both node and edge weights. In this context, another type of constraint on compositional approaches can be making the graph graph incomplete, i.e.: forbidding transitions between some skills. This constraint can be imposed as a rule, but in practice is generally a physical constraint because some skills are impossible to follow by other skills. For example, the WAG Uneven Bars event apparatus consists of two different bars, a high bar and a low bar; some skills must begin on the high bar, which is physically impossible to start if an athlete ends the previous skill on the low bar. The graph of possible connections among 8 such uneven bars skills is shown in Fig. 1. More generally, the position, momentum, or idiosyncratic training of an athlete may disallow transitioning from one skill directly into another.

More formally, an INCOMPLETEGRAPH rule can be modelled as an edge (s_i, s_j) . Alternately, all of the INCOMPLETEGRAPH rules can naturally be encoded into an $n \times n$ graph adjacency matrix C , where $c_{s_i, s_j} = 0$ if (s_i, s_j) in an INCOMPLETEGRAPH rule and $c_{s_i, s_j} = 1$ otherwise. Unlike the previous rules, which amounted to modifications to the score, this is a hard constraint on whether a routine is allowed at all. Equivalently, we deem that any routine that breaks this rule is scored 0, which can be written as a multiplicative modification of the existing rules

$$f_{\text{INCOMPLETEGRAPH}}(S) = \left(\sum_{i=1}^m (p(s_i) - d(s_i)) \right) \prod_{i=1}^{m-1} c_{s_i, s_{i+1}}. \quad (6)$$

Naturally, this also can be combined with any of the other rules-sets.

4 Relationship between rule classes and polynomial reductions

You may notice that although the rule classes are written differently in the Olympic codes of points, the first two and the latter two seem to cover different kinds of conditions. ANTI-REPETITION and ELEMENTGROUP rules do not care about the ordering of skills,

17:10 Routine Construction Complexity

but just how many times each skill is performed. On the other hand, CONNECTION and INCOMPLETEGRAPH are both solely concerned with consecutive orderings of skills. In this section, we show that when considering the NP-hardness of rule sets, those rule sets are very closely related. Indeed, we'll find that for the ordering-free rules, whether the similarity classes overlap is substantially more important than the difference between ANTI-REPETITION and ELEMENTGROUP.

► **Lemma 1.** *INCOMPLETEGRAPH can be reduced to CONNECTION.*

Proof. In CONNECTION, the scoring function can be modelled as a fully connected weighted directed graph $G = (V, E)$ with skills for vertices and connections for edges. Let $w(s)$ and $w(s_1, s_2)$ be the respective weights on a vertex $s \in V$ or directed edge $(s_1, s_2) \in E$, corresponding to the effective scores after deductions. A routine is precisely a path in G , and the score of that routine is precisely the sum of the vertex and edge weights along that path, including the start and end nodes.

In INCOMPLETEGRAPH, because all scores are bounded by M , INCOMPLETEGRAPH can be reduced to CONNECTION by setting the weights for existing edges to 0, and then adding edges with weight $-10Mm$ for each missing edge to make a complete directed graph. Clearly, these edges cannot be traversed in any optimal path, so solving CONNECTION would also solve INCOMPLETEGRAPH. This means that INCOMPLETEGRAPH is no harder than CONNECTION. Furthermore, INCOMPLETEGRAPH+CONNECTION can still be reduced to just CONNECTION by keeping existing edge weights and adding $-10Mm$ weight edges for all the missing edges. ◀

► **Lemma 2.** *ELEMENTGROUP can be reduced to ANTI-REPETITION.*

Proof. First, notice that we have defined the ELEMENTGROUP rules to be non-overlapping; this is crucial for this transformation. As before, a penalty for missing element groups is equivalent to a bonus for the very first time a skill within a group is performed. Let's duplicate the set of skills s_1, \dots, s_n to include s'_1, \dots, s'_n , where the s'_i skills are used to denote the first time a skill is done in routine. Thus, given that element groups are disjoint, for a skill $s \in \rho$ we can simply add the element group penalty p_ρ as a bonus to the point value $p(s') = p(s) + p_\rho$ of a skill s , while keeping the deduction $d(s') = d(s)$. Then, we add an ANTI-REPETITION rule $(\rho', 1)$, where ρ' is the copy of ρ within the duplicated skills. As a result, it never makes sense to use a particular duplicated skill more than once in a routine, as replacing it with the original skills is always better (or at least as good). We have thus encoded the ELEMENTGROUP penalty as an ANTI-REPETITION rule.

In fact, we further claim that ELEMENTGROUP+ANTI-REPETITION can be reduced to just ANTI-REPETITION. Importantly, unlike ELEMENTGROUP rules, which are disjoint, ANTI-REPETITION rules are allowed to be overlapping, which is crucial for the reduction. We start by duplicating the skills, as above, and encoding all of the ELEMENTGROUP rules as new ANTI-REPETITION rules. However, for each existing ANTI-REPETITION rule in the original version of the problem, (ρ, k) , we replace it with a new ANTI-REPETITION rule in the duplicated skills world of $(\rho \cup \rho', k)$. That completes the encoding. ◀

Furthermore, although ANTI-REPETITION rules are allowed to overlap, most of the time, similarity classes are either subsets of one another, or entirely disjoint. For example, there are often ANTI-REPETITION rules for each individual skill, as well as ELEMENTGROUP penalties. Even when there are other similarity classes, very often, they will be subsets of ELEMENTGROUPS. It turns out that this hierarchical case is much easier than when arbitrary overlaps are allowed in the ANTI-REPETITION rules, so we will consider the *Hierarchical* and *Non-hierarchical* cases separately.

► **Definition 3** (*Hierarchical ANTI-REPETITION structure*). Consider a set of ANTI-REPETITION rules $\{(\rho_1, k_1), \dots, (\rho_q, k_q)\}$. If there exists a pair (ρ_i, ρ_j) where $\rho_i \cap \rho_j \neq \emptyset$ and $\rho_i \cap \rho_j \neq \rho_i$ and $\rho_i \cap \rho_j \neq \rho_j$, then the ANTI-REPETITION rules are *Non-hierarchical*.

Note that we consider *Non-hierarchical* to imply *Hierarchical*, because there is always a subset of the *Non-hierarchical* rules that is *Hierarchical*.

5 Hardness proofs

5.1 Pure rule sets

We organize our proofs in increasing order of hardness, in terms of what types of rules are used. In this subsection, we discuss pure rule sets, where only a single type of rules is used. As an aside, instead of separating ELEMENTGROUP and ANTI-REPETITION rule sets, which are largely mathematically equivalent, we instead prove hardness results for *Hierarchical ANTI-REPETITION* (which includes ELEMENTGROUP) and *Non-hierarchical ANTI-REPETITION*, as that's where the complexity changes.

► **Theorem 4.** *Basic compositional string scoring is in P.*

Proof. There is a trivial poly-time algorithm to construct the highest scoring routine of length m for f_{BASIC} . Choose the skill s that maximizes $p(s) - d(s)$. If $p(s) - d(s) \geq 0$, then the optimal routine is to simply repeat s a total of m times. Otherwise, $p(s) - d(s) < 0$, so the optimal routine is the empty string. ◀

► **Theorem 5.** *Hierarchical ANTI-REPETITION is in P.*

Proof. First, notice that our limit m on the total length of the routine can be encoded as an ANTI-REPETITION rule (Σ, m) , and that this is compatible with *Hierarchical ANTI-REPETITION*. In the absence of specific ANTI-REPETITION rules on individual skills, we can just add ANTI-REPETITION rules for each skill s of the form $(\{s\}, m)$.

We now transform the problem into a minimum-cost maximum flow problem, which can be solved in polynomial time using e.g. linear programming [24]. We can write down all of the ANTI-REPETITION similarity classes ρ_i 's into a tree T encoding their hierarchical structure, with the top-level root being Σ , and the leaves being the individual skills $\{s\}$ for all $s \in \Sigma$. On each of the nodes of the tree T , we have a capacity k corresponding to the number of times that similarity class can be repeated due to the corresponding ANTI-REPETITION rule. Furthermore, on the leaves of the tree, we also have a point value $p(s)$.

Let's construct an auxiliary graph G based off of T , but with the following transformations:

1. We start by making G a directed copy of T , where all the edges point towards the root, and have capacity m .
2. We use a standard trick to convert node capacities into edge capacities. For each vertex v with capacity k , split it into two vertices v^{in} and v^{out} , where v^{in} has all of the incoming edges and v^{out} has all the outgoing edges.
3. Furthermore, there is a directed edge from v_i^{in} to v_i^{out} with capacity k .
4. For each new edge $(\{s\}_{in}, \{s\}_{out})$ corresponding to a leaf $\{s\}$ of T , we assign a cost $d(s) - p(s)$, and we assign a cost of 0 for every other edge.
5. Now augment the graph with a source node with directed edges pointing at each of the leaves, all with capacity m and cost 0.
6. Also augment the graph with a target node with a single edge coming from the root of the tree, with capacity m and cost 0.

17:12 Routine Construction Complexity

7. Lastly, we add another n edges directly from the source to the target corresponding to performing an unrecognized skill; each edge corresponds to a single skill, and has capacity m and cost $d(s)$.

Importantly, this new graph G is still a directed acyclic graph, so there are no negative cycles. After constructing this graph G , integer solutions to the minimum-cost maximum flow problem from the source to the target will correspond to an optimal scoring routine. One way to ensure that we get integer solutions is to put an epsilon perturbation on all of the costs, so that no two paths have the exact same cost, preventing flow from being split in non-integer ways. ◀

► **Theorem 6.** *Non-hierarchical ANTI-REPETITION is NP-hard.*

Proof. We prove this via reduction from the positive one-in-three 3-SAT problem (1-in-3-SAT+), which is NP-hard [18, 26]. In this problem, we are given a family of Boolean variables $\Sigma = \{s_1, \dots, s_n\}$ and a collection of triples $\tau_i \subseteq \Sigma$, where $|\tau_i| = 3$. The goal is to determine whether or not there is an assignment of all the variables such that every triple has exactly one true value.

We now show that we can encode this into a string scoring problem with *Non-hierarchical ANTI-REPETITION* rules. Specifically, we first encode the problem of finding a solution with exactly m variables set to true. Repeating the construction for all m completes the proof.

1. Let Σ be the family of skills, so we have one skill for each Boolean variable.
2. For every skill s , assign a point value $p(s) = 1$ and deduction $d(s) = 0$.
3. For every skill s , add an ANTI-REPETITION rule $(\{s\}, 1)$, so that each skill can only be performed once for credit.
4. Add another ANTI-REPETITION rule corresponding to each triple in the 1-in-3-SAT+ problem, $(\tau_i, 1)$. Thus, at most one skill in each triple can be performed for credit.
5. We also need to ensure that at least one skill in each triple is performed, so add another ANTI-REPETITION rule with the complement of a triple, $(\Sigma \setminus \tau_i, m - 1)$.

Notice that successfully performing a skill for credit gives only 1 point, so the maximum possible score is m points for m skills. However, if even one of those skills is not recognized, or for any routine that is length $< m$, then the maximum possible score would be $< m$. Thus, if an optimal routine could be found that scores m points, then that implies that there is a 1-in-3-SAT+ solution with m variables set to true.

Since there are n variables, the number of true variables in a solution must be between 1 and n . We thus run n instances of the reduction, and if 1-in-3-SAT+ has any satisfying variable assignment with $0 < m' \leq n$ Trues, that will be found in the reduction where we let $m = m'$. Thus, we have a polynomial-time reduction from 1-in-3-SAT+ to *Non-hierarchical ANTI-REPETITION* routine scoring. ◀

► **Theorem 7.** *CONNECTION is in P.*

Proof. We exhibit a dynamic programming algorithm below to find an optimal routine of fixed length m . We model the scoring function as a fully connected directed graph with vertices Σ and edge weights given by the matrix C , where c_{s_i, s_j} is the connection value. Assign weights on the vertices by $w(s) = p(s) - d(s)$. A routine is precisely a path in G , and the score of that routine is precisely the sum of the vertex and edge weights along that path, including the start and end nodes.

For any given starting node, we can use dynamic programming to find the maximum scoring routine of length m in polynomial $O(n^3m)$ time:

1. Compute an $n \times n \times m$ tensor D , where $D(x, y, z)$ is the weight of highest scoring routine of length z starting at skill x and ending at skill y :
 - a. Initialize D with $D(x, x, 0) = w(x)$ for all nodes x , and $D(x, y, 0) = 0$ for any $x \neq y$.
 - b. Recursively compute D by $D(x, y, z) = \max_{y' \in \Sigma} D(x, y', z - 1) + c_{y', y} + w(y)$
2. The actual optimal routines themselves can be stored in an auxiliary data structure.

The reason we were able to do this is because revisiting nodes is allowed without penalty. ◀

► **Corollary 8.** *INCOMPLETEGRAPH is in P.*

Proof. Note that INCOMPLETEGRAPH reduces to CONNECTION in Lemma 1. ◀

5.2 Mixing rule classes

Non-hierarchical ANTI-REPETITION is already NP-hard, so clearly combining it with anything else would still be NP-hard. Additionally though, we show below that combining together *Hierarchical* ANTI-REPETITION with either Connections or Incomplete graph makes things NP-hard as well.

► **Theorem 9.** *Hierarchical ANTI-REPETITION+INCOMPLETEGRAPH is NP-hard.*

Proof. We proceed by reduction from the Hamiltonian path problem, which is NP-complete [18]. In the Hamiltonian path problem, the goal is to find a simple path through a directed graph $G = (V, E)$ such that every node is visited exactly once. Encode each node as a skill s . For each skill s , set its point value $p(s) = 1$ and its deduction $d(s) = 0$. Add an ANTI-REPETITION rule for each skill. Then, the first time a skill is performed, it will be worth 1 point, but every subsequent time, it will be worthless. Set $m = n$, so we are therefore looking for a routine that is as long as there are skills. Clearly, if there is a Hamiltonian path, that corresponds to precisely a routine that does every single skill exactly once, which would be worth m points. Thus, if there is an optimal solution in *Hierarchical* ANTI-REPETITION+INCOMPLETEGRAPH, then there is also a Hamiltonian path in the underlying graph. ◀

► **Corollary 10.** *Hierarchical ANTI-REPETITION+CONNECTION is NP-hard.*

Proof. Recall Lemma 1, which showed that we can encode INCOMPLETEGRAPH as CONNECTION. Then apply Theorem 9 above. ◀

► **Corollary 11.** *CONNECTION+INCOMPLETEGRAPH is in P.*

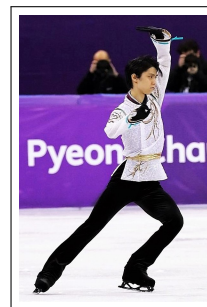
Proof. This follows from Lemma 1 and Theorem 7, because without an ANTI-REPETITION rule, CONNECTION+INCOMPLETEGRAPH is equivalent to just CONNECTION, which we proved to be in P. ◀

► **Corollary 12.** *Any scoring function combining three of Non-hierarchical ANTI-REPETITION, Hierarchical ANTI-REPETITION, CONNECTION, and INCOMPLETEGRAPH is NP-hard.*

Proof. This is clear from the fact that every combination of three rule types includes at least one of the NP-hard combinations above. ◀



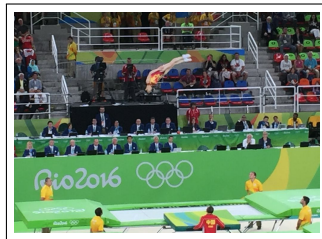
(a) Freestyle Skiing Men's Aerials Final (2010 Winter Olympics) [25].



(b) Free skating (2018 Winter Olympics) [15].



(c) Rhythmic gymnastics (2016 Olympics) [20].



(d) Trampoline (2010 Olympics) [29].



(e) Pommel Horse (2016 Olympics) [23].

■ **Figure 7** Images of each of the Olympic sports (though not all events) we examine.

6 Classification of Olympic Sports

6.1 A Note on Counting “Events”

At the Olympics, an “Event” refers to a contest in which individuals or teams accumulate points over a series of games or performances, and for which a single set of medals is awarded. Under this definition, Free Skate in figure skating (the main component of figure skating with a significant routine construction component) is not an event, but is instead a component of four other events (Men’s and Women’s Single Skating, Pair Skating, and Team Event), all of which contain other performances as well. This is confusing for the purposes of analyzing events with routine construction, since Free Skate is the only component of figure skating featuring difficult routine construction with objective scoring.

For lack of a better term, we use “event” to mean a program during the Olympics in which an athlete or team is asked to perform athletic feats and receives a distinct score, where two events are the same if the activities performed and scoring rules are both substantially the same. Under this definition, Single and Pair Free Skate are both their own event, but Men’s and Women’s Free Skate are the same event.

6.2 Figure Skating

There are seven main events in figure skating: (Men’s and Women’s) Short Program, (Men’s and Women’s) Free Skate, Pair Short Program, Pair Free Skate, Rhythm Dance, Pattern Dance, and Free Dance. Of these, the two Free Skate events are most relevant to our analysis. The dance events are scored primarily for artistry and forbid many of the difficult tricks that characterize the rest of figure skating. In Short Program, athletes perform a sequence of 7 required skills in any order. While this is technically a Hamiltonian Path problem, it is trivially attainable, as skaters reset their position between skills and thus

can complete any skill in any order. We focus our analysis on Single and Pair Free Skate, in which skaters perform a routine of up to 12 technical skills, separated by artful skating around the ice. In the technical component of a routine’s score, each recognized skill is scored according to a Level of Difficulty for the skill, plus a Grade of Execution (GoE) between -5 and $+5$ determined by the judges. While the rules do not bar individual skills from being repeated, routines in the Singles program are limited to a maximum number of skills in each of four categories. [9] The technical Elements Score is joined by a more subjective Program Components score, consisting of both technical (e.g.: Skating) and artistic (e.g.: Composition) components. The relative weights of these scores differs by event.

Focusing only on the more objective Elements score, Free Skate is most naturally modeled as basic compositional scoring with *Hierarchical ANTI-REPETITION*. But there are two complexities in modeling figure skating scoring into our framework. First, some skills actually can be connected to each other, in either a “jump combination” or a “jump sequence.” Fig. 3 contains several examples of these, such as the Triple Flip-euler-triple toe loop combination (3F+1Eu+3S). However, in many ways, such combined elements are treated as a single element: they are given only a single GoE score, and only count as one element towards the maximums. Even if an athlete can combine any of their skills into a jump sequence or combination, so long as their maximum length is bounded, encoding these possibilities into an expanded output is at most a polynomial blowup in the input size.

Second, there is a 10% “fatigue bonus” for up to 3 jumps performed in the second half of a routine in Single Skating, such as the elements marked with an X in Fig. 3. However, since all other routine construction rules are indifferent to element order, this can be optimized by using the algorithm of Thm. 5 to choose the set of elements in the highest scoring routine, and then placing the hardest jumps in the second half, as it appears the athlete has actually done in Fig. 3. An alternative modeling choice is to create a directed graph encoding both the position of skills in a routine and how competitors may not be able to perform their most difficult skills at the end, though it is unclear whether this is actually realistic, and in any case we are not modeling fatigue in any of the other sports.

6.3 Skiing

While most events in Olympic skiing are races scored purely on time to complete a course, there are five “freestyle” events which are judged based on aerial tricks performed: Aerials, Big Air, Slopestyle, Halfpipe, and Mogul. In Aerials and Big Air, skiers complete a series of twists and turns during a single jump off a large ramp; in Slopestyle and Halfpipe, skiers execute a series of tricks as they progress down a course. Mogul skiing consists primarily of athletes constantly skiing around bumps in the snow (“moguls”) while occasionally jumping off a ramp and performing a trick; we exclude it from our analysis as this “Air” component is only 20% of their score, although the Air component is identical to the other events for complexity modeling purposes.

In each of the included freestyle events, competitors perform a sequence of tricks across one or multiple jumps or on other obstacles. Each trick is scored by its Degree of Difficulty along with deductions. This score is then multiplied or augmented by other judged factors. For example, in Aerials, the Degree of Difficulty is multiple by a number between 0 and 10 built out of components for the take-off, height-and-distance, form, and landing for the jump[3]. Although the tricks performed may impact this multiplier, which then applies to all tricks in the jump, we shamelessly ignore this complexity.

The possible sequences of skills in all skiing events are an Incomplete Graph: it is not possible to flip both forwards and backwards in a single jump in aerials, and an athlete may not be able to chain together arbitrary tricks across consecutive jumps in Halfpipe.

17:16 Routine Construction Complexity

While competitors may perform the same trick multiple times in a single jump, and receive separate Degree of Difficulty scores for them, the ANTI-REPETITION rules become muddled for aeriels and big air: they are actually given two jumps across two separate runs, and are forbidden from performing the exact same “routine” in both jumps. However, this is most easily modeled by saying that the goal of a skier is actually to construct the two highest scoring routines, justifying the decision to model skiing events as not containing an ANTI-REPETITION rule. Although commentators discuss that competitors are encouraged to perform different tricks in Slopestyle and Halfpipe, we have been unable to find anything in the rules to that effect. We therefore model all skiing events as basic Compositional Scoring with INCOMPLETEGRAPH.

6.4 Rhythmic Gymnastics

In rhythmic gymnastics, competitors manipulate and juggle a small “apparatus” object such as a ball or hoop while performing a series of dance and contortionist moves, all in time to music. The major emphasis is on simultaneously performing a “Difficulty of Body” (DB) with a “Difficulty of Apparatus” (DA). While DAs recognized do differ with the apparatus, and athletes are required to give multiple performances with different apparatus, the routine construction and scoring rules are the same. However, Team differs significantly from Individual, as e.g.: team members are required to hand or throw their apparatus to another (a “Difficulty of Exchange” [DE]). We thus follow the Olympic classification and deem there to be only two events in rhythmic gymnastics: Individual and Team.

As they are performed simultaneously, we have a choice of modeling DAs and DBs either as a single element (on the grounds they must be practiced together) or as separate elements. However, the latter choice is a more natural fit for the scoring rules: both the DB and DA components have ELEMENTGROUP constraints with minimum and maximum requirements, and both have their own ANTI-REPETITION rules for individual elements. Even without this choice, skills and positions flow smoothly into each other, forming an INCOMPLETEGRAPH. The scores of each element are summed into the Difficulty Score, while their deductions are used to compute the Execution Score. These are summed with an Artistry score to obtain the final score. Of note is that the Artistry Score contains a 0.1 point deduction for “illogical connections,” which can presumably be transformed into a known penalty per pair of elements. Even without including this CONNECTIONS rule, rhythmic gymnastics scoring contains *Hierarchical* ANTI-REPETITION and INCOMPLETEGRAPH, and is thus NP-HARD.

6.5 Trampoline Gymnastics

Trampoline Gymnastics, sometimes just “Trampoline,” is almost self-explanatory: athletes jump on a trampoline and perform a series of twists and flips. The only Olympic events are Men’s and Women’s Individual Trampolining, consisting of both compulsory and voluntary (free) routines (called “exercises”). In Voluntary Exercise, competitors design a routine of 10 elements. Each element consists of a jump with a series of flips, twists, or other acrobatic body manipulations. Repetitions are disallowed. The Difficulty score is then the sum of the scores for each element, while the Execution score is 20 minus the deductions for the elements. These are then added to electronically-determined scores for horizontal displacement and time of flight, which are effectively another form of per-element execution deduction. [8]

It is difficult to connect arbitrary skills in trampolining. A trampoline athlete who can do both front and back flips will need additional training to perform them in sequence. Thus, we model trampoline scoring as *Hierarchical* ANTI-REPETITION with INCOMPLETEGRAPH. However, it may effectively form a complete graph to sufficiently advanced athletes, as all moves are supposed to start and end in identical straight jump positions.

Outside of Individual Trampoline at the Olympics, Trampoline Gymnastics features three other non-Olympic events: Synchronized Trampoline, where two athletes perform the same routine on separate trampolines; Power Tumbling (aka “Tumbling”), where athletes execute a series of handsprings and flips down a 25-meter sprung track; and Double Mini-Trampoline, where competitors run at a small trampoline, and perform a skill off it onto a second trampoline, whence they perform a second skill onto a mat. Synchronized Trampoline and Power Tumbling have the same structure as Individual Trampoline with regards to our analysis. Double Mini-Trampoline contains exactly two skills, and would be excluded from our analysis. Also of note is that the requirements for junior age groups pose a rare example of *Non-Hierarchical* ELEMENTGROUP requirements, requiring e.g.: one element *to* front or back and one element *from* front or back, although the same element may not count for both.

6.6 Artistic Gymnastics

Artistic gymnastics, usually just referred to as “gymnastics,” is among the most recognizable Olympic sports, being one of the original 9 sports featured in the 1896 Summer Olympics. Artistic Gymnastics is split into Men’s Artistic Gymnastics (MAG) and Women’s Artistic Gymnastics (WAG). The MAG events are Floor, Pommel Horse, Rings, Vault, Parallel Bars, and Horizontal Bar (a.k.a.: High Bar), while the WAG are Vault, Uneven Bars, Balance Beam, and Floor. Unlike other sports such as figure skating with separate men’s/women’s divisions, the shared apparatus between MAG and WAG (Floor and Vault) have very different scoring rules and sets of recognized moves; most saliently, WAG floor routines are set to music and have dance requirements, while MAG floor routines are silent and have none.

In both MAG and WAG Vault, the athlete performs only one skill, and hence we exclude both from our analysis. The other 8 events all demand lengthy routines subject to complicated scoring rules. We summarize below the official Code of Points for both disciplines. [5, 6]

All MAG events share some common rules of scoring and routine construction, as do all WAG events. We describe the common principles behind both here. A routine consists of a sequence of elements. The Code of Points provides a long list of recognized elements per event, split into a handful of element groups. All MAG events have 3 or 4 element groups and require one skill from each; Repeated elements are never counted. All events except for Rings and Pommel Horses feature CONNECTIONS, with a small Connection Bonus awarded for performing two difficult skills in succession. Repeated skills are never counted. Routines are capped at 10 scored skills,

The total score of a routine is the sum of a Difficulty (“D-jury”) and Execution (“E-jury”) score. The Difficulty score consists primarily of the sum of the difficulty value of each element, along with any Connection Bonuses, and, in WAG, awards for Composition Requirements as described below. Each element is given a difficulty between A and G, with A skills being worth 0.1 points, B skills being worth 0.2, etc. The Execution score is 10 minus the deductions for all the skills performed, where each skill has an enumerated set of possible flaws and the deduction associated with each.

Position and momentum are very important in initiating skills, and so all events form an INCOMPLETEGRAPH. For example, in both Parallel Bars and Rings, most moves can only be initiated from either the position of handstand, of torso above the apparatus (support), or of torso below the apparatus (hang).

The WAG events feature cross-cutting Composition Requirements; see, e.g.: Fig. 5b. Because of these, both Uneven Bars and WAG Floor have *Non-hierarchical* ANTI-REPETITION rules. Pommel horse is the only MAG event to have *Non-hierarchical* ANTI-REPETITION rules, thanks in large part to Article 11.2.2 paragraph 3b, limiting the competitor to two “Russian

Wendeswing” moves, a set that crosses multiple element groups. The requirement to use all parts of the pommel horse can also be modeled as an extra element group which cuts across the other element groups, as certain moves are only done off the center or sides of the horse. In classifying these events but not the others as *Non-hierarchical* ANTI-REPETITION, we make the somewhat arbitrary extrapolation that additional ANTI-REPETITION rules added to these events but not others may intersect non-hierarchically with existing ELEMENTGROUP and ANTI-REPETITION rules. However, all of these events would be NP-hard even without any ANTI-REPETITION rules.

Outside of these, gymnastics events feature a staggering number of special scoring and composition rules (see e.g.: Figs 4, 5, 6), and our modeling captures most but not all of them. Two notable rules not captured are the Short Routine penalty, a deduction for any routine under 6 moves, and the requirement that any MAG floor routine pass through all four corners of the floor. But any high-scoring routine will already satisfy these, and we have shown the problem is NP-hard even without considering these constraints.

7 Discussion

7.1 Other Sports

We decided to restrict the scope of this paper to Olympic sports, lest we be compelled to investigate competition rules for every form of dance. Several other Olympic sports feature judged routines, but were excluded for various reasons. Equestrian (Dressage) and snowboarding (Big Air, Halfpipe, and Slopestyle) lack an objective Code of Points. [4, 10] Diving, Vault in artistic gymnastics, and Double Mini-Trampoline in trampoline gymnastics all feature “routines” of at most one or two skills, and are trivially computationally tractable.

Artistic Swimming was a close call for inclusion. Its Technical Routine event features objective scoring assigning a “Degree of Difficulty” and deductions to a series of acrobatic elements, but the set and sequence of skills is fixed. Its “Free Routine” and “Highlight Routine” events allow competitors to construct their own routines (with ELEMENTGROUP constraints for Highlight Routine). However, these events lack an objective scoring component. Indeed, the authors confused these events and originally wrote this paper with Artistic Swimming included as a 6th sport. However, there is a new ruleset under proposal that will add element-based objective scoring to all Artistic Swimming events. [22] This new ruleset bases the difficulty score on the sum of per-element scores for both acrobatic and “hybrid” elements (long sequences of underwater leg movements), where each hybrid is composed of a large number of individually-scored leg movements. Whether the new rules make routine construction NP-hard depends on the tractability of hybrid scoring, and whether Free Routine and Highlight Routine keep their current lack of ANTI-REPETITION rules.

We should emphasize that our results here depend on a generalization where we allow the routine length, number of potential skills, and number of rules to grow. Thus, we are not measuring directly the hardness of routine construction in a specific sport, but rather instead the hardness of routine construction in a hypothetical sport that is allowed to use the rule types we consider.

Furthermore, although we attempted to choose our rule types to cover as much of the technical scoring functions given in the codes of points, many sports have one-off rules that do not exactly fit into any of our classes directly. In the previous section, we discuss some of these idiosyncrasies. In the interest of full disclosure, we should note that this article is written by two club gymnasts, and there is possibly a gymnastics-centric bias in which aspects of the sports we chose to model and how we chose to generalize them, in particular in our choice to ignore “subjective” artistic scores, which also often encourage a diversity of skills and interesting connections.

7.2 Related Work

Although we are to our knowledge the first to examine the complexity of routine construction, there does exist a body of literature on the strategy surrounding multiteam tournament [21, 16]. Many sports that are structured as 2-player competitive games; in soccer or hockey, the objective is to score more points than the opposing team, but the opposing team can (and should!) interfere with your attempts to score. However, often, there are more than 2 countries competing in a tournament, so there is some complexity in the means through which pair-wise games are used to determine a single gold medalist; often, there may not be a well-defined ordering such that the winner would necessarily beat every other player in a direct competition. E.g., some variants of FIFA soccer rules make it NP-hard to ask if a specific team still has a chance of winning or coming in a particular place [21].

Outside of sports, string scoring functions have also played a role in bioinformatics, such as in predicting the melting temperature of nucleic acids [28] or the prediction of cleavage sites in protein sequences [14]. However, most of these scores are based on k -mer composition, rather than explicitly considering ordering in the way we do here. The more complex bioinformatics string applications tend to be in directly comparing two similar strings, such as in sequence alignment [27], rather than in scoring individual strings.

References

- 1 U.S. Figure Skating Scoring Guide. URL: <https://usfigureskating.org/sites/default/files/media-files/Scoring%20Cheat%20Sheet.pdf>.
- 2 Element Symbols for Men's Artistic Gymnastics, May 2015. URL: https://www.gymnastics.sport/publicdir/rules/files/en_MAG%20Element%20Symbols%20Booklet.pdf.
- 3 FIS Freestyle Skiing Judging Handbook, October 2018. URL: https://assets.fis-ski.com/image/upload/v1540187845/fis-prod/Freestyle_Skiing_Judging_Handbook.pdf.
- 4 Judges Handbook, Snowboard & Freeski, June 2019. URL: https://assets.fis-ski.com/image/upload/v1610953451/fis-prod/assets/FIS_SB_FK_JudgesHandbook_20_21.pdf.
- 5 2022 – 2024 Code of Points: Men's Artistic Gymnastics, February 2020. URL: https://www.gymnastics.sport/publicdir/rules/files/en_MAG%20CoP%202022-2024.pdf.
- 6 2022 – 2024 Code of Points: Women's Artistic Gymnastics, February 2020. URL: https://www.gymnastics.sport/publicdir/rules/files/en_WAG%20CoP%202022-2024.pdf.
- 7 2022 – 2024 Code of Points: Rhythmic Gymnastics, December 2021. URL: https://www.gymnastics.sport/publicdir/rules/files/en_2022-2024%20RG%20Code%20of%20Points.pdf.
- 8 2022 – 2024 Code of Points: Trampoline Gymnastics, May 2021. URL: https://www.gymnastics.sport/publicdir/rules/files/en_TRA%20CoP%202022-2024.pdf.
- 9 Special Regulations & Technical Rules: Single & Pair Skating and Ice Dance, 2021, June 2021. URL: <https://www.isu.org/figure-skating/rules/fsk-regulations-rules/file>.
- 10 Dressage Rules; 25th edition, effective 1st January 2014, January 2022. URL: https://inside.fei.org/sites/default/files/FEI_Dressage_Rules_2022_Clean_Version_V2.pdf.
- 11 Aviv Adler, Jeffrey Bosboom, Erik D Demaine, Martin L Demaine, Quanquan C Liu, and Jayson Lynch. Tatamibari is NP-complete. *arXiv preprint arXiv:2003.08331*, 2020.
- 12 Aviv Adler, Erik D Demaine, Adam Hesterberg, Quanquan Liu, and Mikhail Rudoy. Clickomania is Hard, Even With Two Colors and Columns. *The Mathematics of Various Entertaining Subjects: Research in Games, Graphs, Counting, and Complexity*, 2:325, 2017.
- 13 Greg Aloupis, Erik D Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo games are (Computationally) Hard. *Theoretical Computer Science*, 586:135–160, 2015.

17:20 Routine Construction Complexity

- 14 Christina Backes, Jan Kuentzer, Hans-Peter Lenhof, Nicole Comtesse, and Eckart Meese. GraBCas: a Bioinformatics Tool for Score-Based Prediction of Caspase-and Granzyme B-Cleavage Sites in Protein Sequences. *Nucleic Acids Research*, 33(suppl_2):W208–W213, 2005.
- 15 David W. Carmichael. File:2018 Winter Olympics - Yuzuru Hanyu FS (1).jpg. [https://commons.wikimedia.org/wiki/File:2018_Winter_Olympics_-_Yuzuru_Hanyu_FS_\(1\).jpg](https://commons.wikimedia.org/wiki/File:2018_Winter_Olympics_-_Yuzuru_Hanyu_FS_(1).jpg), February 2018.
- 16 Jan Christensen, Anders Nicolai Knudsen, and Kim S Larsen. Soccer is Harder than Football. *International Journal of Foundations of Computer Science*, 26(4):477–486, 2015.
- 17 Erik D Demaine, Martin L Demaine, Sarah Eisenstat, Adam Hesterberg, Andrea Lincoln, Jayson Lynch, and Y William Yu. Total Tetris: Tetris with Monominoes, Dominoes, Trominoes, Pentominoes,... *Journal of Information Processing*, 25:515–527, 2017.
- 18 Michael R Garey and David S Johnson. *Computers and Intractability*, volume 174. Freeman San Francisco, 1979.
- 19 Olympics Gymnastics. Women’s Uneven Bars Final | Tokyo Replays. https://www.youtube.com/watch?v=I84it6P5_i8, October 2021.
- 20 Ilgar Jafarov. File:Rhythmic gymnastics at the 2016 Summer Olympics, Marina Durunda 30.jpg. https://commons.wikimedia.org/wiki/File:Rhythmic_gymnastics_at_the_2016_Summer_Olympics,_Marina_Durunda_30.jpg, November 2016.
- 21 Walter Kern and Daniël Paulusma. The New FIFA Rules are Hard: Complexity Aspects of Sports Competitions. *Discrete Applied Mathematics*, 108(3):317–323, 2001.
- 22 Christina Marmet. What to Expect From the New Scoring System? URL: <https://insidesynchro.org/2021/04/08/what-to-expect-from-the-new-scoring-system/>.
- 23 Javid Nikpour/Tasnimnews. Gymnastics at the 2016 Summer Olympics. <https://www.tasnimnews.com/en/news/2016/08/12/1155951/gymnastics-at-the-2016-summer-olympics/photo/3>, August 2016.
- 24 James B Orlin. A Polynomial Time Primal Network Simplex Algorithm for Minimum Cost Flows. *Mathematical Programming*, 78(2):109–129, 1997.
- 25 Duncan Rawlinson. Freestyle Skiing Men’s Aerials Final 19 . <https://www.flickr.com/photos/44124400268@N01/4392687057>, February 2010.
- 26 Thomas J Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.
- 27 Temple F Smith, Michael S Waterman, et al. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.
- 28 Masamichi Tsuboi. On the Melting Temperature of Nucleic Acid in Solution. *Bulletin of the Chemical Society of Japan*, 37(10):1514–1522, 1964.
- 29 Sander van Ginkel. Rio 2016 Summers Olympics. <https://flickr.com/photos/139991533@N04/28558139233>, August 2016.

How Brokers Can Optimally Abuse Traders

Manuel Lafond  

University of Sherbrooke, Canada

Abstract

Traders buy and sell financial instruments in hopes of making profit, and brokers are responsible for the transaction. There are several hypotheses and conspiracy theories arguing that in some situations, brokers want their traders to lose money. For instance, a broker may want to protect the positions of a privileged customer. Another example is that some brokers take positions opposite to their traders', in which case they make money whenever their traders lose money. These are reasons for which brokers might manipulate prices in order to maximize the losses of their traders.

In this paper, our goal is to perform this shady task optimally – or at least to check whether this can actually be done algorithmically. Assuming total control over the price of an asset (ignoring the usual aspects of finance such as market conditions, external influence or stochasticity), we show how in quadratic time, given a set of trades specified by a stop-loss and a take-profit price, a broker can find a maximum loss price movement. We also look at an online trade model where broker and trader exchange turns, each trying to make a profit. We show in which condition either side can make a profit, and that the best option for the trader is to never trade.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Algorithms, trading, graph theory

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.18

Funding The author acknowledges financial support from the Natural Sciences and Engineering Research Council (NSERC) and from the Fonds de Recherche du Québec – Nature et technologies (FRQNT) for this research.

1 Introduction

Trading is the practice of buying or selling financial assets with the aim of making a profit. A trader can buy an instrument at some price p and sell it at price p' , making a profit of $p' - p$ (which might be negative). The trader can also sell an unowned instrument at some price p with the obligation to buy it back someday, say at a time where the new price is p' , making a profit of $p - p'$ (this is called *shorting*). A *broker* is usually responsible for the execution of a trade, taking care of the technical aspects of the transaction.

This power over trade execution has given rise to several conspiracy theories. This has been especially prevalent in the year 2021 where unprecedented financial events occurred. One of them was the rise and fall of the GameStop (GME) stock that started in January. In short, large institutions had significantly shorted the stock and, in a concerted counteract effort, retail traders massively bought it and made its price go up 30-fold. At this point, several traders were unable to buy more GME stocks, the transaction being blocked by their broker (see [3]). This may have been caused by technical issues, but of course the Internet accused brokers of manipulating prices to protect the short positions of their large clients. The stock went back near its original price, allowing the shorts to limit their losses, and then stabilized a bit higher in the months after. The year 2021 has also seen cryptocurrencies gain monstrous gains in value. They are under less regulations than stocks and have been suspected of shady price manipulation techniques, for instance pumps and dumps, since their rise in popularity [1]. As a final example, the foreign exchange currency market is largely managed by brokers called *market-makers*. These place trades in the opposite direction of their traders – if a trader wants to buy an asset, the market-maker will sell it, and if the



© Manuel Lafond;

licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 18; pp. 18:1–18:19

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

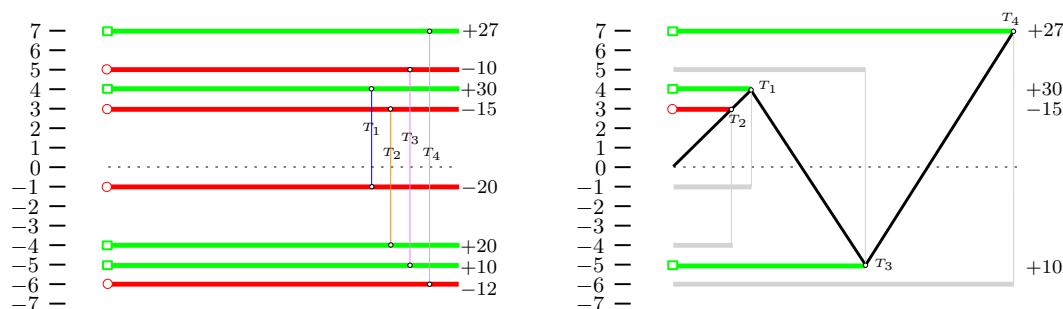
trader wants to sell it, they will buy it. After all, there has to be two parties involved in a transaction, and market-makers assume one of the roles. This gives them an incentive for their clients to perform poorly. See [2] for a gentle introduction on market makers and [14] for a thesis on the topic. Price manipulation theories are based on the arguments that large brokerage firms have access to enough funds to control prices to some extent, along with statistics showing that a majority of traders lose money (see [4]).

Now, taking such accusations seriously is likely to make economists jump out of their seats, since there are too many factors driving asset prices for a single entity to control it. But here, we rather take an algorithmic perspective on these theories. To take it to the extreme, suppose that brokers have total control over the prices. Armed with the knowledge of every trade that is currently open, their goal is to make people lose as much money as possible. The question is: even with the ultimate power of price manipulation, *can they?* Is this optimization problem easy? In this paper, we wear the hat of (would-be) mischievous brokers and devise price manipulation algorithms to do our evil bidding. Note that this question has another interpretation: as a trader, what is the worst that could happen with a set of opened trades¹? There seems to be no algorithmic answer in the literature for this simple question. Algorithms exist for the seemingly related notions of *value at risk* [10] and *maximum loss* [13], but these measures are based on stochastic prices, operate on multiple assets and are subject to various market conditions, unlike here where we assume full control.

Our contributions. We model trades as bounded by two closing prices - a winning and a losing price. This is typical in trading: traders want to limit their risk and often set up a *stop-loss*, a price at which the trade closes automatically when too much losses are incurred. This is usually accompanied with a *take-profit* price, which closes the trade when its profit is high enough. Brokers can use these two pieces of information to their advantage, leading to several problems. First, we study the *offline* problem where, given a set of trades, we ask for a price movement that maximizes trader losses. We show that this problem reduces to finding a maximum independent set on the *trade conflict graph*, which exhibits which trades cannot be won simultaneously. This graph is bipartite and thus our problem can be solved in polynomial time using maximum flow techniques [11, 12]. By digging a bit deeper, we develop a specialized $O(n^2)$ time algorithm by characterizing trade conflict graphs geometrically. That is, we define an equivalent graph class called *bicolored plane domination graphs*, which are shown to be chordal bipartite (see [5]), and on which dynamic programming can be performed for our purposes. This class is interesting in itself and leads to several open algorithmic and graph theoretical problems.

Second, we look at the *online* setting. That is, the trader can add a new trade or close a trade at any given time, and the broker still has to maximize losses. This leads to a two-player game where the trader and broker exchange turns. We show that there is essentially one strategy that the broker must use, which is to always move the price greedily in the direction of maximum *potential* profit. In particular, using the independent set algorithm from above in the online setting incurs infinite losses against an optimal trader. We conclude by showing that if the broker uses this strategy, the trader should simply not open any trade and get rich through other means.

¹ This suggests that traders could open trades in opposing directions. This appears to make little sense, but it has its advantages – in fact, this is a well-known risk management technique called *hedging*.



■ **Figure 1** Left: an example input for the Maximum Trader Abuse problem with $|\mathcal{T}| = \{T_1, T_2, T_3, T_4\}$. Each trade has a winning price (green lines that start with a square) and a losing price (red lines that start with a circle). Each winning and losing price has a corresponding profit indicated on the right. For instance, $T_1 = (4, -1, f)$ where $f(4) = 30$ and $f(-1) = -20$. (b) An optimal price movement M for \mathcal{T} that makes a total profit of $30 - 15 + 10 + 27 = 52$. The green/red (square/circle) lines now indicate the lifespan of each trade according to M . The profit points not realized by M are grayed-out. Only T_2 is lost since price 3 is reached before -4 .

2 Preliminary notions

We use the notation $[a] = \{1, 2, \dots, a\}$ and $[a, b] = \{x \in \mathbb{Z} : a \leq x \leq b\}$. Given an integer $x \in \mathbb{Z}$, the sign of x is denoted $\text{sgn}(x)$, which is either 1 or -1 . All graphs in this paper are finite and simple. For a graph G and $X \subseteq V(G)$, $G[X]$ denotes the subgraph induced by X . A *weighted graph* is a pair (G, h) where G is a graph and $h : V(G) \rightarrow \mathbb{R}$ assigns a weight to each vertex. For $X \subseteq V(G)$, we write $h(X)$ for the sum of weights of vertices in X . Some of our routines will require to sort integers. We will write *integer sorting time* to refer to the time required to sort a list of n arbitrary integers, when n is clear from the context (this can range from $O(n \log \log n)$ to $O(n)$ depending on conditions that we prefer to leave out of consideration [9, 8]).

Regarding the financial notions used in this paper, a *price* is simply an integer, possibly negative. Using integral prices is justified by the fact that in trading, prices are usually handled to the fourth of fifth decimal and may easily be treated as integers. All trades are done on a single asset, and for our purposes no fee is required to open a trade (although the fees could be embedded in the profit functions described below). We assume that the current price of the asset is 0.

A *trade* $T = (w, \ell, f)$ consists of three parameters: w is the winning price from the broker's point of view, ℓ is the losing price from the broker's point of view, and $f : \{w, \ell\} \rightarrow \mathbb{R}$ is the profit made by the broker if the price is attained before the other (see Figure 1 and the description of price movements below). Note that profits can be negative. We require that either $\ell < 0 < w$, in which case T is an *up* trade, or $w < 0 < \ell$, in which case T is a *down* trade. We will assume that $f(w) \geq f(\ell)$ (otherwise, we can flip the roles of w and ℓ).

Note that in a typical real-life trade setting, we have $f(w) > 0$ since w represents a profit point and $f(\ell) < 0$ since ℓ it represents a loss point. Moreover, profits and losses are usually linearly related, i.e. there is some δ such that $f(w) = f(\ell) + \delta \cdot |w - \ell|$ (as is the case in Figure 1). However, such restrictions will not be needed for our algorithms until we discuss online trades, and we prefer to keep f generic. As another side note, in our terminology, an *up* trade corresponds to a *sell* from the trader since the mischievous broker makes profit when the trader loses, and likewise a *down* trade is a *buy* from the trader. Since we take the broker's viewpoint in this paper, trades are described with the broker's preferred direction.

18:4 How Brokers Can Optimally Abuse Traders

A price movement $M = (m_1, \dots, m_k)$ is a sequence where $m_i \in \{+1, -1\}$ for each $i \in [k]$. The current price after the j -th movement is $\sum_{i=1}^j m_i$. Consider a trade $T = (w, \ell, f)$ and a price movement M . If, under M , price w is reached before ℓ , then T is *won* and a profit of $f(w)$ is made, and if instead price ℓ is reached before w , then T is *lost* and a profit of $f(\ell)$ is incurred. The first price in $\{w, \ell\}$ reached by M is denoted $close_M(T)$ and is called the *closing price* of T under M . If M does not reach either w or ℓ , then $close_M(T)$ is undefined. If it is defined, then the broker's profit made from T under price movement M can be written as $f(close_M(T))$. Given a set of trades \mathcal{T} , we say that price movement M is *valid for \mathcal{T}* if $close_M(T)$ is defined for every $T \in \mathcal{T}$. If M is valid, the *total profit* of M for \mathcal{T} is

$$profit_{\mathcal{T}}(M) = \sum_{T=(w,\ell,f) \in \mathcal{T}} f(close_M(T))$$

The problem statement follows:

The **Maximum Trader Abuse** problem:

Given: a set of trades \mathcal{T} ;

Find: a valid price movement M that maximizes $profit_{\mathcal{T}}(M)$.

We will denote $n = |\mathcal{T}|$ unless stated otherwise. We will say that $\mathcal{T}' \subseteq \mathcal{T}$ is an *optimal set of trades* if there exists a price movement M that maximizes $profit_{\mathcal{T}}(M)$ such that the set of trades won by M is precisely \mathcal{T}' . This definition implies that the problem is equivalent to finding an optimal set of trades.

Observe that we could have removed the validity requirement on M in the problem definition. This would allow the broker to leave some trades open forever by never reaching one of their closing prices. This appears to be an important difference, and we leave it as an open question whether our techniques can handle this variant. On another note, let us briefly discuss the size of M . Consider a trade $T = (w, \ell, f) \in \mathcal{T}$, and notice that w or ℓ are not necessarily polynomial in n . Since M has to close T , representing M as a sequence of unit movements might yield an exponential-size output, which is not necessary. Instead, M can be described by the sequence of prices in which it changes direction, from which the $+1/-1$ sequence can easily be inferred. That is, we may represent the movement as a sequence of prices $M = (p_1, p_2, \dots, p_k)$, where $p_1 = 0$ is the initial price. Then for each $i \in \{0, \dots, k-1\}$, $|p_{i+1} - p_i|$ steps must be performed in the same direction (up if $p_{i+1} > p_i$, and down otherwise). In fact, each p_i can be assumed to be one of the values in $\bigcup_{(w,\ell,f) \in \mathcal{T}} \{w, \ell\}$ since only these determine profits, which guarantees that the output can be made of linear size.

3 Maximizing trader abuse in quadratic time

We first show that finding an optimal M reduces to finding a maximum weight independent set in a bipartite graph, which we call the *trade conflict graph*, that describes which pairs of trades cannot be won together. The latter problem can be solved in polynomial time using standard maximum flow techniques. We then study the trade conflict graph a bit more in-depth and show that its properties lead to a $O(n^2)$ time algorithm. This is achieved by representing the trade as a set of green and red points on the 2D plane where conflicts correspond to bicolored domination relationships.

We say that a set of trades \mathcal{T} is *compatible* if there exists a price movement M such that for all $(w, \ell, f) \in \mathcal{T}$, we have $close_M(T) = w$. Otherwise \mathcal{T} is *incompatible*. Two trades T_1, T_2 are compatible if the set $\{T_1, T_2\}$ is compatible (and otherwise incompatible). Pairs of incompatible trades are easy to characterize.

► **Lemma 1.** *Let $T_1 = (w_1, \ell_1, f_1)$ and $T_2 = (w_2, \ell_2, f_2)$ be two trades. Then T_1 and T_2 are incompatible if and only if $\text{sgn}(w_1) \neq \text{sgn}(w_2)$, $|\ell_1| \leq |w_2|$ and $|\ell_2| \leq |w_1|$.*

Proof. First assume that w_1 and w_2 have a different sign, and that $|\ell_1| \leq |w_2|$ and $|\ell_2| \leq |w_1|$ hold. As both trades are of opposite signs, it is impossible to reach price w_2 without reaching ℓ_1 first (or simultaneously if $w_2 = \ell_1$), and it is impossible to reach w_1 without reaching ℓ_2 first (or simultaneously). Thus, one of the loss prices must be hit, making the trades incompatible.

We prove the converse direction by contraposition. Assume that one of the three conditions does not hold. Suppose first that $\text{sgn}(w_1) = \text{sgn}(w_2)$. If $w_1, w_2 > 0$, then the price movement that always goes up wins both trades, and if $w_1, w_2 < 0$, we can make the price go down. In either case, T_1 and T_2 are compatible. We may thus assume that $\text{sgn}(w_1) \neq \text{sgn}(w_2)$. If $|\ell_1| > |w_2|$, we first win T_2 by moving the price to w_2 . This does not reach ℓ_1 and so T_1 is still open at this point. It then suffices to go from w_2 to w_1 , winning both trades. The same idea applies when $|\ell_2| > |w_1|$. ◀

For example, trades T_1 and T_2 in Figure 1 are incompatible.

Obviously, if a set of trades \mathcal{T} has two trades T_1, T_2 that are incompatible, then they cannot both be won and trivially \mathcal{T} is incompatible. Luckily, the converse also holds.

► **Lemma 2.** *Let \mathcal{T} be a set of trades in which every pair of trades is compatible. Then \mathcal{T} is compatible. Moreover, one can construct in integer sorting time a price movement M that wins every trade of \mathcal{T} .*

Proof. We use induction on $|\mathcal{T}|$. The case $|\mathcal{T}| = 1$ is trivial: it suffices to go to the single winning price. Now assume that $|\mathcal{T}| > 1$. Note that for any proper subset \mathcal{T}' of \mathcal{T} , each pair of \mathcal{T}' is compatible. We may thus assume by induction that the statement holds for any such \mathcal{T}' . For a price p , denote $\mathcal{T}(p) = \{(w, \ell, f) \in \mathcal{T} : p = w \text{ or } p = \ell\}$. Let p^+ be the minimum value above 0 such that $\mathcal{T}(p^+) \neq \emptyset$, and let p^- be the maximum value below 0 such that $\mathcal{T}(p^-) \neq \emptyset$. If all trades of $\mathcal{T}(p^+)$ are won at price p^+ , we can raise the price from 0 to p^+ , win these trades, bring the price back to 0 and apply induction on $\mathcal{T} \setminus \mathcal{T}(p^+)$ to win every other trade. The same applies if all trades of $\mathcal{T}(p^-)$ are won at p^- . So suppose that there are $T_1 = (w_1, \ell_1, f_1) \in \mathcal{T}(p^+)$ and $T_2 = (w_2, \ell_2, f_2) \in \mathcal{T}(p^-)$ that are losing, i.e. such that $\ell_1 = p^+$ and $\ell_2 = p^-$. Then T_1 is a *down* trade and T_2 is an *up* trade, and hence $\text{sgn}(w_1) \neq \text{sgn}(w_2)$. Moreover, $w_2 \geq \ell_1 > 0$ by our choice of p^+ , and similarly $w_1 \leq \ell_2 < 0$. These imply that $|\ell_1| \leq |w_2|$ and $|\ell_2| \leq |w_1|$ and, by Lemma 1, T_1 and T_2 are incompatible, a contradiction. It follows that some price movement can win every trade of \mathcal{T} .

One can derive a sorting time algorithm from the above argument. First obtain a sorted list A_+ of the all the trades (w, ℓ, f) with respect to $\max\{w, \ell\}$ in ascending order, and a sorted list A_- of all the trades with respect to $\min\{w, \ell\}$ in descending order. The first element of A_+ has price p^+ , and the first price in A_- is p^- . By advancing through A_+ (resp. A_-) as long as the trade prices are p^+ (resp. p^-), we build the set L_+ (resp. L_-) of the trades that are lost at price p^+ (resp. p^-). By the above argument, one of L_+ or L_- is empty. If, say, $L_+ = \emptyset$, we move the price to p^+ , win every trade at that price, and add those trades to a set C of closed trades. We also remove from L_- those trades that just closed. We then move forward in A_+ to find the next p^+ and the next L_+ and, if necessary, we advance through A_- and find the next L_- . Again, one of A_+ or A_- will be empty, so we repeat. The process is the same if L_- is empty instead. Note that in subsequent iterations, one must check whether a trade is in C before adding it to L_+ or L_- . The whole procedure can be done by traversing A_+ and A_- once, and by adding/removing each trade in L_+ or L_- at most once, and $O(n)$ price movements are added, which takes overall $O(n)$ time after sorting. ◀

18:6 How Brokers Can Optimally Abuse Traders

Lemma 2 essentially states that we can devote our efforts to finding a set of pairwise-compatible trades $\mathcal{T}' \subseteq \mathcal{T}$ that maximizes profit, although we need to account for the losses incurred by the trades *not* in \mathcal{T}' . We can thus reformulate the problem in graph-theoretical terms.

► **Definition 3.** *The trade conflict graph $G(\mathcal{T}) = (V, E)$ of a set of trades \mathcal{T} is the graph with vertex set $V = \mathcal{T}$, and in which there is an edge between each pair of incompatible trades.*

Furthermore, the weighing of \mathcal{T} is the function $h : \mathcal{T} \rightarrow \mathbb{R}$ that assigns the weight $f(w) - f(\ell)$ to each trade $(w, \ell, f) \in \mathcal{T}$.

Note that all weights of h are non-negative since $f(w) \geq f(\ell)$ is assumed to hold for all trades. Also observe that by Lemma 1, testing compatibility on a pair of trades can be done in constant time, and thus $G(\mathcal{T})$ can be built in time $O(n^2)$. Perhaps $G(\mathcal{T})$ could be constructed faster with more refined ideas, but we shall not dwell on this. Now, as per Lemma 2, $\mathcal{T}' \subseteq \mathcal{T}$ is compatible if and only if \mathcal{T}' forms an independent set in $G(\mathcal{T})$ (a set of vertices with no shared edges). As we show below, the h weighing is adjusted so that our problem reduces to finding a maximum weight independent set.

► **Theorem 4.** *Let \mathcal{T} be a set of trades. Then $\mathcal{T}^* \subseteq \mathcal{T}$ is an optimal set of winning trades if and only if \mathcal{T}^* is a maximum weight independent set in $(G(\mathcal{T}), h)$, where h is the weighing function of \mathcal{T} .*

Proof. Let M be any price movement that closes every trade and let \mathcal{T}' be the set of trades won by M . Note that \mathcal{T}' must be an independent set of $G(\mathcal{T})$. Moreover, the profit incurred by M is

$$\begin{aligned} \sum_{T \in \mathcal{T}} \text{close}_M(T) &= \sum_{(w, \ell, f) \in \mathcal{T}'} f(w) + \sum_{(w, \ell, f) \in \mathcal{T} \setminus \mathcal{T}'} f(\ell) \\ &= \sum_{(w, \ell, f) \in \mathcal{T}'} f(w) + \left(\sum_{(w, \ell, f) \in \mathcal{T}} f(\ell) - \sum_{(w, \ell, f) \in \mathcal{T}'} f(\ell) \right) \\ &= \sum_{(w, \ell, f) \in \mathcal{T}'} (f(w) - f(\ell)) + \sum_{(w, \ell, f) \in \mathcal{T}} f(\ell) \\ &= \sum_{T \in \mathcal{T}'} h(T) + \sum_{(w, \ell, f) \in \mathcal{T}} f(\ell) \end{aligned}$$

The term $t := \sum_{(w, \ell, f) \in \mathcal{T}} f(\ell)$ does not depend on the choice of M and can be ignored since it does not contribute to the optimization criterion. Therefore, each M of profit $k + t$ corresponds to an independent set \mathcal{T}' of weight k , and conversely each independent set of weight k corresponds to a price movement of profit at least $k + t$, by Lemma 2. It follows that finding an optimal M is equivalent to finding an independent set in $G(\mathcal{T})$ of maximum weight with respect to h . ◀

It is not hard to see that $G(\mathcal{T})$ is bipartite. Indeed, letting $\mathcal{T}^+ = \{(w, \ell, f) \in \mathcal{T} : w > 0\}$ and $\mathcal{T}^- = \{(w, \ell, f) \in \mathcal{T} : w < 0\}$, one can see by Lemma 1 that $\{\mathcal{T}^+, \mathcal{T}^-\}$ is a partition of \mathcal{T} into two independent sets. Finding a maximum weight independent set in a bipartite graph G can be done in polynomial-time using a maximum-flow reduction. This can be implemented to run in time $O(n^3)$ using the Stoer-Wagner min-cut algorithm [12] or a variety of max-flow algorithms that run in time $O(nm)$, where m is the number of conflicts [11].

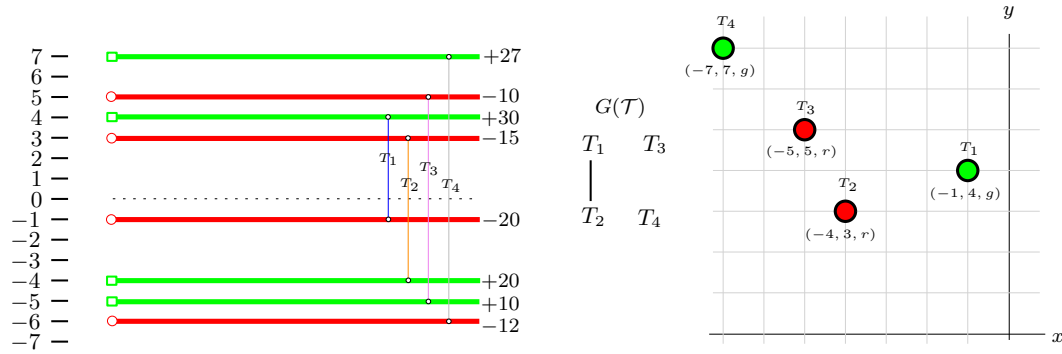


Figure 2 Left: the four trades from the previous example. Middle: the graph $G(\mathcal{T})$, which has only one incompatible pair. Right: A bicolored plane domination model for $G(\mathcal{T})$. This is the model used in the proof of Theorem 5.

3.1 Bicolored plane domination models for trade conflict graphs

We now reformulate trade conflict graphs in terms of two-colored points on the 2D plane. A *colored point* is a triple (x, y, c) where x and y refer to horizontal and vertical plane coordinates, respectively, and $c \in \{green, red\}$ is the color of the point. We say that a colored point (x, y, c) *dominates* another colored point (x', y', c') if $x \geq x'$ and $y \geq y'$.

Let $G = (V, E)$ be a graph. We say that G is a *bicolored plane domination graph* if one can assign to each $v \in V$ a colored point $p_v = (x_v, y_v, c_v)$ such that $uv \in E(G)$ if and only if $c_u \neq c_v$ and the green point of $\{p_u, p_v\}$ dominates the red point of $\{p_u, p_v\}$. In words, vertices correspond to colored points, and there is an edge between two vertices if their points are green and red, and if the green is to the upper-right of the red. If this is the case, the multiset of colored points p_u is called a *bicolored plane domination model* for G (multiset because two vertices could be assigned points with the same coordinates and color). As it turns out, this coincides with trade conflict graphs. See Figure 2 for an illustration.

► **Theorem 5.** *A graph G is a trade conflict graph if and only if G is a bicolored plane domination graph.*

Moreover, given a set of trades \mathcal{T} , a bicolored plane domination model for $G(\mathcal{T})$ can be found in time $O(n)$.

Proof. Let \mathcal{T} be a set of trades and let $G := G(\mathcal{T})$ be its trade conflict graph. Let \mathcal{T}^+ be the set of trades of \mathcal{T} that are won at a positive price, and let $\mathcal{T}^- = \mathcal{T} \setminus \mathcal{T}^+$. For each $T = (w, \ell, f) \in \mathcal{T}$, assign the corresponding colored point of T to

$$(x_T, y_T, c_T) = \begin{cases} (\ell, w, green) & \text{if } T \in \mathcal{T}^+ \\ (w, \ell, red) & \text{if } T \in \mathcal{T}^- \end{cases}$$

This model is illustrated in Figure 2. Now consider two distinct trades $T_1 = (w_1, \ell_1, f_1)$ and $T_2 = (w_2, \ell_2, f_2)$ of \mathcal{T} . First assume that T_1 and T_2 are incompatible. Then by Lemma 1, $sgn(w_1) \neq sgn(w_2)$. Assume without loss of generality that T_1 is in \mathcal{T}^+ , and thus that T_2 is in \mathcal{T}^- . By incompatibility, $|\ell_1| \leq |w_2|$. Since $w_2 < 0$, it follows that $\ell_1 \geq w_2$. Similarly, $|\ell_2| \leq |w_1|$ and, since $w_1 > 0$, we have $w_1 \geq \ell_2$. Therefore, the point $(\ell_1, w_1, green)$ corresponding to T_1 dominates the point (w_2, ℓ_2, red) corresponding to T_2 , and the model correctly puts an edge between T_1 and T_2 .

Assume that T_1 and T_2 are compatible. If $sgn(w_1) = sgn(w_2)$, then the points corresponding to T_1 and T_2 are either both green and both red and the model correctly puts a non-edge between T_1 and T_2 . We may thus assume without loss of generality that $T_1 \in \mathcal{T}^+$ and

18:8 How Brokers Can Optimally Abuse Traders

$T_2 \in \mathcal{T}^-$. By compatibility, one of $|\ell_1| > |w_2|$ or $|\ell_2| > |w_1|$ holds. Suppose that $|\ell_1| > |w_2|$. Since ℓ_1 is negative, we have $\ell_1 < w_2$, which means that the green point $(\ell_1, w_1, \text{green})$ does not dominate the red point $(w_2, \ell_2, \text{red})$. So suppose that $|\ell_2| > |w_1|$. Since ℓ_2 is positive, we have $\ell_2 > w_1$, and again $(\ell_1, w_1, \text{green})$ does not dominate $(w_2, \ell_2, \text{red})$. In either case, T_1 and T_2 do not share an edge according to the model. We deduce that G is a bicolored plane domination graph using the model described above.

Note that constructing the above set of colored points does not require constructing $G(\mathcal{T})$. It only requires looping through \mathcal{T} and outputting each point in $O(1)$ time, which takes total time $O(n)$. This proves the second part of the theorem statement.

Now consider the converse direction of the equivalence. Assume that G is a bicolored plane domination graph. We show that there exists a set of trades \mathcal{T} such that G is isomorphic to $G(\mathcal{T})$. Take any model for G and, for $v \in V(G)$, denote by (x_v, y_v, c_v) the colored point assigned to v . Notice that bicolored plane domination models are robust to translation. That is, if we translate every point by the same vector, the domination relationships remain unchanged. Using this, we can assume that every point is in the upper left quadrant of the plane, i.e. that every point (x_v, y_v, c_v) satisfies $x_v < 0$ and $y_v > 0$ (which can be done by translating to the left and upwards by large enough amounts). For each $v \in V(G)$ such that $c_v = \text{green}$, add to \mathcal{T} the trade (y_v, x_v, f_v) , where f_v is arbitrary for our purposes. Then for each $w \in V(G)$ such that $c_w = \text{red}$, add to \mathcal{T} the trade (x_w, y_w, f_w) again with arbitrary f_w .

We show that dominating pairs of green-red points coincide with incompatible trades. Consider a green point (x_v, y_v, green) that dominates a red point (x_w, y_w, red) . Then the corresponding trades (y_v, x_v, f_v) and (x_w, y_w, f_w) are incompatible because $\text{sgn}(y_v) \neq \text{sgn}(x_w)$ and $y_w \leq y_v$ by domination, which implies $|y_w| \leq |y_v|$ since the y 's are positive, and because $x_w \leq x_v$ by domination, which implies $|x_w| \leq |x_v|$ since the x 's are negative. Next, consider two incompatible trades T_1, T_2 of \mathcal{T} . They must be of the form $T_1 = (y_v, x_v, f_v)$ and $T_2 = (x_w, y_w, f_w)$. By our construction, the corresponding points are (x_v, y_v, green) and (x_w, y_w, red) . By incompatibility, $|x_v| \leq |x_w|$ and $|y_w| \leq |y_v|$. Since all the y 's are positive and all the x 's are negative, this implies $x_w \leq x_v$ and $y_w \leq y_v$ and thus (x_v, y_v, green) dominates (x_w, y_w, red) . It follows that $G(\mathcal{T})$ has exactly the same edges as G . ◀

The above equivalence arguably simplifies our trading framework, but it is not immediately convenient to design algorithms using a generic bicolored plane domination model. We proceed to show that any such model can be transformed into an equivalent one on a discrete grid in which each column has exactly one point, and each row has exactly one point, without gaps. This will allow us to design a relatively simple dynamic programming algorithm.

We say that a multiset of n colored points P has the *permutation matrix property* if $\{x : (x, y, c) \in P\} = \{y : (x, y, c) \in P\} = [n]$. In other words, each point of P has a distinct x coordinate in $[n]$, and a distinct y coordinate in $[n]$. The property is named after the fact that if an $n \times n$ matrix is filled with 0s, but that we put at 1 in each (x, y) that occurs in P , then we have a permutation matrix.

► **Lemma 6.** *Let P be a multiset of n colored points representing a bicolored plane domination model of some graph G . Then in integer sorting time, one can construct from P another bicolored plane domination model P^* for G such that P^* has the permutation matrix property.*

Proof. Suppose that P does not already have the permutation matrix property. We will assume that the points of P are in the upper-right quadrant of the plane, i.e. that each (x, y, c) satisfies $x > 0$ and $y > 0$. This can be achieved through translation as we did previously.

We first argue that we can find another model P' for G in the same quadrant such that each x -coordinate is distinct and each y -coordinate is distinct. First, multiply the x coordinate of each point of P by n , and notice that we obtain a model of the same graph since this does not alter domination relationships. Then, sort the points of P in ascending lexicographic order, where *red* comes before *green*. That is, points are of the form (x, y, c) and we sort by x coordinates first, then by y , and then c to break ties (again, note that if there are points that have the same x and y coordinates, those that are *red* occur before the *green*). Traverse the points using this order and suppose that, for some x , we encounter several points $(x, y_1, c_1), \dots, (x, y_k, c_k)$ with the same x -coordinate, in this sorted order. Replace them by the points $(x, y_1, c_1), (x+1, y_2, c_2), \dots, (x+k-1, y_k, c_k)$. Because we previously multiplied all coordinates by n , these points now all have an x -coordinate not shared with any other point. Moreover, one can check that our sorting criteria ensure that domination relationships are unchanged (even for green-red pairs having the same x and y positions since the green is moved to the right of the red). After traversing all the points in this fashion, all x values are distinct. Note that none of the y values have changed in this process. We can thus repeat the same idea with the y -coordinates, resulting in another model P' for G with no repeated x nor y values. This requires sorting time, plus one pass through the points in time $O(n)$.

To get the permutation matrix property, it suffices to “squish” all the x 's and the y 's in $[n]$. That is, sort P' in ascending x -coordinates and let $(x_1, y_1, c_1), \dots, (x_n, y_n, c_n)$ be the obtained ordering (there are no ties this time). Replace the points by $(1, y_1, c_1), (2, y_2, c_2), \dots, (n, y_n, c_n)$, keeping the colors the same, and note that again, the domination relationships are unchanged. This takes integer sorting time, assuming we have to sort again. We can repeat with the y 's and obtain an equivalent model with the permutation matrix property in total integer sorting time. ◀

We now proceed to finding a maximum weight independent set. Let (G, h) be a weighted bicolored plane domination graph with a model P that has the permutation matrix property. For simplicity, we will denote the vertices of G by their points, so that $V(G) = P$. For $i, j \in [n]$, denote by

$$V(i, j) = \{(x, y, c) \in V(G) : 1 \leq x \leq i \text{ and } j \leq y \leq n\}$$

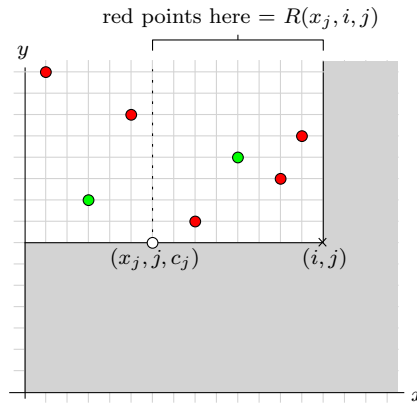
In other words, $V(i, j)$ is the set of vertices that correspond to points located in the box with corners $(1, n)$ and (i, j) . We will denote by $I(i, j)$ the maximum weight of an independent set of $G[V(i, j)]$ with respect to the h weights. The maximum weight of an independent set of G is $I(n, 1)$, the value we are ultimately looking for.

Since P has the permutation matrix property, for $j \in [n]$, we will denote by (x_j, j, c_j) the unique point of P whose y -coordinate is j . One last notation: for $i_1, i_2, j \in [n]$, denote by $R(i_1, i_2, j) = \{(x, y, c) \in P : i_1 \leq x \leq i_2 \text{ and } j \leq y \leq n \text{ and } c = \text{red}\}$. See Figure 3 for an illustration.

For i, j such that $i \notin [n]$ or $j \notin [n]$, the value of $I(i, j)$ is 0. Otherwise, we can use the following recurrence:

$$I(i, j) = \begin{cases} I(i, j+1) & \text{if } x_j > i \\ h((x_j, j, c_j)) + I(i, j+1) & \text{if } x_j \leq i \text{ and } c_j \text{ is green} \\ \max(I(i, j+1), h(R(x_j, i, j)) + I(x_j-1, j+1)) & \text{if } x_j \leq i \text{ and } c_j \text{ is red} \end{cases}$$

► **Lemma 7.** *The above recurrence for $I(i, j)$ correctly represents the maximum weight of an independent set of $G[V(i, j)]$.*



■ **Figure 3** Illustration of the main components of the $I(i, j)$ recurrence. Points in the grayed area are ignored and only the points of $V(i, j)$ are shown. Here, (x_j, i, c_j) is the unique point with y -coordinate j , which could be red or green. The $R(x_j, i, j)$ set consists of the three red points in the area shown.

Proof. This can be shown inductively on j in reverse order. As a base case, notice that for $j > n$ and any i , then $I(i, j) = 0$ is correct since $V(i, j)$ is empty. Assume that $i, j \in [n]$ and that for any $j' > j$ and any i' , the recurrence for $I(i', j')$ is correct. Let W be an independent set of $G[V(i, j)]$ of maximum weight. Consider the point (x_j, j, c_j) of P . If $x_j > i$, then point (x_j, j, c_j) is not in $V(i, j)$ and W does not contain it. Thus all points of W are in $V(i, j + 1)$ and $I(i, j) = I(i, j + 1) = h(W)$ follows by induction. We may assume that $x_j \leq i$. Suppose that (x_j, j, c_j) is green. By the permutation matrix property, (x_j, j, c_j) does not dominate any point in $V(i, j)$. Hence, (x_j, j, c_j) is an isolated vertex of $G[V(i, j)]$ and we may assume that $(x_j, j, c_j) \in W$ since all weights are positive. All other points of W must be in $V(i, j + 1)$ and by induction $h(W \setminus \{(x_j, j, c_j)\}) = I(i, j + 1)$. It follows that $I(i, j) = h((x_j, j, c_j)) + I(i, j + 1) = h(W)$ is correct.

Finally, suppose that (x_j, j, c_j) is red. We first show that $I(i, j) \geq h(W)$ and deal with the converse bound after. There are two cases: either $(x_j, j, c_j) \in W$, or not. If not, then all points of W are in $V(i, j + 1)$ and $h(W) = I(i, j + 1)$. But if $(x_j, j, c_j) \in W$, then no green point (x_k, k, c_k) of $V(i, j)$ such that $x_k > x_j$ can be in W , since any such point dominates (x_j, j, c_j) . Therefore, we may assume that every red point (x_k, k, c_k) of $V(i, j)$ with $x_k > x_j$ is in W . These points are $R(x_j, i, j)$, and so W consists of $R(x_j, i, j)$ plus a maximum weight independent set of $V(x_j - 1, j + 1)$ of weight $I(x_j - 1, j + 1)$. In either case, since $I(i, j)$ takes the maximum of the two possibilities, we get that $I(i, j) \geq h(W)$. For the converse bound, it is not hard to see that there exists an independent set of weight $I(i, j + 1)$ in $V(i, j)$, and an independent set of weight $h(R(x_j, i, j)) + I(x_j - 1, j + 1)$ in $V(i, j)$ (the latter obtained by taking all red points in $R(x_j, i, j)$ and an optimal independent set in $V(x_j - 1, j + 1)$, noticing that none of the points from that set can dominate $R(x_j, i, j)$). The weight of W is at least the maximum of those, and so $I(i, j) \leq h(W)$. This concludes the proof. ◀

Algorithm 1 summarizes the above ideas and computes a maximum weight independent set in a bicolored plane domination graph in $O(n^2)$ time, as we argue below.

The algorithm simply computes $I(i, j)$ for every $j \in [n]$ in reverse order and for every $i \in [n]$ in order. There are $O(n^2)$ values of $I(i, j)$ to compute and each can take time $O(1)$, assuming that each $h(R(x_j, i, j))$ can be accessed in constant time. This can be achieved by storing all the relevant R values during a preprocessing step. Notice that the only relevant

■ **Algorithm 1** How to abuse traders: a geometric interpretation.

```

function abuseTraders( $\mathcal{T}$ )
  Compute the bicolored plane domination model  $P$  for  $G(\mathcal{T})$  (Theorem 5)
  Compute a model  $P^*$  with the permutation matrix property (Lemma 6)
  computeRweights( $P^*$ )
  for  $j = n$  down to 1 do
    for  $i = 1$  to  $n$  do
      Compute  $I(i, j)$  using the recurrence
    end
  end
  return  $I(1, n)$ 

function computeRweights( $P^*$ )
  for  $j = 1$  to  $n$  do
    Let  $(x_j, j, c_j)$  be the unique point of  $P^*$  with  $y$ -coordinate  $j$ 
    weight = 0
    for  $i = x_j$  to  $n$  do
      Let  $(i, y_i, c_i)$  be the unique point of  $P^*$  with  $x$ -coordinate  $i$ 
      if  $c_i = \text{red}$  and  $y_i > j$  then
        weight = weight +  $h((i, y_i, c_i))$ 
         $h(R(x_j, i, j)) = \text{weight}$ 
      end
    end
  end

```

values to store for the recurrence have the form $R(x_j, i, j)$, where x_j is determined by j . Hence, for each x_j and j pair, we can compute $h(R(x_j, i, j))$ for each i in increasing order of i , adding for each i the weight of the new point with x -coordinate i if relevant. For each j , we need linear time to compute the necessary $h(R(x_j, i, j))$ values, so this preprocessing is done in time $O(n^2)$. All the above ideas are shown in Algorithm 1. We note that this algorithm only returns the maximum weight, but a standard backtracking procedure can find an actual set of optimal winning trades, which in turn can be converted to a price movement. We have argued the following.

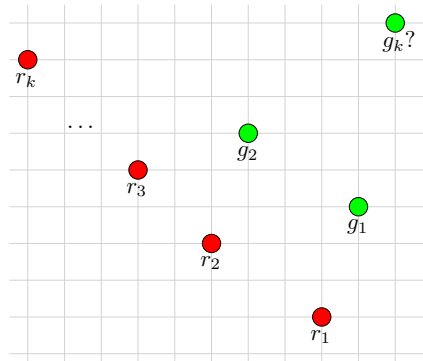
► **Theorem 8.** *A maximum weight independent set of a bicolored plane domination graph can be found in time $O(n^2)$.*

3.2 Some open questions on trade conflict graphs

If we abstract away the financial motivations of this work for a moment, trade conflict graphs (i.e. bicolored plane domination graphs) can be studied from a purely graph theoretical perspective. One could ask whether such graphs are easy to recognize, and whether trade conflict graphs coincide, or at least share some similarity with a graph class that is already known. Although we reserve these questions for future work, we can provide a partial answer to the latter. A graph is *chordal bipartite* if it is bipartite and contains no induced cycle with at least 6 vertices.

► **Proposition 9.** *All bicolored plane domination graphs are chordal bipartite.*

Proof. We already know that G is bipartite, so we only need to argue that it has no induced cycle of length 6 or more. Let G be a bicolored plane domination graph and let P be a set of colored points that is a model for G . Assume that G contains an induced cycle



■ **Figure 4** An illustration of how some of the relative locations of the r_i and g_i points are forced.

$r_1 g_1 r_2 g_2 \dots r_k g_k r_1$ with $k \geq 3$. Suppose without loss of generality that the r_i 's correspond to red points and the g_i 's to green points in P (otherwise, start the cycle at g_1 and rename accordingly). We will denote by $r_i.x$ and $r_i.y$ the coordinates of the point corresponding to r_i in P , and use the same notation $g_i.x, g_i.y$ for g_i , where $i \in [k]$.

Observe that for any $i \in [k]$, it is not possible that $r_i.x \geq r_{i+1}.x$ and $r_i.y \geq r_{i+1}.y$ both hold (where r_{k+1} is taken to be r_1). Indeed, this would imply that any green point that dominates r_i would also dominate r_{i+1} , and we know that g_{i-1} dominates r_i but not r_{i+1} (note that this is where we need the cycle to be of length at least 6). By a symmetric argument, we cannot have $r_{i+1}.x \geq r_i.x$ and $r_{i+1}.y \geq r_i.y$ simultaneously.

The rest of the following argument is illustrated in Figure 4. Suppose without loss of generality that $r_2.x \leq r_1.x$ (otherwise, start the cycle at r_2 and list the vertices in reverse order by renaming accordingly). We then deduce from the above observation that $r_2.y > r_1.y$. Moreover, g_2 dominates r_2 but not r_1 , which means that $g_2.y \geq r_2.y > r_1.y$ but $g_2.x < r_1.x \leq g_1.x$. Now consider the location of r_3 , which is dominated by g_2 . We must have $r_3.x \leq g_2.x \leq g_1.x$ and, since g_1 does not dominate r_3 , we must have $r_3.y > g_1.y \geq r_2.y$. Again using our previous observation, we get that $r_3.x \leq r_2.x$.

In other words we have deduced from $r_2.x \leq r_1.x$ and $r_2.y > r_1.y$ that $r_3.x \leq r_2.x$ and $r_3.y > r_2.y$. This argument can be used inductively to infer that $r_k.x \leq r_{k-1}.x \leq \dots \leq r_1.x$ and $r_k.y > r_{k-1}.y > \dots > r_1.y$. Now consider the point g_k , which dominates both r_k and r_1 . One can see that such a point must dominate each point in r_1, r_2, \dots, r_k , a contradiction. Thus the cycle cannot exist. ◀

Let us mention that chordal bipartite graphs are known to admit $O(|V| + |E|)$ time algorithms for the maximum *unweighted* independent set problem, assuming that an ordering of the vertices called a *strong ordering* is given [6]. However, it is not clear whether the same time bound can be achieved for the weighted version. In any case, the geometric perspective may lead to even better algorithms in the future. We close this section with some question that this leads us to.

1. Is the class of bicolored plane domination graphs equal to the class of chordal bipartite graphs?
2. If not, can bicolored plane domination graphs be characterized as bipartite graphs that forbid induced cycles of length at least 6, plus a finite set of forbidden induced subgraphs?
3. Given a graph G , can one decide in polynomial time whether G is a bicolored plane domination graph?

4. Given a bicolored plane domination model P that has the permutation matrix property, can a maximum weight independent set be found in time $O(n)$ (regardless of the number of edges of the underlying graph)?

A positive answer to (1) would be a purely accidental find, and would be surprising since bicolored plane domination graphs appear to have a simple structure. However, it seems hard to construct a chordal bipartite graphs that can be shown to not be a trade conflict graph. This would probably give clues on (2) though. As for (3), trade conflict graphs are probably polynomial-time recognizable, given their simplicity, but this requires a more careful look. As for (4), it is plausible that $o(n^2)$ could be achieved, since our recurrence needs to go through lots of (i, j) locations that contain no point, and only n such locations appear truly relevant.

4 Online trades

We now allow the trader to interrupt the broker's price manipulation at any time by adding a new trade or closing an open trade. One may ask whether, given this new power, the trader can make money. It does appear that a small proportion of real-life traders makes profit, so perhaps the broker can be beaten. On the other hand, if there was a strategy for the trader that guarantees profit even against conspiratorial prices, someone would have figured it out by now. But surprisingly, there doesn't seem to be a clear answer in the literature. One impossibility result is given by the *efficient market hypothesis* (EMH) [7], which, roughly speaking, states that if prices perfectly reflect their environment, then no strategy can win *consistently*, i.e. against every price movement (note that the author is far from being an economist and that the actual theory is much deeper). In the same vein, assuming the price is a random walk, the best trader strategy has expected profit 0, so there exists a price movement with no win for the trader (unless the trader has infinite funds, in which case a Martingale betting system can be used for profit). Thus the broker can "mimic" a worst-case EMH-driven price or a random walk to guarantee that, on expectation, traders make no money. The broker can still lose money if unlucky, raising the question of whether there is a strategy for the broker that guarantees profitability. To our knowledge, none of aforementioned frameworks is applicable to our model.

4.1 The online model

In the online setting, a trade can be opened at any current price, unlike the previous section where the trade opening price was always assumed to be 0. Moreover, a trader could decide to close a trade at any moment, not only when w or ℓ is reached. To model this, an *online trade* T will be seen as a quadruple (w, ℓ, p, f) where w and ℓ are the winning and losing prices, respectively, $p \in \mathbb{Z}$ is the opening price and $f : [\min(w, \ell), \max(w, \ell)] \rightarrow \mathbb{R}$ gives the profit the broker makes at any possible closing price. We require that $f(p) = 0$, and that one of the following holds:

- $\ell < p < w$, in which case $f(q) > 0$ for each $q > p$ and $f(q) < 0$ for each $q < p$ (*up*); or
- $w < p < \ell$, in which case $f(q) < 0$ for each $q > p$ and $f(q) > 0$ for each $q < p$ (*down*).

Note that in a real-life setting, the trader does not have to communicate the desired limit prices w and ℓ . However in our model, this would allow the trader to leave losing trades open forever. Moreover, a limit on losses always exists, since brokers will never allow the trader's asset values to go below the account equity. In other words, the broker can always set a worst-case w and ℓ value if not given by the trader.

18:14 How Brokers Can Optimally Abuse Traders

This online model can then be seen as a two-player game as follows. The game alternates turns between the trader and the broker, with the trader starting. At any turn i , there is a current price p_i and a set of open online trades \mathcal{T}_i . On the first turn $i = 1$, the price is $p_1 = 0$ and $\mathcal{T}_1 = \emptyset$.

On the i -th turn, the trader acts first and can apply the following actions any number of times:

- add an online trade $T = (w, \ell, p_i, f)$ to \mathcal{T}_i ;
- close any open trade $T = (w, \ell, p, f)$ in \mathcal{T}_i . This has the effect of removing T from \mathcal{T}_i and adding an amount of $f(p_i)$ to the broker's profit.

We say that the trader is *passive* if the only action used by the trader is to add trades (never closing them unless w or ℓ is reached). When the trader is done, the broker looks at \mathcal{T}_i and p_i , and acts by choosing one of the two following actions:

- move the price up by one unit, so that the price for the next turn becomes $p_{i+1} = p_i + 1$;
- move the price down by one unit, so that the price for the next turn becomes $p_{i+1} = p_i - 1$.

Finally, we impose three restrictions on the broker:

- R1.** if $\mathcal{T}_i \neq \emptyset$, the broker must eventually close a trade if the trader does nothing;
- R2.** if $\mathcal{T}_i = \emptyset$ on the broker's turn, the broker moves towards price 0;
- R3.** the broker's decision is based solely on the current open trades \mathcal{T}_i and the price p_i , and not on the past closed trades.

These can be justified as follows. R1 ensures constant activity, as it prevents the broker from zig-zagging in the same price area indefinitely. R2 states that when no trade is active, the game can be reset and the current price p_i might as well be interpreted as 0. R3 is akin to treating the broker as a Markov chain, as it makes sure that it always acts in the same manner when facing the same trade state. Another way to justify this restriction is that the broker may assume optimal play from the trader, in which case the same move should always be preferred in a given configuration. In real-life, traders are imperfect and brokers might profit from analyzing their past behavior, but this is beyond the scope of this work.

After the broker's turn, any trade $T = (w, \ell, p, f)$ in \mathcal{T}_i such that $p_{i+1} \in \{w, \ell\}$ becomes closed. In this case, T is removed from \mathcal{T}_i and a profit of $f(p_{i+1})$ is added to the broker's profit. The updated \mathcal{T}_i becomes \mathcal{T}_{i+1} . We also restrict \mathcal{T}_i to be finite at any turn, meaning the trader must satisfy the requirement that there exists $t \in \mathbb{N}$ such that, regardless of the broker's strategy, for each turn i we have $|\mathcal{T}_i| \leq t$.

It is not hard to show that if we impose no restriction on f , then the trader wins effortlessly. For instance on turn 1 at price 0, the trader can simply open two trades $T_1 = (1, -1, 0, f_1)$ and $T_2 = (-1, 1, 0, f_2)$ such that $f_1(1) = f_2(-1) = -2$ and $f_1(-1) = f_2(1) = 1$. Whichever direction the broker chooses, a profit of -1 will be incurred and both trades will be closed. By R2, the trader can then wait until price 0 is reached. The game resets and by R3, the trader can repeat this forever, forcing a profit of $-\infty$.

For this reason, we will require trades to be linear, as defined below.

► **Definition 10.** An online trade $T = (w, \ell, p, f)$ is linear if there exists a positive rational number $\delta \in \mathbb{Q}_{>0}$ such that:

- if $\ell < p < w$, then $f(q) = \delta \cdot (q - p)$ for all $\ell \leq q \leq w$;
- if $w < p < \ell$, then $f(q) = \delta \cdot (p - q)$ for all $w \leq q \leq \ell$.

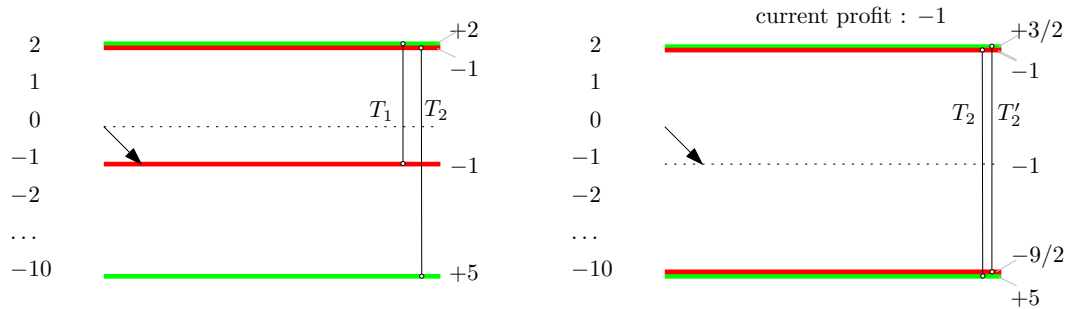
Note that real-life trading is done on linear trades and δ is sometimes called the *lot size*.

4.2 The only good online broker strategy

Perhaps the first strategy that comes to mind is to use the results from the previous section in the online setting. That is, on the broker's i -th turn, we look at \mathcal{T}_i , translate the trades so that the current price can be interpreted as 0, and compute an optimal price trajectory M using a maximum weight independent set on $G(\mathcal{T}_i)$. We then go in the same direction as M for one price unit, and this M will be recalculated on the $(i + 1)$ -th turn. Let us call this the *offline strategy*. Unfortunately, this is not a good strategy.

► **Proposition 11.** *If the broker uses the offline strategy, then the trader can force the broker to have an infinite negative profit, even if the trader is passive and restricted to linear trades.*

Proof. The trader's strategy is illustrated in Figure 5.



■ **Figure 5** The offline strategy is not good.

On turn 1 at price $p_1 = 0$, the trader opens a linear trade $T_1 = (2, -1, 0, f)$ with $f(-1) = -1, f(1) = 1, f(2) = 2$, and another trade $T_2 = (-10, 2, 0, g)$ such that $g(-10) = 5$ and $g(2) = -1$ (and the other values made linear with $\delta = 1/2$). These trades are incompatible, so the broker using the offline strategy will choose to lose T_1 and win T_2 , and will thus go down on its first turn. The price will become -1 and the T_1 trade will be lost, incurring a (negative) profit of -1 for the broker.

On turn 2 at price $p_2 = -1$, the passive trader opens the trade $T'_2 = (2, -10, -1, g')$ with $g'(-10) = -9/2$ and $g'(2) = 3/2$ (linear with $\delta = 1/2$). At this point the trader only waits until T_2 and T'_2 are closed, which must happen by R1. If price 2 is reached first, both trades close and the broker's profit for them is $-1 + 3/2 = 1/2$. If price -10 is reached first instead, then the profit is $5 - 9/2 = 1/2$. In either case, the broker's total profit is $-1 + 1/2 = -1/2$. At this point, all trades are closed. By R2 and R3, the trader can repeat this pattern indefinitely. ◀

The above is actually part of a more general result that says that any strategy that does not go for the maximum *potential* from the start loses infinitely. The interest of Proposition 11 is that it serves as a concrete example of losing to a concrete strategy. We next show that in fact, maximizing potential profits is essentially the only good strategy. Anything that diverges from it is either suboptimal or has infinite negative profit.

For a set of online trades \mathcal{T} and a price p , the *potential profit* of p on \mathcal{T} is defined as

$$potent(\mathcal{T}, p) = \sum_{(w, \ell, q, f) \in \mathcal{T}} f(p)$$

which represent the profit that would be made if every trade closed at price p .

18:16 How Brokers Can Optimally Abuse Traders

The *maximum potential* strategy is defined as follows. On the broker's i -th turn,

- if $\text{potent}(\mathcal{T}_i, p_i + 1) \geq \text{potent}(\mathcal{T}_i, p_i - 1)$, then move the price *up*;
- otherwise if $\text{potent}(\mathcal{T}_i, p_i + 1) < \text{potent}(\mathcal{T}_i, p_i - 1)$, then move the price *down*.

Intuitively speaking, this strategy is based on the fact that the trader either has more buys than sells or vice-versa. Under linear trades, one of the direction has to be worse than the other (unless they both achieve profit 0) and we simply go in the direction that is worse for the trader. Do note that this strategy can change direction even if the trader does nothing, since winning a trade can change the direction of maximum potential.

We now analyze this strategy against others, but from a general perspective where the broker starts applying it after a certain number of turns i .

► **Theorem 12.** *Suppose that the trader is restricted to linear online trades. Let \mathcal{T}_i be a set of open trades at the start of the i -th turn, let p_i be the current price, and let $\text{profit}(i)$ be the total profit of the broker at the start of turn i . Then the following holds:*

1. *if $\text{profit}(i) + \text{potent}(\mathcal{T}_i, p_i) \geq 0$, then if the broker applies the maximum potential strategy from turn i and onwards, it achieves a total profit of at least $\text{profit}(i) + \text{potent}(\mathcal{T}_i, p_i)$ against any trader. Moreover, this is the maximum possible profit achieved against a trader with optimal play, passive or not;*
2. *if the broker does not move the price in the direction of maximum potential profit on turn i , then the broker makes a profit that is strictly less than $\text{profit}(i) + \text{potent}(\mathcal{T}_i, p_i)$ against a trader with optimal play, passive or not;*
3. *if $\text{profit}(i) + \text{potent}(\mathcal{T}_i, p_i) < 0$, then the broker incurs an infinite negative profit against a trader with optimal play, passive or not.*

Proof. Let us first consider the first part of (1). Let \mathcal{T}'_i be the set of trades after the trader has finished the i -th turn, and let $\text{profit}'(i)$ be the profit of the broker at this point. Note that if the trader is passive, then $\mathcal{T}_i \subseteq \mathcal{T}'_i$, but if it is not passive, a certain set of trades might be closed. In this case, for each $(w, \ell, q, f) \in \mathcal{T}_i \setminus \mathcal{T}'_i$, a profit of $f(p_i)$ is added for the broker and an amount of $f(p_i)$ is removed from the potential of p_i . Also, recall that for each new trade $(w, \ell, q, f) \in \mathcal{T}'_i \setminus \mathcal{T}_i$, we have $f(p_i) = 0$. It follows that

$$\text{profit}(i) + \text{potent}(\mathcal{T}_i, p_i) = \text{profit}'(i) + \text{potent}(\mathcal{T}'_i, p_i)$$

In other words, the trader's turn does not affect the potential profit of the broker. We next argue that one of the price directions does not decrease this potential. Write $\mathcal{T}'_i = \{T_1, \dots, T_m\}$ and for $j \in [m]$, let δ_j be the linear factor affecting trade T_j . We say that $\text{sgn}(T_j) = 1$ if T_j is an *up* trade and $\text{sgn}(T_j) = -1$ if T_j is a *down* trade. Then by linearity, raising the price by one unit changes the potential of T_j by $\text{sgn}(T_j) \cdot \delta_j$ (*up* trades increase in potential, and *down* trades decrease), and lowering the price by one unit changes it by $-\text{sgn}(T_j) \cdot \delta_j$. In other words,

$$\text{potent}(\mathcal{T}'_i, p_i + 1) - \text{potent}(\mathcal{T}'_i, p_i) = \sum_{(w, \ell, p_j, f_j) \in \mathcal{T}'_i} (f_j(p_i + 1) - f_j(p_i)) = \sum_{T_j \in \mathcal{T}'_i} \text{sgn}(T_j) \cdot \delta_j$$

and likewise

$$\text{potent}(\mathcal{T}'_i, p_i - 1) - \text{potent}(\mathcal{T}'_i, p_i) = \sum_{T_j \in \mathcal{T}'_i} -\text{sgn}(T_j) \cdot \delta_j$$

One of $\sum_{T_j \in \mathcal{T}'_i} \text{sgn}(T_j) \cdot \delta_j$ or $\sum_{T_j \in \mathcal{T}'_i} -\text{sgn}(T_j) \cdot \delta_j$ is greater than or equal to 0, and so one direction leads to a potential greater than or equal to $\text{potent}(\mathcal{T}'_i, p_i)$. Suppose that this direction is up. Let $\text{profit}(i+1)$ be the broker's profit after changing the price to $p_{i+1} = p_i + 1$ and let \mathcal{T}_{i+1} be the set of trades still open at this point. Some trades (w, ℓ, q, f) might become closed at price $p_i + 1$ and incur a profit of $f(p_i + 1)$, but remove an amount of $f(p_i + 1)$ from $\text{potent}(\mathcal{T}'_i, p_i + 1)$. That is, similarly as we did above, we obtain that

$$\text{profit}(i+1) + \text{potent}(\mathcal{T}_{i+1}, p_i + 1) = \text{profit}'(i) + \text{potent}(\mathcal{T}'_i, p_i + 1)$$

which is greater than or equal to $\text{profit}(i) + \text{potent}(\mathcal{T}_i, p_i)$. The idea is the same if going down is better for the potential instead. By repeating this argument, this shows that the profit plus potential can be made to never decrease, guaranteeing the broker at least a profit of $\text{profit}(i) + \text{potent}(\mathcal{T}_i, p_i)$ when every trade is closed and the potential has reached 0. Also note that it is not hard to see that unless the trader opens new trades, the broker will keep the same direction until at least one trade is closed, as required by our model. This shows the first part of (1).

We now argue that this is what the optimal trader can achieve. If the trader is not passive, then the trader can close every trade at price p_i at the start of the i -th turn and stop playing. The broker's profit will be exactly $\text{profit}(i) + \text{potent}(\mathcal{T}_i, p_i)$, which is the best the trader can hope for since this is a lower bound. This proves (1) in the case of a non-passive trader.

We next prove a claim that will be useful for the remaining statements of the theorem that concern passive traders. Let \mathcal{T}_j be any set of open trades at some current price p_j , and assume that it is the trader's turn. We claim that the passive trader can add a set of trades to \mathcal{T}_j such that the broker's profit on \mathcal{T}_j is forced to be exactly $\text{potent}(\mathcal{T}_j, p_j)$. This effectively simulates the action of closing all trades at current price. Let $\mathcal{T}_j^+ = \{(w, \ell, p, f) \in \mathcal{T} : w > p_j\}$ and let $\mathcal{T}_j^- = \mathcal{T}_j \setminus \mathcal{T}_j^+$. Consider $T = (w, \ell, p, f) \in \mathcal{T}_j^+$ and let δ be the linear factor affecting T . Add the corresponding linear trade $T' = (\ell, w, p_j, f')$ such that $f'(\ell) = \delta(p_j - \ell)$ and $f'(w) = \delta(p_j - w)$. Suppose that price $q \in \{w, \ell\}$ is reached first. Since the trades are linear with the same δ the profit on T and T' will be $f(q) + f'(q) = \delta(q - p) + \delta(p_j - q) = \delta(p_j - p)$. Again by linearity, this is exactly $f(p_j)$. By symmetry, one can add a similar trade T' for each $(w, \ell, p, f) \in \mathcal{T}_j^-$ to force a profit of $f(p_j)$ for T and T' . Because of restriction R1, the trader can wait until the broker has closed every trade. It follows that the trader can force the current trades to close for a total profit of $\sum_{(w, \ell, p, f) \in \mathcal{T}_j} f(p_j) = \text{potent}(\mathcal{T}_j, p_j)$.

This claim shows that (1) holds also for passive traders, since the trader can force a final profit of $\text{profit}(i) + \text{potent}(\mathcal{T}_i, p_i)$ at the start of the i -th turn.

Statement (2) is now easy to prove. If the broker moves against the optimal potential, then we will get $\text{profit}(i+1) + \text{potent}(\mathcal{T}_{i+1}, p_{i+1}) < \text{profit}(i) + \text{potent}(\mathcal{T}_i, p_i)$. As we have seen, the trader (passive or not) can force a final profit of $\text{profit}(i+1) + \text{potent}(\mathcal{T}_{i+1}, p_{i+1})$, which is strictly less than what the maximum potential strategy gives.

As for statement (3), suppose that the trader has done a sequence of $i-1$ actions such that $\text{profit}(i) + \text{potent}(\mathcal{T}_i, p_i) < 0$. Again, the trader, passive or not, can force this negative profit on the broker. After the latter has closed every trade, the trader can simply repeat the same sequence of actions to make the broker lose the same amount again indefinitely (which works because of restrictions R2 and R3, which imply that the game always resets after this negative profit, and that the broker repeats the same mistake every time). ◀

As a corollary, we see that if the trader makes a move that makes the profit plus potential positive, then the broker will make profit. Conversely, if the broker makes a mistake to make this value negative, then the trader will make it lose infinitely. Therefore, the only source of

positive profit is due to a trader mistake, and the only source of negative profit is due to a broker mistake. If both players play optimally, no mistake is ever made and a profit of 0 is made. However, in real life trading, each trade opened by the trader has a fee that goes into the broker's pockets. We conclude that the best option for the trader is to not trade and try to get rich by other means.

5 Conclusion

In this work, we have provided some useful algorithmic tools for malevolent brokers to plot against traders. Our models are combinatorial and do not consider any form of stochasticity. This may serve as criticism towards our model, but one must reckon that stochastic trading models have been studied for a very long time. Our work offers a new perspective on trade analysis and shows how price movements can be optimized when given full control over them. Let us also mention that our graph theoretical view on trading is novel and allowed us to discover a potentially new graph class (although they might just be chordal bipartite graphs – the future will tell).

One future direction would be to reach a middle ground between combinatorial and stochastic by incorporating various forms of randomness into our price manipulation possibilities. For instance, we may be able to move prices in the desired direction only with certain probabilities. Also, in the online model, we often assumed a trader with optimal play, whereas a randomized trader could be considered more realistic. Finally, one can ask whether a trader knowledgeable of the broker's scheme is able to profit from that information. In the offline setting, if the trader was able to see every trade, he or she could predict the broker's movements and add appropriate winning trades. This would need to be done so that the broker would not alter its trajectory because of the new trades, leading to another algorithmic problem of interest.

References

- 1 How crypto investors can avoid the scam that captured \$2.8 billion in 2021. time.com: <https://time.com/nextadvisor/investing/cryptocurrency/protect-yourself-from-crypto-pump-and-dump/>. Accessed: February 17, 2022.
- 2 Market makers vs. electronic communications networks. investopedia.com: <https://www.investopedia.com/articles/forex/06/ecnmarketmaker.asp>. Accessed: February 17, 2022.
- 3 Robinhood restricts trading in gamestop, other names involved in frenzy. cnbc.com: <https://www.cnbc.com/2021/01/28/robinhood-interactive-brokers-restrict-trading-in-gamestop-s.html>. Accessed: February 17, 2022.
- 4 Why do many forex traders lose money? here is the number 1 mistake. dailyfx.com: https://www.dailyfx.com/forex/fundamental/article/special_report/2015/06/25/what-is-the-number-one-mistake-forex-traders-make.html. Accessed: February 17, 2022.
- 5 Andreas Brandstädt, Van Bang Le, and Jeremy P Spinrad. *Graph classes: a survey*. SIAM, 1999.
- 6 Feodor F Dragan. Strongly orderable graphs a common generalization of strongly chordal and chordal bipartite graphs. *Discrete applied mathematics*, 99(1-3):427–442, 2000.
- 7 Eugene F Fama. Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, 25(2):383–417, 1970.

- 8 Gianni Franceschini, Shan Muthukrishnan, and Mihai Pătraşcu. Radix sorting with no extra space. In *European Symposium on Algorithms*, pages 194–205. Springer, 2007.
- 9 Yijie Han. Deterministic sorting in $o(n \log \log n)$ time and linear space. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 602–608, 2002.
- 10 Nicklas Larsen, Helmut Mausser, and Stanislav Uryasev. Algorithms for optimization of value-at-risk. In *Financial engineering, E-commerce and supply chain*, pages 19–46. Springer, 2002.
- 11 James B Orlin. Max flows in $O(nm)$ time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 765–774, 2013.
- 12 Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.
- 13 Gerold Studer. *Maximum loss for measurement of market risk*. PhD thesis, ETH Zurich, 1997.
- 14 Mu Yang. *Fx spot trading and risk management from a market maker's perspective*. Master's thesis, University of Waterloo, 2011.

Wordle Is NP-Hard

Daniel Lokshtanov ✉

University of California Santa Barbara, CA, USA

Bernardo Subercaseaux ✉ 

Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Wordle is a single-player word-guessing game where the goal is to discover a secret word w that has been chosen from a dictionary D . In order to discover w , the player can make at most ℓ guesses, which must also be words from D , all words in D having the same length k . After each guess, the player is notified of the positions in which their guess matches the secret word, as well as letters in the guess that appear in the secret word in a different position. We study the game of Wordle from a complexity perspective, proving NP-hardness of its natural formalization: to decide given a dictionary D and an integer ℓ if the player can guarantee to discover the secret word within ℓ guesses. Moreover, we prove that hardness holds even over instances where words have length $k = 5$, and that even in this case it is NP-hard to approximate the minimum number of guesses required to guarantee discovering the secret word. We also present results regarding its parameterized complexity and offer some related open problems.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases wordle, np-hardness, complexity

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.19

Related Version *Full Version*: <https://arxiv.org/abs/2203.16713>

Funding *Daniel Lokshtanov*: supported by BSF award 2018302, and NSF award CCF-2008838.

Bernardo Subercaseaux: supported by NSF awards CCF-2015445, CCF-1955785, and CCF-2006953.

Acknowledgements We thank anonymous reviewers for helpful suggestions, Mark Polyakov for pointing out a minor bug in a previous proof of Theorem 1, and Wassim Bouaziz for pointing out some imprecisions in a previous version of this paper. The second author thanks Jérémy Barbay for his insightful conversation, and his awesome friends: Saranya Vijayakumar for proofreading the English in an earlier draft of this paper, and Bailey Miller for providing him a place (and tea) to work on this article.

1 I N T R O

Wordle (<https://www.nytimes.com/games/wordle/index.html>) is a single-player web-based word-guessing game that combines principles of classic games such as Mastermind, Hangman, and Jotto, which have been studied from a computational perspective before [1, 6, 7, 12, 14]. Created by Josh Wardle for his partner [13], and published in October 2021, Wordle has reached an unprecedented level of virality and gained millions of daily players [4, 11]. As for the popularity of the game, Wardle suggested that part of the game’s success is due to its artificial scarcity, as it can only be played once a day [9]. Others have pointed at the distinctive colorful patterns with which players share their results [10]. This article can be seen as yet another reason for Wordle’s success: its intrinsic complexity. The relationship between in games and their computational complexity has been proposed by multiple authors [5, 8], and thus by establishing theoretical results of hardness for Wordle we can explain part of its challenging nature, and make the reader feel better about their unsuccessful guesses.

The game is played over a dictionary of words D , all of which have length k , that is fully known to the player. Formally, for a given alphabet Σ , $D \subseteq \Sigma^k$. The player, who we will denote the *guesser*, wishes to discover a secret word $w \in D$ through a series of at



© Daniel Lokshtanov and Bernardo Subercaseaux;
licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

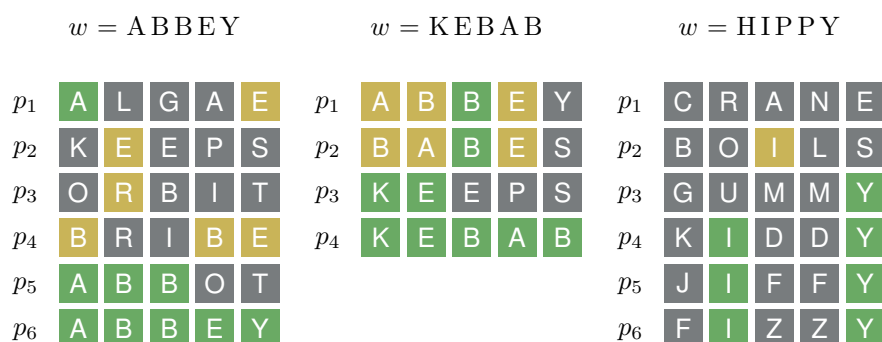
Editors: Pierre Fraigniaud and Yushi Uno; Article No. 19; pp. 19:1–19:8

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

19:2 Wordle Is NP-Hard



■ **Figure 1** Illustration of three different instances of Wordle. In all of them $k = 5$ and $\ell = 6$. The first two games are won by the guesser, while the third one is lost.

most ℓ guesses p_1, p_2, \dots, p_ℓ , all of which must be words belonging to D . The guesser is said to win if one of its guesses p_i is exactly equal to w , and she loses if no such match occurs after ℓ guesses. Whenever a guess p is made, the guesser receives some information about the relation between p and the secret word w : she is notified of every position i such that $p[i] = w[i]$, and also of positions i such that the letter $p[i]$ is present in the word w but not in position i . More precisely, to handle the case of repeated letters, we will use a notion of *marking* with colors. For an index $1 \leq i \leq k$, $w[i]$ marks $p[i]$ with green iff $w[i] = p[i]$. If $w[i] \neq p[i]$, and the set

$$S_i = \{j : p[j] = w[i], p[j] \text{ is unmarked}\}$$

is not empty, then $w[i]$ marks $p[\min(S_i)]$ with yellow. All letters $p[j]$ that were not marked at this point are marked with gray. A few games of Wordle are illustrated in Figure 1.

Despite its apparent simplicity, Wordle allows for encoding hard problems, as we present in the next section.

2 Complexity Results

In order to study the complexity of Wordle, we need to define an appropriate formal decision problem for it. Let us assume that the guesser wants to design a strategy that ensures winning the game (i.e., guessing correctly within ℓ attempts) regardless of what the secret word was chosen to be. Thus, we can pose the following problem:

PROBLEM : **Wordle**
 INPUT : (D, ℓ) with $D \subseteq \Sigma^k$ the dictionary,
 and an integer $\ell \geq 1$, the number of guesses allowed.
 OUTPUT : **Yes**, if there is a winning strategy for the guesser,
 and **No** otherwise.

Note that Σ and k are not explicitly part of the input, but can trivially be deduced from D . We claim now that if a guesser knew a polynomial time algorithm A for this problem, then they could use A to win the game whenever it was possible, as we explain next. First, we will define the notion of feasible words. Observe that at any point in the game, the information the guesser has received thus far (i.e., the letters that have been marked or not in all the previous guesses) defines a dictionary $D' \subseteq D$ of words that could happen to be

the secret word, which we call *feasible* words. For example, after the first guess illustrated in the left-most game of Figure 1, the word **GAMES** would be infeasible with the information received for the first guess, as the letter **G** was not marked. Similarly, the word **KEEPS** used in the second guess of that same game is also infeasible with the result of the first guess, as it does not contain the letter **A** that was marked in the first position. The word **AMAZE** would not be feasible either, as in the first guess the letter **E** at the end is marked with yellow, and thus cannot appear in that position in the secret word. In contrast, the word **ANNEX** would be feasible, and naturally **ABBEY**, the actual secret word of that game, is feasible too. Formally, a word $p \in D$ is feasible after a sequence of guesses p_1, \dots, p_n if and only if the following conditions hold:

1. No letter $p[i]$ was marked gray in a previous guess p_j , for $1 \leq i \leq k, 1 \leq j \leq n$.
2. No letter $p[i]$ was marked yellow in a previous guess p_j , such that $p_j[i] = p[i]$, for $1 \leq i \leq k, 1 \leq j \leq n$.

■ **Algorithm 1** WordleGuesser(D, ℓ).

```

1.1 if  $\ell = 0$  then
1.2   | gesser loses.
1.3 if  $|D| = 1$  then
1.4   | guess the only word in  $D$  and win.
1.5 for  $p \in D$  do
1.6   | potentialGuess  $\leftarrow$  true
1.7   | for  $w \in D$  do
1.8     |  $m \leftarrow$  marking for guess  $p$  if  $w$  is the secret word
1.9     |  $D' \leftarrow \{p' \in D \mid p' \text{ is feasible after } m\}$ 
1.10    | if WordleGuesser( $D', \ell - 1$ ) is a loss for the gesser then
1.11      | | potentialGuess  $\leftarrow$  false
1.12      | | break
1.13    | if potentialGuess then
1.14      | | Gesser can win using  $p$  as its next guess.
1.15 If this point is reached, then the gesser loses.

```

Now, provided a gesser can solve the Wordle problem efficiently, they can play optimally by following Algorithm 1. To see correctness, we can proceed by induction over ℓ . The base case $\ell = 0$ is trivially correct. For the inductive case, if $|D| = 1$ then also correctness is trivial. In any other case, if there is a winning strategy for the gesser within ℓ attempts, that implies that after the first guess, regardless of what the secret word is, it will be possible to win within $\ell - 1$ attempts over the dictionary restricted to the corresponding feasible words. Thus, the inductive hypothesis guarantees that calls to the algorithm with parameter $\ell - 1$ will be correct, thus implying correctness of the algorithm. This justifies the choice of Wordle as a decision problem. Our next step is to study its complexity. We will prove the following theorems:

► **Theorem 1.** *Wordle is NP-hard, and it is W[2]-hard when parameterized by ℓ , the number of guesses allowed.*

► **Theorem 2.** *Wordle is NP-hard even when restricted to instances where $k = 5$.*

19:4 Wordle Is NP-Hard

► **Theorem 3.** *Wordle can be solved in polynomial time if the alphabet size $\sigma = |\Sigma|$ is constant.*

In order to prove Theorem 1, we will reduce from **Almost Set Cover**. An input of **Almost Set Cover** is a pair (\mathcal{F}, c) , where \mathcal{F} is a family of sets $S_1, \dots, S_{|\mathcal{F}|}$ whose union we denote by U , and c is an integer. The goal is to decide whether there is a sub-family $\mathcal{F}' \subseteq \mathcal{F}$ of at most c sets such that

$$\left| U \setminus \left(\bigcup_{S \in \mathcal{F}'} S \right) \right| \leq 1.$$

In other words, the goal is to decide whether it is possible to cover all the elements of the universe, except for perhaps one of them, with c sets. **Almost Set Cover** is $W[2]$ -hard (parameterized by c), as we show next. It is a simple reduction from standard **Set Cover**, which is known to be $W[2]$ -hard.

► **Lemma 4.** *Almost Set Cover is $W[2]$ -hard, when parameterized by c .*

Proof. Let (\mathcal{F}, c) be an instance of **Set Cover**, and let $U = \bigcup \mathcal{F}$ be its universe. Create first U' of size $2|U|$ in the following way: for each element $u \in U$ put elements u^+ and u^- in U' . Now, create \mathcal{F}' with $|\mathcal{F}|$ sets as follows: for each set $S \in \mathcal{F}$, put into \mathcal{F}' a set

$$S' = \{u^+, u^- \mid u \in S\}.$$

We claim $(\mathcal{F}, c) \in \text{Set Cover}$ iff $(\mathcal{F}', c) \in \text{Almost Set Cover}$. For the forward direction, if $\mathcal{F}^* \subseteq \mathcal{F}$ is a set cover of size c for \mathcal{F} , then its corresponding sub-family

$$\mathcal{F}^{*'} = \{S' \mid S \in \mathcal{F}^*\}$$

is trivially a set cover for \mathcal{F}' . On the other hand, if no sub-family K of at most c sets in \mathcal{F} covers \mathcal{F} , that means that for every such sub-family K , there is at least one element $u \notin \bigcup K$. But now if we consider

$$K' = \{S' \mid S \in K\},$$

then both u^+ and u^- are not covered by K' . This implies that every sub-family $K' \subseteq \mathcal{F}'$ of size at most c leaves at least two elements outside. This concludes the reduction, as it can be clearly computed in FPT-time. ◀

We are now ready to prove Theorem 1.

Proof of Theorem 1. Let (\mathcal{F}, c) be an instance of **Almost Set Cover**. We will build a dictionary $D_{\mathcal{F}}$ as follows. First, we will need $|U| + 2$ symbols, that we will denote $\Sigma_{\mathcal{F}} = \{\perp, 1, s_1, \dots, s_{|U|}\}$. Now, $D_{\mathcal{F}}$ will contain two different kinds of words, *element-words* and *set-words*, both of which will have length $|U|$. For every element $u_i \in U$, build its corresponding word w_{u_i} as:

$$w_{u_i}[j] = \begin{cases} 1 & \text{if } i = j \\ s_i & \text{otherwise.} \end{cases}$$

All words w_{u_i} created this way constitute the set of element-words. Now, for every set $S_i \in \mathcal{F}$, build its corresponding set-word w_{S_i} as:

$$w_{S_i}[j] = \begin{cases} 1 & \text{if } u_j \in S_i \\ \perp & \text{otherwise.} \end{cases}$$

We now claim that $(\mathcal{F}, c) \in \text{Almost Set Cover}$ iff $(D_{\mathcal{F}}, c + 1) \in \text{Wordle}$. For the forward direction, if there is a family of sets \mathcal{F}' , with $|\mathcal{F}'| \leq c$ and such that $|U \setminus \bigcup \mathcal{F}'| \leq 1$, then the guesser can use the set-words w_S for every $S \in \mathcal{F}'$ to guarantee a win. If the secret word w was chosen to be a set-word, then we have two cases after the first guess $w_S, S \in \mathcal{F}'$. Either the secret word w is revealed to be exactly w_S , in which case the guesser wins, or it is another set-word $w_{S'}$, in which case we claim that after guess w_S is made, $w_{S'}$ will be the only feasible word remaining and thus the guesser will win in her second turn. Indeed, for every element $u_i \in U$ there are 4 cases:

1. $(u_i \in S, u_i \in S')$. In this case $w_S[i] = w_{S'} = 1$, and it will be marked green. This makes all words that do not have a 1 in their i -th position infeasible.
2. $(u_i \in S, u_i \notin S')$. In this case $w_S[i] = 1$, and it will not be marked green. This makes all words that have a 1 in their i -th position infeasible.
3. $(u_i \notin S, u_i \in S')$. In this case $w_S[i] = \perp$, and it will not be marked green. This makes all words that have a \perp in their i -th position infeasible.
4. $(u_i \notin S, u_i \notin S')$. In this case $w_S[i] = w_{S'} = \perp$, and it will be marked green. This makes all words that do not have a \perp in their i -th position infeasible.

Note that regardless of the case, words that do not agree with $w_{S'}$ on their i -th position become infeasible, and as this happens for every $i \in [U]$, the only remaining feasible word is $w_{S'}$. As $c + 1 \geq 1$, and we have shown that if the secret word is a set-word then the guesser can win in at most 2 turns, we can safely proceed to the case where the secret word is an element-word. For this case the guesser can first make c guesses corresponding to all the words w_S for $S \in \mathcal{F}'$. Let w_{u_i} be the secret element word. As $|U \setminus \bigcup \mathcal{F}'| \leq 1$, there are two cases: in the first case, there is some $S_j \in \mathcal{F}'$ such that $u_i \in S_j$. Whenever the guess w_{S_j} is made, as it will happen that $w_{S_j}[i] = w_{u_i}[i] = 1$. This will reveal a 1 in the i -th position of the secret word, and as it is an element-word, this unambiguously reveals w_{u_i} to the guesser, who will win in the next turn, and thus in no more than $c + 1$ guesses. The second case is when w_{u_i} , the secret word, is the only element of $U \setminus \bigcup \mathcal{F}'$, and thus the guesser simply say that word in its next turn, also ensuring a win. This concludes the forward direction.

For the backward direction, assume every sub-family $\mathcal{F}' \subseteq \mathcal{F}$ leaves at least two elements u_1 and u_2 of U uncovered. Then we claim that no strategy for the guesser can guarantee a win within $c + 1$ attempts. Indeed, for any set of c initial guesses the guesser can make, we claim that there are at least two feasible words remaining, and thus the guesser cannot ensure a win in her last turn. To see this, consider any sequence $W = w_1, \dots, w_c$ of c guesses, and then define

$$\mathcal{F}_{\text{sets}} = \{S \in \mathcal{F} \mid w_S \in W\} \quad ; \quad \mathcal{F}_{\text{elem}} = \{\{u\} \subseteq U \mid w_u \in W\}.$$

Now observe that every element-word w_u such that $u \notin \bigcup (\mathcal{F}_{\text{sets}} \cup \mathcal{F}_{\text{elem}})$, is feasible after the sequence of guesses W . Also notice that for every $S \in \mathcal{F}_{\text{elem}}$ there is a set $S' \in \mathcal{F}$ such that $S \subseteq S'$. Thus, if for every set $S \in \mathcal{F}_{\text{elem}}$ we choose a set $S' \in \mathcal{F}$ such that $S \subseteq S'$, we get a collection of sets \mathcal{F}' such that $(\bigcup \mathcal{F}_{\text{elem}}) \subseteq (\bigcup \mathcal{F}')$ and $|\mathcal{F}'| = |\mathcal{F}_{\text{elem}}|$. This implies that $(\mathcal{F}_{\text{sets}} \cup \mathcal{F}') \subseteq \mathcal{F}$ and $(\mathcal{F}_{\text{sets}} \cup \mathcal{F}')$ is a collection of at most c sets, and thus there are at least two elements $u_1, u_2 \in U$ that are not on its union. We thus also have that

$$\{u_1, u_2\} \cap \left(\bigcup \mathcal{F}_{\text{sets}} \cup \mathcal{F}_{\text{elem}} \right) = \emptyset,$$

and consequently, both w_{u_1} and w_{u_2} are feasible words, which implies it is not possible to guarantee a win in the next guess. This concludes the proof. \blacktriangleleft

Theorem 2 captures an arguably essential part of Wardle’s Wordle: difficulty does not require long words. Its proof is inspired by that of coNP-hardness for Evil Hangman by Barbay and Subercaseaux [1]. We will use a result in hardness of approximation to prove that Wordle cannot be solved efficiently unless $P = NP$. In particular, we will use that $\gamma(G)$, the size of the smallest dominating set in a 4-regular graph G , is NP-hard to approximate within $1 + \frac{1}{390}$ [2, 3]. For our purpose it will be enough to use that is NP-hard to compute a $(1 + \epsilon)$ -approximation.

Proof of Theorem 2. Let G be a 4-regular graph. We will work with words of length $k = 5$. The alphabet Σ will be the vertex set of G . We build a dictionary D_G from G as follows: for every vertex $v \in G$, we create two words, w_v and w'_v : the word w_v has v as its first symbol is v , and its 4 remaining symbols are the neighbors of v (the order of these 4 symbols can be chosen arbitrarily), while the word w'_v consists simply of the symbol v repeated 5 times. Given a dictionary D , let $W(D)$ be the minimum value of ℓ such that (D, ℓ) is a positive instance of Wordle. We now claim that $\gamma(G) \in [W(D_G) - 4, W(D_G)]$. Note that this claim is enough to prove the theorem, as a polynomial time algorithm for Wordle would imply that $W(D_G)$ can be computed in polynomial time, and thus $\gamma(G)$ could be approximated up to an additive constant, contradicting its $1 + \frac{1}{390}$ hardness of approximation [2, 3]. In order to prove $\gamma(G) \geq W(D_G) - 4$, we show that a dominating set of size d for G implies a guessing strategy that guarantees a win within $d + 4$ guesses. Assume G has a dominating set S of size at most d and the secret word is either w_u or w'_u for some $u \in V(G)$. Now consider the sequence of guesses $w_v, v \in S$. If $u \in S$ we have two cases, either the secret word was w_u , and thus the secret word was already guessed in d guesses, or the secret word was w'_u , in which case the first symbol of guess w_u was marked green, leaving only w'_u as a feasible word, and thus allowing a guaranteed win within $d + 1$ guesses. If $u \notin S$, then as S is a dominating set u must be a neighbor of some vertex v^* in S . If the secret word was w_u , then the symbol u in guess w_{v^*} must have been marked yellow, which allows to find w_u by guessing the sequence of words $w_{u'}$ for $u' \in N(v^*)$, which must contain w_u , as u is a neighbor of v . This guessing strategy has at most $d + 4$ guesses and it is guaranteed to succeed. On the other hand, if the secret word was w'_u , then the symbol u in guess w_{v^*} must have been marked green, which makes w'_u the only feasible word, and thus allows to win in $d + 1$ guesses. In order to prove $\gamma(G) \leq W(D_G)$, we show that is not possible to guarantee a win within $\gamma(G) - 1$ guesses. Indeed, any sequence σ of $\gamma(G) - 1$ guesses induces a set S of vertices by considering the first symbol of each guess. As $|S| \leq \gamma(G) - 1$, there needs to be a vertex v that is not dominated by S , the word w'_v is still a feasible word that is not part of σ , and thus in the case where every request in σ was entirely marked with gray, then σ does not make the guesser win. As this is true for any sequence σ of length $\gamma(G) - 1$, we conclude $W(D_G) \geq \gamma(G)$. ◀

We naturally obtain hardness of approximation as a corollary.

► **Corollary 5.** *Let $W(D)$ be the minimum number of guesses ℓ such that the guesser can guarantee to win a game of Wordle over dictionary D within ℓ guesses, i.e., $(D, \ell) \in \text{Wordle}$. Then it is NP-hard to approximate $W(D)$ within $1 + \frac{1}{390}$.*

Interestingly, both the proof of Theorem 1 and Theorem 2 crucially depend on variable length alphabets. This is further confirmed by Theorem 3, which we will prove next. First, consider the following lemma:

► **Lemma 6.** *Wordle restricted to instances where $\ell \leq c$, for some fixed constant c , can be solved in polynomial time.*

Proof. This can be seen by analyzing the branching factor and depth of the associated game-tree. The game-tree has a branching factor of $O(D)$, as at any point the guesser can choose one of the $O(D)$ feasible words remaining, and for each such word there are at most $O(D)$ possible responses the guesser can get, depending on what the secret word w was at that point. The game-tree has also depth at most ℓ , which implies thus that the size of the game-tree is at most $O(D^\ell) = D^{O(1)}$, and thus the optimal guess at any given point can be computed in polynomial time. The result follows from this. ◀

Now we can prove a key lemma relating $\sigma = |\Sigma|$ and ℓ .

► **Lemma 7.** *On a game of Wordle over an alphabet Σ of size $\sigma = |\Sigma|$, the guesser can always win in at most σ guesses.*

Proof. Consider the following simple algorithm for the guesser: *at any point of the game, choose any feasible word as a guess.* We will show that this algorithm requires at most σ guesses to find the secret word. Indeed, for each index $1 \leq i \leq k$, let us define sets $S(i) \subseteq \Sigma$ as:

$$S(i) = \{s \in \Sigma \mid \text{there is a feasible word } w \in D, \text{ such that } w[i] = s\}.$$

Note that, before the first guess, $S(i) = \Sigma$ for every i . The key idea is now the following: after a feasible word w is guessed by the aforementioned strategy, the following holds for every $1 \leq i \leq k$: $w[i]$ can either be marked green, in which case $S(i) \leftarrow \{w[i]\}$, or it can be marked with gray or yellow, in which case $S(i) \leftarrow S(i) \setminus \{w[i]\}$. This implies that after every feasible guess each set $S(i)$ either becomes a singleton or decreases its size by 1. Therefore, by doing $\sigma - 1$ feasible guesses, either the secret word has been found (in which case we are done), or it happens that every set $S(i)$ must be a singleton, and thus there will be only one remaining feasible word, which allows the guesser to win within σ guesses in total. ◀

We are now ready to use the previous lemmas and prove Theorem 3.

Proof of Theorem 3. Consider an instance (D, ℓ) of Wordle where $|\Sigma| \in O(1)$. By Lemma 7, if $|\Sigma| \leq \ell$, then simply answer **Yes**. Otherwise $\ell \leq |\Sigma| = O(1)$, and thus by Lemma 6 the result follows. ◀

3

O P E N

Problems

We now offer problems that our analysis leaves open.

1. Is Wordle in NP, or is it PSPACE-complete? It is not hard to see that Wordle is in PSPACE, by simply computing its associated game-tree, which has polynomial depth by Lemma 7.
2. Is Wordle in FPT when parameterized by $\sigma = |\Sigma|$? The proof of Theorem 3 provides an algorithm in time $D^{O(\ell)}$, and it seems challenging to obtain a $\text{poly}(D) \cdot f(\ell)$ algorithm for some computable function ℓ .
3. How does is the complexity of Wordle affected by having a dictionary that is not presented as a list of word, but rather in some more compact representation as a finite automaton? We can envision D to be provided as a finite automaton, for which all accepted words of length k are considered part of the dictionary. This affects for example the proof of Theorem 3, as now a branching factor of $|D|$ can be exponential in the input size.

References

- 1 Jérémy Barbay and Bernardo Subercaseaux. The computational complexity of evil hangman. In *10th International Conference on Fun with Algorithms (FUN 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- 2 M. Chlebík and J. Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Information and Computation*, 206(11):1264–1275, November 2008. doi:10.1016/j.ic.2008.07.003.
- 3 Miroslav Chlebík and Janka Chlebíková. Approximation hardness of edge dominating set problems. *Journal of Combinatorial Optimization*, 11(3):279–290, May 2006. doi:10.1007/s10878-006-7908-0.
- 4 New York Times Company. Wordle is joining the new york times games. <https://www.nytimes.com/press/wordle-new-york-times-games/>, 2022. Accessed: 2022-02-17.
- 5 David Eppstein. Computational complexity of games and puzzles. <https://www.ics.uci.edu/~eppstein/cgt/hard.html>, 1997. Accessed: 2022-02-17.
- 6 Sam Ganzfried. Computing strong game-theoretic strategies in jotto. In *Lecture Notes in Computer Science*, pages 282–294. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-31866-5_24.
- 7 Michael T. Goodrich. On the algorithmic complexity of the mastermind game with black-peg results. *Information Processing Letters*, 109(13):675–678, June 2009. doi:10.1016/j.ipl.2009.02.021.
- 8 Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. A. K. Peters, Ltd., USA, 2009.
- 9 Nicole Holliday and Ben Zimmer. Wordle’s creator thinks he knows why the game has gone so viral. <https://slate.com/culture/2022/01/wordle-game-creator-wordle-twitter-scores-strategy-stats.html>, 2022. Accessed: 2022-02-17.
- 10 Meeta Malhotra. Wordle’s viral success is based on design. <https://thehardcopy.co/wordles-viral-success-is-based-on-design/>, 2022. Accessed: 2022-02-17.
- 11 Erin Sebo. Codecracking, community and competition: why the word puzzle wordle has become a new online obsession. <https://theconversation.com/codecracking-community-and-competition-why-the-word-puzzle-wordle-has-become-a-new-online-obsession-174878>, 2022. Accessed: 2022-02-17.
- 12 Jeff Stuckman and Guo-Qiang Zhang. Mastermind is np-complete, 2005. arXiv:cs/0512049.
- 13 Daniel Victor. Wordle is a love story. <https://www.nytimes.com/2022/01/03/technology/wordle-word-game-creator.html>, 2022. Accessed: 2022-02-17.
- 14 Giovanni Viglietta. Hardness of mastermind. In Evangelos Kranakis, Danny Krizanc, and Flaminia Luccio, editors, *Fun with Algorithms*, pages 368–378, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

Mirror Games Against an Open Book Player*

Roey Magen ✉

Weizmann Institute of Science, Rehovot, Israel

Moni Naor ✉

Weizmann Institute of Science, Rehovot, Israel

Abstract

Mirror games were invented by Garg and Schnieder (ITCS 2019). Alice and Bob take turns (with Alice playing first) in declaring numbers from the set $\{1, 2, \dots, 2n\}$. If a player picks a number that was previously played, that player loses and the other player wins. If all numbers are declared without repetition, the result is a draw. Bob has a simple mirror strategy that assures he won't lose and requires no memory. On the other hand, Garg and Schenier showed that every deterministic Alice needs memory of size linear in n in order to secure a draw.

Regarding probabilistic strategies, previous work showed that a model where Alice has access to a secret random perfect matching over $\{1, 2, \dots, 2n\}$ allows her to achieve a draw in the game w.p. a least $1 - \frac{1}{n}$ and using only polylog bits of memory.

We show that the requirement for secret bits is crucial: for an “open book” Alice with no secrets (Bob knows her memory but not future coin flips) and memory of at most $n/4c$ bits for any $c \geq 2$, there is a Bob that wins w.p. close to $1 - 2^{-c/2}$.

2012 ACM Subject Classification Theory of computation → Streaming models

Keywords and phrases Mirror Games, Space Complexity, Eventown-Oddtown

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.20

Acknowledgements We thank Boaz Menuhin for very useful discussions and comments and thank Ehud Friedgut for discussions concerning an extension of Berlekamp's Theorem. We thank the anonymous referees for numerous helpful comments.

1 Introduction

In the mirror game, Alice (the first player) and Bob (the second player) take turns picking numbers belonging to the set $\{1, 2, \dots, 2n\}$. A player loses if they repeat a number that has already been picked. After $2n$ rounds, when no more numbers are left to say, then the result of the game is a draw. With perfect memory, both players can keep track of all the numbers that have been announced and avoid losing.

Bob, who plays second, has a simple deterministic, low memory strategy (which can actually be performed even by humans): Bob fixes any perfect matching on the elements $\{1, 2, \dots, 2n\}$; for example $(1, 2), (3, 4), \dots, (2n - 1, n)$. For every number picked by Alice, Bob responds with the matched number. Unless Alice repeats a number, this allows Bob to pick in every turn a number that has not appeared so far. Note that Bob needs to remember just the last number Alice played, i.e. $O(\log n)$ bits, in order to execute this strategy.

This game was suggested by Garg and Schneider [4] as a very simple example of a mirror strategy (hence the name). Given the limited computational resources they require, the question is when can they be implemented. In particular whether Alice has a lower memory strategy as well and if not what is it that breaks the symmetry between the two players.

* Research supported in part by grants from the Israel Science Foundation (no.2686/20), by the Simons Foundation Collaboration on the Theory of Algorithmic Fairness.



What Garg and Schneider showed is that every deterministic “winning” (drawing) strategy for Alice requires space that is linear in n . I.e. there is no mirror like strategy for her.

What about probabilistic strategies? Here we have to consider the model carefully. Suppose that Alice has a secret memory of some bounded size (we call this the “closed book” case) as well as a supply of random bits (that Bob cannot access). Garg and Schneider showed a randomized strategy that manages to draw against any strategy of Bob with probability at least $1 - \frac{1}{n}$ and requires $O(\sqrt{n} \log^2 n)$ bits of memory while relying on access to a *secret random perfect matching oracle*. Feige [2] showed an improved randomized strategy, under the same settings, that draws with the same probability and needs only $O(\log^3 n)$ bits of memory. Menuhin and Naor [6] showed that such strategies can be implemented with either $O(n \log n)$ bits of long lasting randomness (in addition to the $O(\log^3 n)$ bits of memory), or to replace the assumptions with cryptographic ones (assuming that Bob cannot break a one-way function) and obtain a low memory strategy for Alice.

1.1 An Open Book Alice

In contrast to the above model, we consider the case where Alice has *no secret memory*. Alice has $m < 2n$ memory bits, but their content is known to Bob at any point. She has an “unlimited” supply of randomness: in each turn Alice can ask for any number of additional random bits, and then she has access to those bits throughout the rest of the game. However, since Alice has no secrets, Bob knows her random bits *after she asks for them*¹. The question we consider is whether Bob has a strategy that promises to win (forces Alice to lose) with high probability when Alice has such limited memory?

As usual, we assume that Bob is aware of Alice’s strategy and creates his own strategy accordingly. The question is whether for any Alice that has sublinear memory there is a Bob that makes her lose with certain probability. As we shall see, this is indeed the case. We will first see a strategy that makes Alice lose with probability about 1/2 (Theorem 12) and then show how to amplify this to any constant (Theorem 15): any Alice that has memory of size smaller than n/c loses except with probability at most $2^{-\gamma c}$ for some constant $\gamma > 0$.

Note that if Alice chooses all her random bits in advance, then a simple argument shows that she will lose with probability 1: once the random bits are chosen, then her strategy is deterministic and from the work of Garg and Schneider there is the best choice of Bob’s strategy that makes her lose. So the whole difficulty revolves around the issue of Alice choosing the random bits on the fly.

2 Combinatorial Preliminaries

2.1 Oddtown-Eventown

Garg and Schneider [4], in showing their linear lower bound on the space complexity of any deterministic winning or drawing strategy of Alice, used the famous Eventown-Oddtown Theorem of Berlekamp from extremal combinatorics. This theorem puts a bound on the maximum number of even sets where every pair has an odd intersection:

► **Definition 1.** An “Oddtown” is a collection \mathcal{F} of subsets of $\{1, 2, \dots, N\}$ where every subset $s \in \mathcal{F}$ is of even cardinality, but the intersection of every pair of distinct subsets $s_i, s_j \in \mathcal{F}$ has an odd cardinality.

¹ This is similar to the “full information model” in distributed computing.

► **Theorem 2.** *Every Oddtown contains at most N subsets.*

A proof can be found, for instance, in [3]. We will use the following corollary of the theorem:

► **Corollary 3.** *Let \mathcal{F} be a collection of subsets of $\{1, 2, \dots, N\}$ where every subset $s \in \mathcal{F}$ has an even cardinality. For every integer j , if $|\mathcal{F}| > N + j$ then \mathcal{F} contains at least $\lceil \frac{j}{2} \rceil$ disjoint pairs of subsets from \mathcal{F} , where for each pair (s_i, s_j) the union $s_i \cup s_j$ is of even cardinality. We call this a matching P of pairs.*

Proof. As long as \mathcal{F} contains more than N elements, by Theorem 2 it contains pair of distinct subsets that have an intersection of even cardinality. Since every subset in \mathcal{F} has even cardinality, the union of the pair has even cardinality as well. Put the pair in the matching P , delete the pair from \mathcal{F} and continue the process. ◀

▷ **Claim 4.** Let $\mathcal{A}_1, \dots, \mathcal{A}_m$ be events whose union is the entire sample space. Let $H_b = \{\mathcal{A}_i \mid \Pr(\mathcal{A}_i) > b, i \in \{1, \dots, m\}\}$ be the set of events with relative “high” probability. Let \mathcal{A} be the event that one of the events from H_b happens. If $b < \frac{1}{m}$, then $\Pr[\mathcal{A}] \geq 1 - (m-1) \cdot b$.

► **Definition 5.** *A function $f: \mathbb{N} \rightarrow \mathbb{R}$ is negligible if for every positive integer c there exists an integer N_c such that for all $x > N_c$, $|f(x)| < \frac{1}{x^c}$. We use the term **negl** to denote some negligible function.*

3 Introduction to the Mirror Game

3.1 Strategies and memory

Alice has $m < n$ bits of memory, and we refer to her state as $x \in \{0, 1\}^m$. In addition, at the end of turn $2i$ (after Bob picks a number) Alice chooses random bits $r_i \in \{0, 1\}^\ell$, when this r_i is added to the set of random strings she can access. I.e. at the i th round Alice has at her disposal the memory state $x \in \{0, 1\}^m$ and $R_A^i = (r_1, r_2, \dots, r_i)$. The size of ℓ can be arbitrary large. If the value of i is clear from the context, we will simply use R_A .

Formally, the description of Alice’s strategy consists of two functions:

1. **Next move function** $next: \{0, 1\}^m \times \{0, 1\}^{i \cdot \ell} \rightarrow \{1, 2, \dots, 2n\}$. receiving the current memory state, the random bits chosen in steps r_1, \dots, r_i and outputting a number in $\{1, 2, \dots, 2n\}$.
2. **State transition function** $upd: \{1, \dots, 2n\} \times \{0, 1\}^m \times \{1, \dots, 2n\} \times \{0, 1\}^{i \cdot \ell} \rightarrow \{0, 1\}^m$.

Receiving the last move of Bob, the current memory state, turn number $2i + 1$, the random bits chosen in steps r_1, \dots, r_i and assigning a new memory state.

Note that Bob knows these functions. In the context of the random bits of Alice, when Alice receives a new sequence of random bits r_i , Bob has access to r_i as well (but not to the future bits, before they are selected).

Let R_A and R_B be the random variables that represent the random bits of Alice and Bob respectively. Note that in the beginning of turn $2i + 1$, $R_A^i = (r_1, r_2, \dots, r_i)$ and at that point Bob knows the value of R_A^i . Let S^i be the random variable that represents the subset of numbers that have been picked by either player until turn i . Similarly, X_i is the random variable that represents the memory state of Alice in the i th turn. If the value of i is clear, we will simply use X and S .

Given $x \in \{0, 1\}^m$, turn $i \in \{1, 2, \dots, 2n\}$ and R_A , for a subset $s \subseteq \{1, 2, \dots, 2n\}$ of size i we will consider the following notation:

- $\Pr[S = s] := \Pr_{R_B}[\text{set of used numbers is } s \mid R_A = r_A]$

20:4 Mirror Games Against an Open Book Player

■ $\Pr[X_i = x] := \Pr_{R_B}[X_i = x | R_A = r_A]$, and if i is clear we will use $\Pr[X = x]$.

Now we will consider the conditional probability that s was the subset of chosen numbers that lead to memory state x :

$$\Pr[S = s | x] := \Pr_{R_B}[\text{The set of used numbers is } s | \text{memory state is } x, R_A^i = r_A].$$

► **Definition 6.** For turn $1 \leq i \leq 2n$, memory state $x \in \{0, 1\}^m$ and random strings r_A , denote with $\mathcal{F}_x^{r_A}$ the collection of sets $s \subseteq \{1, 2, \dots, 2n\}$ of size i s.t. it is possible that the state of memory of Alice at turn i is x when the set of numbers played is equal to s and Alice's random string is r_A . Call those sets the feasible sets.

For ease of notation we sometime use \mathcal{F}_x , instead of $\mathcal{F}_x^{r_A}$.

4 Bob's Strategy Against an Open Book Alice

We present a strategy for Bob against any Alice with $o(n)$ memory and without secrets. Namely, Bob has access to Alice's memory state, random bits as well as her strategy, namely the state transition function upd and next move function $next$, but not to any future random bits. We will first see in Section 4.1 a strategy that makes Alice lose with probability about $1/2$. We will then see in Section 4.2 how to amplify this strategy and reduce Alice's probability of winning to any $1/2^c$ constant at the price of limiting her memory to be at most n/ac for some constant a .

The rough outline of the strategy is for Bob to start by a sequence of random moves that do not include any of the numbers selected. At some midpoint Bob changes course. He selects a pair of subsets, so that one of them is the correct one played up to this point and the other one is a "decoy". The pair is selected so that from Alice's point of view these two sets are indistinguishable. Bob continues by avoiding elements that belong to this pair. Alice has to be the first one to venture into this territory and make a guess who is the real one and who is the decoy. She loses with probability $1/2$. The strategy is not computationally efficient, even if Alice has an efficient strategy, since Bob needs to come up with the pairing.

4.1 Bob Can Win With Probability Close to Half

We present a strategy for Bob that makes him win with probability close to half. Let $1 \leq k \leq [n]$ be a value that will be determined later (we will see that $k = 0.4n$). Until turn $2k$ Bob chooses a number uniformly at random from those that have not been picked so far.

Let s be the actual set of used numbers and let $x \in \{0, 1\}^m$ be the memory state of Alice after turn $2k$. We will show that if Alice has low memory, then with high probability Bob can find a partition of the feasible sets $\mathcal{F}_x^{r_A}$ (Definition 6) into pairs. For every pair (s_1, s_2) in the partition, (i) the two subsets s_1 and s_2 have similar conditional probabilities that they are the sets actually played given memory state $x \in \{0, 1\}^m$ and randomness r_A , and (ii) the cardinality of the union of the pair is even. Namely,

1. $\Pr_{R_B}[S = s_1 | X = x, R_A = r_A] \approx \Pr_{R_B}[S = s_2 | X = x, R_A = r_A]$.
2. $|s_1 \cup s_2|$ is even.

Note that this partition does not depend on s , just on R_A and x . But as we show w.h.p s belongs to some pair in the partition, meaning that there exists (s_1, s_2) in the partition s.t. $s \in \{s_1, s_2\}$. Bob's strategy after turn $2k$ is just to pick some number that has not been used and that is not in $s_1 \cup s_2$.

We claim that since $|s_1 \cup s_2|$ is even, Alice must be the one that chooses the first value in $s_1 \cup s_2$ after turn $2k$. Since s_1 and s_2 have similar probabilities, even if a “little birdy” tells Alice about s_1 and s_2 , she will not be able to differentiate between them in real-time. Namely, she does not know whether $s = s_1$ or $s = s_2$. Therefore she loses with probability close to half.

Bob’s strategy for making Alice lose with probability 1/2

1. From turn 2 until turn $2k$: Bob chooses uniformly at random a number that has not been picked so far.
2. Following turn $2k$ Bob finds two sets s_1 and s_2 of size $2k$ s.t. the actual set of number picked so far s is equal to one of them, and $|s_1 \cup s_2|$ is even.
3. From turn $2k$ till the end, Bob picks some number that has not been used and that is not in $s_1 \cup s_2$.

Note that after turn $2k$ Bob’s strategy can be deterministic. For example pick the smallest number that has not been used and that is not in $s_1 \cup s_2$.

▷ **Claim 7.** For every subset $s \subseteq \{1, 2, \dots, 2n\}$ and for every assignment r_A to the random bits of Alice, if $|s| = 2k$ then

$$\Pr_{R_B}[S = s | R_A = r_A] \leq \left(\frac{2k}{2n}\right)^k$$

Proof. For every turn $i \in \{2, 4, \dots, 2k\}$, if all of the numbers picked so far belong to s , then the cardinality of the set of numbers that have not been used and belongs to s is exactly $2k - i + 1$. Thus the probability that Bob chooses a number from s is exactly $\frac{2k-i+1}{2n-i+1}$ independent from r_A . Hence, let $Z = \{2, 4, \dots, 2k\}$ and we get:

$$\Pr_{R_B}[S = s | R_A = r_A] \leq \prod_{i \in Z} \frac{2k - i + 1}{2n - i + 1} \leq \prod_{i \in Z} \frac{2k - 1}{2n - 1} = \left(\frac{2k - 1}{2n - 1}\right)^k \leq \left(\frac{2k}{2n}\right)^k \quad \triangleleft$$

► **Corollary 8.** For every subset $s \subseteq \{1, 2, \dots, 2n\}$ and for every assignment r_A to the random bits of Alice, if $k = 0.4n$ and $|s| = 2k$ then

$$\Pr_{R_B}[S = s | R_A = r_A] \leq 0.4^{0.4n} = 0.4^{0.2 \cdot 2n} \leq 0.84^{2n}$$

These numbers were chosen for convenience, where the goal is to get an upper bound on $\Pr_{R_B}[S = s]$ of the form α^{2n} for some constant $0 < \alpha < 1$.

We call a memory state x *useful* for r_A if $\Pr_{R_B}[X_{2k} = x | R_A = r_A] > 0.9^{2n}$. Once again, this number was chosen for convenience. We want to get a lower bound on $\Pr[X = x | R_A = r_A]$ of the form β^{2n} for some constant $0 < \beta < 1$ s.t. $\alpha < \beta$. Let $U_{r_A} \subseteq \{0, 1\}^m$ be the set of useful memory for r_A . We argue that w.h.p. Alice’s memory state in turn $2k$ is useful.

▷ **Claim 9.** Let X be the random variable representing the memory state after turn $2k$. For any assignment to Alice’s coin flips r_A , if $m < 0.2n$, then

$$\Pr_{R_B}[X \in U_{r_A} | R_A = r_A] \geq 1 - \text{negl}(n).$$

Proof. Let A_x be the event that $X = x$. Note that for every $x \in \{0, 1\}^m$ we get that $\Pr[A_x] \leq 0.9^{2n} \leq \frac{1}{2^m}$. Then by Claim 4:

$$\Pr_{R_B}[x \in U_{r_A}] \geq 1 - (2^m - 1) \cdot 0.9^{2n} > 1 - 0.97^{2n} + 0.9^{2n} = 1 - \text{negl}(n).$$

where the last inequality is true since $2^m \cdot 0.9^{2n} \leq 2^{0.2n} \cdot 0.9^{2n} < 0.97^{2n}$. ◁

20:6 Mirror Games Against an Open Book Player

▷ **Claim 10.** For any assignment r_A , if x is useful, then Bob can find a subset $W_x^{r_A} \subseteq \mathcal{F}_x^{r_A}$ and a partition $P_x^{r_A}$ of $W_x^{r_A}$ into pairs s.t.:

1. $\Pr_{R_B}[S \in W_x^{r_A} | x \in U_{r_A}, R_A = r_A] > 1 - \text{negl}(n)$.
2. For every pair $(s_1, s_2) \in P_x^{r_A}$ we have that $|s_1 \cup s_2|$ is even.
3. For every pair $(s_1, s_2) \in P_x^{r_A}$ the ratio of their conditional probabilities is close to 1:

$$\frac{2n}{2n+1} \leq \frac{\Pr_{R_B}[S = s_1 | x, R_A = r_A]}{\Pr_{R_B}[S = s_2 | x, R_A = r_A]} \leq \frac{2n+1}{2n}$$

Proof. The idea is to partition $\mathcal{F}_x^{r_A}$ into layers of the possible subsets that lead to memory state x according to the values of their conditional probability. The pair $(s_1, s_2) \in P_x^{r_A}$, only if $\Pr[S = s_1 | x]$ and $\Pr[S = s_2 | x]$ belong to the same layer and $|s_1 \cup s_2|$ is even. We consider $(2n)^3$ different layers. For $j \in \{1, \dots, (2n)^3\}$ let the j th interval of probabilities be

$$I_j = \left[\left(\frac{2n}{2n+1} \right)^j, \left(\frac{2n}{2n+1} \right)^{j-1} \right].$$

Every layer L_j represents the collection of subsets whose conditional probability given memory state x and r_A is in the interval I_j . Thus, $L_j = \{s \in \mathcal{F}_x^{r_A} | \Pr_{r_B}[S = s | X = x] \in I_j\}$. Note that if s_1 and s_2 in L_j , then

$$\frac{2n}{2n+1} \leq \frac{\Pr_{r_B}[S = s_1 | x]}{\Pr_{r_B}[S = s_2 | x]} \leq \frac{2n+1}{2n}.$$

Note that every subset in L_j has even cardinality. Thus, if layer L_j contains more than $2n$ subsets, then by Claim 3 it contains at least $\frac{|L_j| - N}{2}$ pairs with an even intersection. Call this set of pairs P_j . Let $P_x^{r_A}$ be the collection of all pairs from $P_1, P_2, \dots, P_{(2n)^3}$. Denote the collection of all subsets that belong to some pair in $P_x^{r_A}$ as $W_x^{r_A}$. Namely,

$$P_x^{r_A} := \{(s_1, s_2) \in P_j | j \in [(2n)^3]\} W_x^{r_A} := \{s \subseteq \{1, 2, \dots, 2n\} | \exists s', (s, s') \in P_x^{r_A}\}.$$

We will see that $P_x^{r_A}$ and $W_x^{r_A}$ meet all the three conditions of the claim:

1. Since x is useful and by Corollary 8 we get that for every $s \in \mathcal{F}_x^{r_A}$:

$$\Pr_{R_B}[S = s | X = x, R_A = r_A] \leq \frac{\Pr[S = s | R_A = r_A]}{\Pr[X = x | R_A = r_A]} \leq \frac{0.84^{2n}}{0.9^{2n}} < 0.934^{2n}$$

Therefore for every $F \subseteq \mathcal{F}_x^{r_A}$, if F contains at most a polynomial number of subsets, then $\Pr[S \in F | X = x]$ is negligible. There are $(2n)^3$ layers, and in every layer i , there are at most $2n$ subsets that do not belong to P_i . Thus we get at most $(2n)^4$ subsets that not belong to $P_x^{r_A}$. In addition define:

$$L_x = \{s \in \mathcal{F}_x^{r_A} | \Pr[S = s | x] < \frac{1}{2^{4n}}\}.$$

Since the cardinality of L_x is at most 2^{2n} , we get that

$$\Pr[S \in L_x | X = x] < 2^{2n} \cdot \frac{1}{2^{4n}} = 2^{-2n}.$$

We choose $(2n)^3$ layers, since $\left(\frac{2n}{2n+1}\right)^{(2n)^3-1} \leq \frac{1}{2^{4n}}$ for large enough n . Overall $W_x^{r_A}$ contains all of the subsets from $\mathcal{F}_x^{r_A}$, except some of the subsets in L_x and at most $(2n)^4$ other subsets, then the claim is follows.

2. Correct by the construction of $P_x^{r_A}$.
3. By construction for every pair $(s_1, s_2) \in P_x^{r_A}$ there exists j s.t $(s_1, s_2) \in L_j$. ◁

► **Observation 11.** *Since $|s_1 \cup s_2|$ is even, then Alice is the first to pick an element of $s_1 \cup s_2$ after turn $2k$.*

Now we will show that even if from turn $2k$ and on Alice has an unlimited size memory and randomness, and even if Alice knows $W_x^{r_A}$ for every memory state x , she still loses with probability close to half.

► **Theorem 12.** *If Alice has no secrets, and memory of size at most $0.2n$ bits, then there is a strategy for Bob where Alice loses with probability at least $\frac{1}{2} \cdot \frac{2n}{2n+1} - \text{negl}(n)$*

Proof. Let r_A be assignment to the random bits of Alice. If a memory state X is useful for r_A and $S \in W_x^{r_A}$, then there exists $(s_1, s_2) \in P_x^{r_A}$ s.t. $S \in (s_1, s_2)$, $|s_1 \cup s_2|$ is even and

$$\frac{2n}{2n+1} \leq \frac{\Pr[S = s_1|x]}{\Pr[S = s_2|x]} \leq \frac{2n+1}{2n}$$

From turn $2k$, Bob chooses a number that has not been used and not in $s_1 \cup s_2$. Then by Observation 11 the first one after turn $2k$ to pick a number from $s_1 \cup s_2$ is Alice. Intuitively, since s_1 and s_2 have similar conditional probability given X and r_A , Alice cannot determine whether $S = s_1$ or $S = s_2$. Then the best choice Alice can make the first time she picks a number from $s_1 \cup s_2$, is to choose a number from the set with the lowest conditional probability. But even then, she loses w.p. similar to half.

Formally, let a be the first number from $s_1 \cup s_2$ that Alice picks after turn $2k$. We want to analyze:

$$\Pr_{R_B}[a \in S | X \text{ is useful}, S \in W_X^{r_A}, R_A = r_A]$$

Note that if $a \in S$, then Alice loses. Indeed, for any randomness r_A and useful memory state $x \in U_{r_A}$:

$$\begin{aligned} & \Pr_{r_B}[a \in S | X = x, S \in \{s_1, s_2\}, (s_1, s_2) \in P_x^{r_A}, R_A = r_A] \\ & \geq \min_{s' \in \{s_1, s_2\}} \Pr_{r_B}[S = s' | X = x, S \in \{s_1, s_2\}, (s_1, s_2) \in P_x^{r_A}, R_A = r_A] \end{aligned} \quad (1)$$

$$\begin{aligned} & = \min_{s' \in \{s_1, s_2\}} \frac{\Pr_{r_B}[S = s' | X = x, R_A = r_A]}{\Pr_{r_B}[S \in \{s_1, s_2\} | X = x, R_A = r_A]} \\ & \geq \min_{s' \in \{s_1, s_2\}} \frac{\Pr_{r_B}[S = s' | X = x, R_A = r_A]}{\Pr_{r_B}[S = s_1 | X = x, R_A = r_A] + \Pr_{r_B}[S = s_2 | X = x, R_A = r_A]} \\ & \geq \frac{\left(\frac{2n}{2n+1}\right)^j}{2 \left(\frac{2n}{2n+1}\right)^{j-1}} \end{aligned} \quad (2)$$

$$= \frac{2n}{2n+1} \cdot \frac{1}{2} \quad (3)$$

Where all the probabilities are over Bob's coins R_B . Inequality 1 is true since given $S \in (s_1, s_2)$, without knowing S , decide if $S = s_1$ or $S = s_2$ is hard in terms of information at least as choosing number from S . Inequality 2 is true by property number 3 of Claim 10.

By Claim 9 the memory state x in turn $2k$ is useful with probability at least $1 - \text{negl}(n)$. By Claim 10 (first property) $S \in W_x^{r_A}$ with probability of at least $1 - \text{negl}(n)$. Finally, by Equation 3 we get that Alice loses w.p. at least $\frac{2n}{2n+1} \cdot \frac{1}{2} - \text{negl}(n)$. ◀

4.2 Amplifying the Probability that Bob Wins

For simplicity, we assume that n is even. Let $c \geq 2$ s.t. n/c is even as well. We will show a strategy for Bob where he wins with probability about $1 - (\frac{1}{2})^{c/2}$ against any open book Alice with bounded memory. The idea is to use similar arguments to the previous section, but instead of finding only one pair of subsets with similar conditional probability and an even union, we would like to find a collection of such pairs. We define $c/2$ breakpoints in time where each breakpoint represents some even numbered turn. Each breakpoint and memory state x , define a new partition to pairs of subsets (where the subsets are possible one that could have been used since the last breakpoint). The subsets are of size n/c and the matched subsets should have similar conditional probabilities.

For $1 \leq t \leq c/2$ and a set s of the numbers used between breakpoint $t-1$ and breakpoint t , we will show that w.h.p. s belongs to some pair of subsets (s_1^t, s_2^t) in the partition of feasible subsets for the given memory and Alice's coins; as before, s_1^t and s_2^t have similar conditional probability and an even union. Then Bob adds this pair to the collection of pairs that he already has from previous breakpoints. He uses this collection of pairs in order to define his winning strategy against Alice. Let us formalize the foregoing discussion.

Breakpoint number t for $t \in \{0, 1, \dots, c/2\}$ is chosen at turn number $k_t := t \cdot \frac{n}{c}$. Since n/c is even then k_t is even.

The span of turns between two successive breakpoints is called an **epoch**. In particular, for $t \in \{1, \dots, c/2\}$, the span of n/c turns between turns $k_{t-1} + 1$ and k_t is called a t -**epoch** (note that the epoch ends at the t th breakpoint.)

At the end of each breakpoint $t \in \{1, \dots, c/2\}$, Bob comes up with new pair (s_1^t, s_2^t) . We will show later how Bob finds these pairs. Denote with $C_t = (s_1^1, s_2^1), \dots, (s_1^t, s_2^t)$ the collection of pairs that Bob comes up until breakpoint number t . Let $C_0 := \emptyset$ and $C_t := C_{t-1} \cup \{(s_1^t, s_2^t)\}$.

Bob's amplified strategy:

Phase 1

- **From turn 2 Until turn $n/2$:** during the t th epoch Bob chooses *uniformly at random* among the elements that do not belong to any subset s that is part of a pair in C_{t-1} . Namely, from

$$\{1, \dots, 2n\} \setminus \bigcup \{s \mid s \in \{s_1^j, s_2^j\} \text{ and } (s_1^j, s_2^j) \in C_{t-1}\}$$

- Let $C = C_{c/2} = \{(s_1^1, s_2^1), \dots, (s_1^{c/2}, s_2^{c/2})\}$ be the collection of pairs of subsets from the first $n/2$ turns. For any turn $n/2 < i \leq 2n$, let C_i^* be all pairs $(s_1^i, s_2^i) \in C$ such that no member of $s_1^i \cup s_2^i$ was played from breakpoint t until turn i .

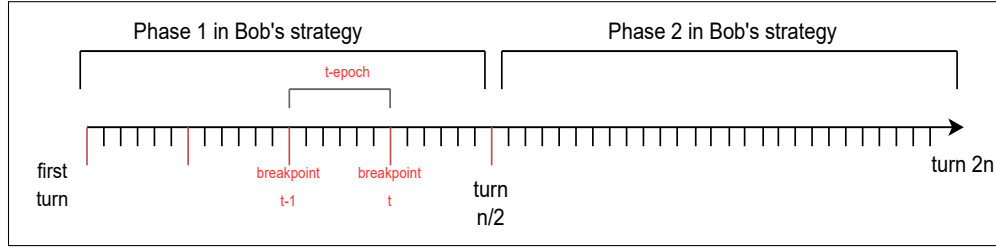
Phase 2

- **For every turn $i \in \{n/2 + 2, n/2 + 4, \dots, 2n\}$,** Bob picks some number that has not been used and not in

$$\bigcup \{s \mid s \in \{s_1^j, s_2^j\} \text{ and } (s_1^j, s_2^j) \in C_i^*\}$$

We still need to specify how Bob sets the pairs (s_1^t, s_2^t) in each breakpoint t , and what is the meaning of similar conditional probability in this case.

Let X^t be the random variable that represents the memory state of Alice at breakpoint number t , let S^t be the r.v. that represents the set of numbers that have been used during the t th epoch, let R_A^t be the r.v. that represents the randomness of Alice until breakpoint



■ **Figure 1** The red lines represent the breakpoints, the black lines represent the turns throughout the game. Not to scale.

number t and let H^t be the r.v. that represent all the history of the game until breakpoint number t . Namely,

$$H^t := (S^1, \dots, S^t, X^1, \dots, X^t, C^t)$$

Note that H_t represents the set of numbers that were played, the memory states of Alice in each breakpoint and the pairs that Bob chooses, until break point t .

Given history h^{t-1} and randomness r_A . Denote with $\mathcal{F}_x^{h^{t-1}, r_A}$ the collection of feasible sets $s \subseteq \{1, \dots, 2n\}$ of size n/c s.t. it is possible that the set of numbers picked during the t th epoch was s given all the history h^{t-1} until breakpoint $t - 1$ and Alice's random string in breakpoint number t is r_A .

For simplicity, we use the notation \mathcal{F}_x^t , instead of $\mathcal{F}_x^{h^{t-1}, r_A}$. We want to find a partition into pairs P_t of \mathcal{F}_x^t that includes most of the subsets in \mathcal{F}_x^t . First we show an upper bound for any set s on the probability that the set played in the $(t) - \text{epoch}$ is s . The number of values from $\{1, \dots, 2n\}$ that are not in C_{t-1} is at least n . Therefore, in each turn during epoch t , Bob has at least n numbers from which he chooses one at random one in Phase 1 of his strategy. If size of s is n/c , then, similarly to Claim 7, we get

$$\Pr_{R_B}[S^t = s | R_A = r_A, H^{t-1} = h^{t-1}] \leq \left(\frac{n/c}{n}\right)^{n/2c} = \left(\frac{1}{c}\right)^{2n/4c}$$

Let $\alpha = \left(\frac{1}{c}\right)^{1/4c}$. Note that $\alpha < 1$. Let $\alpha < \beta < 1$. We call a memory state x **useful** if $\Pr_{R_B}[X^t = x | H^{t-1} = h^{t-1}, R_A = r_A] > \beta^{2n}$. We want to show that w.h.p. Alice's memory state in each breakpoint is useful. Choose $\delta \in (0, 1)$ s.t. $2^\delta \cdot \beta < 1$. Recall that the motivation to define a useful memory state is to have an upper bound on the conditional probability of any set of numbers to be the one used in the $t - \text{epoch}$ over Bob's coins.

Let $U_t^{r_A} \subseteq \{0, 1\}^m$ be the set of useful memory states.

▷ **Claim 13.** Let x be the memory state after turn k_t (breakpoint t). For every randomness r_A and history h^{t-1} , if $m < \delta \cdot 2n$, then $\Pr_{R_B}[X \in U_t^{r_A} | R_A = r_A, H^{t-1} = h^{t-1}] \geq 1 - \text{negl}(n)$.

Proof. By Claim 4:

$$\Pr[x \in U_t^{r_A} | R_A = r_A, H^{t-1} = h^{t-1}] \geq 1 - (2^m - 1) \cdot \beta^{2n} > 1 - (2^\delta \beta)^{2n} + \beta^{2n}$$

which is $1 - \text{negl}(n)$ ◁

20:10 Mirror Games Against an Open Book Player

Recall that the motivation to define a useful memory state is to have an upper bound on the conditional probability of set of used numbers during (t) – epoch over Bob’s coins. In our case, $\Pr[S^t = s | X^t = x, H^{t-1} = h^{t-1}, R^A = r_A, x \in U_t^{r_A}]$ is bounded from above by a^{2n} , for some constant a that is smaller than one. If x is useful we get,

$$\begin{aligned} & \Pr[S^t = s | X^t = x, H^{t-1} = h^{t-1}, R_A = r_A] \\ & \leq \frac{\Pr[S^t = s | R_A = r_A, H^{t-1} = h^{t-1}]}{\Pr[X^t = x | R_A = r_A, H^{t-1} = h^{t-1}]} \\ & \leq \left(\frac{\alpha}{\beta}\right)^{2n} \end{aligned}$$

Thus Claim 10 holds in this case as well. Namely,

▷ **Claim 14.** For any assignment r_A . If x is useful, then Bob can find a subset $W_x^t \subseteq \mathcal{F}_x^t$ and a partition P_x^t of W_x^t into pairs s.t:

1. $\Pr[S^t \in W_x^t | X^t = x, x \in U_t^{r_A}, H^{t-1} = h^{t-1}] > 1 - \text{negl}(n)$
2. For every pair $(s_1, s_2) \in P_x^t$, $|s_1 \cup s_2|$ is even.
3. For every pair $(s_1, s_2) \in P_x^t$:

$$\frac{2n}{2n+1} \leq \frac{\Pr[S^t = s_1 | X^t = x, x \in U_t^{r_A}, H^{t-1} = h^{t-1}]}{\Pr[S^t = s_2 | X^t = x, x \in U_t^{r_A}, H^{t-1} = h^{t-1}]} \leq \frac{2n+1}{2n}$$

Claim 14 gives us the appropriate definition of similar probability for two subsets. Therefore, in every breakpoint t and for a given set s of values picked during t – epoch, Bob choose the pair (s_1^t, s_2^t) s.t. $s \in (s_1^t, s_2^t)$ and $(s_1^t, s_2^t) \in P_x^t$. This pair is then added to C_{t-1} to get C_t .

► **Theorem 15.** *If Alice has no secrets, and memory of size of at most $\delta \cdot 2n$ bits, then there exists a strategy for Bob such that against it Alice loses with probability of at least*

$$1 - \left(\frac{n+1}{2n+1}\right)^{c/2} - \text{negl}(n)$$

Proof. If memory state x^t is useful and $S^t \in W_x^t$, then Bob can find a pair $(s_1^t, s_2^t) \in P_x^t$ s.t. $S^t \in (s_1^t, s_2^t)$, $|s_1^t \cup s_2^t|$ is even and

$$\frac{2n}{2n+1} \leq \frac{\Pr[s_1^t | x, H^{t-1} = h^{t-1}]}{\Pr[s_2^t | x, H^{t-1} = h^{t-1}]} \leq \frac{2n+1}{2n}.$$

From breakpoint t and on, Bob chooses a number that has not been used and not in C_t . Moreover, at every round in epoch $1 \leq t' \leq c/2$ there are at least n numbers that have not been used and are not in $C_{t'}$. Thus Bob has probability of picking a specific number is at most $1/n$.

As in Observation 11, for every $1 \leq t \leq c/2$, the first one to pick a number from $s_1^t \cup s_2^t$ after breakpoint t is Alice, since Bob avoids picking from these sets until Alice does so (and by parity she will be the first to do so). Call the element chosen a_t and the number of turn q_t (note that the q_t ’s are not necessarily increasing in t ; furthermore, q_t might be smaller than $n/2$). Therefore, as in Theorem 12, Eq. 3 and applying Claim 14 we get that,

$$\begin{aligned} & \Pr[a_t \in S^t | X^t \in U_t, S^t \in \{s_1^t, s_2^t\}, (s_1^t, s_2^t) \in P_x^t, H^{t-1} = h^{t-1}, R_A = r_A] \\ & \geq \min_{s' \in \{s_1^t, s_2^t\}} \Pr[S^t = s' | X^t \in U_t, S^t \in \{s_1, s_2\}, (s_1, s_2) \in P_x^t, H^{t-1} = h^{t-1}, R_A = r_A] \\ & \geq \frac{2n}{2n+1} \cdot \frac{1}{2} \end{aligned} \tag{5}$$

In turn q_t Alice needs to determine whether $S^t = s_1^t$ or $S^t = s_2^t$. We claim that even if Alice already guessed (before turn q_t) the values of $S_{j_1}, \dots, S_{j_\ell}$ for breakpoints j_1, \dots, j_ℓ different from t , she still has the same probability of guessing S^t correctly as in Eq. 5. The reason is that s_1^t and s_2^t have similar conditional probability given all of the history until breakpoint $t - 1$. Therefore knowledge of those guesses does not give her any advantage at turn q_t .

By Claim 13 the memory state x^t in breakpoint t is useful with a probability of at least $1 - \text{negl}(n)$. By Claim 14 $S^t \in W_x^t$ with probability of at least $1 - \text{negl}(n)$. Overall we get that for every turn q_t Alice picks a number that has already been played with probability at least

$$\frac{2n}{2n+1} \cdot \frac{1}{2} - \text{negl}(n)$$

In order not to repeat the same number twice and lose the game, Alice needs to guess the right set for S^t for every $t = 1, 2, \dots, \frac{c}{2}$ (in some order). But whenever she makes such a guess, her probability of succeeding given that she has succeeded so far is roughly $(n+1)/(2n)$. Therefore the probability she succeeds in all of the cases is at most

$$\left(\frac{n+1}{2n+1}\right)^{c/2} + \text{negl}(n) \quad \blacktriangleleft$$

► **Corollary 16.** *If Alice has no secrets, and memory of size at most $\frac{n}{4c}$ bits (for $c \geq 2$), then there exists a strategy for Bob that for every $\epsilon > 0$ and large enough n , Alice loses with probability at least*

$$1 - \left(\frac{1}{2} + \epsilon\right)^{c/2}.$$

Proof. The only requirement on δ in Theorem 15 is that $2^\delta \beta < 1$. Therefore we can choose δ s.t.:

$$\delta \in \left(-\frac{3}{4} \log \beta, -\log \beta\right).$$

The only requirement on $\beta \in (0, 1)$ is that $\beta > \alpha = (\frac{1}{c})^{1/4c}$. By choosing β s.t.: $\log \beta \in (\log \alpha, \frac{2}{3} \log \alpha)$ we get that $\delta > -\frac{3}{4} \log \beta > -\frac{3}{4} \cdot \frac{2}{3} \log \alpha = -\frac{1}{2} \log \alpha$. Therefore

$$\delta > -\frac{1}{2} \log \left(\left(\frac{1}{c}\right)^{1/4c}\right) = \frac{1}{8c} \log c > \frac{1}{8c}.$$

By Theorem 15 the corollary follows. ◀

5 Further Work

Note that the strategy for Bob requires exponential computation. The work suggests the question of whether Bob's strategy can be made to be computationally efficient. In particular, a possible direction for getting an efficient strategy for Bob is to simulate Alice on various future selections, with the hope that this reveals which set and matching she is actually storing in her memory and therefore yield a move to play that will make her lose.

Another question is whether it is possible to have a closed book Alice with little memory and only a few bits of long term randomness where even an all powerful Bob will not be able to beat her. Recall that under computational limitations on Bob this is possible, as pointed out by Menuhin and Naor [6].

Is there an extension of Berlekamp's Theorem that will tell us that given enough subsets not only are there two whose union is of even cardinality but there are c sets such that the union of any c' of them is of even cardinality for all $c' \leq c$. This type of result would have made the amplification much simpler.

What happens if both Alice and Bob have limited memory? Is there some sort of stable solution where neither party wants to deviate from a scripted solution (assuming they prefer winning over drawing and that over losing)?

Finally, what is the precise bound on the the memory Alice actually needs in order to survive this ordeal. We can think of a strategy that requires around n bits. Is this tight? Also is there a better amplification where you get that a $o(n)$ memory Alice has a probability of winning that is exponentially small in n .

Adversarial streams and sampling. Recently there has been a lot of exciting research on streaming in adversarial environments. A streaming algorithm is called adversarially robust if its performance holds even when the elements in the stream are chosen adaptively and in an adversarial manner after seeing the current approximation or sample. The question is whether tasks that can be done with low memory against a stream that does not change as a function of the seeing the current state can be done with low memory even if the status is public. For instance, on the positive side, Ben-Eliezer, Jayaram, Woodruff and Yogev [1] showed general transformations for a family of tasks, for turning a streaming algorithm to be adversarially robust (with some overhead). On the other hand, Kaplan, Mansour, Nissim and Stemmer [5] showed a problem that requires polylogarithmic amount of memory in the static case but any adversarially robust algorithm for it requires a much larger memory.

The general question we ask is under what circumstance can we argue that a game where a player has no deterministic low memory strategy does not have an open book randomized one as well.

References

- 1 Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 63–80. ACM, 2020. doi:10.1145/3375395.3387658.
- 2 Uriel Feige. A randomized strategy in the mirror game. *arXiv preprint arXiv:1901.07809*, 2019.
- 3 Jacob Fox. Lecture notes for applications of linear algebra, mat 307, combinatorics. URL: <https://math.mit.edu/~fox/MAT307-lecture15.pdf>.
- 4 Sumegha Garg and Jon Schneider. The space complexity of mirror games. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 36:1–36:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.36.
- 5 Haim Kaplan, Yishay Mansour, Kobbi Nissim, and Uri Stemmer. Separating adaptive streaming from oblivious streaming using the bounded storage model. In *Advances in Cryptology - CRYPTO 2021, August 16-20, 2021, Proceedings, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 94–121. Springer, 2021. doi:10.1007/978-3-030-84252-9_4.
- 6 Boaz Menuhin and Moni Naor. Keep that card in mind: Card guessing with limited memory. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 107:1–107:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.107.

Fun with FUN

Fabien Mathieu ✉

Swapcard, Paris, France

Sébastien Tixeuil¹ ✉

Sorbonne Université, CNRS, LIP6, Paris, France

Abstract

The notions of *scientific community* and *research field* are central elements for researchers and the articles they publish. We propose to explore the evolution of the FUN conference community since its creation from the articles listed in DBLP, authors, program committees, and advertised themes, by means of a novel symmetric embedding, and carefully crafted software tools. Our results make it possible on the one hand to better understand the evolution of the community, and on the other hand to easily integrate new themes or researchers during future editions.

2012 ACM Subject Classification Information systems → Similarity measures; Information systems → Information extraction

Keywords and phrases Natural Language Processing, Relevance Propagation, Bibliometry, Community, Scientific Fields

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.21

Supplementary Material *Software (Source Code)*: https://github.com/balouf/conference_analysis/tree/main/FUN; archived at `swh:1:dir:8b96e766c426c12f0b9a7901b52230bc09d04a8b`

Acknowledgements This work was done at LINCS (<https://www.lincs.fr/>).

1 Introduction

Understanding the progress of science by studying how new scientific knowledge is created is an important societal challenge. In particular, there are many studies regarding how scientific knowledge spreads through scientific literature. For example, the field of bibliometrics is concerned with measuring the properties of the research corpus, and leads to measures of importance, based on notions such as the number of citations of an article, the impact factor of a journal, and an author's *h*-index. Furthermore, the social aspects associated with scientific research, such as the *sociology of scientific knowledge*, including the social structures and processes of scientific activity, as well as its political aspects, have also been the subject of studies [6]. The reader interested in such topics can benefit from the recent textbook by Wang and Barabási [12] and references therein.

In this article, we are interested in the analysis of scientific communities, from both a spatial and temporal point of view. The spatial (that is, community structure and/or habits) dimension is often assessed via the researchers' co-publications graph [7] to obtain scientific communities through clustering, but also thematically, for example by examining the vocabulary used in the main text of published articles [3]. The temporal dimension accounts for the evolution of the featured aspects measured over time. Most often, the temporal dimension concentrates on research impact or on evolution of collaborating relations [12].

Our focus in this paper is orthogonal. We focus on a known scientific community (here, the community involved in the many editions of the FUN conference), and aim to assess the evolution of its researchers (featured as conference authors or as PC members) and its associated themes (as described in the calls for papers, but also those associated to the PC

¹ Corresponding author.



members and authors of various editions) over the lifespan of the conference. This simple motivation raises several intriguing questions: how to map a researcher to her research themes? how to map research themes to researchers? how to represent the evolution of mappings over time?

Our contribution, for which the source code of the implementation is public, is threefold:

1. we present a new symmetric embedding augmented with a random walk mechanism to obtain a powerful cross mapping mechanism;
2. we design a methodology to collect and iterate modifications on data from various sources about a given conference or journal in IT to obtain spatio-temporal information about the venue, using the previously defined embedding;
3. we collect data for the FUN conference, apply our methodology, and describe our findings about the evolution of the conference since its creation.

We argue that our methodology is not only valuable to observe the past of a scientific community, but also to envision its future (*e.g.* by strategically choosing future PC member with respect to promoted themes).

The rest of the paper is organized as follows. Section 2 describes our new embedding. Section 3 presents our methodology to apply the embedding to a scientific venue, using the FUN conferences as running example. Section 4 describes the actual outputs of our methodology applied to FUN. Finally, Section 5 provides some concluding remarks.

2 Embedding documents and words, and vice versa

Searching for documents and extracting relevant information out of them is a task everyone faces on a regular basis. Common examples include looking for a relevant Internet page, digging a crucial e-mail from thousands of unread messages, or finding out relevant papers for a given topic. An effective search typically relies on a precise and well-organized search engine. The majority of current search engine techniques combine a keyword search with structural information (ontologies, relationships between elements) to order the documents in a corpus by relevance.

To increase the search performance, a standard approach consists in *embedding* documents or keywords, i.e. representing them by vectors in some space. A popular such example is *Term Frequency, Inverse Document Frequency* (TF-IDF) (further described in Section 2.1). In this section, we propose a novel approach that extends TF-IDF to enhance the quality and the flexibility of the embedding. This new approach will be instrumental in the analysis of the FUN community. This section is organized as follows: Section 2.1 presents the TF-IDF embedding; Section 2.2 introduces our symmetrized version called *Term Frequency - Inverse Document & Term Frequency* (TF-IDTF); Section 2.3 adds a random-walk approach on top of TF-IDTF to refine the quality of the embedding.

2.1 Texts are made of words

TF-IDF is a metric commonly used in language processing to estimate the importance of a word based on its frequency in a document and its rarity in a corpus. The metric postulates that a document can be represented by the set of the words it contains and that a rare word in the corpus is more important than a common one.

In details, consider a corpus X made of n documents. These documents are made of words. Let Y be the set of the m (unique) words that are present in the corpus. We can build a simple bipartite graph $G = (X \cup Y, E)$ of components X and Y by creating an (undirected)

edge between $x \in X$ and $y \in Y$ if, and only if, x contains the word y . If $x.y$ denotes the number of occurrences of y in x (the frequency of y in x), TF-IDF attributes the following weight to the edge (x, y) :

$$w(x, y) = x.y \log \left(\frac{n}{d(y)} \right), \text{ where } d(\cdot) \text{ is the degree in } G. \quad (1)$$

Using Equation (1), one can associate to each document x the m -dimensional vector \vec{x} on Y defined by $\vec{x}_y = w(x, y)$. This is called an *embedding* of X into Y . This embedding is usually *sparse*: if the corpus is large and various, a typical document contains a fraction of the available words, so most components of its embedding are null.

Embeddings have many uses, including the possibility to measure the *cosine similarity* between two documents. If x_1 and x_2 are two documents of X , their cosine similarity is:

$$\text{sim}(x_1, x_2) = \frac{\vec{x}_1 \cdot \vec{x}_2}{\|\vec{x}_1\|_2 \cdot \|\vec{x}_2\|_2}. \quad (2)$$

Despite its simplicity and the loss of meaning induced by reducing a text to a *bag of words*, the embedding induced by Equation (1) is surprisingly effective in practice. Several explanations have been proposed to explain the success of TF-IDF. One of the most elegant ones comes from information theory [1]: assume that I need to find one specific document among a large set. I know that the document contains a specific word, so I can restrict the search to the documents that contain it. If only the desired document contains the word, the search has been made trivial by the knowledge of the word. If all documents contain the word, the knowledge is useless. In general, if one expresses the quantity of information, in the sense of information theory, brought by the knowledge of the presence of the word, we get the *Inverse Document Frequency* (IDF) of the word (the logarithmic term in Equation (1)).

In other words, $w(x, y)$ can be seen as the product of the strength of the relationship between x and y (estimated by $x.y$) by the quantity of information given by y (expressed by the inverse document frequency).

2.2 Words are defined by texts

The graph G we consider in Section 2.1 represents an inclusion relationship, which is asymmetric. However, the graph itself is a simple graph, and contains no information to distinguish the *documents* part X from the *words* part Y . For example, if we represent each document x of X by a unique word y' (not necessarily from Y), and each word y of Y by a document x' made by assembling the representatives of all documents that contain y , we end up with the original graph G except that X and Y are reversed. Based on this symmetry between X and Y , it is natural to investigate what happens if one switches them in Equation (1).

This would introduce $\log \left(\frac{m}{d(x)} \right)$ (the logarithm of the ratio between the total number of unique words in X and the total number of unique words in x).

This *Inverse Term Frequency* (ITF) can be interpreted as a dual version of IDF: while IDF relies on the assumption that a document is defined by the words it contains, and favors scarcity (of words), ITF relies on the assumption that a word is defined by its context (the documents that contain it), and favors conciseness (of documents). So, if a document is concise and contains only a few words, knowing that a word belongs to that document gives a lot of information for separating that specific word from the others. Conversely, knowing that a word belongs to a lengthy document with many distinct words gives little information about the meaning of that specific word.

Based on this observation, we introduce a refinement of TF-IDF that we call TF-IDTF. TF-IDTF attributes the following weight to the edge (x, y) :

$$w(x, y) = (1 + \log(x.y)) \log\left(\frac{1+n}{1+d(y)}\right) \log\left(\frac{1+m}{1+d(x)}\right) \text{ if } y \in x, 0 \text{ otherwise.} \quad (3)$$

Compared to Equation (1), the ITF is added to the formula, to convey the information that document x brings about word y . Equation (3) also introduces the following classical modifications (cf. for example the TF-IDF implementation in the scikit-learn library [10]):

- $x.y$ is logarithmically smoothed to limit possible over-representation when a word is widely used in a document.
- A shift of one unit is introduced in the expressions of the inverse frequencies. It corresponds to the addition of a fictitious document containing all the words, and of a fictitious word appearing in all the documents. These additions smooth the weights.

The introduction of the ITF does not drastically change the resulting embedding of documents. In fact, the embedding of one document is just scaled by its ITF, and in particular the cosine similarity is unaffected. The real change is that the new weights provides a new dual embedding, not on documents but on *words*: one can associate to each word y the n -dimensional vector \vec{y} on X defined by $\vec{y}_x = w(x, y)$. With this embedding, two words are considered close if they are often co-occurrent in documents, with more importance given to co-occurrence in short documents. This allows for example to identify words that belong to the same lexical field. Just like the IDF weighting prevents frequent words from polluting the embedding of documents, the ITF weighting prevents lengthy documents from polluting the embedding of words.

The two embeddings (document and word) can be unified by considering the $n \times m$ matrix W defined by $W_{x,y} = w(x, y)$: each line (resp. each column) of W represents the embedding of a document (resp. a word) in Y (resp. in X).

Now that we have the tools to compare two documents or two words, we will see how to compare words and documents.

2.3 The friend of my friend is my friend

One major caveat of embeddings like TF-IDF (or TF-IDTF) is that they are blind to synonyms: if one document on graphs uses the term *node* and another the term *vertex*, these two words will make the embeddings of the documents less similar although it should be the opposite. Another issue is the impossibility to directly compare a word and a document beyond the term frequency metric.

To address these issues, we propose to refine the embedding of documents and words by considering a random walk on G . The idea is that through our symmetric document embedding, any vector on X , seen as a weighted set of documents, can be turned into a vector on Y . Reciprocally, any vector on Y , seen as a weighted set of words, can be turned into a vector on X by the word embedding. In addition to allowing translations between embeddings, this process may enable *idea associations*: if one document on graphs uses the terms *node*, *edge*, and *graph* and another the terms *vertex*, *edge* and *graph*, going back and forth between words and documents will uncover the similarity between *node* and *vertex* as words that belong to documents that contain the words *edge* and *graph*. With this approach, it is possible to find out the similarity between two documents that use the same lexical field, even if they have no word in common (so, their similarity is 0 according to Equation (2)). This mitigates (without fully nullifying) the impact of synonyms.

This idea of extracting information from graph exploration is highly reminiscent of the *random surfer* model used in *PageRank*, the algorithm behind the Google search engine [2]. In fact, PageRank has already been used in language processing to produce extractive summaries of documents [9]. Note that in the case of extractive summaries, the walk transitions from documents to words are often weighted according to IDF to avoid pollution of frequent words. However, transitions from words to documents are usually chosen uniformly, which can induce pollution from lengthy documents. As we do not want the pure length of a document to be a competitive advantage in the random walk, we propose to use TF-IDTF to handle both types of transitions (from documents to words, and from words to documents).

For the actual random walk computation, we propose to use D-Iteration [5], a PageRank variation adapted to the exploration of the neighborhood of a part of a graph. Intuitively, D-iteration consists in diffusing a finite quantity of evanescent fluid on the graph vertices from an initial distribution (like a word or a document) and measuring the quantity of fluid flowing through the vertices.

The D-Iteration algorithm takes as parameters a stochastic matrix A (which represents a random walk on a graph), a vector Z that represents the start of the walk, and an attenuation coefficient $\alpha \in (0, 1)$. In our case, the matrix A derives from W , Z derives from the words or documents one wants to analyze, and α is a parameter to be chosen carefully.

In detail, A is a matrix of size $(n + m) \times (n + m)$ defined by $A = \begin{pmatrix} 0_{n,n} & S(W) \\ S(W^t) & 0_{m,m} \end{pmatrix}$, where $S(M)$ denotes the stochastic renormalization of M , which consists of dividing each non-zero line of a positive matrix M by the sum of its elements. A defines a random walk on G where each edge is chosen proportionally to its TF-IDTF weight.

If z represents a set of words (or documents) one wants to analyze, possibly weighted, it can be represented by a vector Z on $X + Y$ whose components are equal to 1 (or their weight) if they correspond to an element of z , and 0 otherwise.

The D-Iteration algorithm associates to z a vector $P(z)$ defined by :

$$P(z) = \sum_{k \geq 1} \alpha^k Z A^k. \quad (4)$$

Each term of the sum corresponds to a random walk of length k from Z , weighted by α^k . This implies that the average length of the random steps that $P(z)$ aggregates is $\frac{1}{1-\alpha}$. In other words, the attenuation coefficient α controls the span of the graph exploration, which is subject to a trade-off: when α is close to 0, the walk lengths are close to 1, and $P(z)$ converges to a straight TF-IDTF embedding of z . Conversely, when α is close to 1, the walk lengths are long. This enables idea associations, but also “blurs” the result: the underlying Markov chain, while not being ergodic (the graph is bipartite, hence periodic with period 2), is irreducible on its connected components, which means that all information but the starting bipartite component are forgotten on long walks.

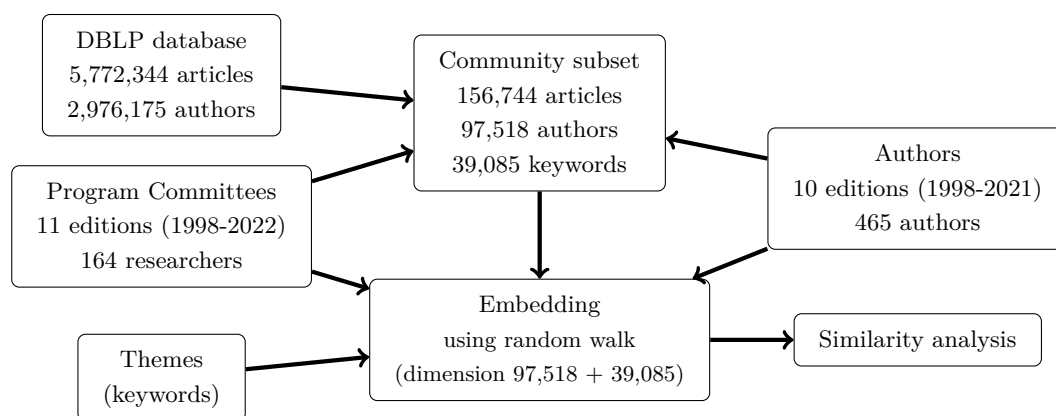
The vector $P(z)$ has multiple uses. One possibility is to look at the components with the greatest values to get the documents and words that are the closest to z . Another possibility is to treat $P(z)$ as an embedding of z in the space $X + Y$. Using a random walk of finite length to represent a weighted subset of vertices in a graph is not a new approach [4, 11]. However, as stated above, the bipartite property of the graph creates a natural asymmetry depending on whether the starting point of the walk is a document or a word. For example, if we start from a document, walks of even length always return documents, and walks of odd length always return words. This induces a natural distortion that makes it difficult to compare a document with a word: by design, the embedding of a word (resp. a document)

obtained with Equation (4) systematically gives more weights to documents (resp. words). To mitigate this effect, we renormalize the components of $P(z)$ on documents and words so that the total weight on words is always equal to the total weight on documents.

3 Embedding FUN

Section 2 exposed a generic way to embed arbitrary documents and their content. In this section, we expose how to use and adapt those general mathematical tools to analyze the FUN community. The originality of our approach lies in the fact that instead of trying to compute the supposed importance of a community or a theme compared to another, we want to characterize the links of similarity that connect them. For this purpose, we employ the methodology summarized in Figure 1:

- From the *Digital Bibliography & Library Project* (DBLP) database, we extract a community subset centered around FUN;
- From the community subset, we build a graph between researchers and words that allows to represent them in the same space, and thus to compare them;
- By calculating the cosine similarity between the vectors, we investigate the links between authors, program committees, and themes.



■ **Figure 1** Our methodology in a nutshell.

The rest of the section is organized as follows: Section 3.1 presents DBLP and the actual dataset that is used; Section 3.2 describes the use of authors instead of words to describe a scientific paper and the extraction of a community graph; Section 3.3 explains how to merge author and word descriptions altogether; lastly, Section 3.4 presents the implementation of our methodology. The actual analysis is presented in Section 4.

3.1 Judging a book by its cover

The DBLP project indexes English-language publications in the IT field. The database, publicly available at the address <https://dblp.uni-trier.de/xml/dblp.xml.gz>, contains the majority of bibliographical references in IT. In particular, for each bibliographical reference, the database contains its title and authors. At the time this article was written, the database referenced 5,772,344 articles written by 2,976,175 unique authors.

Sadly, DBLP does not provide in its downloadable version any information on the content of the articles beyond their title. In particular, the paper abstract, introduction, and keywords are missing. At first glance, it seems highly insufficient to carry out a relevant analysis,

which is true if one wants to analyze one single article. Luckily, our approach uses articles by batches, like all articles written by a researcher, so we can hope that quantity will compensate the noisy semantic quality conveyed by titles to identify highlighting trends.

However, for the particular case of the FUN conference, the hypothesis that a paper title conveys information is hindered. Indeed, FUN articles are typically written in a fun and amusing way, and it reflects on their titles. What themes can we infer from titles like *Kings, Name Days, Lazy Servants and Magic* or *Urban Hitchhiking*? For this reason, if one wants to study the FUN community, it is probably better to consider the people of that community instead of the titles of the articles they publish in FUN (but still consider the titles of the papers they write outside FUN).

3.2 Articles are made of authors, and vice versa

Traditionally, the elements of an embedding are called *features*, as they yield a description of a document. Obviously, the authors of an article give information on that article. For example, they can hint at the topic of the article. If one uses authors instead of words as features of the articles, can we re-interpret TF-IDF?

- We assign a term frequency of 1 for all authors, which means that we assume that all authors have the same importance for a considered article. This is of course debatable, but given the limited amount of information available, this is the only sensible choice, and we can hope that this is true *in average*.
- The IDF term means that for a given article, more weight is given to authors that have few publications. This reduces the natural bias towards prolific authors. We emphasize that this is no indication of the intrinsic value of people, but just a measure of how much an article can be characterized by the presence of a given author. A starting researcher has a big weight as it narrows down the corpus of articles to a small subset. Paul Erdős (about 1,500 publications) has low weight as the corpus reduction is smaller. Didier Raoult has 4 articles referenced in DBLP so he has a big weight on the corpus of IT publications; on the other hand with more than 3,100 publications referenced in PubMed², he has a very low weight on the corpus of medical publications.
- The ITF term means that for a given author, more weight is given to the articles that have few authors. Observe that this reduces the natural bias towards articles with many authors. The interpretation, in terms of information theory, is that with less co-authors, or no co-author at all, an article is more representative of the production of a researcher. Conversely, it is likely that an article with dozens of co-authors provides little information on the profile of one single author.

All things considered, we can see that all the reasoning behind the introduction of TF-IDF also makes sense if one considers authors instead of words.

Based on this observation, we carried out a filtering of the complete database centered on the FUN community. For this, we collected all the program committees and authors of the different editions, and manually disambiguated each of them with respect to their DBLP entry (homonyms or near homonyms have specific entries in the database). For each edition, we computed $P(z)$ using a default value $\alpha = 1/2$ (cf Equation (4))³, and we aggregated the

² <https://pubmed.ncbi.nlm.nih.gov/>

³ The value $\alpha = 1/2$ is chosen empirically. It is based on the quality of the results roughly assessed by the authors of this paper. To give a sense of comparison, extractive summaries typically use smaller values for α , usually less than 0.1 (see e.g. the parameter $1 - d$ used by Otterbacher et al. [9]), to remain very close to the starting point, while the original PageRank algorithm uses the empirical value $\alpha = 0.85$ to

most valued articles according to each embedding. This allowed us to select 156,744 articles representing the FUN community and its (large) neighborhood. The goal of this reduction was twofold: first, improving computation time by working on a smaller dataset; second, improving the corpus quality with respect to the FUN community by removing irrelevant publications from unrelated fields.

3.3 The words of my papers are my words

From the articles of the FUN community, we can extract two bipartite graphs with stochastic transitions: one between articles and words, one between articles and researchers. As said before, we have little interest in the articles themselves, which convey little information, so it is natural to combine the two graphs into a new bipartite graph (with stochastic transitions) between researchers and words. The new graph links each researcher to all words she used in her articles, with more weights on the rare words that appear on articles with few co-authors. Conversely, each word is linked to all researchers that have used it, with more weight on the non-prolific researchers that prefer small titles. This graph will be used to perform the embedding for our analysis.

We did not take into account the years of publication when constructing the graphs. In particular, this means that each author is analyzed throughout their entire career, even if their own research interests have evolved.

3.4 Our implementation

To perform the actual analysis of the FUN community, we used a Python package called *Gismo* (*Generic Information Search with a Mind of its Own*) [8]. *Gismo* performs most of the “heavy lifting”, from the DBLP interface to the building of embeddings, and allows to focus on the FUN part. The code for the FUN part is available at the following address: https://github.com/balouf/conference_analysis/tree/main/FUN.

The repository mainly contains two *Jupyter Notebook* files, one dedicated to the creation of the community subset and associated embedding, the other to the actual similarity analysis. Some technical details (word pre-processing, incorporation of multi-words like *stochastic geometry*, ...) are left out of this paper but are available in the notebooks.

4 Results: a brief history of FUN

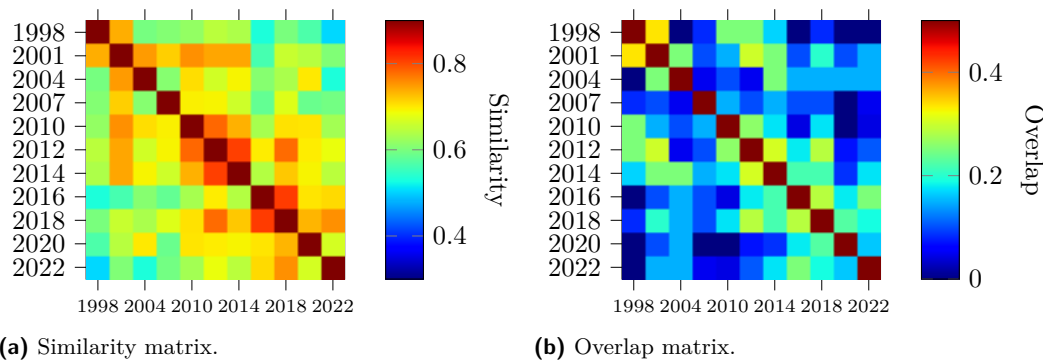
We propose to conduct our analysis in two parts. Section 4.1 studies the evolution of the PC members and authors over the different editions, while Section 4.2 focuses on the dynamics of FUN themes as put forward in the call for papers.

4.1 It’s a small world, after all

Figure 2a presents the matrix of similarities of scientific communities induced by the PC members along the different editions of FUN since its creation. A warm color indicates strong similarity, while a cool color indicates weak similarity. The following observations can be made. First, the heatmap is generally warm, meaning that the scientific community induced by the PC members remains similar along time. Second, we can distinguish pairs of FUN

give more room to exploration.

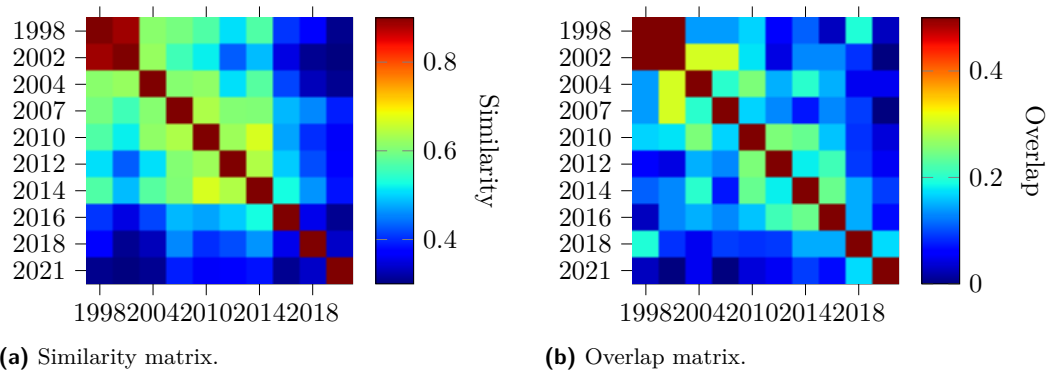
editions that are very strongly similar: 1998-2001, 2001-2004, 2010-2012, 2012-2014, 2016-2018, and 2018-2020. By contrast, the most dissimilar consecutive editions are 2004-2007. Two editions are remarkable, 2001 is highly similar to the next five editions (until 2014), while 2016 is quite dissimilar to early editions. Some of those observations can be explained looking at Figure 2b that describes the overlap matrix of PC members along editions (that is, the proportion of PC members common between two editions). We observe again that pairs of consecutive years have (relatively) high overlap: 1998-2001, 2001-2004, 2010-2012, 2012-2014, 2016-2018, and 2018-2020. Also, the pair 2004-2007 has the less overlap in PC members. However, the particular edition of 2001 show very little overlap with the 2010 edition, despite being similar from a scientific community point of view.



■ **Figure 2** Evolution of the FUN community through its PC members along FUN conference editions.

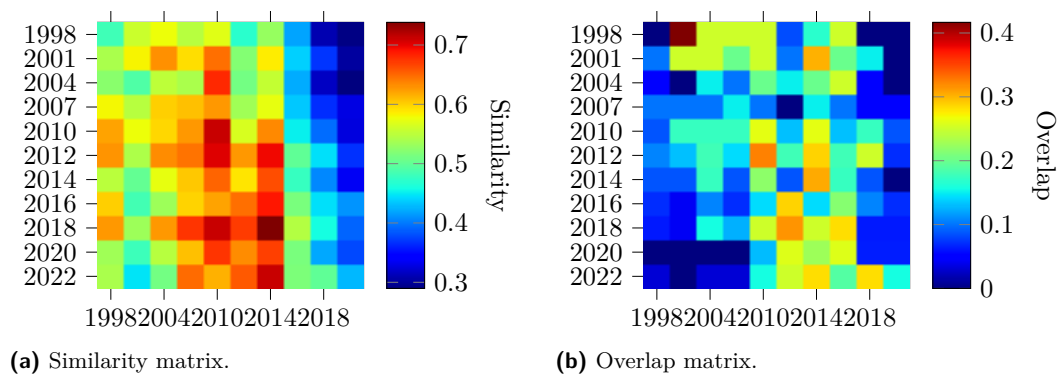
Arguably, a scientific venue such as FUN can also be analyzed through its authors (that is, the scholars that publish papers appearing in the conference). Figure 3a presents the matrix of similarities of scientific communities induced by the authors along the different past editions of FUN. There, the years correspond to the years of publication of the papers, so the latest available data at the time of writing is from FUN 2020, published in 2021. We observe that the heatmap is much cooler overall than the one generated from the PC members. A possible explanation of the lesser intensity of similarity between PC members induced communities and author induced communities could come from the average pool size. Program Committees are smaller than the authors that publish in a given edition, and smaller groups tend to have more consistency. We also observe some clusters of consecutive editions with similar authors: 1998-2002 mainly, and to some extent 2004-2007-2010 and 2007-2010-2012-2014. However, editions become quickly dissimilar with other editions further away in time, implying that authors induced communities do not last. Again, those results are partly explained by Figure 3b that presents the overlap of FUN authors. It confirms that the turnover of authors is slightly higher than that of PC members, although some continuity is sometimes preserved: editions 1998-2002 have a huge overlap that fades afterwards, and occasionally we observe periods with significant overlap of authors, like 2002-2004-2007, 2010-2012-2014-2016, or 2018-2020.

At this point, one might wonder about explaining the evolution of a scientific community. Figure 4a presents the similarity of the community induced by the PC members (on the y axis) versus the community induced by the authors (on the x axis). One striking observation is that some author years (column-wise) are highly similar to many previous PC (2010, 2012, and 2014), while some others (2002 and 2018-2021) are less so (note that the coolest colors here are still quite similar in absolute value). Another interesting observation is that as lines



■ **Figure 3** Evolution of the FUN community through its authors along FUN conference past editions.

go down (*i.e.*, when the PC members evolve), the similarity gets higher, which means that the PC members are selected according to the authors of previous editions interests. This is confirmed by Figure 4b that presents the overlap between authors and PC members along the years. For example, the PC members of FUN 2022 are heavily selected from the authors of FUN 2012 to 2020, but less so from authors of the previous editions. As we go back in time for PC members selection, we also go back to the earlier editions of the conference to select them from the authors' pool. A few outliers are worth mentioning: the PC members of 1998, 2001 (the first two editions), and 2012 are longstanding contributors to the conference as authors (roughly a span of 20 years).

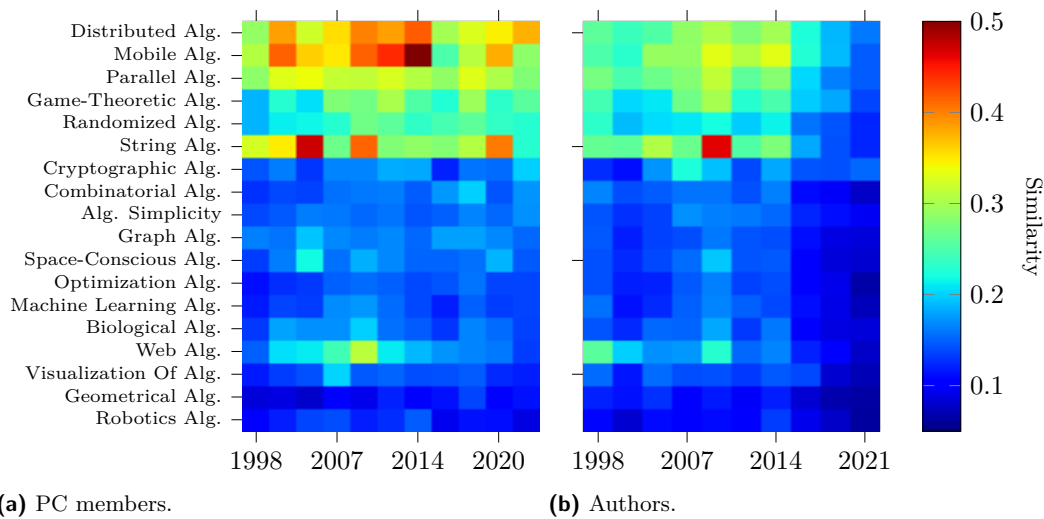


■ **Figure 4** Evolution of the equilibrium between PC members and authors along FUN conference editions.

4.2 Hot topics

To analyze the evolution of the FUN themes, we considered three editions: 1998, 2010, and 2022. For each edition, we compared the similarity between the advertised themes, the PC member of all editions, and the authors of all past editions. Results are displayed in Figures 5–7.

Figure 5 focuses on the 2022 edition. The main observation is that most of the similarity with the FUN community (PC members and authors alike) is concentrated in about one third of the themes, which perhaps show a lack of equilibrium between the community and the selected themes.



■ **Figure 5** Evolution of the themes presented in the call for papers of FUN 2022. Themes are ordered by similarity with the 2022 PC members.

Some outlier themes are worth mentioning. *Mobile algorithms* and *Distributed Algorithms* are rotating as themes highly similar with PC members since the second FUN edition, and although they seem to have peaked in 2014, they are still going strong. Also, *Game-theoretic algorithms* is gaining popularity along the years (still when comparing with PC members). *String algorithms* is intricate, as it goes in burst in some editions (2004, 2010, 2020 for PC members, only 2010 for authors) but not others. Then, *web algorithms* used to be well represented by both PC and author induced communities (with a peak in 2010) but seems to have periclitated.

Another outlier is the *Space-conscious algorithms* theme that exhibit a burst in some early editions of FUN (2004 for PC members, 2010 for authors), and mostly disappears afterwards.

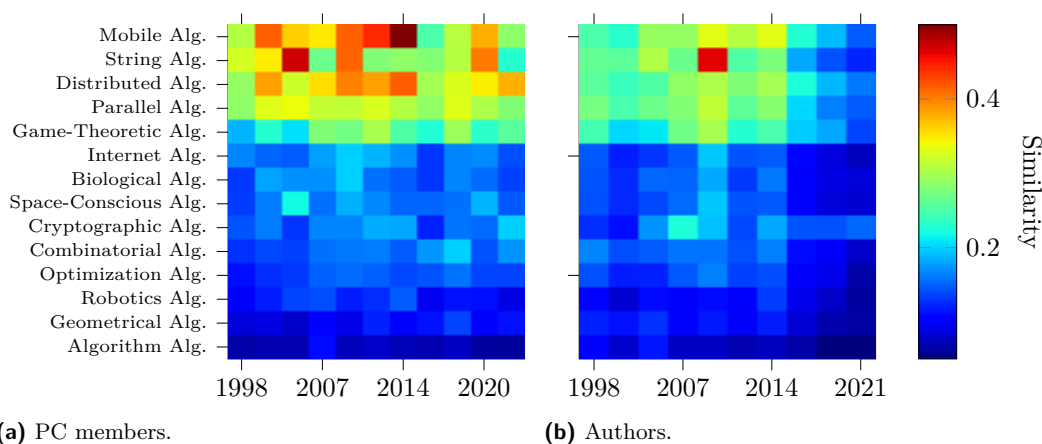
Another trend worth mentioning is that the similarity between themes and authors seems to decrease with time, possibly indicating that as time passes, compliance with the advertised themes is less required by the selection process.

Figure 6 focuses on the 2010 edition. The trend is globally the same as in Figure 5, which is not surprising as many themes are common to both editions. We still observe a concentration on five main themes (*Mobile algorithms*, *String algorithms*, *Distributed Algorithms*, *Parallel algorithms*, and *Game-theoretic algorithms*).

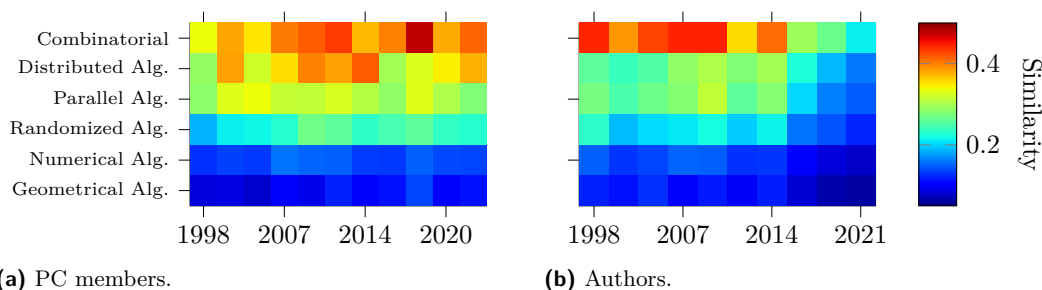
One interesting outlier is the missing one: “web algorithms”, which peaked in 2010, is not present in the advertised. This seems to indicate that when published papers themes are put forward too late (that is, once the momentum is gone), it may be difficult to attract good submissions on those topics later.

By contrast, in Figure 7, we observe that the first edition has two thirds of themes that are similar to the FUN communities induced by its PC members and authors. Arguably, the low number of proposed themes (6) could be an explanation.

However, in the FUN 1998 edition, the theme *Combinatorial* seems to be an outlier, as it is a single word (versus all other themes being pairs of words), which translated to increased similarity (for example, the pair *Combinatorial algorithms* is dissimilar throughout the years, as shown by Figures 5 and 6), hence the real proportion for FUN 1998 is half-half.



■ **Figure 6** Evolution of the themes presented in the call for papers of FUN 2010. Themes are ordered by similarity with the 2010 PC members.



■ **Figure 7** Evolution of the themes presented in the call for papers of FUN 1998. Themes are ordered by similarity with the 1998 PC members.

5 The future of FUN

We have shown how, from the only knowledge of the members of the program committees, the conference authors, the calls for contributions, and the DBLP database, it is possible to analyze the themes and communities of a conference and to observe their evolution.

Beyond the analytical aspect, our approach can also be used to assist in the development of a program committee, in particular to avoid a weak similarity between the scientific community induced by the program committee and the themes promoted, as seen in Section 4. The source code that accompanies this article (see Section 3.4) includes in particular tools to obtain suggestions from PC members relevant to (possibly new) themes, by setting a renewal rate (that is, picking a suitable proportion of PC members from previous ones).

The same tool can be used for more individual purposes: suppose a researcher wants to write a paper about a FUN theme for the next edition of the conference, but is a bit ignorant on some topic or technique that would be instrumental in carrying out the research. Our implementation can be used to suggest suitable collaborators for the task, possibly issued from the FUN community.

References

- 1 A. Aizawa. An information-theoretic perspective of TF-IDF measures. *Info. Proc. & Manag.*, 39(1):45–65, 2003. doi:10.1016/S0306-4573(02)00021-3.

- 2 S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998. doi:10.1016/S0169-7552(98)00110-X.
- 3 Graham Cormode, S. Muthukrishnan, and Jinyun Yan. Scienceography: The study of how science is written. In Evangelos Kranakis, Danny Krizanc, and Flaminia L. Luccio, editors, *Fun with Algorithms - 6th International Conference, FUN 2012, Venice, Italy, June 4-6, 2012. Proceedings*, volume 7288 of *Lecture Notes in Computer Science*, pages 379–391. Springer, 2012. doi:10.1007/978-3-642-30347-0_37.
- 4 Bruno Gaume and Fabien Mathieu. PageRank Induced Topology for Real-World Networks. working paper or preprint, May 2016. URL: <https://hal.archives-ouvertes.fr/hal-01322040>.
- 5 D. Hong, T. D. Huynh, and F. Mathieu. D-Iteration: diffusion approach for solving PageRank. *CoRR*, abs/1501.06350, 2015. arXiv:1501.06350.
- 6 John H. Marburger III, Julia I. Lane, Stephanie S. Shipp, and Kaye Husbands Fealing, editors. *The Science of Science Policy: A Handbook*. Stanford University Press, 2011. URL: <http://www.sup.org/books/title/?id=18746>.
- 7 Michael Kuhn and Roger Wattenhofer. The theoretic center of computer science. *SIGACT News*, 38(4):54–63, 2007. doi:10.1145/1345189.1345202.
- 8 Fabien Mathieu. Generic Information Search with a Mind of its Own (Gismo). URL: <https://gismo.readthedocs.io/en/latest/>.
- 9 J. Otterbacher, G. Erkan, and D. Radev. Using random walks for question-focused sentence retrieval. In *HLT/EMNLP*, pages 915–922, 2005. URL: <https://www.aclweb.org/anthology/H05-1115>.
- 10 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- 11 Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008. doi:10.1073/pnas.0706851105.
- 12 Dashun Wang and Albert-László Barabási. *The Science of Science*. Cambridge University Press, 2021. doi:10.1017/9781108610834.

All Your bases Are Belong to Us: Listing All Bases of a Matroid by Greedy Exchanges

Arturo Merino  

Department of Mathematics, TU Berlin, Germany

Torsten Mütze   

Department of Computer Science, University of Warwick, Coventry, UK

Department of Theoretical Computer Science and Mathematical Logic, Charles University, Prague, Czech Republic

Aaron Williams   

Department of Computer Science, Williams College, Williamstown, MA, UK

Abstract

You provide us with a matroid and an initial base. We say that a subset of the bases “belongs to us” if we can visit each one via a sequence of base exchanges starting from the initial base. It is well-known that “All your base are belong to us”. We refine this classic result by showing that it can be done by a simple greedy algorithm. For example, the spanning trees of a graph can be generated by edge exchanges using the following greedy rule: Minimize the larger label of an edge that enters or exits the current spanning tree and which creates a spanning tree that is new (i.e., hasn’t been visited already). Amazingly, this works for any graph, for any labeling of its edges, for any initial spanning tree, and regardless of how you choose the edge with the smaller label in each exchange. Furthermore, by maintaining a small amount of information, we can generate each successive spanning tree without storing the previous trees.

In general, for any matroid, we can greedily compute a listing of all its bases matroid such that consecutive bases differ by a base exchange. Our base exchange Gray codes apply a prefix-exchange on a prefix-minor of the matroid, and we can generate these orders using “history-free” iterative algorithms. More specifically, we store $O(m)$ bits of data, and use $O(m)$ time per base assuming $O(1)$ time independence and co-independence oracles.

Our work generalizes and extends a number of previous results. For example, the bases of the uniform matroid are combinations, and they belong to us using homogeneous transpositions via an Eades-McKay style order. Similarly, the spanning trees of fan graphs belong to us via face pivot Gray codes, which extends recent results of Cameron, Grubb, and Sawada [Pivot Gray Codes for the Spanning Trees of a Graph ft. the Fan, COCOON 2021].

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Combinatorics; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Matroids, base exchange, Gray codes, combinatorial generation, greedy algorithms, spanning trees

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.22

Supplementary Material *Software (Source Code)*: <https://gitlab.com/amerinof/greedy-matroids>
archived at `swh:1:dir:40f8b84af501e2f96ef57c47d492a12d2072b99f`

Funding *Arturo Merino*: German Science Foundation grant 413902284 and ANID Becas Chile 2019-72200522.

Torsten Mütze: German Science Foundation grant 413902284 and Czech Science Foundation grant GA 22-15272S.

Acknowledgements The authors would like to thank the anonymous referees who contributed significantly to the quality of this paper.



© Arturo Merino, Torsten Mütze, and Aaron Williams;
licensed under Creative Commons License CC-BY 4.0

11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 22; pp. 22:1–22:28

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Every graph theorist knows that a maximal spanning forest of a graph can be constructed greedily: Add an edge so long as it doesn't create a cycle with some previously added edges. Similarly, every linear algebraist knows that a base for the column space of a matrix can be constructed greedily: Add a column so long as it isn't linearly dependent with some previously added columns. In both cases, the approach works regardless of how the elements (i.e., edges or columns) are ordered. Or put another way, if there are multiple elements that can be added during the next step, then ties can be broken arbitrarily. This property motivated Whitney's [40] generalization to matroids, where bases can be constructed in a similar manner: Add an element so long as it doesn't create a circuit with some previously added elements. Again, this approach works regardless of how ties are broken. Furthermore, it can be used to generate any base.

This greedy algorithm is so powerful, it brings to mind the enemy CATS in the side-scrolling arcade shooter *Zero Wing* (1989) by the Japanese developer Toaplan [42]. In the year 2101, the alien overlord CATS breaks his peace treaty with the United Nations and attacks Earth. This backstory was expanded upon in the opening cutscene of the European port of *Zero Wing* to the SEGA Mega Drive in 1991, where CATS exclaims "ALL YOUR BASE ARE BELONG TO US". This poor translation [17] birthed one of the first widespread internet memes [41, 14]. Besides providing comic relief, the poor grammar can also cause confusion for game players. Is CATS saying that the player has one base, and all of that one base belongs to him? Or is CATS saying that the player has multiple bases, and all of them belong to him?¹ In this paper, we hope to create similar confusion with greedy algorithms and matroids, as shown by Figure 1.



Figure 1 An internet meme in (a), along with (b) classic and (c) new results involving greedily generating the bases of a matroid.

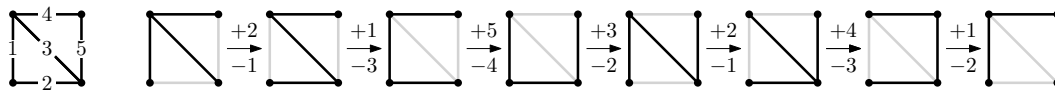
To fully appreciate our results, we recommend that the reader is familiar with the basic properties of spanning trees, as analogous concepts involving matroids will be introduced. For additional background reading on matroids, we suggest Oxley [23], while Mütze [21] provides a new survey on Gray codes. We also encourage the reader to (re)familiarize themselves with the *All Your Base Are Belong To Us* meme, including the video by Bad_CRC et al. that was originally posted as a Flash animation on Newgrounds in 2001 [2].

¹ The latter is correct according to fandom.com: “[CATS] breaks the treaty and takes over all of Japan’s space colonies” [8].

1.1 New results

1.1.1 Matroid bases including spanning trees

We prove that an exhaustive listing of all the bases of a matroid can be generated greedily. Specifically, if you provide us with a matroid and an initial base, then we can list the matroid's bases one at a time as follows: From the current base perform an exchange that creates a new base and minimizes the larger element involved in the exchange. By repeating this simple greedy rule, every base will be generated exactly once. This works for any matroid, for any ordering of its elements, for any initial base, and regardless of how ties are broken to choose the second element in the exchange. In the case of graphic matroids, this greedy rule translates to the following: From the current spanning tree, exchange one edge with another edge in such a way that a new spanning tree is generated and the larger edge in the pair is minimized. We refer to the resulting order as an *edge-exchange Gray code*, or simply an *exchange Gray code*, in reference to the eponymous minimal-change order known as the binary reflected Gray code (see Section 1.3). Figure 2 provides an illustration.



■ **Figure 2** Listing the spanning trees of our favorite graph on the left. The list is an exchange Gray code, i.e., successive spanning trees differ in adding one edge and removing another edge. For example, the first spanning tree is transformed into the second by adding edge 2 and removing edge 1. The list is constructed greedily by always minimizing the larger edge label involved in the exchange. Ties can be broken arbitrarily; here the smaller edge in the exchange is maximized.

1.1.2 Weighted matroids including maximum spanning trees

A *weighted matroid* is a matroid with some weight associated with each element. The weight of a subset of elements is the sum of the weights of the elements in the subset. Edmonds [7] showed that the standard greedy algorithm can be modified to build a maximum weight base: Consider the elements by monotonically decreasing weight and add an element so long as it doesn't create a circuit with some previously added elements. By instead sorting the elements in monotonically increasing order, the standard greedy algorithm will instead build a minimum weight base. These and similar ideas support the well-known algorithms by Kruskal and Prim for finding optimal spanning trees.

The optimization problem on a matroid $M = (E, \mathcal{I})$ with weights $w : E \rightarrow \mathbb{R}$ can also be modeled by an unweighted matroid $M' = (E, \mathcal{I}')$: If $S \subseteq E$ is a subset of a maximum weight base M , then $S \in \mathcal{I}'$. In particular, the bases of M' are precisely the maximum weight bases of M . For this reason, our greedy algorithms also allow generating the maximum weight bases in weighted matroids. Furthermore, by replacing the weight function w with $-w$ we are also able to generate *minimum* weight bases in weighted matroid.

1.2 Efficiency and meta-algorithms

Our greedy algorithm is not efficient as stated – it requires exponential space to ‘remember’ the bases that have previously been created. Fortunately, we are able to completely remove this dependency, and to make our algorithm *history-free*, meaning that the history of previous objects is not stored. In other words, we can determine the next “new” base without storing the “old” bases that have been listed. Specifically, for a matroid $M = (E, \mathcal{I})$ with $m = |E|$ elements, we can generate the bases using just $O(m)$ bits of storage. Furthermore, each

successive base can be determined using $O(m)$ calls to the matroid’s independence and co-independence oracle (see Section 2.1.5 for a discussion on oracles). Again, these results do not depend on the matroid, the initial base, or the specific manner in which ties are broken (so long as the associated computations do not dominate the time or memory requirements).

Our new greedy algorithm should be also viewed as a *meta-algorithm* that can be specialized into specific algorithms that can be optimized in a variety of ways. For example, by specializing to certain uniform matroids, we provide a new algorithm for generating $(2n, n)$ -combinations that runs in *constant amortized time* per generated combination. The algorithm produces an interesting new order of $(2n, n)$ -combinations that shares similar properties to the well-known listing by Eades and McKay [6].

1.3 Relationship to previous work

1.3.1 Sublist Gray codes

The most well-known minimal-change orders, or Gray codes, involve listing the set $B(n)$ of n -bit binary strings so that successive strings differ in one bit. In other words, they trace a Hamilton path in the n -dimensional hypercube. In particular, many readers will be familiar with the eponymous *binary reflected Gray code* (BRGC) attributed to Frank Gray [9]. This order can be defined recursively as $C_n = 0C_{n-1}, 1 \text{ rev}(C_{n-1})$, where rev denotes reversal. The $n = 4$ order is below.

$$C_4 = 0000, 0001, 0011, 0010, 0110, 0111, 0101, 1101, 1111, 1110, 1010, 1011, 1001, 1000.$$

The BRGC has been the source of many additional minimal-change orders. For example, the k -subsets of the set $[n] := \{1, 2, \dots, n\}$ are often referred to as *combinations*, and their characteristic vectors form the set $B_k(n)$, which contains the n -bit binary strings with weight k (i.e., exactly k many 1-bits). If the BRGC is restricted or filtered to only include $B_k(n)$, then successive bitstrings differ by an exchange of a 0 and a 1 [33]. Such an exchange is often referred to as a *transposition*, and this particular transposition Gray code is known as the revolving door Gray code for combinations, since one element leaves the k -set and one element enters the k -set at each step. More broadly, various classes of sublist Gray codes of the BRGC have been studied in the literature [37, 27]. This is also true for the well-known Steinhaus-Johnson-Trotter or *plain change* order of permutations [15] for sublists with a continuous range of inversions [38] or avoiding certain factors or patterns [26]. Similarly, the cool-lex Gray code for binary strings [30, 31] can be filtered into a prefix-shift Gray code for combinations [25] or more broadly, a shift Gray code for any bubble language [24], with similar results holding for the cool-lex order of multiset permutations [43, 45, 28]. In contrast, our algorithms are *adaptive* in the sense that they do not produce orders that are sublists of the BRGC or any other particular order. In particular, our meta-algorithm can start at any base. Due to this adaptivity, our algorithms are similar to Algorithm J, which is a greedy algorithm based on “jumps” that has recently been used to generate Gray codes for a large variety of combinatorial objects [10, 11, 12, 20, 4], based on encoding them as permutations.

1.3.2 Hamiltonicity

Previously, it was known that the bases of any matroid can be ordered by exchanges. In graph-theoretic terms, this means that the base exchange graph of a matroid – which contains one vertex per base, and an edge between bases that differ by an exchange – has a Hamilton path. In fact, Holzmann and Harary [13] showed that this graph has a Hamilton cycle including any of its edges, and excluding any of its edges. Furthermore, Naddef and

Pulleyblank [22] proved that the graph is either Hamilton-connected, i.e., it has a Hamilton path between any two end vertices, or it is a hypercube, in which case it has a Hamilton path between any two end vertices from distinct partition classes; see also [1]. The bases of a matroid can also be generated in a non-Gray code manner, without using a single exchange operation in each step. In particular, Uno [36] showed that the bases of a matroid can be generated in $O(m)$ space and the time is dependent on using $O(m)$ independence and contraction and deletion operations on the matroid. In contrast, our implementations do not rely on contracting and deleting elements from the matroid.

1.3.3 Spanning trees and column spaces

The Hamiltonicity of the base exchange graph implies that the spanning trees of a connected graph can be generated by edge exchanges. A specific order was generated efficiently by Smith [29] using $O(n^2)$ space and $O(1)$ amortized delay per tree, where n is the number of vertices of the graph; also see Knuth’s detailed coverage in [15, Sec. 7.2.1.6]. Our combinatorial result provides new edge-exchange Gray codes for spanning trees, and several avenues for developing efficient algorithms by selecting specific initial spanning trees and tiebreaker rules.

The second original source for matroids is linear algebra. In this context, the Hamiltonicity of the base exchange graph implies that the bases for the column space of a matrix can be generated by column exchanges. One of our column-exchange Gray codes is illustrated in Figure 3. To our knowledge, there are no previous algorithmic results in this particular case.

$$\begin{array}{cccccc}
 1 & 2 & 3 & 4 & 5 & & 1 & 2 & 3 & & 1 & 3 & 4 & & 1 & 4 & 5 & & 3 & 4 & 5 & & 2 & 3 & 5 & & 1 & 2 & 5 \\
 \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{array} \right] & & \left[\begin{array}{ccc} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{array} \right] \xrightarrow{+4} & \left[\begin{array}{ccc} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array} \right] \xrightarrow{-2} & \xrightarrow{+5} & \left[\begin{array}{ccc} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{array} \right] \xrightarrow{-3} & \xrightarrow{+3} & \left[\begin{array}{ccc} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{array} \right] \xrightarrow{-1} & \xrightarrow{+2} & \left[\begin{array}{ccc} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{array} \right] \xrightarrow{-4} & \xrightarrow{+1} & \left[\begin{array}{ccc} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{array} \right]
 \end{array}$$

■ **Figure 3** Listing the bases of the column space of the full-rank matrix over $GF(2)$ on the left. Notice that the even columns are identical, so they are not in a base together. Similarly, the odd columns are linear combinations of each other, so they do not form a base. The list is a column-exchange Gray code, i.e., successive bases differ in adding one column and removing another column. The list is constructed greedily by always minimizing the larger column label involved in the exchange. Ties can be broken arbitrarily; here the smaller column in the exchange is maximized.

1.4 Outline

Section 2 provides background information on matroids and Gray codes, and Section 3 introduces the prefix-exchange property. Our combinatorial and algorithmic results are in Section 4 and 5, respectively. We sharpen our general results for specific matroids in Section 6. Finally, we conclude with additional remarks in Section 7.

2 Preliminaries

2.1 Matroids

Matroids were introduced by Whitney in 1935 [40]. Originally conceived as a generalization of *independence* in vector spaces, they capture related notions of independence in set systems. We begin by considering the following more general notion.

► **Definition 1.** An independence system \mathcal{F} over a finite ground set E is a pair (E, \mathcal{I}) where $\mathcal{I} \subseteq 2^E$ satisfies the following two conditions:

1. $\emptyset \in \mathcal{I}$.
2. \mathcal{I} is closed under taking subsets. Specifically, if $Y \in \mathcal{I}$ and $X \subseteq Y$, then $X \in \mathcal{I}$.

The elements of \mathcal{I} are known as the *independent sets* of \mathcal{F} . The subsets of E which are not in \mathcal{I} will be known as *dependent sets*. We highlight that as E is finite, we usually think of it as $E = [m]$ for some m (said another way, we think of E as linearly ordered). In particular, statements like $e > f$ or $e \geq f$ make sense for $e, f \in E$. We also remark that elements of the ground set E will usually be denoted by e or f with $e > f$.

Of interest are typically the *maximal* independent sets. Thus, a *base* in an independence system $\mathcal{F} = (E, \mathcal{I})$ is an independent set which is maximal with respect to inclusion. Before defining matroids in terms of their bases, we introduce some notation: For a set A and an element a we use $A + a$ and $A - a$ as shorthands for $A \cup \{a\}$ and $A - \{a\}$ respectively. We now give the definition of a matroid.

► **Definition 2.** We say that an independence system $M = (E, \mathcal{I})$ is a matroid if for every pair T_1, T_2 of bases in M and $e \in T_1 - T_2$ there is an element $f \in T_2 - T_1$ such that both $T_1 - e + f$ and $T_2 + e - f$ are bases.

This condition is known as the (strong) base exchange property, as one is always allowed to exchange elements from bases while keeping the base property. It is not hard to see that the base exchange property implies that all bases have the same size. Thus, if T is a base, instead of writing “ $T + e - f$ is a base” or “ $T - e + f$ is a base” we can write the equivalent “ $T \Delta \{e, f\}$ is a base” as it is clear that one element must be swapped in and the other out.

If B is a base of $M = (E, \mathcal{I})$ we say that its complement $E - B$ is a *cobase*. Furthermore, if $S \subseteq E$ is contained in *some* cobase, then we will say that S is *coindependent*. Note that, by definition, a cobase is an inclusion-wise maximal coindependent set.²

2.1.1 Examples

In this subsection we formally define the example matroids mentioned in Section 1.

Let $n, k \in \mathbb{N}$ such that $n \geq k$. The *uniform matroid* $U(n, k)$ is the matroid with ground set $[n]$ and a subset $S \subseteq [n]$ is independent if and only if its size is at most k . The bases of $U(n, k)$ are the subsets of $[n]$ of size exactly k , its cobases are the subsets of $[n]$ of size exactly $n - k$ and its coindependent sets are the subsets of $[n]$ of size at most $n - k$. If $k = n$ this is also known as the *free matroid* of rank n .

Let $G = (V, E)$ be a graph. The *graphic matroid* $M(G)$, is the matroid with ground set E and a subset $S \subseteq E$ is independent if and only if it is a forest in G . The bases of $M(G)$ are the spanning forests of G , and its coindependent sets are subsets of edges that do not form cuts in G . If G is connected, then the bases of $M(G)$ are the spanning trees of G .

Consider a field \mathbb{F} and a matrix $A \in \mathbb{F}^{m \times n}$. The *column matroid* $M_{\mathbb{F}}(A)$ is the matroid with the columns of A as the ground set and a subset of the columns of A is independent if and only if it is linearly independent over \mathbb{F} . The bases of $M_{\mathbb{F}}(A)$ are linearly independent subsets of the columns of A of size $\text{rank}(A)$.

Consider a matroid $M = (E, \mathcal{I})$ and a weight function $w : E \rightarrow \mathbb{R}$ on the ground set. We say that a base B of M is *w-optimal* if it maximizes $\sum_{e \in B} w(e)$ among all bases of M . The *optimality matroid* M_w is the matroid with E as a ground set and a subset $S \subseteq E$ is independent if and only if it is contained in some *w-optimal* base. The bases of M_w are exactly the *w-optimal* bases of M .

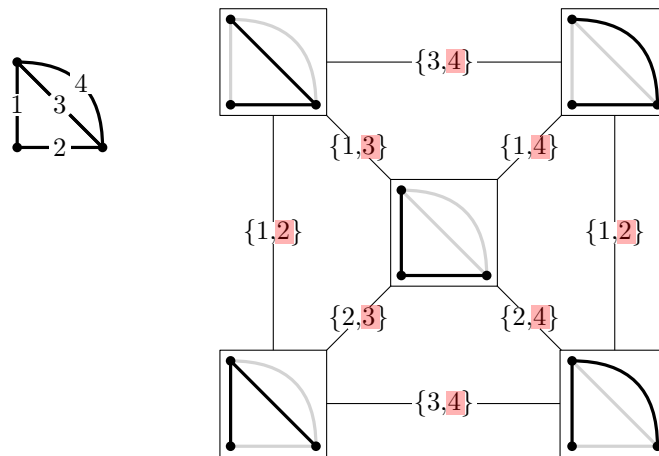
² It is interesting to note that the coindependent sets form another matroid called the dual matroid.

2.1.2 Base exchange graph

We consider the base exchange graph as originally defined by Maurer [18, 19].

► **Definition 3.** Let $M = (E, \mathcal{I})$ be a matroid. The base exchange graph of M is a graph $\mathcal{G}(M)$ with the bases of M as vertices and two bases T, T' are connected by an edge if and only if there exist $e, f \in E$ such that $T' = T \Delta \{e, f\}$. We refer to the edges in the graph $\mathcal{G}(M)$ as exchanges.

Note that for $e > f$ and the edge $(T, T \Delta \{e, f\})$ there are two possibilities for the directions in which the elements are exchanged; either (1) $e \in T$ and $f \notin T$ in which case the edge is $(T, T - e + f)$ or (2) $e \notin T$ and $f \in T$ in which case the edge is $(T, T + e - f)$.



■ **Figure 4** (Left) A graph H with $n = 3$ vertices and $m = 4$ edges. (Right) The base exchange graph $\mathcal{G}(M(H))$. Recall that the maximum size independent sets are the bases of $M(H)$, which are the spanning trees of H . These spanning trees are the vertices of $\mathcal{G}(M)$, and its edges join spanning trees that differ by an exchange of two edges.

For brevity, we use the term *exchange graph* to refer to $\mathcal{G}(M)$. Observe that the edges of the exchange graph have the form $(T, T \Delta S)$ for some non-empty set $S \subseteq E$. In Figure 4, each edge is labeled with its corresponding set S , and its largest element is highlighted.

As mentioned in Section 1.3, it is known that $\mathcal{G}(M)$ has a Hamilton cycle for all M with at least three bases, and in fact much stronger Hamiltonicity properties are known.

2.1.3 Circuits, cocircuits, loops and coloops

A *circuit* is a minimal dependent set. Unlike bases, which all have the same size, the size of circuit can vary. In particular, an element that forms a circuit by itself is a *loop*. Similarly, a *cocircuit* is a minimal set of elements that intersect with every base. In particular, an element that forms a cocircuit by itself is a *coloop*.

Note that the circuits of $U(n, k)$ are the sets of size $k + 1$, similarly the cocircuits of $U(n, k)$ are the sets of size $n - k + 1$. For a graph G , the circuits of $M(G)$ are exactly the cycles of G . On the other hand, the cocircuits of $M(G)$ are minimal cuts of G and, consequently, the coloops of $M(G)$ are the *bridges* of G .

The following remarks describe the circuits and cocircuits that are formed by adding or removing an element to a base, respectively. Most readers will be familiar with these remarks in the special case of graphic matroids. More specifically, Remark 4 implies that adding an edge to a spanning tree creates a unique cycle, whereas Remark 5 implies that removing an edge from a spanning tree creates a unique minimal cut.

► Remark 4. If $M = (E, \mathcal{I})$ is a matroid and T is a base with $e \in T$, then $T + e$ contains a unique circuit $C(T, e)$ called the *fundamental circuit of $T + e$* . Furthermore, $T \Delta \{e, f\}$ is a base, if and only if, f is in this unique circuit.

► Remark 5. If $M = (E, \mathcal{I})$ is a matroid and T is a base with $e \notin T$, then $T - e$ contains a unique cocircuit $C^*(T, e)$ called the *fundamental cocircuit of $T - e$* . Furthermore, $T \Delta \{e, f\}$ is a base, if and only if, f is in this unique cocircuit.

2.1.4 Operations

We now define minor operation on matroids (cf. minor operations on graphs) which will be useful to do induction on matroids.

► Definition 6. Let $M = (E, \mathcal{I})$ be a matroid and $X \subseteq E$. We define the deletion of X as a new matroid $M - X = (E - X, \mathcal{I}')$ where

$$\mathcal{I}' = \{I \subseteq E - X : I \in \mathcal{I}\}.$$

We now define a dual operation to deletion. Let T be a maximal independent set contained in X . We define the contraction of X as a new matroid $M/X = (E - X, \mathcal{I}'')$ where

$$\mathcal{I}'' = \{I \subseteq E - X : I \cup T \in \mathcal{I}\}.$$

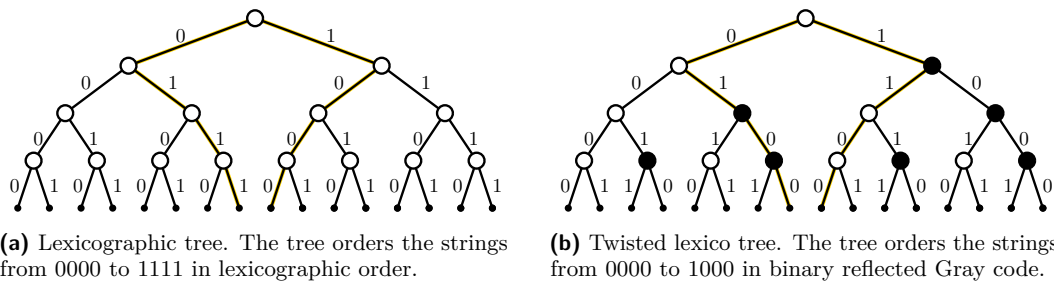
It is well known that the contraction and deletion of X are matroids and the contraction of X does not depend on the choice of maximal independent set $T \subseteq X$. The matroids which can be obtained by a sequence of contractions and deletions are known as the *minors* of M . It is also known that the operations of contraction and deletion are associative and commutative, thus every minor of M can be uniquely written as $M - A/B$ for $A, B \subseteq E$ disjoint.

2.1.5 Algorithmic considerations for matroids

When dealing with computational aspects for arbitrary matroids, it is important to discuss how they are given as an input to algorithms. A naive approach to the input problem is to encode the matroid M by a list of all the elements in E and all the bases of M . This approach is typically avoided because the number of bases can be incredibly large. Furthermore, it does not make much sense for generation algorithms to already have access to all the bases of the matroid, since the problem of generating all of the bases would have already been solved. Thus, we assume only oracle-based access to the matroids, that is, given a set $S \subseteq E$ we can test whether it is independent, coindependent or a base with an independence, coindependence and base oracle respectively. Furthermore, in our algorithmic results we always make explicit which type of oracle is needed and how many calls to them we need.

2.2 Lexico and colexico trees

Figure 5a illustrates a binary tree whose leaves are the binary strings of length $n = 4$ in lexicographic order. More specifically, the root has two children and the branches to these children are labeled 0 and 1. In turn, the branches at the second level fix the second bit to 0 or 1, and so on. Then each labeled path from the root to a leaf spells out a different member of $B(4)$ from the first bit to the last bit. The same approach can be applied for any n , and the resulting tree is the *lexicographic tree* for $B(n)$. By reversing the children of every second node on each level of a lexicographic tree, we obtain a *twisted lexico tree*. This is precisely how the binary reflected Gray code is created, as illustrated in Figure 5b.



■ **Figure 5** The strings in $B(4)$ ordered according to (a) the lexicographic tree, and (b) the twisted lexico tree. The large white nodes have two children with their branches labeled 0 and 1; the large black nodes reverse the labels to 1 and 0. The consecutive pair of highlighted paths differ in all bits in (a) and only one bit in (b).

More generally, we can construct lexicographic and twisted lexico trees for any subset of $B(n)$, although some internal nodes will have only one child. For example, consider the following set of binary strings, written in lexicographic order,

$$B_f = \{01011, 01101, 01110, 10011, 10101, 10110, 11001, 11010\}. \tag{1}$$

These strings are the incidence vectors of spanning tree edges in our favorite graph from Figure 2. In particular, 10110 corresponds to the edge set $\{1, 3, 4\}$, which is the first spanning tree in 2. The lexicographic tree for B_f is in Figure 6a, and it does not provide an exchange Gray code. More specifically, consecutive binary strings do not always differ by a transposition of a 0 and 1, and so the corresponding spanning trees do not differ by an edge exchange.

It is often more convenient to work with *colexicographic order*, which orders strings from right-to-left instead of left-to-right. For example, B_f is written below in this order,

$$B_f = \{11010, 10110, 01110, 11001, 10101, 01101, 10011, 01011\}. \tag{2}$$

The advantage of colexicographic order comes from the fact that *colexicographic trees* fix the value of the largest element at the top of the tree. For example, the colexicographic tree for B_f appears in Figure 6b, and the branches from the root determine the fifth bit, or edge number 5, in the corresponding spanning tree. This leaves³ the first four bits, or the edges numbered 1–4, to be determined, which makes for cleaner inductive results.

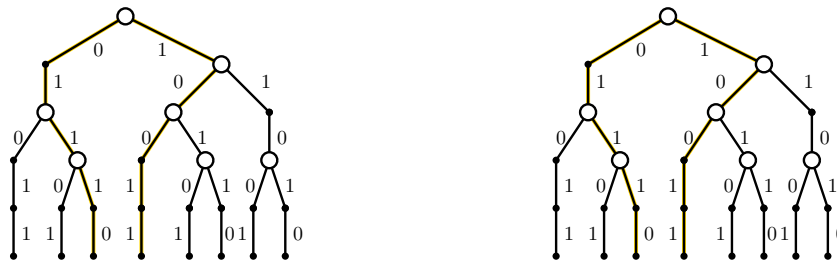
While twisted lexico trees have been used to generate a variety of Gray codes – see [32] where the lexico terminology originates – this singular approach does not create minimal change orders for matroid bases. Instead we’ll need to consider the more broad definition of a *lexico tree* from [32], or more precisely a *colexico tree*, in which *some* of the nodes have reversed children⁴. For example, our exchange Gray code in Figure 2 is backed by a colexico tree, as seen in Figure 7a. Furthermore, if we change the tiebreaker rule from closest to furthest, then another colexico tree is obtained, as seen in Figure 7b. In fact, Remark 7 asserts that something much more general is true.

► **Remark 7.** Every order generated by our generic greedy algorithms has a colexico tree.

Remark 7 should come as somewhat of a surprise. For example, note that the root node in a colexico tree has only two branches. As a result, Remark 7 implies that the largest element changes only once during all of our orders. While our greedy rule always tries to

³ No pun intended!

⁴ Swapping the first and last child of some nodes has also been considered for larger alphabets (see [16]).



(a) Lexicographic tree. The leaves are ordered lexicographically from 01011 to 11010 as in (1). (b) Colexicographic tree. The leaves are ordered colexicographically from 11010 to 01011 as in (2).

■ **Figure 6** Lexicographic and colexicographic trees for B_f , which contains the edge incidence vectors of the spanning trees of our favorite labeled graph from Figure 2. The trees in (a) and (b) are structurally the same due to the fact that $b_1b_2b_3b_4b_5 \in B_f \iff b_5b_4b_3b_2b_1 \in B_f$, which in turn is due to the labeling used. However, the root-to-leaf paths encode reversed strings with respect to each other. In particular, (a) begins with 01011 which corresponds to the spanning tree $\{2, 4, 5\}$, while (b) begins with 11010 which corresponds to the spanning tree $\{1, 2, 4\}$. The pairs of highlighted paths show that these orders are not exchange Gray codes.

minimize the larger element involved in an exchange, there is no immediate reason why it will be involved in only one exchange throughout the order. We'll see that Remark 7 follows from the inductive structure provided in the proof of Theorem 10.

2.2.1 Relabeling

Our main result holds regardless of how the elements are labeled. What does this mean? Before our algorithm is run, the elements of the matroid are given a labeling that has a total order, and then the algorithm creates an order of the bases that has a colexico tree with respect to this labeling. As mentioned earlier, the use of colexicographic order tends to give cleaner inductive results, but otherwise it is not “special” in any particular way.

3 Prefix minors and the prefix exchange property

In this section, we introduce the matroid property that is central to the proper functioning of our greedy algorithm. We also discuss its implications in terms of colexico trees. The property is based on the notion of a prefix minor, which is defined next.

3.1 Prefix minors

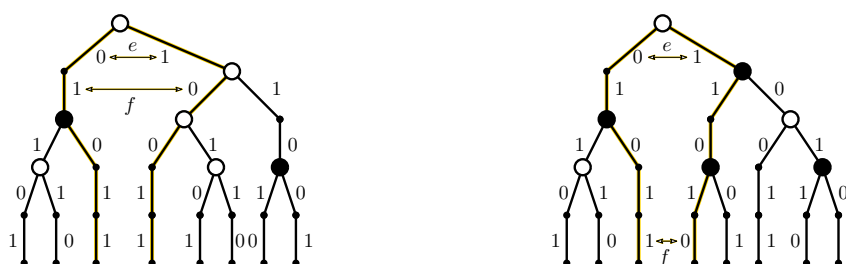
When the ground set of a matroid has a total order, we can consider minors in which some number of the largest elements are deleted or contracted, or equivalently, some number of the smallest elements remain. This leads to the following definition.

► **Definition 8.** $M' = (E', \mathcal{I}')$ is an e -prefix minor of $M = (E, \mathcal{I})$ if it is a minor of M and $E' = \{e \in E : f \leq e\}$.

We say that M' is a *prefix minor* of M if M' is an e -prefix minor of M for some $e \in E$.

3.2 Prefix exchange property for bases

The following lemma is a specialization of the standard base exchange property of matroids.



(a) Colexico tree arising from Algorithm G_{close} . The tree orders the strings from 10110 to 10011 in the same exchange Gray code seen in Figures 2 and 9. In particular, the spanning trees corresponding to the highlighted paths are $T = \{1, 2, 5\}$ and $T' = T\Delta\{e, f\} = \{1, 2, 4\}$ for $e = 5$ and $f = 4$.

(b) Colexico tree arising from Algorithm G_{far} . The tree orders the strings from 10110 to 10101 in another exchange Gray code from the same initial base. In particular, the spanning trees corresponding to the highlighted paths are $T = \{1, 2, 4\}$ and $T' = T\Delta\{e, f\} = \{2, 4, 5\}$ for $e = 5$ and $f = 1$.

■ **Figure 7** The incidence vectors of edges in the spanning trees of our favorite graph are found in (1) (and also (2)). These binary strings are ordered above by two different colexico trees. The branches at the roots determine the inclusion or exclusion of edge 5 (not edge 1). The large nodes are as in Figure 5, and the remaining internal nodes have one child. The consecutive pair of highlighted paths differ in only two bits are changed and the larger is $e = 5$. The algorithm generating (a) selects the smaller change $f = 4$ to be as high up the tree as possible, whereas the algorithm generating (b) selects $f = 1$ to be as low in the tree as possible.

► **Lemma 9** (Prefix exchange property for matroid bases). *Let M' be an e -prefix minor of M such that e is not a loop or coloop in M' . For every base T of M' there exists $f < e$ such that $T\Delta\{e, f\}$ is a base of M' .*

Proof. Let $M' = (E', \mathcal{T})$ be an e -prefix minor of $M = (E, \mathcal{T})$, and suppose that e is not a loop or coloop in M' . We proceed in two cases.

- **Case 1:** $T \subseteq E'$ is a base of M' with $e \in T$. Since e is not a coloop in M' , there is a base $T' \subseteq E'$ of M' with $e \notin T'$. Since $e \in T - T'$, the base exchange property implies that there exists an $f \in T' - T$ such that $T\Delta\{e, f\}$ is a base of M' . Since e is the largest element in E' , we know that $f < e$. Hence, the result holds.
- **Case 2:** $T \subseteq E'$ is a base of M' with $e \notin T$. A similar argument proves this case. ◀

We now illustrate Lemma 9 for two matroids. In the case of a uniform matroid every prefix minor is also a uniform matroid. Consider a specific uniform matroid $M' = U(8, 5)$, which is an 8-prefix minor of $M = U(14, 7)$. Let $T = \{1, 2, 4, 5, 8\}$ be a base of M' . Since $e = 8$, we can finish the exchange by adding any $f \in [8] - T = \{3, 6, 7\}$. On the other hand, if $T = \{1, 2, 4, 6, 7\}$ and $e = 8$ as before, we can finish the exchange by removing any $f \in T$. For a graphic matroid, once again every prefix minor is also a graphic matroid. So, if M' is the graphic matroid of some graph and e is the largest labelled edge, then if $e \notin T$ the exchange can be completed by removing a smaller labelled edge f from the unique cycle in $T + e$, and if $e \in T$ the exchange can be completed by removing a smaller labelled edge f from the unique minimal cut in $T - e$; such an f exists since e is neither a loop nor coloop.

3.3 Tree interpretation

It is instructive to consider the implications of the prefix exchange property in terms of colexico trees. The condition that e is not a loop or coloop in M' implies that there are two subtrees under the node corresponding to M' in any colexico tree of the bases of the matroid M . We don't know which of these two children is first or second in the colexico tree, but we do know that both subtrees contain at least one leaf (which corresponds to a base in M).

The rest of the property implies the following: For every leaf in either subtree, there is a base exchange that results in some leaf of the other subtree. This turns out to be a very strong and helpful property in the context of our greedy algorithm. As our algorithm is building a colexico tree, there will always be an exchange that moves from the last leaf within the first subtree to some leaf within the second subtree; the specific details of which subtree is first or second, and which leaf is last within the first subtree simply do not matter. In other words, our algorithm can never get stuck; this will be formalized in Theorem 10.

This interpretation of the prefix exchange property brings to mind the ZIG ship in *Zero Wing*. Regardless of how “SOMEBODY SET UP US THE BOMB” [17], there is always a ZIG that can escape and continue the mission to catch CATS. This connection is illustrated in Figure 8.

4 Combinatorial result: Hamilton paths

We now describe our main result: the following simple greedy algorithm generates all the bases of a matroid in a Gray code order.

Algorithm G (*Greedy bases*). This algorithm attempts to greedily generate the bases of a matroid M starting from an initial base T_0 .

G1. [Initialize] Visit the initial base T_0 .

G2. [Greedy] Generate an unvisited base of M by performing *any* exchange whose larger element is as small as possible. If no such exchange exists, then terminate. Otherwise visit the resulting base and repeat G2.

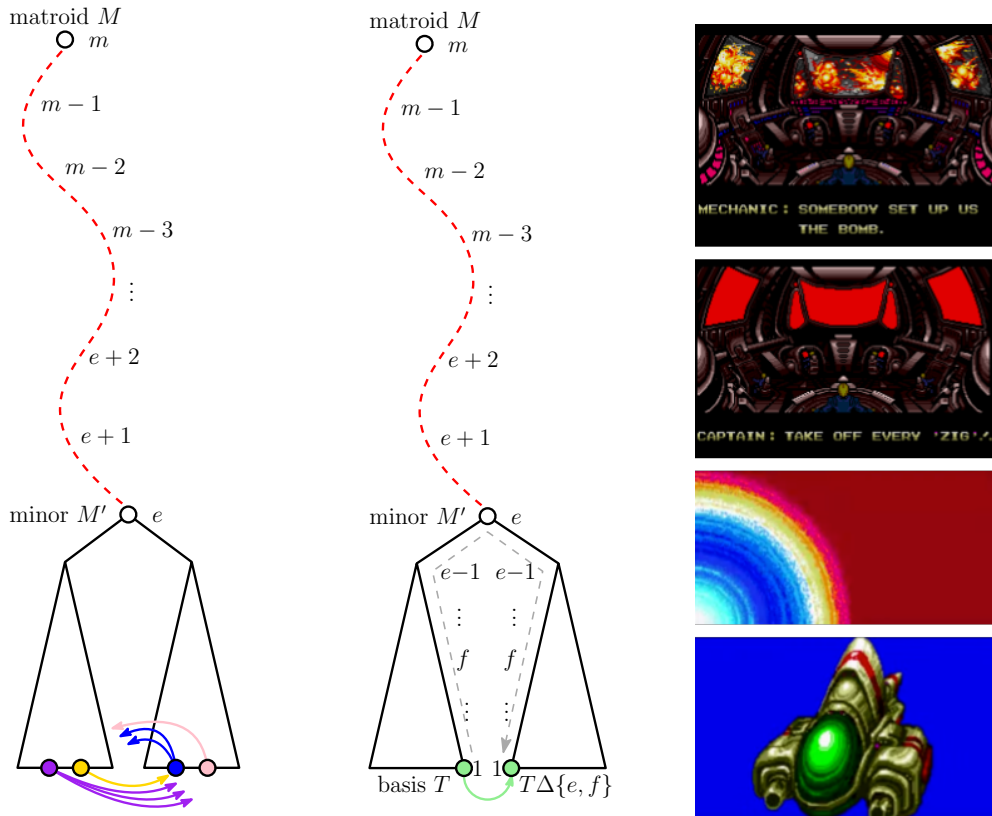
We add the following four notes on Algorithm $G(M, T_0)$.

1. *Efficiency.* It is *not* efficient as presented. This is because the previously visited bases must be maintained. In Section 5, we present several history-free implementations.
2. *Tiebreakers.* It can proceed in many different ways depending on the choices made for “any exchange”. This is discussed in more detail in Section 4.1.
3. *Restricted operations.* In many cases, it is possible to prove that the algorithm operates using only a small subset of the exchanges that it would use in general. Several examples of this appear in Section 6.
4. *Starting base.* Even though the algorithm succeeds *independently* of the initial base, choosing a particular initial base may lead to nicer and/or stronger properties for the listings produced. Examples of this appear in Section 6.

Now we prove that the algorithm works as intended.

► **Theorem 10.** *If $M = (E, \mathcal{I})$ is a matroid and T is a base of M , then $G(M, T)$ always provides a Hamilton path of $\mathcal{G}(M)$. In other words, Algorithm G creates an exchange Gray code for the bases of M starting from any initial base T , regardless of how ties are broken.*

Proof. We prove the statement of the theorem by induction on the number of elements, $|E|$. If $|E| = 1$, then there is at most one base, and so the base case holds. Otherwise, suppose that the statement holds for all matroids with k elements and all initial bases. Now consider a matroid $M = (E, \mathcal{I})$ with $k + 1 = |E|$ elements, and an initial base T . Without loss of generality, label the elements in M as $E = [k + 1]$. Let $M_1 = M/e$ and $M_2 = M - e$ be the two prefix minors with respect to the largest element $e = k + 1 \in E$. Finally, let m , m_1 , and m_2 be the number of bases in M , M_1 , and M_2 , respectively. Note that m_1 and m_2 are the number of bases of M that do and do not contain e , respectively. We proceed in two cases.



(a) The prefix-exchange property ensures that there is at least one exchange from each leaf in either subtree to a leaf in the other subtree. For example, the blue arrows are meant to indicate that there are two different exchanges from the blue leaf in the second subtree to some leaves in the first subtree. Similarly, there is one exchange from the gold leaf that moves from the first subtree to the second subtree.

(b) Algorithm \mathcal{G} escapes the first subtree via an exchange on the last leaf in the first subtree, which is labeled as base T . Visually, \mathcal{G} goes up the colexico tree to e , which is the lowest node with an unvisited branch. Then it switches branches and goes down the same branches it went up, except for switching at f to complete the exchange. This gives a new base $T' = T \Delta \{e, f\}$, which becomes the first leaf in the second subtree.

(c) No matter how many times *Zero Wing* is played, there will always be a ZIG that escapes the bomb. This ZIG continues the pursuit of CATS. Thus, the meme never ends – just as Algorithm \mathcal{G} can never get stuck.

■ **Figure 8** The prefix exchange property for a matroid $M = (E, \mathcal{I})$ with $E = [m]$. The red path illustrates a sequence of contractions or deletions of $m, m - 1, \dots, e + 1$ that result in an e -prefix minor M' in which e is neither a loop nor a co-loop. Hence, there are two non-empty subtrees below the node labeled e : one subtree for bases that include e , and the other subtree for bases that exclude e . An exchange can always “escape” the first subtree, just as a ZIG can always escape the exploding ship in *Zero Wing*.

- Case 1: $e \in T$. Observe that $T_1 = T - e$ is a base of M_1 . Since M_1 has only k elements, we know by induction that $\mathbf{G}(M_1, T_1)$ creates an exchange Gray code for all m_1 bases of M_1 starting at T_1 , regardless of how ties are broken. This also implies that $\mathbf{G}(M, T)$ will begin by generating an exchange Gray code for its bases that include e . More precisely, $\mathbf{G}(M, T)$ can begin by generating an ordered list of m_1 bases of M starting from T , if and only if, $\mathbf{G}(M_1, T_1)$ generates the same list but with e removed from each base. This is due to the fact that algorithm \mathbf{G} always minimizes the maximum element involved in an exchange, and hence, $\mathbf{G}(M, T)$ will not use an exchange involving e until it has no other options (and by induction this happens only after every base including e has been generated). If $m_2 = 0$, then $\mathbf{G}(M, T)$ has successfully generated every one of its bases, and the induction is complete. For the remainder of the proof, we assume that $m_2 > 0$, and this implies that e is neither a loop nor a co-loop in M .

Let T'_1 be an arbitrary last base generated by $\mathbf{G}(M_1, T_1)$. Hence, $T' = T'_1 + e$ is an arbitrary last base of M that contains element e . By the prefix exchange property, there is a base of M that differs from T' by an exchange involving e and a smaller element f . That is, there exists $T_2 = T' \Delta \{e, f\}$ with $f < e$. Therefore, algorithm $\mathbf{G}(M, T)$ will be able to continue generating bases by making an exchange to some such base T_2 .

Since T_2 is a base of M that does not contain element e , we know that it is a base of M_2 . Hence, by induction, $\mathbf{G}(M_2, T_2)$ creates an exchange Gray code for all m_2 bases of M_2 starting at T_2 , regardless of how ties are broken. This also implies that $\mathbf{G}(M, T)$ will end by generating an exchange Gray code for its bases that do not include e . More precisely, $\mathbf{G}(M, T)$ can end by generating an ordered list of m_2 bases of M starting from T' , if and only if, $\mathbf{G}(M_2, T_2)$ generates the same list of bases. Again, this is due to \mathbf{G} minimizing the maximum element involved in an exchange.

The previous three paragraphs have shown that $\mathbf{G}(M, T)$ generates an exchange Gray code for all of its m_1 bases including e , followed by an exchange to some base T' that does not include e , followed by an exchange Gray code for all of its m_2 bases that do not include e . Hence, the inductive statement is true.

- Case 2: $e \notin T$. This case is nearly identical to the previous case by dual arguments. ◀

4.1 Tiebreaker rules

Theorem 10 proves that Algorithms \mathbf{G} is always successful in generating minimal change orders starting from any base. However, the specific orders that they generate will depend on how ties are broken. These tiebreaker choices can affect how efficiently the resulting order can be generated or ranked and unranked, and even the specific types of operations that are used. For example, in Section 6.1 we'll see that one tiebreaker rule causes Algorithm \mathbf{G} to use homogeneous transpositions, while another causes it to use consistent transpositions.

While tiebreaker rules can be quite creative, they must adhere to some basic principles. For example, if the larger element e involved in an exchange is entering (exiting) the base, then the smaller element f must be exiting (entering) to satisfy Theorem 10. This observation is further refined below according to Remarks 4–5.

► **Remark 11.** If Algorithm $\mathbf{G}(M, T)$ performs an exchange on base T' and e is the larger element in the exchange, then its smaller element f must satisfy the following:

- If $e \notin T'$, then f is in the fundamental circuit of $T + e$.
- If $e \in T'$, then f is in the fundamental cocircuit of $T - e$.

The existence of such an f is guaranteed by Lemma 9.

This leads us to the following pair of simple and fundamental tiebreaker rules in Sections 4.1.1–4.1.2. See Figures 7a–7b for an example of how they differ.

4.1.1 Maximum/closest tiebreaker rule

The following tiebreaker rule is *generic* in the sense that it can be applied to any matroid without any additional information. When applied to matroid bases, it simply maximizes the smaller element involved in an exchange.

Algorithm G_{close} (*Greedy bases with maximum tiebreaker*). This algorithm greedily generates the bases of a matroid M starting from an initial base T_0 .

C1. [Initialize] Visit the initial base T_0 .

C2. [Greedy] Generate an unvisited base of M by performing the exchange whose larger element is minimized, and then breaking ties by maximizing the smaller element. If no such exchange exists, then terminate. Otherwise visit the resulting independent set and repeat C2.

We note that the term “maximizing the smaller element” is in reference to the set S in an edge of the form $(T, T\Delta S)$ in the underlying exchange graph. More specifically, the larger element of such an edge is $\max(S)$, and the smaller element is $\min(S)$. Seen another way, the tiebreaker rule chooses the smaller element to be as close as possible to the larger element, hence the terminology *maximum/closest tiebreaker rule*.

4.1.2 Minimum/furthest tiebreaker rule

The following tiebreaker rule is opposite to the previous, in the sense that it minimizes the smaller element involved in an exchange. Since it doesn’t depend on the particular matroid, we also refer to it as a generic tiebreaker rule.

Algorithm G_{far} (*Greedy bases with minimum tiebreaker*). This algorithm is identical to Algorithm G_{close} , but with *minimizing the smaller element*.

This tiebreaker rule chooses the smaller element to be as far away as possible from the larger element, hence the terminology *minimum/furthest tiebreaker rule*.

5 Algorithmic results

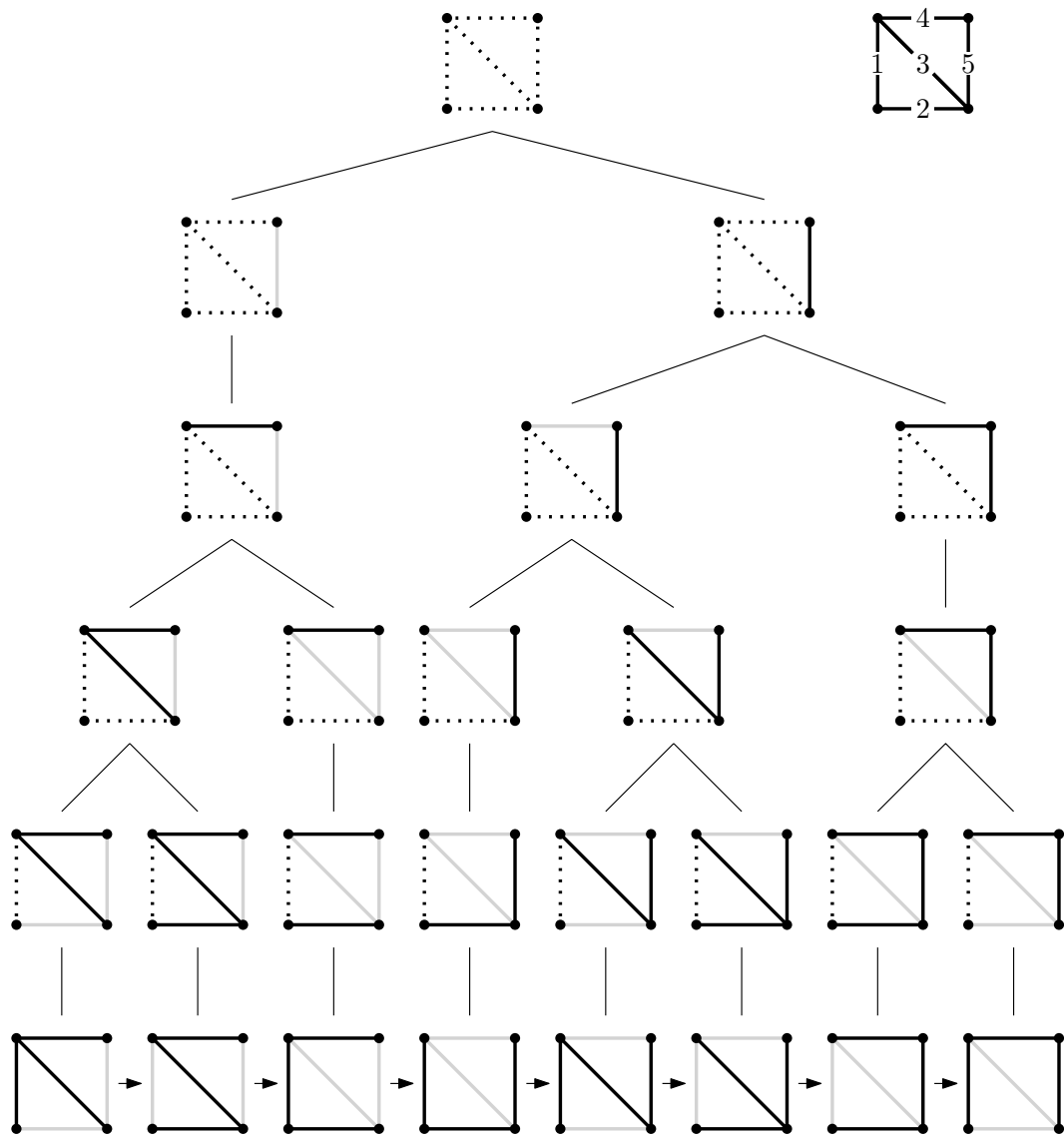
In this section, we provide history-free specializations of the meta-algorithm from Section 4. In other words, we can remove the “which hasn’t appeared earlier” part of Algorithm G by maintaining additional data structures. More specifically, we give history-free implementations of Algorithms G_{close} and G_{far} . The implementations works for any matroid and initial base.

5.1 History-free implementation

We now present history-free iterative implementations of our generic greedy algorithm. To do so, the implementations maintain an additional data structure, like an array or a stack, that allows them to implicitly navigate the colexico tree, as in Figure 9. In particular, these algorithms do not store minors of the matroid (cf. [36]).

5.1.1 Active array

Let $M = ([m], \mathcal{I})$ be a matroid and T_0 an initial base of M . In this subsection we show how to implement $G_{\text{close}}(M, T_0)$ with an active array. Let T be a base of M and recall that T is a leaf on the colexico tree of $G_{\text{close}}(M, T_0)$. We also introduce the notation $P(T)$ for the path between T and the root of the colexico tree.



■ **Figure 9** A colexico tree of our favorite graph. More specifically, it is the colexico tree generated by algorithm G_{close} , which is the specialization of greedy algorithm G using the maximum tiebreaker rule. Observe that the bottom row provides the exchange Gray code for the spanning trees of our favorite graph seen in Figure 2. The same tree with incidence vectors is in Figure 7a.

The main idea behind the active array approach is that whenever we generate T , we also store $A(T)$ which contains $i \in [m]$ if and only if the i -th edge on $P(T)$ (counting from T towards the root) was the leftmost (i.e. first) branch on the colexico tree. Said another way, $A(T)$ stores the potential directions we could follow when traversing the lexico tree. Note that initially $A(T_0) = [m]$ as every edge on $P(T_0)$ was a first branch. Furthermore, if we are given the state $(T, A(T))$ we can compute the next state $(T', A(T'))$ as follows:

1. Computing T' . Let $e \in A(T)$ be such that there exists $f \leq e$ and $T \Delta \{e, f\}$ is a base. Moreover, if there are multiple choices for f we pick the maximum possible ones due to the tie-breaker rule. Note that $T \Delta \{e, f\}$ is the next base as e is the minimum larger exchange possible. This can be computed by checking whether $T \Delta \{e, f\}$ is a basis for all pairs of possible e 's and f 's.

2. Computing $A(T')$. Note that the path only changed for elements smaller than e . We observe that e is now in its second branch, so it should not appear on $A(T')$. Furthermore, as we just moved to the second branch labelled by e , this implies that everything smaller than e is in its first branch, so it should appear in $A(T')$. This implies that

$$A(T') = [e - 1] \cup \{g \in A(T) : g > e\}.$$

These observations translate into the pseudocode in Algorithm 1.

Algorithm 1 $G_{\text{close}}(M, T_0)$.

Input: A matroid $M = ([m], \mathcal{I})$ and an initial base T_0 .

```

1:  $T \leftarrow T_0$ 
2:  $A \leftarrow [m]$ 
3: Visit  $T$ 
4: repeat
5:    $\text{visited} \leftarrow \text{False}$ 
6:   for  $e = 2, \dots, m$  do
7:     if  $e \in A$  then
8:       for  $f = e - 1, \dots, 1$  do
9:         if  $T \Delta \{e, f\}$  is a base and not visited then
10:           $T \leftarrow T \Delta \{e, f\}$ 
11:           $A \leftarrow [e - 1] \cup \{a \in A : a > e\}$ 
12:          Visit  $T$ 
13:           $\text{visited} \leftarrow \text{True}$ 
14: until not visited

```

We make the following observations on the pseudocode implementation.

- The tiebreaker rule only comes into play in Line 8; this for loop is done in a descending way as to prioritize the f which is closest to e . In particular, if one would like to implement the furthest tiebreak, one only needs to do this for loop in an ascending manner. In fact, any (computable) tiebreaker rule can be implemented by following a similar scheme.
- This implementation of G_{close} requires $O(m^2)$ calls to the base oracle. Additionally, we can replace every call to the base oracle for a set S to checking if $|S| = |T_0|$ and S is independent. Thus, this implementation of G_{close} requires $O(m^2)$ calls to the independence oracle. Moreover, we only need to keep track of the state $(T, A(T))$ and the counters, obtaining an algorithm which uses $O(m)$ space.

Further implementation details can be seen in the Appendix.

5.1.2 Stack

In this subsection we provide a stack implementation of $G_{\text{far}}(M, T)$. The main idea behind the stack approach is very similar to the active vector one: Whenever we generate T , we store alongside $S(T)$ which contains $i \in [m]$ if and only if the edge labelled i on $P(T)$ has an unvisited second branch the colexico tree. Said another way $S(T)$ stores the unvisited directions when traversing the lexico tree.

We now characterize when a node has two children in the colexico tree. To this end, we define the following: (1) T^* is the complement of the base T ; (2) for $e \in [m]$ the partial base $\partial_e T$ and cobase $\partial_e T^*$ are the “suffix” of size $m - e$ of T and T^* respectively. More formally,

$$\partial_e T := T \cap \{e + 1, \dots, m\}, \quad \partial_e T^* = T^* \cap \{e + 1, \dots, m\}.$$

Recall that every node on the colexico tree has a prefix minor associated to it. Furthermore, if M' is an e -prefix minor whose node associated lies on $P(T)$ we have that $M' = M - \partial_e T^* / \partial_e T$. The key observation here, is that the node associated to M' has two children if and only if $\partial_e T + e$ is independent and $\partial_e T^* + e$ is coindependent. Since M' has two children if and only if both taking element e and not taking e lead to bases, said another way $\partial_e T + e$ can be extended to a base and $\partial_e T^* + e$ can be extended to a cobase.

With this observation, we can compute the state $(T', S(T'))$ that follows $(T, S(T))$.

1. Computing T' . Let $e \in S(T)$ be the smallest element in the stack. We now consider the set $T\Delta\{e\}$, this has either a fundamental circuit or cocircuit. In particular, any $f \leq e$ which lies on the fundamental circuit or cocircuit will form a possible base (this exists as there was a branching on this node of the colexico tree). Moreover, if there are multiple choices for f we pick the minimum possible one due to the tie-breaker rule. Note that $T\Delta\{e, f\}$ is the next base as e is the minimum larger exchange possible.
2. Computing $S(T')$. Note that the path only changed for elements smaller than e . Thus, the only elements which can enter the stack are smaller than e . In view of this, we update the stack by checking if for every element $f \leq e$ the partial bases and cobases $\partial_f T'$ and $\partial_f T'^*$ are independent and coindependent respectively.
3. Delay. Note that if $\partial_f T'$ is *not* independent, f is forced to be out of the next base in order to actually reach a leaf. Similarly, whenever $\partial_f T'$ is *not* coindependent, f is forced to be inside of the next base in order to actually reach a leaf. Perhaps surprisingly, whenever we are not forced we do not take a choice and delay the decision to complete the exchange at a later point (See 5.1.3 for details on why this works).

These observations translate into the pseudocode in Algorithm 2.

■ **Algorithm 2** $G_{\text{far}}(M, T_0)$.

Input: A matroid $M = ([m], \mathcal{I})$ and an initial base T_0 .

- 1: $T \leftarrow T^0$
- 2: $S \leftarrow \emptyset$
- 3: **for** $e = m, \dots, 1$ **do**
- 4: **if** $\partial_e T + e \in \mathcal{I}$ and $\partial_e T^* + e \in \mathcal{I}^*$ **then**
- 5: $S.\text{push}(e)$
- 6: Visit T
- 7: **while** $S \neq \emptyset$ **do**
- 8: $e \leftarrow S.\text{pop}()$
- 9: **for** $f = e - 1, \dots, 1$ **do**
- 10: **if** $\partial_f T + f$ is independent and $\partial_f T^* + f$ is coindependent **then**
- 11: $S.\text{push}(f)$
- 12: **else if** $\partial_f T + f$ is independent and $f \notin T$ **then**
- 13: $T \leftarrow T\Delta\{f\}$
- 14: $T^* \leftarrow T^*\Delta\{f\}$
- 15: **else if** $\partial_f T^* + f$ is coindependent and $f \in T$ **then**
- 16: $T \leftarrow T\Delta\{f\}$
- 17: $T^* \leftarrow T^*\Delta\{f\}$
- 18: Visit T

We make the following observations on the pseudocode implementation.

- The tiebreaker of G_{far} comes into play with the delay strategy. More specifically, we always delay the decision of picking f until it gets forced on us. Note that it is not hard to replace this tiebreaker with another by adding some conditions to select f .



■ **Figure 10** The implementation of the minimum/furthest tiebreaker rule seems dangerous: Is it safe to delay choosing f until we are forced to do so? Don't worry, "WE KNOW WHAT WE DOING".

- This implementation of \mathbf{G}_{far} requires $O(m)$ calls to the independence and coindependence oracles. Moreover, we only need to keep track of the state $(T, S(T))$ and the counters, obtaining an algorithm which uses $O(m)$ space.

Further implementation details can be seen in the Appendix.

5.1.3 Delay and the furthest tiebreaker rule

Recall that the furthest tiebreaker rule delays switching branches until it is forced to do so. For example, the smaller change occurs at the very bottom of the colexico tree in Figure 7b.

At first, this approach to implementing the furthest tiebreaker rule may seem to be dangerous. *How do we know that there will be a smaller element f that is forced? How do we know that the remaining smaller values can be chosen without any additional changes?* This approach works due to Remarks 4–5. More specifically, the minimum smaller element f is precisely the smallest element on the fundamental circuit $C(T, e)$ or cocircuit $C^*(T, e)$ created by adding or removing the larger element e . If we have not changed the other elements on this circuit or cocircuit, then this f will be a loop or coloop, respectively, and so the algorithm will be forced into choosing it to be the smaller element in the exchange. In the case of Figure 7b, the first highlighted base is the spanning tree with edges $\{1, 2, 4\}$ in the graph found in Figure 2; adding edge $f = 5$ at the top of the colexicotree creates the fundamental cycle 1234 whose smallest element is $e = 1$ at the bottom of the colexico tree. The spirit behind this technique is illustrated in Figure 10.

5.1.4 Example

We illustrate the execution of the iterative algorithms using the *Vámos matroid*. First described in an unpublished manuscript of Peter Vámos [39], the Vámos matroid $\mathcal{V} = ([8], \mathcal{I})$ has $[8]$ as a ground set and its bases are all subsets of size four *except* $\{1, 2, 3, 4\}$, $\{1, 2, 5, 6\}$, $\{1, 2, 7, 8\}$, $\{3, 4, 5, 6\}$ and $\{3, 4, 7, 8\}$ which are marked as faces in Figure 11. Thus, it has $\binom{8}{4} - 5 = 65$ bases. The remaining independent sets are all subsets of size at most three. The Vámos matroid is a very interesting example to run our algorithms, as it is not a column matroid for any field. We show the run of $\mathbf{G}_{\text{close}}(\mathcal{V}, \{1, 2, 3, 5\})$ on Table 1.

5.2 Analysis

In this subsection we provide running times for specific matroids, without referencing oracle calls. We are particularly interested in the *delay* of the algorithm, that is, the time it takes to visit the next base.

We highlight that the algorithms only need to check for *incremental independence* and *incremental coindependence*. Specifically, if we are given an independent (resp. coindependent) set $S \subseteq E$, we must be able to check whether $S + s$ is independent (resp. coindependent) for

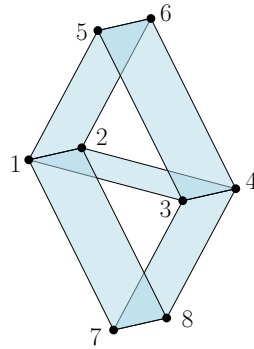


Figure 11 A representation of the five circuits of size four in the Vámos Matroid. Each of $\{1, 2, 3, 4\}$, $\{1, 2, 5, 6\}$, $\{1, 2, 7, 8\}$, $\{3, 4, 5, 6\}$, $\{3, 4, 7, 8\}$ appears as a face in the figure.

Table 1 Vámos matroid bases with variable traces from the array and stack implementations.

#	base T	Active array $A(T)$	Stack $S(T)$	#	base T	Active array $A(T)$	Stack $S(T)$
1	1, 2, 3, 5	1, 2, 3, 4, 5, 6, 7, 8	8, 7, 6, 4	34	3, 5, 6, 8	1, 7	7, 6, 5, 3
2	1, 2, 4, 5	1, 2, 3, 5, 6, 7, 8	8, 7, 6, 3	35	2, 5, 6, 8	1, 2, 5, 6, 7	7, 6, 5, 2
3	1, 3, 4, 5	1, 2, 5, 6, 7, 8	8, 7, 6, 2	36	1, 5, 6, 8	1, 5, 6, 7	7, 6, 5
4	2, 3, 4, 5	1, 5, 6, 7, 8	8, 7, 6	37	1, 4, 6, 8	1, 2, 3, 4, 6, 7	7, 6, 4, 3, 2
5	2, 3, 4, 6	1, 2, 3, 4, 5, 7, 8	8, 7, 5, 4, 3, 2	38	2, 4, 6, 8	1, 3, 4, 6, 7	7, 6, 4, 3
6	1, 3, 4, 6	1, 3, 4, 5, 7, 8	8, 7, 5, 4, 3	39	3, 4, 6, 8	1, 2, 4, 6, 7	7, 6, 4
7	1, 2, 4, 6	1, 2, 4, 5, 7, 8	8, 7, 5, 4	40	2, 3, 6, 8	1, 2, 3, 6, 7	7, 6, 3, 2
8	1, 2, 3, 6	1, 2, 3, 5, 7, 8	8, 7, 5	41	1, 3, 6, 8	1, 3, 6, 7	7, 6, 3
9	1, 3, 5, 6	1, 2, 3, 4, 7, 8	8, 7, 4, 2	42	1, 2, 6, 8	1, 2, 6, 7	7, 6
10	2, 3, 5, 6	1, 3, 4, 7, 8	8, 7, 4	43	1, 2, 5, 8	1, 2, 3, 4, 5, 7	7, 5, 4, 3
11	2, 4, 5, 6	1, 2, 3, 7, 8	8, 7, 2	44	1, 3, 5, 8	1, 2, 4, 5, 7	7, 5, 4, 2
12	1, 4, 5, 6	1, 3, 7, 8	8, 7	45	2, 3, 5, 8	1, 4, 5, 7	7, 5, 4
13	1, 4, 5, 7	1, 2, 3, 4, 5, 6, 8	8, 6, 5, 4, 3, 2	46	2, 4, 5, 8	1, 2, 3, 5, 7	7, 5, 3, 2
14	2, 4, 5, 7	1, 3, 4, 5, 6, 8	8, 6, 5, 4, 3	47	1, 4, 5, 8	1, 3, 5, 7	7, 5, 3
15	3, 4, 5, 7	1, 2, 4, 5, 6, 8	8, 6, 5, 4	48	3, 4, 5, 8	1, 2, 5, 7	7, 5
16	2, 3, 5, 7	1, 2, 3, 5, 6, 8	8, 6, 5, 3, 2	49	2, 3, 4, 8	1, 2, 3, 4, 7	7, 4, 3, 2
17	1, 3, 5, 7	1, 3, 5, 6, 8	8, 6, 5, 3	50	1, 3, 4, 8	1, 3, 4, 7	7, 4, 3
18	1, 2, 5, 7	1, 2, 5, 6, 8	8, 6, 5	51	1, 2, 4, 8	1, 2, 4, 7	7, 4
19	1, 2, 4, 7	1, 2, 3, 4, 6, 8	8, 6, 4, 3	52	1, 2, 3, 8	1, 2, 3, 7	7
20	1, 3, 4, 7	1, 2, 4, 6, 8	8, 6, 4, 2	53	1, 3, 7, 8	1, 2, 3, 4, 5, 6	6, 5, 4, 2
21	2, 3, 4, 7	1, 4, 6, 8	8, 6, 4	54	2, 3, 7, 8	1, 3, 4, 5, 6	6, 5, 4
22	1, 2, 3, 7	1, 2, 3, 6, 8	8, 6	55	2, 4, 7, 8	1, 2, 3, 5, 6	6, 5, 2
23	1, 2, 6, 7	1, 2, 3, 4, 5, 8	8, 5, 4, 3	56	1, 4, 7, 8	1, 3, 5, 6	6, 5
24	1, 3, 6, 7	1, 2, 4, 5, 8	8, 5, 4, 2	57	1, 5, 7, 8	1, 2, 3, 4, 6	6, 4, 3, 2
25	2, 3, 6, 7	1, 4, 5, 8	8, 5, 4	58	2, 5, 7, 8	1, 3, 4, 6	6, 4, 3
26	2, 4, 6, 7	1, 2, 3, 5, 8	8, 5, 3, 2	59	3, 5, 7, 8	1, 2, 4, 6	6, 4
27	1, 4, 6, 7	1, 3, 5, 8	8, 5, 3	60	4, 5, 7, 8	1, 2, 3, 6	6
28	3, 4, 6, 7	1, 2, 5, 8	8, 5	61	4, 6, 7, 8	1, 2, 3, 4, 5	5, 4
29	3, 5, 6, 7	1, 2, 3, 4, 8	8, 4, 3	62	3, 6, 7, 8	1, 2, 3, 5	5, 3
30	2, 5, 6, 7	1, 2, 4, 8	8, 4, 2	63	2, 6, 7, 8	1, 2, 5	5, 2
31	1, 5, 6, 7	1, 4, 8	8, 4	64	1, 6, 7, 8	1, 5	5
32	4, 5, 6, 7	1, 2, 3, 8	8	65	5, 6, 7, 8	1, 2, 3, 4	
33	4, 5, 6, 8	1, 2, 3, 4, 5, 6, 7	7, 6, 5, 4				

$s \notin S$. This is enough, as ∂T is independent and ∂T^* is coindependent, which are the only sets checked by the oracles. Furthermore, if incremental independence can be checked in time t_1 and incremental coindependence in time t_2 , then this gives an $O(|E|(t_1 + t_2))$ -delay algorithm for generating bases.

Note that the space requirements for bases algorithms are the sum of the space it takes to store one base and one cobase and any extra data structures we would maintain for testing incremental independence/coindependence.

1. **Uniform matroid:** For the uniform matroid $U(n, k)$ let $S \subseteq [n]$ and $s \notin S$, in addition to S we also store its size. We can check incremental independence by checking if $|S| + 1 \leq k$. Similarly, the coindependence oracle can be implemented by checking if $|S| + 1 \leq n - k$. These checks and updates take $O(1)$ time, thus we obtain a running time of $O(n)$ per generated base.

An interesting phenomenon occurs when we consider a fixed $\lambda > 1$ and $n = \lambda k$. In this case, the implementation of $\mathsf{G}_{\text{close}}$ is such that the larger element e , on average, is bounded by a constant (which depends on λ , but not on n). Since, the time actually needed to generate a new base is $O(e)$, we obtain a bound of $O_\lambda(1)$ time on average. Details on the exact bound are provided in Section 6.1.2.

2. **Graphic matroid:** Given a graph $G = (V, E)$ such that $|V| = n$ and $|E| = m$, we consider now the graphic matroid $M(G)$. The problem of incremental independence/coindependence has been well studied on graphic matroids, as in particular, these appear as subproblems on algorithms like Kruskal and Reverse-Kruskal. Tarjan [34] showed that incremental independence can be checked in time $O(\alpha(m, n))$ where α is the inverse Ackermann function. Thorup [35] showed that incremental coindependence can be checked in time $O(\log n (\log \log n)^3)$. Thus, we obtain an $O(m \log n (\log \log n)^3)$ -delay algorithm for the spanning trees of a graph.
3. **Column matroid:** Let A be an n by m matrix and consider its column matroid. Here, we note that we can check independence/coindependence by solving an $n \times n$ system of linear equations in time $O(n^3)$. Thus, we obtain a time of $O(mn^3)$ per generated base.

We summarize these results in Table 2 and have the following theorem as bookkeeping.

► **Theorem 12.** *Run-times and memory usage for algorithms in this section are in Table 2.*

■ **Table 2** The running times and memory usage by the modified greedy algorithms.

Matroid	Preprocessing	Space	Delay
Generic Matroid	$O(E)$	$O(E)$	$O(E)$ calls to the independence and coindependence oracle.
Graphic $M(G)$	$O(m + n)$	$O(m + n)$	$O(m \log n (\log \log n)^3)$
Uniform $U(n, k)$	$O(n)$	$O(n)$	$O(n)$
Uniform $U(\lambda k, k)$	$O(k)$	$O(k)$	$O_\lambda(1)$ amortized
Column $M_{\mathbb{F}}(A)$	$O(mn^2)$	$O(mn)$	$O(mn^3)$

6 Special cases

In this section, we specialize our results to a couple of interesting matroids. The goal is not to cover all of the interesting cases that can be derived from our meta-algorithms, nor is it to obtain the sharpest results in any specific case. Instead, the goal is to exemplify that our algorithms are very versatile and can be refined in various ways. Collectively, the examples in this section illustrate the four notes provided in Section 4.

- *Efficiency.* We include an algorithm that runs in constant amortized time.
- *Tiebreakers.* We illustrate that the different tiebreaker rules result in different orders.
- *Restricted operations.* We include orders that use restricted types of exchanges, including homogeneous transpositions and edge pivots.
- *Starting base.* We include orders that make use of particular starting bases and obtain stronger properties like cyclicity or homogeneity.

6.1 All your combinations: uniform matroid bases

In this subsection we consider the particular case when M is the uniform matroid $U(n, k)$. Recall that the bases in this case are all subsets of $[n]$ of size k . Furthermore, in this Section we will usually think of these base in terms of their *characteristic vector*. The characteristic vector of a set S of $[n]$ is simply a bitstring $x \in \{0, 1\}^n$ which has ones in the positions given by S . More formally,

$$x_i = \begin{cases} 1 & \text{if } i \in S, \\ 0 & \text{otherwise.} \end{cases}$$

Note that the characteristic vectors of bases of $U(n, k)$ are exactly the bitstrings $x \in \{0, 1\}^n$ with exactly k ones. We call such vectors (n, k) -combinations. In this context, a base exchange will mean simply the transposition of a 1 and a 0 in the vector. Finally, we remark that we will usually make no distinction between the subsets and their characteristic vectors.

The generation of (n, k) -combinations has been well studied and is covered in Section 7.2.1.3 of The Art of Computer Programming [15]. It is not hard to see that restriction of the binary reflected Gray code for n to (n, k) -combinations gives a *transposition* Gray code, but nevertheless, stronger constraints on the type of transpositions are also possible. We mention two which are relevant for the section. Given (n, k) -combinations x, y , we say that y is obtained by a *homogeneous transposition* from x if it is obtained by transposing elements which have only zeros in between, i.e. x_i and x_j are transposed with $i < j$ and $x_{i+1} \dots x_{j-1} = 0^{j-i-1}$. Due to a result by Eades and McKay [6] homogeneous transpositions Gray codes are known to exist and moreover, they are obtained by the following recursion for $n, k > 0$

$$EM_{n,k} = EM_{n-1,k}0, \text{ rev}(EM_{n-2,k-1})01, EM_{n-2,k-2}11, \quad EM_{n,0} = 0^n, EM_{n,n} = 1^n$$

with the convention that $EM_{n,k} = \emptyset$ for n or k negative. Note that with a homogeneous transposition Gray code, we can play all k -note chords on an n -note piano by moving exactly one finger at a time. We also consider the weaker notion of a *consistent transposition* which only requires that the bits in between the transposed elements are the same, i.e. if x_i and x_j were transposed with $i < j$, then $x_{i+1} \dots x_{j-1}$ is either 0^{j-i-1} or 1^{j-i-1} . In the piano interpretation, a consistent transposition Gray code allows us to play all k -note chords on an n -note piano by either shifting a block of contiguous fingers one note or moving exactly one finger (across possibly many notes).

6.1.1 A closer look into the closest tiebreaker

We already know that the greedy Algorithm allows us to generate combinations by transpositions, but much more can be obtained by closer examination of the listings provided by Algorithm G_{close} . A first interesting observation is that, if x_i and x_j are transposed for $i < j$ then the bits in between are either all zeros or all ones. Otherwise, there would have been a closer transposition with the same largest element j . This directly implies that the listings obtained by Algorithm G_{close} are *consistent transposition Gray codes*. We analyze this further by obtaining a recursive description of the listing obtained by Algorithm G_{close} . To this end, we introduce the *prefix function* $p : \{0, 1\}^n \rightarrow \{0, 1\}^{n-1}$ as a function which maps $x_1 \dots x_n$ to its longest proper prefix $x_1 \dots x_{n-1}$. A simple inductive argument gives the following recursive definition.

► **Lemma 13.** *Let x be an (n, k) -combination and $L(n, k, x)$ the Gray code computed by $G_{\text{close}}(U(n, k), x)$. The following recursive formulas hold for $n > k \geq 1$.*

- *If $x_n = 1$: $L(n, k, x) = L(n-1, k-1, p(x))1, L(n-1, k, 1^{k-1}0^{n-k-1}1)0$.*
- *If $x_n = 0$: $L(n, k, x) = L(n-1, k, p(x))0, L(n-1, k-1, 0^{n-k-1}1^{k-1}0)1$.*

The following remark is a direct consequence of Lemma 13

► **Remark 14.** There are only two possible final bitstrings in the execution of Algorithm G_{close} for (n, k) -combinations: $1^k 0^{n-k}$ or $0^{n-k} 1^k$. In particular, note that only the final bit of the starting bitstring decides this: if it is a 1 the final (n, k) -combination will be $1^k 0^{n-k}$ and if it is a 0 the final one will be $0^{n-k} 1^k$.

By Remark 14, we know that starting Algorithm G_{close} with $1^{k-1} 0^{n-k} 1$ will end up with a *cyclic* Gray code. Furthermore, this will be a consistent transposition Gray code. We summarize the results of this subsection in the following corollary.

► **Corollary 15.** *Running Algorithm G_{close} on $U(n, k)$ starting from any (n, k) -combination gives a consistent transposition Gray code. Moreover, this Gray code is cyclic (in the consistent transposition sense) if and only if the starting bitstring is either $1^{k-1} 0^{n-k} 1$ or $0^{n-k-1} 1^k 0$.*

6.1.2 Amortized analysis: Average larger label

In this subsection we state the result anticipated in Section 5.2. The proof is omitted due to space considerations.

► **Lemma 16.** *Let x be an (n, k) -combination. We denote by $\text{avg}(n, k, x)$ the average larger label involved in a transposition in the listing produced by $G(U(n, k), x)$. The following holds:*

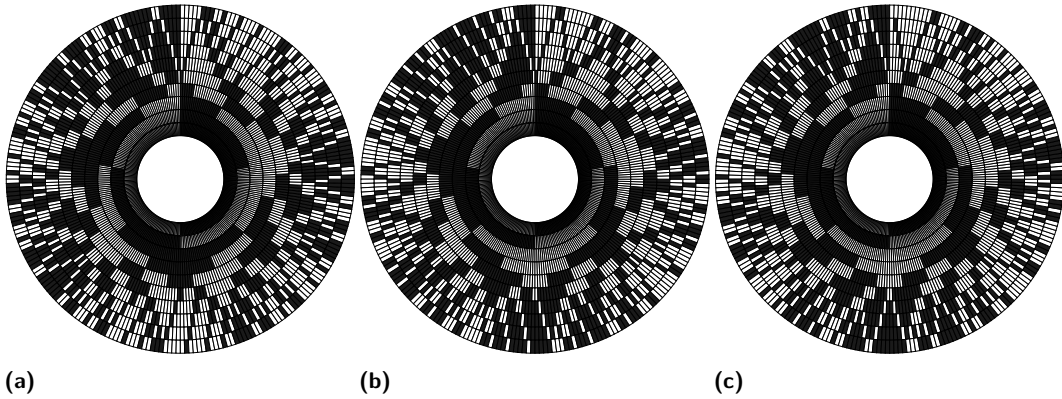
$$\text{avg}(n, k, x) \leq \frac{5n^2}{k(n-k)} + O(1).$$

In particular, if $\lambda > 1$ and $n = \lambda k$, then

$$\text{avg}(\lambda k, k, x) \leq \frac{5\lambda^2}{(\lambda-1)} + O(1) = O_\lambda(1).$$

6.1.3 Homogeneous transpositions

The reader may wonder whether Algorithm G_{close} gives a *homogeneous* transposition Gray code. This is not true as transpositions over contiguous blocks of ones are utilized (see Figure 12). Interestingly, we can fulfill this stronger requirement just by choosing a suitable tiebreaker for Algorithm G . Furthermore, the tiebreaker rule is very natural as it just adds the *homogeneous* requirement to the furthest tiebreaker rule as follows.



■ **Figure 12** Illustration of three gray codes for $(10, 5)$ -combinations. The strings appear in clockwise order, starting at 12 o'clock. The inner track is the leftmost bit and the outer track the rightmost bit, where 1-bits are drawn white and 0-bits are drawn black. (a) The homogeneous transposition Gray code for $(10, 5)$ -combinations produced by the Eades-McKay algorithm (b) The homogeneous transposition Gray code for $(10, 5)$ -combinations produced by $G_{\text{hom}}(10, 5, 1111100000)$ (c) The consistent transposition Gray code for $(10, 5)$ -combinations produced by $G_{\text{close}}(10, 5, 1111100000)$.

Algorithm G_{hom} (*Greedy combinations w/ homogeneous transpositions*). This algorithm attempts to generate all (n, k) -combinations by homogeneous transpositions starting from x^0 .

H1. [Initialize] Visit the initial (n, k) -combination x^0 .

H2. [Greedy] Generate an unvisited (n, k) -combination by performing a homogeneous transposition that minimizes the larger element, and then breaks ties by minimizing the smaller element. If no such transposition exists, then terminate. Otherwise visit the resulting (n, k) -combination and repeat H2.

This is not the first greedy approach for homogeneous transposition Gray codes, as such an approach was given by Williams in 2013 [44]. It is interesting to note that the greedy approach is different to both the one by Eades-McKay (see Figure 12 for a sample comparison with Eades-McKay's approach). On the other hand, even though the approach by Williams has a different description, it is not hard to see that it generates the same listings as G_{hom} .

We note that it is not immediate from Theorem 10 that Algorithm G_{hom} generates *all* (n, k) -combinations, as we have restricted ourselves to only homogeneous transpositions (instead of all possible transpositions or base exchanges). In particular, it is not true that Algorithm G_{hom} generates all (n, k) -combinations independently of the starting combination, as it fails to do so for some of them. Interestingly enough, we can ensure that Algorithm G_{hom} works, whenever the starting bitstring has all of its ones consecutively. More formally, for $a \in \{0, \dots, n-k\}$ define $s_{n,k,a}$ as the (n, k) -combination which has all of its ones consecutively and begins with a zeroes, that is $0^a 1^k 0^{n-k-a}$. We have the following theorem.

► **Theorem 17.** *For every $a \in \{0, \dots, n-k\}$, $G_{\text{hom}}(n, k, s_{n,k,a})$ generates all (n, k) -combinations.*

The proof is by induction and it is omitted due to space considerations.

6.2 Your biggest fan: graphic matroid bases and pivot Gray codes

As mentioned earlier, the bases of a graphic matroid are the spanning trees of the associated graph. Our results provide *edge-exchange Gray codes*, meaning that one edge is added and another one is deleted to obtain the next spanning tree. Recently, a new type of spanning tree Gray code was introduced by Cameron, Grubb, and Sawada [3]. A *pivot Gray code* refines an exchange as follows: The two edges involved in the exchange must share a common vertex u . In other words, the operation is *u-pivot* (or simply a *pivot*), meaning that an edge changes its other endpoint by pivoting around u .

Besides introducing this new concept, the authors of [3] provide a specific pivot Gray code for the family of graphs known as fans. The *fan graph* F_n contains n vertices and $(n-2) + (n-1) = 2n-3$ edges that are organized into a *path* of length $n-1$, along with an edge from each of these path vertices to an additional vertex known as the *handle*. The name of these graphs comes from their natural planar embedding (i.e., with every vertex on the outer face) that resembles an unfolded fan.

The Gray code in [3] is also constructed greedily, but in a vertex-centric manner. They label the vertices of the path $2, 3, \dots, n$ and the handle as ∞ , and then they aim to minimize the vertex that supports a pivot to a new spanning tree. More specifically, their greedy rule is stated as follows:

Prioritize the pivots u from smallest to largest and then for each pivot, prioritize the edges uv that can be removed from the current tree in increasing order of the label on v , and for each such v , prioritize the edges uw that can be added to the current tree in increasing order of the label on w .

By starting at a specific spanning tree – the path together with edge 2∞ – an understandable recursive structure is obtained. From this recursive structure, the authors of [3] are able to obtain algorithmic results including efficient generation and ranking / unranking.

At first glance, the results in [3] appear to have little in common with the general results presented in this paper. After all, we consider exchange operations not pivots, and we prioritize edges rather than vertices. However, that initial judgement proves to be incorrect.

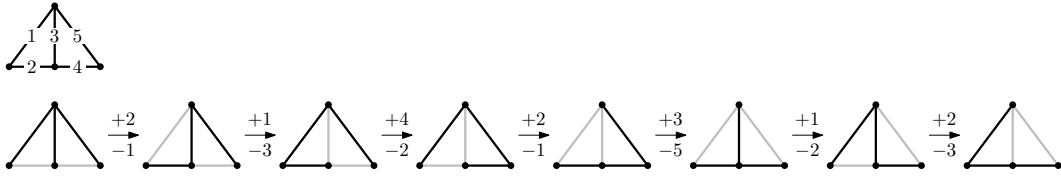
By carefully setting our initial conditions, we are able to recreate the combinatorial result in [3]. In other words, we can also create pivot Gray codes for the fan graph. In fact, we strengthen the combinatorial result in two ways.

1. There is a *face pivot Gray code* of F_n 's spanning trees. This means that the pivot operation is further refined as follows: The added and deleted edges are consecutive in one of the faces of F_n 's natural planar embedding⁵.
2. There is a face pivot Gray code starting from any spanning tree of F_n .

To obtain our results, we label the edges in increasing order from left-to-right as shown in Figure 13. Then Algorithm G_{close} will generate a face pivot Gray code, regardless of the starting spanning tree, with one such order illustrated in Figure 13.

► **Theorem 18.** *Algorithm G_{close} generates a face pivot Gray code for the spanning trees of the fan graph F_n , regardless of the initial spanning tree, so long as it uses the left-to-right labeling.*

⁵ This stronger property is not directly mentioned in [3], but it might hold. In particular, the example order F_6 in Figure 4 only uses face pivots.



■ **Figure 13** A face pivot Gray code of the fan graph F_4 .

Proof. By Theorem 10, we know that Algorithm G_{close} generates exchange Gray codes under these conditions. Consider one such Gray code, and let T be an arbitrary spanning tree in this order, so long as it is not the last one. Let $T' = T \Delta \{e, f\}$ be the next spanning tree in the order with $f < e$. To complete the proof, we simply need to show that this exchange is a face pivot. We proceed in two cases.

- Case 1: $e \notin T$. Let us consider the fundamental cycle C in $T + e$. Observe that C must include exactly two handle edges, say g and h , and some positive number of path edges, say p_1, p_2, \dots, p_k . Without loss of generality, we can assume $g < p_1 < p_2 < \dots < p_k < h$ due to the left-to-right labeling. More specifically, we have the following equalities:

$$g = p_1 - 1 \quad p_1 = p_2 - 2 \quad p_2 = p_3 - 2 \quad \dots \quad p_{k-1} = p_k - 2 \quad p_k = h - 1 \quad (3)$$

We now consider the possibilities for e in turn.

- Note that $e \neq g$, since otherwise e would be a loop in the e -prefix minor, and hence, Algorithm G_{close} would not use it in an exchange.
- Consider $e = h$. In this case, the closest tiebreaker used in Algorithm G_{close} will select $f = p_k$ due to the rightmost equality in (3). Since $e = h$ and $f = p_k$ are on the same triangle face, this exchange is a face pivot.
- Consider $e = p_1$. In this case, the closest tiebreaker used in Algorithm G_{close} will select $f = g$ due to the leftmost equality in (3). Since $e = p_1$ and $f = g$ are on the same triangle face, this exchange is a face pivot.
- Consider $e = p_i$ for some $i > 1$. Note that the handle edge $p_i - 1 \notin T$, since otherwise there would be a cycle in T . Therefore, Algorithm G_{close} cannot select $f = p_i - 1$. The next smallest edge is $f = p_i - 2 = p_{i-1}$ due to the non-edge cases of (3), so Algorithm G_{close} will select this by using the closest tiebreaker. Since $e = p_i$ and $f = p_{i-1}$ are on the outer face, this exchange is a face pivot.
- Case 2: $e \in T$. This case is similar to the previous case. ◀

7 Final remarks

In this paper, we proved that greedy algorithms are fundamental to matroids in a new way. More specifically, we showed that a simple meta-algorithm can be used to generate exhaustive lists of bases using exchanges. We then provided iterative implementations for generating these lists. Finally, we specialized our results for specific matroids and tiebreaker rules. The authors have observed that similar results can be established for the independent sets of a matroid, and we look forward to establishing these results in the full version of this paper. Furthermore, we also plan to provide efficient implementations for specific matroids in the Combinatorial Object Server [5].

References

- 1 B. Alspach and G. Liu. Paths and cycles in matroid base graphs. *Graphs Combin.*, 5(3):207–211, 1989.
- 2 Bad_CRC. all your base are belong to us. www.newgrounds.com/portal/view/11940, 2001.
- 3 Ben Cameron, Aaron Grubb, and Joe Sawada. A pivot gray code listing for the spanning trees of the fan graph. In *International Computing and Combinatorics Conference*, pages 49–60. Springer, 2021.
- 4 Jean Cardinal, Arturo Merino, and Torsten Mütze. Efficient generation of elimination trees and hamilton paths on graph associahedra. In *Proceedings of the Thirty-third ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*. SIAM, Philadelphia, PA, 2022.
- 5 The Combinatorial Object Server: <http://www.combos.org/>.
- 6 P. Eades and B. McKay. An algorithm for generating subsets of fixed size with a strong minimal change property. *Inform. Process. Lett.*, 19(3):131–133, 1984.
- 7 Jack Edmonds. Matroids and the greedy algorithm. *Mathematical programming*, 1(1):127–136, 1971.
- 8 Fandom contributors. CATS - Villains Fandom. villains.fandom.com/wiki/CATS, 2022.
- 9 Frank Gray. Pulse code communication, 1953.
- 10 E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1214–1225. SIAM, 2020.
- 11 E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. I. Fundamentals. *Transactions of the American Mathematical Society*, 2022.
- 12 Hung P. Hoang and Torsten Mütze. Combinatorial generation via permutation languages. II. Lattice congruences. *Israel J. Math.*, 244:359–417, 2021.
- 13 C. A. Holzmänn and F. Harary. On the tree graph of a matroid. *SIAM J. Appl. Math.*, 22:187–193, 1972.
- 14 Know your meme contributors. All your base are belong to us – Know your meme. knowyourmeme.com/memes/all-your-base-are-belong-to-us, 2022.
- 15 D. E. Knuth. *The art of computer programming. Vol. 4A. Combinatorial algorithms. Part 1*. Addison-Wesley, Upper Saddle River, NJ, 2011.
- 16 Yue Li and Joe Sawada. Gray codes for reflectable languages. *Inf. Proc. Letters*, 109(5):296–300, 2009.
- 17 Mandelin, Clyde. How Zero Wing’s “all your base” translation compares with the Japanese script. <https://legendsoflocalization.com/lets-take-a-peek-at-zero-wings-all-your-base-translation/>, 2014.
- 18 Stephen B. Maurer. Matroid basis graphs. I. *J. Comb. Theory Ser. B*, 14:216–240, 1973.
- 19 Stephen B. Maurer. Matroid basis graphs. II. *J. Comb. Theory Ser. B*, 15:121–145, 1973.
- 20 Arturo I. Merino and Torsten Mütze. Efficient generation of rectangulations via permutation languages. In *Proceedings of the 37th International Symposium on Computational Geometry (SoCG 2021), Buffalo, NY, USA, June 7–11 (Virtual Conference)*, volume 189 of *LIPIcs*, pages Art. No. 54, 18 pp. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2021.
- 21 Torsten Mütze. Combinatorial Gray codes – an updated survey. *arXiv preprint 2202.01280*, 2022.
- 22 D. J. Naddef and W. R. Pulleyblank. Hamiltonicity and combinatorial polyhedra. *J. Combin. Theory Ser. B*, 31(3):297–312, 1981.
- 23 J. G. Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.
- 24 Frank Ruskey, Joe Sawada, and Aaron Williams. Binary bubble languages and cool-lex order. *J. Comb. Theory, Series A*, 119(1):155–169, 2012.
- 25 Frank Ruskey and Aaron Williams. The coolest way to generate combinations. *Discrete Mathematics*, 309(17):5305–5320, 2009.
- 26 Ahmad Sabri. *Gray codes and efficient exhaustive generation for several classes of restricted words*. PhD thesis, Université de Bourgogne, 2015.

- 27 J. Sawada, A. Williams, and D. Wong. Inside the binary reflected gray code: Flip-swap languages in 2-gray code order. In *International Conference on Combinatorics on Words*, pages 172–184. Springer, 2021.
- 28 Joe Sawada and Aaron Williams. A universal cycle for strings with fixed-content (which are also known as multiset permutations). In *Workshop on Algorithms and Data Structures*, pages 599–612. Springer, 2021.
- 29 Malcolm J. Smith. Generating spanning trees. Master’s thesis, University of Victoria, 1997.
- 30 Brett Stevens and Aaron Williams. The coolest order of binary strings. In *International Conference on Fun with Algorithms*, pages 322–333. Springer, 2012.
- 31 Brett Stevens and Aaron Williams. The coolest way to generate binary strings. *Theory of Computing Systems*, 54(4):551–577, 2014.
- 32 Tadao Takaoka. $O(1)$ time algorithms for combinatorial generation by tree traversal. *The Computer Journal*, 42(5):400–408, 1999.
- 33 D. T. Tang and C. N. Liu. Distance-2 cyclic chaining of constant-weight codes. *IEEE Trans. Computers*, C-22:176–180, 1973.
- 34 Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM (JACM)*, 22(2):215–225, 1975.
- 35 M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC ’00, pages 343–350, New York, NY, USA, 2000. Association for Computing Machinery.
- 36 Takeaki Uno. A new approach for speeding up enumeration algorithms and its application for matroid bases. In *Computing and combinatorics (Tokyo, 1999)*, volume 1627 of *Lecture Notes in Comput. Sci.*, pages 349–359. Springer, Berlin, 1999.
- 37 Vincent Vajnovszki. Gray code order for Lyndon words. *Discrete Math. Theor. Comput. Sci.*, 9(2):145–151, 2007.
- 38 Vincent Vajnovszki and Rémi Vernay. Restricted compositions and permutations: from old to new gray codes. *Information Processing Letters*, 111(13):650–655, 2011.
- 39 P. Vámos. On the representation of independence structures. Unpublished manuscript., 1968.
- 40 Hassler Whitney. On the abstract properties of linear dependence. *Amer. J. Math.*, 57(3):509–533, 1935.
- 41 Wikipedia contributors. All your base are belong to us – Wikipedia, the free encyclopedia. en.wikipedia.org/wiki/All_your_base_are_belong_to_us, 2022.
- 42 Wikipedia contributors. Zero wing – Wikipedia, the free encyclopedia. en.wikipedia.org/wiki/Zero_Wing, 2022.
- 43 Aaron Williams. Loopless generation of multiset permutations using a constant number of variables by prefix shifts. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 987–996. SIAM, 2009.
- 44 Aaron Williams. The greedy Gray code algorithm. In *Algorithms and data structures*, volume 8037 of *Lecture Notes in Comput. Sci.*, pages 525–536. Springer, Heidelberg, 2013.
- 45 Aaron M. Williams. *Shift Gray codes*. PhD thesis, University of Victoria, 2009.

Playing Guess Who with Your Kids

Ami Paz 

LISN, CNRS & Paris Saclay University, France

Liat Peterfreund 

LIGM, CNRS & Gustav Eiffel University, Champs-sur-Marne, France

Abstract

Guess who is a two-player search game in which each player chooses a character from a deck of 24 cards, and has to infer the other player’s character by asking yes-no questions. A simple binary search strategy allows the starting player find the opponent’s character by asking 5 questions only, when the opponent is honest.

Real-life observations show that in more realistic scenarios, the game is played against adversaries that do not strictly follow the rules, e.g., kids. Such players might decide to answer all questions at once, answer only part of the questions as they do not know the answers to all, and even lie occasionally. We devise strategies for such scenarios using techniques from error-correcting and erasure codes. This connects to a recent line of work on search problems on graphs and trees with unreliable auxiliary information, and could be of independent interest.

2012 ACM Subject Classification Theory of computation → Representations of games and their complexity; Theory of computation → Sorting and searching; Theory of computation → Theory and algorithms for application domains

Keywords and phrases Guess Who?, Binary Search, Error Correcting Codes, Erasure Codes

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.23

Acknowledgements The authors are grateful to Yonatan, Yuval and Ari for inspiring this work, and to Ben Lee Volk for discussions on error correcting codes.

1 Introduction

Guess Who? is a two-player board game where the goal of each player is to guess the identity of its opponent chosen character. Each player has a board that includes cartoon images of 24 characters. At the beginning of the game, each player chooses one of the characters, and the goal of the other player is to guess this character. The objective of the game is to be the first player to determine which character was selected by the opponent. Players alternately ask their opponents various yes-no questions on their selected card, such as “Does your character wear glasses?”, “Is your character a woman?” etc. The player then eliminates candidates based on the opponent’s response by flipping the appropriate characters down. The game ends as soon as there is a player with a single character left on their board.

Nica [18] performed a very interesting game-theoretical analysis of the game. By analyzing it as a zero-sum stochastic game, he was able to show that at each round, a player with more cards left should take a “bold move”, and ask a question that has the potential of eliminating many characters, while the leading player (with less characters left) should stick to a traditional binary search strategy.

In this work, we take an algorithmic perspective. We hence ignore the game-theoretic competitive considerations that paved the way in Nica’s work, and instead, focus only on the underlying algorithmic search problem. To this end, we analyze a version of the game where one player is the *chooser*, which chooses a character answers questions on it, while the



© Ami Paz and Liat Peterfreund;
licensed under Creative Commons License CC-BY 4.0
11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 23; pp. 23:1–23:10



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

other player is the *guesser*, who has to ask questions in order to correctly guess the chosen character. Thus, the focus of this paper is on strategies for the guesser. Doing so, we follow the footsteps of Knuth [15], who applied a similar approach studying the Mastermind game.

Naturally, against a well-behaved chooser, the optimal strategy will be a binary search. However, in real-world applications, such as playing Guess Who with young kids, a guesser may need to devise a strategy that yields a correct guess despite a chooser that *deviates* from the game protocol. Inspired by real-life experience, we consider several ways in which a chooser might deviate from the protocol.

A *Not-At-Time-ANswering* (NATAN) adversary is an impatient chooser, which refuses to answer one question at a time, and instead, gives the guesser a single opportunity to present all their questions, and answers them altogether. In terms of theoretical computer science, we can think of this as an asynchronous adversary, or alternatively, we can say that the guesser is *non-adaptive*. In Section 3, we show that there is a simple strategy that allows to perform a binary search even against this adversary, thus guessing in an optimal number of $\lceil \log n \rceil$ questions.

An *Answers Relatively Insecure* (ARI) adversary is a chooser that is having hard time determining if a character has some property, and thus, might answer “I do not know” and not only “Yes” or “No”. This adversary, in addition to modeling a young kid, is also relevant in the context of communication in a noisy channel. Thus, it may not come as a surprise to the reader that in Section 4, we show how to overcome the indecisiveness of the adversary by using *erasure codes*.

Finally, a Valid At Large (VAL) player is a chooser that has a flexible view of the truth. Hence, this chooser might answer incorrectly to some fraction of the questions. In a game, we give this player the benefit of the doubt that they are merely mistaken, but in theoretical computer science, we would say we are dealing with a *malicious* player, or with a *Byzantine* agent. Section 5 shows that this behavior can be overcome using *error correcting codes*, with an overhead of a few questions.

The applicability of our approach in real-world scenarios is still to be proven, and an experimental follow-up work is now being considered. Regardless of this aspect, we believe that our work presents simple applications of basic theoretical computer science concepts such as binary search, erasure codes and error correcting codes, and as such can be used for introductory examples for a wide audience.

Related Work

As mentioned, Nica [18] was the first to conduct a rigorous analysis of the Guess Who game. His work stayed in the framework of the game, and used a stochastic analysis for determining when a player should make a “bold guess”, that in a small probability will result in the elimination of many characters. We note that such an analysis is useless, e.g., against the NATAN adversary, as it crucially relies on the round-based nature of the game.

The ARI adversary resembles scenarios in which some of the information is missing. Handling incomplete or missing information is one of the longstanding topics in database research [16, 14, 19, 4]. In particular, the study of certain answers (answers that are true regardless of how one completes the missing entries of the queried database) has attracted lots of attention in the database community [22, 7].

The VAL adversary presents a setting of a search with partially incorrect information, that is of high research interest in recent years. This is the case, e.g., when studying search and navigation problems in graphs, where the information regarding the target’s location might be unreliable [3, 12, 13, 8]. A question even more similar to ours is that of playing twenty questions game with a liar [6], which has tight connections to learning theory.

(a) Characters.¹

	glasses	no glasses	hat	blond	back hair	brown hair	white hair	glasses and bald
Bill	0	1	0	0	0	0	0	0
Charles	0	1	0	1	0	0	0	0
Claire	1	0	1	0	0	0	0	0
Joe	1	0	0	1	0	0	0	0
Maria	0	1	1	0	0	1	0	0
Max	0	1	0	0	1	0	0	0
Sam	1	0	0	0	0	0	1	1
Susan	0	1	0	0	0	0	1	0
Tom	1	0	0	0	1	0	0	1

(b) A corresponding $\mathbf{T}_{9 \times 8}$ matrix.

■ Figure 1 Running Example.

Guess Who was invented as a simple version of the *Bulls and Cows* game (whose commercial version is known under the name “Mastermind”), another search game where the questions and answers are more involved than the binary questions applied here. In Mastermind, players have to guess their opponent’s 4-peg colorful pattern, while the opponent replying to each guess by specifying the number of correct pegs used in place, and those that are correct but misplaced. A word-based variant of Mastermind is the trendy Wordle game. While the scientific literature on Wordle is rather limited (though numerous analyses, discussions and strategies can be found online), Mastermind was studied in several occasions. In 1977, Knuth [15] devised an optimal strategy for Mastermind, in only 5 guesses. Interestingly, he analyzed the game as a game with a single guesser and ignored game-theoretic aspects yielding from the fact that the game has two players, as we do here. In recent decades, several variants [21, 1] and approaches to Mastermind were studied, most notably is the *evolutionary* approach [9, 5, 11, 10, 2].

2 Formal Framework

In this section, we set the formal settings based on our algorithmic perspective.

2.1 Characters and Traits

We denote the number of *characters* by n , and refer to each character by a unique identifier in the set $[n] := \{1, \dots, n\}$. We denote possible (physical) *traits* (of the characters) by $[m]$. A trait can be being brunette, not wearing a hat, having a big nose and being blond etc. Since players can only ask yes-no questions, we assume that all possible questions are of the form

$$q(j) := \text{Does your character possess trait } j?$$

where $1 \leq j \leq m$.

¹ Image taken from iSLCOLLECTIVE.com

23:4 Playing Guess Who with Your Kids

The information on the different characters and their traits can be represented as a binary matrix whose rows correspond with characters' identifiers, and columns with traits. Formally, the *traits' matrix*, is a binary $n \times m$ matrix \mathbf{T} , where $M_{i,j} = 1$ if character i possess trait j , and $M_{i,j} = 0$ if character i does not possess trait j . In this way, every row of the matrix describes a character.

► **Example 1.** Figure 1a presents a subset of characters of the original game and Figure 1b a corresponding matrix with traits listed as its columns.

2.2 Distinguishability

We assume that each two different characters are *distinguishable*, that is, have at least one different trait. Formally, for each $i \neq i' \in [n]$ there is $j \in [m]$ such that $\mathbf{T}[i][j] \neq \mathbf{T}[i'][j]$. This ensures that no matter which character was chosen by the chooser, the guesser can ask questions on its traits until revealing its identity.

In fact, similarly to [18], we assume even a stronger assumption – not only do every two characters are distinguishable, but also any subset of characters is distinguishable from the rest. We formalize this as follows:

► **Assumption 1.** For any subset $A \subseteq [n]$ of characters, there is a trait j_A such that $A = \{i \in [n] \mid \mathbf{T}[i][j_A] = 1\}$.

We call j_A the trait that *separates* A .

► **Example 2.** The trait that separates the subset consisting of Sam and Tom from the rest is “having glasses and being bald.” For Sam, Tom, Joe and Claire, it is simply “having glasses”.

From this point on, we only consider games for which Assumption 1 holds.

2.3 Games, Runs, and Classical Strategies

We view an n -character game, or, simply, a game, as a process in which a player, namely *the guesser*, needs to search in the n -characters' space the character chosen by the other player, namely *the chooser*. Classically, this search is done by eliminating elements from $[n]$ until reaching a single one. For simplicity, we assume that a player does not need to explicitly guess the other player's character after narrowing down the list of possible characters to one; on the other hand, a player cannot guess a character if its list is larger than one. Formally, we define a *run* ρ as a sequence:

$$q_1, a_1, I_1, \dots, q_k, a_k, I_k$$

where for every $1 \leq i \leq k$ it holds that $I_i \subseteq [n]$, q_i and a_i are, respectively, the question and answer of round i of the game, and the following hold for every $1 \leq \ell \leq k$:

- if $q_\ell = q(j)$ and $a_\ell = \text{yes}$ then $I_\ell = \{i \in I_{\ell-1} \mid \mathbf{T}[i][j] = 1\}$;
- if $q_\ell = q(j)$ and $a_\ell = \text{no}$ then $I_\ell = \{i \in I_{\ell-1} \mid \mathbf{T}[i][j] = 0\}$,

where we set $I_0 := [n]$.

We say that q_1, \dots, q_k *defines* ρ . In addition, we say that ρ is *successful* if I_k is a singleton. If ρ is successful, we say that q_1, \dots, q_k *defines* a successful run.

► **Example 3.** Consider the run $q_1, a_1, I_1, q_2, a_2, I_3$ where I_0 consists of all characters that appear in Figure 1a, $q_1 = q(\text{hat})$, $a_1 = \text{yes}$, $I_1 = \{\text{Claire, Maria}\}$, $q_2 = q(\text{brown hair})$, $a_2 = \text{yes}$, and $I_3 = \{\text{Maria}\}$.

A *classical strategy* is a procedure that, given a run $q_1, a_1, I_1, \dots, q_i, a_i, I_i$, either chooses the next question q_{i+1} , or terminates if the run is successful. If, for every chosen character, the procedure terminates after choosing at most k questions, we say it is a k -question strategy.

2.4 Our Gallery of Adversaries

In this paper we consider several adversaries who deviate from the standard game protocol. In order to play with each of them, we have to adjust the notion of a strategy.

Impatient NATAN. The first adversary we consider is the *impatient* adversary denoted NATAN, which answers all the questions at once. A run against such an adversary is a normal run, but the strategy is non-adaptive. Formally, a k -question *predetermined-strategy* is a set $\{q_1, \dots, q_k\}$ of questions where $q_\ell := q(j_\ell)$, such that the following set is a *singleton*:

$$\{i \mid \forall \ell : ((a_\ell = \text{yes}) \text{ implies } \mathbf{T}[i][j_\ell] = 1 \text{ and } (a_\ell = \text{no}) \text{ implies } \mathbf{T}[i][j_\ell] = 0)\}$$

where a_ℓ is a correct answer to q_ℓ for every ℓ . That is, there is always a single character determined by answers to these questions. Alternatively, using our previous definitions, we can define k -question *predetermined-strategy* as a set $\{q_1, \dots, q_k\}$ of questions, such that each sequence obtained by a permutation of its elements defines a successful run.

Clueless ARI. The second type of adversaries we consider is *clueless* adversaries that might occasionally say they do not know the answer. ARI is such an adversary that might answer “don’t know” at most once. To deal with clueless adversaries, we slightly extend the definition of runs by allowing $a_\ell = \text{don’t know}$, and setting $I_\ell = I_{\ell-1}$ in this case. A *strategy* against a clueless adversary is defined similarly to a classical strategy, while considering the extended notion of run. It is said to be a k -question *strategy* if it terminates after at most k questions for any chosen character.

In fact, our strategy for the clueless adversary is even more elaborate, and holds against an adversary that is *both impatient and clueless*. As the reader may expect, this adversary answers all the questions at once, and might answer “don’t know” on some fraction of them. We define the notion of k -question *predetermined strategy* similarly to before. Note that the alternative definition based on successful runs is valid also here by considering the extended notion of run.

Liar VAL. Finally, we consider *liar* adversaries. VAL, for example, is such an adversary that can lie at most once in a game.

A k -question *predetermined strategy* for an impatient liar that lies at most d times is a set $\{q(j_1), \dots, q(j_k)\}$ of questions such that for there is a unique $i \in [n]$ for which

$$d_{\text{HAM}}((b_1, \dots, b_k), v^i) \leq d$$

where d_{HAM} stands for the Hamming distance (that is, the number of entries in which two vectors differ), $v^i := (\mathbf{T}[i][j_1], \dots, \mathbf{T}[i][j_k])$, a_ℓ is the answer to $q(j_\ell)$, and $b_\ell := \begin{cases} 1 & a_\ell = \text{yes} \\ 0 & a_\ell = \text{no} \end{cases}$.

Intuitively, since the adversary lies, the guesser can only “approximate” the chosen character, i.e., find a vector that differs from that of the chosen character by at most d bits. The uniqueness of i ensures that the guesser can determine who is the chosen character based on the approximate vector she gets.

3 Playing with Honest (but Sometimes Lazy) Players

We start by considering settings in which the chooser is honest, i.e., always answers correctly. The following result is based on the very basic strategy of binary search.

► **Theorem 4.** *For every n -character game, there is a $\lceil \log n \rceil$ -question strategy. In addition, there is no k -question strategy with $k < \lceil \log n \rceil$.*

Proof. The theorem is achieved by a simple binary search algorithm, defined recursively. Given a set I_ℓ with $|I_\ell| > 1$, we choose $A \subseteq I_\ell$ with $|A| = \lfloor |I_\ell|/2 \rfloor$, and use Assumption 1 to find a trait j with $A = \{i \mid \mathbf{T}[i][j] = 1\}$. We ask $q_{\ell+1} = q(j)$ to get an answer $a_{\ell+1}$, and set $I_{\ell+1} = \{i \mid \mathbf{T}[i][j] = a_{\ell+1}\}$.

It is immediate to see that the algorithm defines a successful run $q_1, a_1, I_1, \dots, q_k, a_k, I_k$ with $|I_k| = 1$. Indeed, using the inequality $|I_{\ell+1}| \leq \lfloor |I_\ell|/2 \rfloor \leq (|I_\ell| + 1)/2$, a simple induction, and the fact that $|I_0| = n$, we get $|I_k| \leq \frac{n+2^k-1}{2^k}$. For $k = \lceil \log n \rceil \geq \log n$, this implies $|I_k| \leq 2 - 1/n$ and the integrality of $|I_k|$ guarantees $|I_k| = 1$ as claimed.

For the second part of the theorem, let us assume that there is a k -question strategy with questions q_1, \dots, q_k . Each successful run that is consistent with this strategy is of the form $I_0, q_1, a_1, I_1, \dots, q_k, a_k, I_k$ where for every ℓ , $a_\ell \in \{\text{yes}, \text{no}\}$ and $I_{\ell+1} \subseteq I_\ell$. Note that $I_{\ell+1}$ is uniquely determined by a_1, \dots, a_k and thus the number of possible I_k s is bounded by the number of possible vectors a_1, \dots, a_k . Hence, there are at most 2^k successful runs that are consistent with this strategy. If $k < \lceil \log n \rceil$ then $2^k < n$ which implies that q_1, \dots, q_k cannot be a k -question strategy (since there are n characters). ◀

Notice that this strategy is adaptive, that is, at each round the guesser chooses the next question based on the outcome of the previous one. Is it possible to find a strategy that is not adaptive, that is, a strategy against an impatient adversary? We answer this in the affirmative.

► **Theorem 5.** *For every n -character game, there is a $\lceil \log n \rceil$ -question predetermined-strategy.*

Proof. For $1 \leq \ell \leq \lceil \log n \rceil$, let $A_\ell = \{i \in [n] \mid i[\ell] = 1\}$, where $i[\ell]$ is the bit in location ℓ of the binary expansion of i (whose first bit is $i[1]$). For each ℓ in the range, we set q_ℓ to be the trait separating A_ℓ . (It exists due to Assumption 1.)

To prove this strategy always assures termination in $\lceil \log n \rceil$ questions, we consider the run $q_1, a_1, I_1, \dots, q_{\lceil \log n \rceil}, a_{\lceil \log n \rceil}, I_{\lceil \log n \rceil}$ induced by the aforementioned questions. The choice of questions guarantees $I_\ell = \{i \mid i[1] = a_1, \dots, i[\ell] = a_\ell\}$. Therefore, $I_{\lceil \log n \rceil}$ contains solely the element i with binary encoding $a_{\lceil \log n \rceil} a_{\lceil \log n \rceil - 1} \dots a_2 a_1$. ◀

This “static” approach can be useful when playing with a *Not-At-Time-ANSwering* (NATAN) player that answers the questions altogether. Therefore, we can conclude the following.

► **Corollary 6.** *The classical Guess Who game has a 5-question predetermined-strategy, i.e., a strategy that works even against NATAN.*

4 Playing with Clueless Players

In this section, we consider *clueless* players, that do not always know the answer to a question. A clueless player might answer, in addition to “yes” and “no”, the answer “I do not know” (“don’t know”, for brevity). One way to circumvent this answer is, whenever the player answers “don’t know” on some trait, to ask a question on a different trait that has exactly

the same answers on all characters. However, this requires the assumption that such a trait always exists, and in addition, it does not work against an adversary that is both clueless and impatient. Instead, we suggest a different solution, using *erasure codes*.

► **Definition 7** ([20]). *An erasure code that can handle d erasures is a collection of binary vectors, namely code-words, such that there is a recovery algorithm that gets as an input each of these vectors with at most d locations replaced by “?”, and returns the original vector.*

We restrict our attention to codes where all the code-words have equal length. A simple erasure code with $d = 1$ can be achieved, for example, by adding a parity bit – the recovery algorithm in this case will replace the “?” with a value complementing the number of 1-s in the vector to be even. A code that can handle more erasures can be achieved, e.g., by taking a collection of low-degree polynomials, and producing a code-word from each polynomial p by setting the i -th place of the code-word to $p(i)$.

Before presenting the result, let us set its formal setup.

► **Theorem 8.** *If there is an erasure code composed of at least n binary vectors of length k , that can handle d erasures, then for every n -character game there is a k -question predetermined-strategy against an adversary that answers “don’t know” at most d times.*

Proof. Assign each character i with a unique vector v_i from the code. For $1 \leq \ell \leq k$, let $A_\ell = \{i \mid v_i[\ell] = 1\}$ be all the characters with the ℓ -th entry of their vector equal to 1. Let j_ℓ be the trait that separates A_ℓ from \bar{A}_ℓ .

We use the question sequence $q_1 := q(j_1), \dots, q_k := q(j_k)$, which defines a run $q_1, a_1, I_1, \dots, q_k, a_k, I_k$. From the sequence (a_1, \dots, a_k) of answers, which contains at most d answers of “?”, we find the unique way to complement the vector into a code-word $v' = (a'_1, \dots, a'_k)$. The character i with $v_i = v'$ is then the desired character.

For correctness, consider the true character i^* chosen, and the sequence (a_1^*, \dots, a_k^*) of the true answers to the questions, where a_ℓ^* is the answer on $q(j_\ell)$. Note that the choice of questions guarantees that the following are equivalent: $a_\ell^* = 1$; $i^* \in A_\ell$; and $v_{i^*}[\ell] = 1$. This concludes in $v_{i^*}[\ell] = a_\ell^*$, from which we have $v_{i^*} = (a_1^*, \dots, a_k^*)$. By the assumption of at most d answers are missing in the answer vector (a_1, \dots, a_k) , and the properties of the erasure code ensure that the completion of this vector is unique, and is v_{i^*} . Hence, the algorithm returns i^* , as desired. ◀

One application of the above theorem is when the adversary may answer “don’t know” at most once. For this, we use simple parity check bit. That is, by concatenating a parity bit to every $\lceil \log n \rceil$ -bit binary vector, we get the following code.

► **Lemma 9.** *For any positive integer n , there is an erasure code of n vectors of length $\lceil \log n \rceil + 1$ allowing to overcome a single deletion.*

Recall that ARI is a player that might answer “don’t know” at most once. By choosing $n = 24$ in the lemma and applying Theorem 8, we get the following corollary.

► **Corollary 10.** *The classical game of Guess Who has a 6-question predetermined-strategy against ARI.*

5 Playing with Liars

While in the previous section, the clueless player was honest (i.e., provided only correct answers), we now turn our attention to a player that might give incorrect answers. These are sometimes called Byzantine or malicious players, but we will give the kids the benefit of the doubt and say they are merely “mistaken”.

A key tool in this section will be *error correcting codes*.

► **Definition 11.** An error correcting code of distance d is a collection of equal-length binary vectors, such that the Hamming distance between every two is at least d .

We will refer to such a code a “distance- d code”. Note that, given such a code and a vector v , if there exists a code word c with Hamming distance at most $\lfloor \frac{d-1}{2} \rfloor$ from v , then c is the unique code-word with this property. The common use of these codes is thus by “correcting” each such vector v to the unique nearest code-word c ; hence, we say that a distance- d code can fix at most $\lfloor \frac{d-1}{2} \rfloor$ errors.

► **Theorem 12.** If there is a distance- d error correcting code composed of at least n binary vectors of length k , then for every n -character game there is a k -question predetermined-strategy against an adversary that performs at most $\lfloor \frac{d-1}{2} \rfloor$ mistakes.

Proof. Assign each character i with a unique vector v_i from the code. For $1 \leq \ell \leq k$, let $A_\ell = \{i \mid v_i[\ell] = 1\}$ be all the characters with the ℓ -th entry of their vector equal to 1. Let j_ℓ be the trait that separates A_ℓ .

We use the question sequence $q_1 = q(j_1), \dots, q_k = q(j_k)$, which defines a run $I_0, q_1, a_1, I_1, \dots, q_k, a_k, I_k$. From the sequence (a_1, \dots, a_k) of answers, we find the unique closest code-word $v' = (a'_1, \dots, a'_k)$, which differs from the sequence of answers by at most $\lfloor \frac{d-1}{2} \rfloor$ answers. The character i with $v_i = v'$ is then the desired character.

For correctness, consider the true character i^* chosen, and the sequence (a_1^*, \dots, a_k^*) of the true answers to the questions, where a_ℓ^* is the answer on $q(j_\ell)$. Note that the choice of questions guarantees that the following are equivalent: $a_\ell^* = 1$; $i^* \in A_\ell$; and $v_{i^*}[\ell] = 1$. This concludes in $v_{i^*}[\ell] = a_\ell^*$, from which we have $v_{i^*} = (a_1^*, \dots, a_k^*)$. By the assumption of at most $\lfloor \frac{d-1}{2} \rfloor$ wrong answers, we know that the answer vector (a_1, \dots, a_k) differs from v_{i^*} in at most $\lfloor \frac{d-1}{2} \rfloor$ positions, which implies that the unique vector closest to the answer vector is indeed v_{i^*} , and the algorithm returns i^* as desired. ◀

One application of the above theorem is for the VAL player, that might make at most a single mistake. For this, we use a code guaranteed to exist by the Gilbert–Varshamov bound (see, e.g. [17]).

► **Lemma 13.** For positive integers k, N satisfying $2^N < \frac{2^k}{k}$, there is a distance-3 code composed of at least 2^N binary vectors of length k .

By choosing $k = 9$, $N = 5$ above, we get a code that implies, using Theorem 12, the following corollary.

► **Corollary 14.** The classical game of Guess Who has a 9-question predetermined strategy against VAL.

6 Concluding Remarks

As mentioned, Guess Who was invented as a simple version of the color-guessing Mastermind game. In Mastermind, the questions and answers are more involved than the binary questions asked in Guess Who, and thus, playing it with players that do not strictly follow the game protocol is an interesting direction we leave for future research.

Similar question holds for the trendy Wordle game, where the goal is not to guess a character but a word. Among the gallery of variations of Wordle that can be found online, one can consider a variant that does not always correctly mark the guessed letters, or where the user does not get feedback for the guesses until the end. These will probably not be the most fun-to-play version of the game, but poses interesting theoretical questions, such as error correction in natural language.

The commercial version of Guess Who was a target for criticism for its gallery of characters, where most characters are white males, and the exceptions (usually 5 out of 24) are women or people of color. One can consider a more inclusive version of the game, not only by splitting the traits more equally, but also by allowing non-binary answers to some questions. This could be done, e.g., by allowing questions such as “what is the color of the character’s hair”. A trinary version of the game (three hair colors, no-glasses/monocle/glasses, etc.) is yet to be presented, and the corresponding algorithmic questions are yet to be studied.

References

- 1 Aaron Berger, Christopher Chute, and Matthew Stone. Query complexity of mastermind variants. *Discret. Math.*, 341(3):665–671, 2018.
- 2 Lotte Berghman, Dries R. Goossens, and Roel Leus. Efficient solutions for mastermind using genetic algorithms. *Comput. Oper. Res.*, 36:1880–1885, 2009.
- 3 Lucas Boczkowski, Uriel Feige, Amos Korman, and Yoav Rodeh. Navigating in trees with permanently noisy advice. *ACM Trans. Algorithms*, 17(2):15:1–15:27, 2021.
- 4 Marco Console, Paolo Guagliardo, and Leonid Libkin. On querying incomplete information in databases under bag semantics. In *IJCAI*, volume 17, pages 993–999, 2017.
- 5 Carlos Cotta, Juan Julián Merelo Guervós, Antonio Mora García, and Thomas Philip Runarsson. Entropy-driven evolutionary approaches to the mastermind problem. In *PPSN*, 2010.
- 6 Yuval Dagan, Yuval Filmus, Daniel Kane, and Shay Moran. The entropy of lies: Playing twenty questions with a liar. In *ITCS*, volume 185 of *LIPICs*, pages 1:1–1:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 7 Claire David, Leonid Libkin, and Filip Murlak. Certain answers for xml queries. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 191–202, 2010.
- 8 Dariusz Dereniowski, Daniel Graf, Stefan Tiegel, and Przemyslaw Uznański. A framework for searching in graphs in the presence of errors. In *SOSA*, 2019.
- 9 Julien Gagneur, Markus C. Elze, and Achim Tresch. Selective phenotyping, entropy reduction, and the mastermind game. *BMC Bioinformatics*, 12:406–406, 2011.
- 10 Juan Julián Merelo Guervós, Pedro A. Castillo, Antonio Mora García, and Anna I. Esparcia-Alcázar. Improving evolutionary solutions to the game of mastermind using an entropy-based scoring method. In *GECCO '13*, 2013.
- 11 Juan Julián Merelo Guervós, Antonio Mora García, Pedro A. Castillo, Carlos Cotta, and Mario García Valdez. A search for scalable evolutionary solutions to the game of mastermind. *2013 IEEE Congress on Evolutionary Computation*, pages 2298–2305, 2013.
- 12 Nicolas Hanusse, David Ilcinkas, Adrian Kosowski, and Nicolas Nisse. Locating a target with an agent guided by unreliable local advice: how to beat the random walk when you have a clock? *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, 2010.
- 13 Nicolas Hanusse, Evangelos Kranakis, and Danny Krizanc. Searching with mobile agents in networks with liars. In *Euro-Par*, 2000.
- 14 Tomasz Imieliński and Witold Lipski Jr. Incomplete information in relational databases. In *Readings in Artificial Intelligence and Databases*, pages 342–360. Elsevier, 1989.
- 15 Donald Ervin Knuth. The computer as master mind. *Journal of Recreational Mathematics*, 9:1–6, 1977.
- 16 Witold Lipski Jr. On databases with incomplete information. *Journal of the ACM*, 28(1):41–70, 1981.
- 17 Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977.


23:10 Playing Guess Who with Your Kids

- 18 Mihai Nica. Optimal strategy in “guess who?”: Beyond binary search. *Probability in the Engineering and Informational Sciences*, 30(4):576–592, 2016. doi:10.1017/S026996481600022X.
- 19 Riccardo Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 356–365, 2006.
- 20 Ron Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006. doi:10.1017/CB09780511808968.
- 21 Geoffroy Ville. An optimal mastermind (4,7) strategy and more results in the expected case. *ArXiv*, abs/1305.1010, 2013.
- 22 Jef Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 179–190, 2010.

How to Physically Verify a Rectangle in a Grid: A Physical ZKP for Shikaku

Suthee Ruangwises  

Department of Mathematical and Computing Science, Tokyo Institute of Technology, Japan

Toshiya Itoh  

Department of Mathematical and Computing Science, Tokyo Institute of Technology, Japan

Abstract

Shikaku is a pencil puzzle consisting of a rectangular grid, with some cells containing a number. The player has to partition the grid into rectangles such that each rectangle contains exactly one number equal to the area of that rectangle. In this paper, we propose two physical zero-knowledge proof protocols for Shikaku using a deck of playing cards, which allow a prover to physically show that he/she knows a solution of the puzzle without revealing it. Most importantly, in our second protocol we develop a general technique to physically verify a rectangle-shaped area with a certain size in a rectangular grid, which can be used to verify other problems with similar constraints.

2012 ACM Subject Classification Security and privacy → Information-theoretic techniques; Theory of computation → Cryptographic protocols

Keywords and phrases Zero-knowledge proof, Card-based cryptography, Shikaku, Puzzles, Games

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.24

1 Introduction

Shikaku is a pencil puzzle introduced by Nikoli, a Japanese publisher that developed many popular pencil puzzles such as Sudoku, Kakuro, and Slitherlink. The puzzle has become popular and many Shikaku mobile apps have been developed [7]. A Shikaku puzzle consists of a rectangular grid of size $m \times n$, with some cells containing a number. The objective of this puzzle is to partition the grid into rectangles such that each rectangle contains exactly one number, which must be equal to the area of that rectangle (see Figure 1). Determining whether a given Shikaku puzzle has a solution is an NP-complete problem [23].

Suppose that Paimon, an expert in Shikaku, created a difficult Shikaku puzzle and challenged her friend Venti to solve it. After a while, Venti could not solve her puzzle and began to doubt whether the puzzle actually has a solution. Paimon wants to convince him that her puzzle indeed has a solution without revealing it (which would render the challenge pointless). To achieve this, Paimon needs a *zero-knowledge proof (ZKP)*.

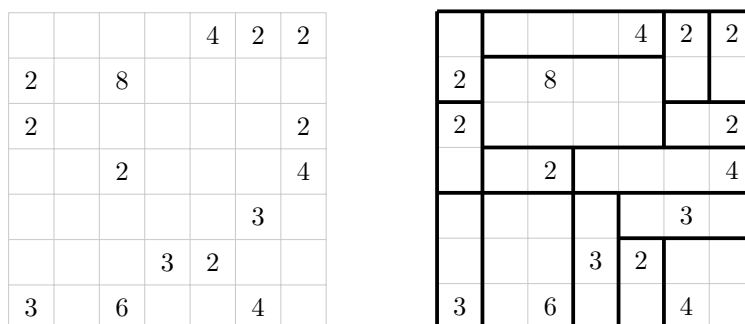


Figure 1 An example of a 7×7 Shikaku puzzle (left) and its solution (right).



© Suthee Ruangwises and Toshiya Itoh;
licensed under Creative Commons License CC-BY 4.0
11th International Conference on Fun with Algorithms (FUN 2022).

Editors: Pierre Fraigniaud and Yushi Uno; Article No. 24; pp. 24:1–24:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Zero-Knowledge Proof

First introduced in 1989 by Goldwasser et al. [6], a ZKP is an interactive protocol between a prover P and a verifier V . Both P and V are given a computational problem x , but only P knows a solution w of x . A ZKP enables P to convince V that he/she knows w without revealing any information about w . It must satisfy the following three properties.

1. **Completeness:** If P knows w , then V accepts with high probability. (In this paper, we consider only the *perfect completeness* property where V always accepts.)
2. **Soundness:** If P does not know w , then V rejects with high probability. (In this paper, we consider only the *perfect soundness* property where V always rejects.)
3. **Zero-knowledge:** V learns nothing about w . Formally, there exists a probabilistic polynomial time algorithm S (called a *simulator*), not knowing w but having an access to V , such that the outputs of S follow the same probability distribution as the ones of the real protocol.

As there exists a ZKP for every NP problem [5], one can construct a computational ZKP for Shikaku. However, such construction requires cryptographic primitives and thus is not intuitive or practical.

Instead, many results so far aimed to develop physical ZKP protocols using a deck of playing cards. These card-based protocols have benefits that they use only portable objects found in everyday life and do not require computers. They also allow external observers to verify that the prover truthfully executes the protocol (which is often a challenging task for digital protocols). In addition, these protocols have great didactic values to teach the concept of a ZKP to non-experts.

1.2 Related Work

Card-based ZKP protocols for many other popular pencil puzzles have been developed, including Sudoku [8, 17, 21], Nonogram [3, 16], Akari [1], Takuzu [1, 12], Kakuro [1, 13], KenKen [1], Makaro [2], Norinori [4], Slitherlink [11], Juosan [12], Numberlink [18], Suguru [14], Ripple Effect [19], Nurikabe [15], Hitori [15], Cryptarithmic [9], and Bridges [20].

In a recent work of Robert et al. [15], the authors posed an open problem to extend the idea of their protocol to verify a solution of Shikaku or other puzzles that require to draw rectangles with certain sizes in a grid.

1.3 Our Contribution

In this paper, we answer the open problem posed by Robert et al. [15] by developing two card-based ZKP protocols with perfect completeness and soundness for Shikaku: a brute force protocol and a more elegant, intuitive *flooding protocol*. The two protocols use $\Theta(m^2n^2)$ cards and $\Theta(mn)$ cards, respectively.

Most importantly, in the flooding protocol we develop a general technique to physically verify a rectangle-shaped area with a certain size in a rectangular grid, which can be used to verify other problems with similar constraints.

2 First Attempt: Brute Force Protocol

Every card used in this paper has an integer on the front side. All cards have indistinguishable back sides denoted by $\boxed{?}$.

Let (x, y) denote a cell located in the x -th topmost row and y -th leftmost column of the Shikaku grid. Let p_2, p_3, \dots, p_{k+1} be the k numbers written on the grid¹, with each number p_i in a cell (x_i, y_i) . Note that we must have $p_2 + p_3 + \dots + p_{k+1} = mn$.

Suppose that in P 's solution, the grid is divided into k rectangles Z_2, Z_3, \dots, Z_{k+1} such that each Z_i contains the number p_i . Each rectangle Z_i is represented by its top-left and bottom-right corner cells (a_i, b_i) and (a'_i, b'_i) , respectively. To verify that the solution is correct, it is sufficient to show that

1. $a_i \leq x_i \leq a'_i$ and $b_i \leq y_i \leq b'_i$ (a cell with the number p_i is inside Z_i) for every $i \in \{2, 3, \dots, k+1\}$,
2. $(a'_i - a_i + 1)(b'_i - b_i + 1) = p_i$ (the area of Z_i is equal to p_i) for every $i \in \{2, 3, \dots, k+1\}$, and
3. $a'_i < a_j$ or $a'_i < a_j$ or $b'_i < b_j$ or $b'_i < b_j$ (Z_i and Z_j do not overlap) for every distinct $i, j \in \{2, 3, \dots, k+1\}$.

These three conditions can be verified by applying the combination of the copy, addition, multiplication, and equality protocols [20], and a protocol to compare two numbers [2].

This protocol, however, involves a lot of messy calculations and thus has lost its didactic values as it becomes more computational and less intuitive. Moreover, it requires up to $\Theta(m^2n^2)$ cards (as we have to multiply integers in modulo mn)², which is far too many to be practical. Instead, we are looking for an elegant and intuitive protocol that uses a reasonable number of cards.

3 Verifying an Area of Connected Cells

In a recent work, Robert et al. [15] developed a *sea formation protocol* that allows the prover P to convince the verifier V that a given area in a grid consists of t cells that are connected to each other horizontally or vertically. We will first show the necessary subprotocols and then explain the sea formation protocol.

3.1 Pile-Shifting Shuffle

Given a $p \times q$ matrix of cards, a *pile-shifting shuffle* rearranges the columns of the matrix by a random cyclic shift, i.e. shifts the columns cyclically to the right by x columns for a uniformly random $x \in \mathbb{Z}/q\mathbb{Z}$, unknown to all parties.

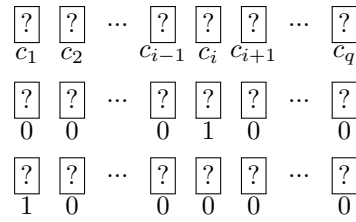
The pile-shifting shuffle was developed by Shinagawa et al. [22]. It can be performed in real world by putting the cards in each column into an envelope and then taking turns to apply *Hindu cuts* (taking several envelopes from the bottom and putting them on the top) to the sequence of envelopes [24].

3.2 Chosen Cut Protocol

Given a sequence of q face-down cards $C = (c_1, c_2, \dots, c_q)$, a *chosen cut protocol* for q cards allows P to select a card c_i he/she wants (to use in other operations) without revealing i to V . This protocol also reverts the sequence C back to its original state after P finishes using c_i . It was developed by Koch and Walzer [10].

¹ We intentionally start the indices at 2 so that our second protocol, which will be introduced later, will be easier to understand.

² In this protocol, an integer x in modulo mn is encoded by a sequence of mn consecutive cards, with all of them being $\boxed{0}$ s except the $(x+1)$ -th leftmost card being a $\boxed{1}$.



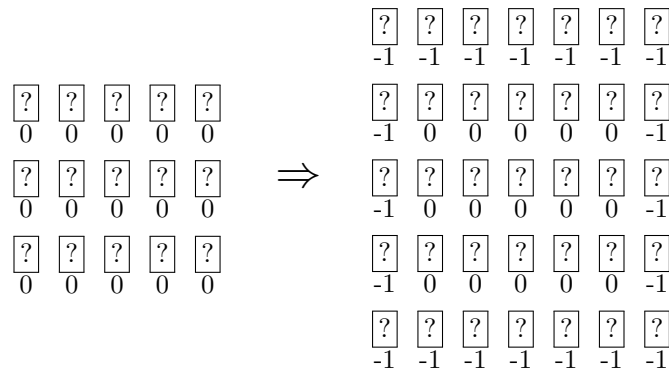
■ **Figure 2** A $3 \times q$ matrix M constructed in Step 1 of the chosen cut protocol.

1. Construct the following $3 \times q$ matrix M (see Figure 2).
 - a. In Row 1, publicly place the sequence C .
 - b. In Row 2, secretly place a face-down $\boxed{1}$ at Column i and a face-down $\boxed{0}$ at each other column.
 - c. In Row 3, secretly place a face-down $\boxed{1}$ at Column 1 and a face-down $\boxed{0}$ at each other column.
2. Apply the pile-shifting shuffle to M .
3. Turn over all cards in Row 2. Locate the position of the only $\boxed{1}$. A card in Row 1 directly above that $\boxed{1}$ will be the card c_i as desired.
4. After we finish using c_i in other operations, place c_i back into M at the same position.
5. Turn over all face-up cards in Row 2 and apply the pile-shifting shuffle to M again.
6. Turn over all cards in Row 3. Locate the position of the only $\boxed{1}$. Shift the columns of M cyclically such that this $\boxed{1}$ moves to Column 1. This reverts M back to its original state.

Note that Steps 3 and 6 of this protocol guarantee that the cards in Row 2 and Row 3 are in a correct format (each row having one $\boxed{1}$ and $q - 1$ $\boxed{0}$ s).

3.3 Sea Formation Protocol

First, publicly place a face-down $\boxed{0}$ on every cell in the Shikaku grid. To handle the case where a selected cell is on the edge of the grid, we publicly place face-down “dummy cards” $\boxed{-1}$ s around the grid. We now have an $(m + 2) \times (n + 2)$ matrix of cards (see Figure 3).



■ **Figure 3** The way we place cards on a 3×5 Shikaku grid during the setup of the sea formation protocol.

Start at the top-left corner of the matrix and pick all cards in the order from left to right in Row 1, then from left to right in Row 2, and so on. Arrange them into a single sequence $D = (d_1, d_2, \dots, d_{(m+2)(n+2)})$. Note that we know exactly where the four neighbors of any

given card are. Namely, the cards on the neighbor to the left, right, top, and bottom of a cell containing d_i are d_{i-1} , d_{i+1} , d_{i-n-2} , and d_{i+n+2} , respectively.

The sea formation protocol to verify a connected area with size t works as follows.

1. P applies the chosen cut protocol for $(m+2)(n+2)$ cards to select a $\boxed{0}$ that he/she wants to replace.
2. P reveals the selected card to V that it is a $\boxed{0}$ (otherwise V rejects) and then replaces it with a $\boxed{1}$.
3. P repeatedly performs the following steps for $t-1$ iterations.
 - a. P applies the chosen cut protocol for $(m+2)(n+2)$ cards to select a $\boxed{1}$ he/she wants.
 - b. P reveals the selected card to V that it is a $\boxed{1}$ (otherwise V rejects).
 - c. P picks the four neighbors of the selected card and applies the chosen cut protocol for four cards to select one of the four neighbors, which is a $\boxed{0}$ that he/she wants to replace.
 - d. P reveals the selected neighbor to V that it is a $\boxed{0}$ (otherwise V rejects) and then replaces it with a $\boxed{1}$.

We can see that in each iteration, the “sea” of $\boxed{1}$ s expands by one cell, while all $\boxed{1}$ s remain connected to each other. Therefore, after $t-1$ steps, V is convinced that there is an area of t $\boxed{1}$ s in the grid that are connected to each other.

4 Idea to Verify a Rectangle-Shaped Area

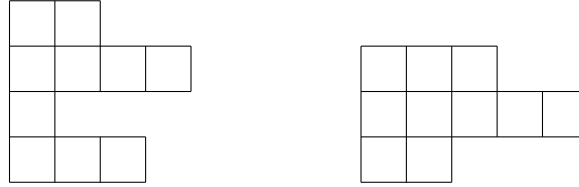
The sea formation protocol, however, does not say anything about the shape of the area. By extending the idea of the sea formation protocol, we propose the following *flooding protocol*, which allows P to convince V that the area is a rectangle with size t .

The idea is to always start at the top-left corner of the rectangle. At first, P changes the card on the top-left corner cell of the rectangle from a $\boxed{0}$ to a $\boxed{1}$. Similarly to the sea formation protocol, in each step P selects a cell with a $\boxed{1}$ and changes the card on one of its neighbor from a $\boxed{0}$ to a $\boxed{1}$. However, the difference from the sea formation protocol is that P can only select the neighbor to the right or to the bottom (but not to the left or to the top). We call this process a *flood*, which starts at the top-left corner and goes downwards or rightwards in each step until it eventually fills the whole rectangle in $t-1$ steps.

To be more specific, at first the flood can only go downwards (i.e. P can only select the neighbor to the bottom) to fill cells along the left edge of the rectangle. Then, right after it just filled all cells along the left edge, the flood suddenly changes direction and can only go rightwards (i.e. P can only select the neighbor to the right) to fill the rest of the cells in the rectangle. In particular, V must not know the exact time when the flood changes direction (otherwise V will know the height of the rectangle).

The technique to achieve this “one-time direction change” is to let P keep a secret variable r , which controls the direction of the flood (if $r = 0$, then the flood goes downwards; if $r = 1$, then the flood goes rightwards). At the beginning, P shows V that $r = 0$. Before each step, P secretly chooses whether to add 1 to r or not, then shows V that $r \neq 2$ (without revealing the actual value of r). This technique works because while $r = 0$, r can become either 0 or 1 in the next step, but once r becomes 1, it must remain 1 forever (see a subprotocol in Section 5.2 on how to make the selected neighbor depend on the value of r).

P performs the above process for $t - 1$ times to change all $\boxed{0}$ s in the rectangle to $\boxed{1}$ s. However, the protocol is not finished yet, as V is not yet convinced that the area is a rectangle. In fact, P has only shown that the area has a straight left edge; it may look like one of the shapes in Figure 4.



■ **Figure 4** Examples of possible shapes with a straight left edge, each with area 10.

To convince V that the area is a rectangle, P needs to perform the “second flood”. The second flood starts at the bottom-right corner and goes into the cells already visited by the “first flood” in the opposite direction from the first flood – originally the flood can only go upwards (i.e. P can only select the neighbor to the top), then right after it just filled all cells along the right edge, the flood changes direction and can only go leftwards (i.e. P can only select the neighbor to the left).

Formally, P starts at a bottom-right corner of the rectangle and replaces a $\boxed{1}$ with a $\boxed{2}$. P sets $r = 0$ and shows it to V . In each step, P secretly chooses whether to add 1 to r or not, then shows V that $r \neq 2$. If $r = 0$ (resp. $r = 1$), P selects a cell with a $\boxed{2}$ and changes the card on its neighbor to the top (resp. to the left) from a $\boxed{1}$ to a $\boxed{2}$. P performs this for $t - 1$ steps to change all $\boxed{1}$ s in the rectangle to $\boxed{2}$ s.

After the second flood, P have shown that the area also has a straight right edge. This is sufficient to convince V that the area is a rectangle with size t (see the proof of Lemma 2 for the full proof of perfect soundness).

In the next section, we will show the necessary subprotocols that enable us to formalize this idea into an actual protocol.

5 Subprotocols

5.1 Addition Protocol for $\mathbb{Z}/3\mathbb{Z}$

We use a sequence of three consecutive cards to encode each integer in $\mathbb{Z}/3\mathbb{Z}$. Namely, we use $\boxed{1}\boxed{0}\boxed{0}$, $\boxed{0}\boxed{1}\boxed{0}$, and $\boxed{0}\boxed{0}\boxed{1}$ to encode 0, 1, and 2, respectively.

Suppose we have sequences R and S encoding integers r and s in $\mathbb{Z}/3\mathbb{Z}$, respectively. This protocol, developed by Shinagawa et al. [22], computes the sum $r + s$ without revealing r or s .

1. Swap the two rightmost cards of S . This modified sequence, called S' , now encodes $-s \pmod{3}$.
2. Construct a 2×3 matrix M by placing S' in Row 1 and R in Row 2.
3. Apply the pile-shifting shuffle to M . Note that Row 1 and Row 2 of M now encode $-s + x \pmod{3}$ and $r + x \pmod{3}$, respectively, for some uniformly random $x \in \mathbb{Z}/3\mathbb{Z}$.
4. Turn over all cards in Row 1 of M . Locate the position of a $\boxed{1}$. Shift the columns of M cyclically such that this $\boxed{1}$ moves to Column 1.
5. The sequence in Row 2 of M now encodes $(r + x) - (-s + x) \equiv r + s \pmod{3}$ as desired.

Note that Step 4 of this protocol guarantees that S is in a correct format (having one $\boxed{1}$ and two $\boxed{0}$ s). In each step of the flooding protocol, P secretly selects $s \in \{0, 1\}$ and places S accordingly. Then, P reveals the rightmost card of S that it is a $\boxed{0}$ to show V that $s \neq 2$. Similarly, after computing the sum $r + s$, P reveals the rightmost card of the resulting sequence to show V that $r + s \neq 2$.

5.2 Neighbor Selection Protocol

In $\mathbb{Z}/2\mathbb{Z}$, we use $\boxed{1}\boxed{0}$ and $\boxed{0}\boxed{1}$ to encode 0 and 1, respectively. Suppose we have two face-down cards c_0 and c_1 , and a sequence R encoding an integer $r \in \mathbb{Z}/2\mathbb{Z}$. We want to select a card c_r to use in other operations without revealing r , and also put c_0 and c_1 back to where they came from.

We can do so by applying the chosen cut protocol for two cards. However, in Step 1.b, we instead place a sequence R in Row 2 (without revealing R). Also, at the end of the chosen cut protocol, M is reverted to its original state, so we can put c_0 and c_1 back to where they came from.

In each step of the flooding protocol, after showing that $r \neq 2$, P picks only the two leftmost cards of a sequence encoding r in $\mathbb{Z}/3\mathbb{Z}$. This truncated sequence encodes r in $\mathbb{Z}/2\mathbb{Z}$ as desired. During the first flood, P chooses the cards on the neighbor to the bottom and to the right of the selected cell as c_0 and c_1 , respectively; during the second flood, P chooses the cards on the neighbor to the top and to the left of the selected cell as c_0 and c_1 , respectively.

6 Formal Steps of the Flooding Protocol

Similarly to the sea formation protocol, we first publicly place a face-down $\boxed{0}$ on every cell in the Shikaku grid, and also place face-down $\boxed{-1}$ s around the grid. We now have an $(m + 2) \times (n + 2)$ matrix of cards (see Figure 3).

Let $h_i = a'_i - a_i + 1$ be the height of a rectangle Z_i ($i \in \{2, 3, \dots, k + 1\}$). To verify that Z_i is a rectangle with area p_i and also contains a cell with the number p_i , P performs the following two phases: the first flood and the second flood.

6.1 First Flood

1. P applies the chosen cut protocol for $(m + 2)(n + 2)$ cards to select a card on the top-left corner cell of Z_i .
2. P reveals the selected card to V that it is a $\boxed{0}$ (otherwise V rejects) and then replaces it with a $\boxed{1}$.
3. P publicly constructs a sequence R of three cards encoding an integer $r = 0$.
4. P repeatedly performs the following steps for $p_i - 1$ iterations.
 - a. P secretly constructs a sequence S of three cards encoding an integer $s \in \{0, 1\}$. If this is the h_i -th iteration, P must choose $s = 1$; otherwise, P must choose $s = 0$.
 - b. P reveals the rightmost card of S to V that it is a $\boxed{0}$ to show that $s \neq 2$ (otherwise V rejects).
 - c. P applies the addition protocol to compute $r + s$ and reveals the rightmost card of the resulting sequence to V that it is a $\boxed{0}$ to show that $r + s \neq 2$ (otherwise V rejects). From now on, set $r := r + s$.
 - d. P applies the chosen cut protocol for $(m + 2)(n + 2)$ cards to select a $\boxed{1}$ he/she wants from the Shikaku grid. If this is during the first $h_i - 1$ iterations, P must choose the bottommost $\boxed{1}$; otherwise, P may choose any card that is the rightmost $\boxed{1}$ in its row and is not located in the rightmost column of Z_i .

- e. P reveals the selected card to V that it is a $\boxed{1}$ (otherwise V rejects).
- f. P chooses the neighbors to the bottom and to the right of the selected card as c_0 and c_1 , respectively, and applies the neighbor selection protocol to select a card c_r (using the two leftmost cards of a sequence encoding r as inputs).
- g. P reveals the selected neighbor to V that it is a $\boxed{0}$ (otherwise V rejects) and then replaces it with a $\boxed{1}$.

After the first flood, all cards on the cells in Z_i are now changed to $\boxed{1}$ s.

6.2 Second Flood

1. P applies the chosen cut protocol for $(m+2)(n+2)$ cards to select a card on the bottom-right corner cell of Z_i .
2. P reveals the selected card to V that it is a $\boxed{1}$ (otherwise V rejects) and then replaces it with an \boxed{i} .
3. P publicly constructs a sequence R of three cards encoding an integer $r = 0$.
4. P repeatedly performs the following steps for $p_i - 1$ iterations.
 - a. P secretly constructs a sequence S of three cards encoding an integer $s \in \{0, 1\}$. If this is the h_i -th iteration, P must choose $s = 1$; otherwise, P must choose $s = 0$.
 - b. P reveals the rightmost card of S to V that it is a $\boxed{0}$ to show that $s \neq 2$ (otherwise V rejects).
 - c. P applies the addition protocol to compute $r + s$ and reveals the rightmost card of the resulting sequence to V that it is a $\boxed{0}$ to show that $r + s \neq 2$ (otherwise V rejects). From now on, set $r := r + s$.
 - d. P applies the chosen cut protocol for $(m+2)(n+2)$ cards to select an \boxed{i} he/she wants from the Shikaku grid. If this is during the first $h_i - 1$ iterations, P must choose the topmost \boxed{i} ; otherwise, P may choose any card that is the leftmost \boxed{i} in its row and is not located in the leftmost column of Z_i .
 - e. P reveals the selected card to V that it is an \boxed{i} (otherwise V rejects).
 - f. P chooses the neighbors to the top and to the left of the selected card as c_0 and c_1 , respectively, and applies the neighbor selection protocol to select a card c_r (using the two leftmost cards of a sequence encoding r as inputs).
 - g. P reveals the selected neighbor to V that it is a $\boxed{1}$ (otherwise V rejects) and then replaces it with an \boxed{i} .

After the second flood, all cards on the cells in Z_i are now changed to \boxed{i} s. Finally, P turns over a card on the cell with the number p_i to show that it is an \boxed{i} , i.e. Z_i contains the cell with the number p_i (otherwise V rejects).

P performs the above two phases for every $i \in \{2, 3, \dots, k+1\}$. If all verification steps pass, then V accepts.

The number of cards used in the flooding protocol is $\Theta(mn)$, which is much lower than the brute force protocol.

7 Proof of Security

We will prove the perfect completeness, perfect soundness, and zero-knowledge properties of the flooding protocol.

► **Lemma 1** (Perfect Completeness). *If P knows a solution of the Shikaku puzzle, then V always accepts.*

Proof. Suppose that P knows a solution of the puzzle. Consider the verification of each Z_i .

In the first flood, during the first $h_i - 1$ iterations P chooses $s = 0$ and chooses the bottommost $\boxed{1}$, so the area of $\boxed{1}$ s expands downwards by one cell. After $h_i - 1$ iterations, all cards along the left edge of Z_i have been changed to $\boxed{1}$ s. In the h_i -th iteration, P chooses $s = 1$ and chooses any $\boxed{1}$, so the flood direction is changed to rightwards and the area of $\boxed{1}$ s expands by one cell. After that, in each iteration P chooses $s = 0$ and chooses any card that is the rightmost $\boxed{1}$ in its row and is not located in the rightmost column of Z_i , so the area of $\boxed{1}$ s expands by one cell inside Z_i . Therefore, at the end of the first flood, all cards in Z_i has been changed to $\boxed{1}$ s.

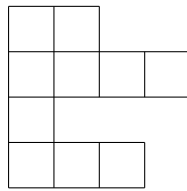
Analogously, in the second flood, during the first $h_i - 1$ iterations P chooses $s = 0$ and chooses the topmost \boxed{i} , so the area of \boxed{i} s expands upwards by one cell. After $h_i - 1$ iterations, all cards along the right edge of Z_i have been changed to \boxed{i} s. In the h_i -th iteration, P chooses $s = 1$ and chooses any \boxed{i} , so the flood direction is changed to leftwards and the area of \boxed{i} s expands by one cell. After that, in each iteration P chooses $s = 0$ and chooses any card that is the leftmost \boxed{i} in its row and is not located in the leftmost column of Z_i , so the area of \boxed{i} s expands by one cell inside Z_i . Therefore, at the end of the first flood, all cards in Z_i has been changed to \boxed{i} s, thus a card on the cell containing the number p_i must also be an \boxed{i} .

Since the verification passes for every Z_i , V always accepts. \blacktriangleleft

► **Lemma 2 (Perfect Soundness).** *If P does not know a solution of the Shikaku puzzle, then V always rejects.*

Proof. We will prove the contrapositive of this statement. Suppose that V accepts, meaning that the flooding protocol passes for every Z_i . We will prove that P must know a solution.

First, note that the chosen cut protocol in Section 3.2 and the addition protocol in Section 5.1 guarantee that the inputs from P must be in a correct format. Consider the verification of Z_i . Suppose that the first flood goes downwards for $h - 1$ steps before changing direction to rightwards. The area that contains $\boxed{1}$ s after the first flood must have a straight left edge with height h , and have a shape like h horizontal bars placing on top of each other. Let $\ell_1, \ell_2, \dots, \ell_h$ be the length of these bars from top to bottom. For example, in Figure 5 we have $h = 4$, $\ell_1 = 2$, $\ell_2 = 4$, $\ell_3 = 1$, and $\ell_4 = 3$.



■ **Figure 5** An example of a possible shape with a straight left edge.

Since all p_i $\boxed{1}$ s in this area have been replaced by \boxed{i} s after the second flood, all cells in the area must be reachable from the starting point of the second flood by moving only upwards or leftwards. Thus, the only possible starting point of the second flood is the rightmost cell of the bottommost bar (the one with length ℓ_h).

Moreover, for any $j < h$, we must have $\ell_j \leq \ell_h$ (otherwise there is a cell in the j -th bar which is located to the right of the starting point and thus not reachable by the second flood). However, if $\ell_j < \ell_h$, the second flood cannot go directly from the starting point to the j -th

bar by only moving upwards; it has to change direction at least twice, a contradiction since the flood can change direction at most once. Therefore, we must have $\ell_j = \ell_h$ for every $j \in \{1, 2, \dots, h-1\}$, which means the area must be a rectangle.

Therefore, Z_i is a rectangle with area p_i that contains a cell with the number p_i for every $i \in \{2, 3, \dots, k+1\}$. Since any two rectangles do not overlap, and $p_2 + p_3 + \dots + p_{k+1} = mn$, they must be a partition of the grid. Hence, we can conclude that P knows a valid solution of the puzzle. ◀

► **Lemma 3 (Zero-Knowledge).** *During the verification phase, V learns nothing about P 's solution of the Shikaku puzzle.*

Proof. To prove the zero-knowledge property, it is sufficient to show that all distributions of cards that are turned face-up can be simulated by a simulator S that does not know P 's solution.

- In Steps 3 and 6 of the chosen cut protocol in Section 3.2, the $\boxed{1}$ has an equal probability to be at any of the q positions, so this step can be simulated by S .
- In Step 4 of the addition protocol in Section 5.1, the $\boxed{1}$ has an equal probability to be at any of the three positions, so this step can be simulated by S .
- In the flooding protocol, during the verification of each Z_i , there is only one deterministic pattern of the cards that are turned face-up. This pattern solely depends on p_i , which is public information, so the whole protocol can be simulated by S . ◀

8 Future Work

We developed a physical ZKP protocol with perfect completeness and soundness for Shikaku using $\Theta(mn)$ cards. Most importantly, we also developed a general technique to physically verify a rectangle-shaped area with a certain size in a rectangular grid.

A possible future work is to develop physical ZKP protocols to verify other geometric shapes or other puzzles with constraints related to shapes (e.g. Shakashaka). Another interesting future work is to develop an equivalent protocol for Shikaku that can be implemented using a deck of all different cards (like the one for Sudoku [17]).

References

- 1 X. Bultel, J. Dreier, J.-G. Dumas, and P. Lafourcade. Physical zero-knowledge proofs for Akari, Takuzu, Kakuro and KenKen. In *Proceedings of the 8th International Conference on Fun with Algorithms (FUN)*, pages 8:1–8:20, 2016. doi:10.4230/LIPIcs.FUN.2016.8.
- 2 X. Bultel, J. Dreier, J.-G. Dumas, P. Lafourcade, D. Miyahara, T. Mizuki, A. Nagao, T. Sasaki, K. Shinagawa, and H. Sone. Physical zero-knowledge proof for Makaro. In *Proceedings of the 20th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 111–125, 2018. doi:10.1007/978-3-030-03232-6_8.
- 3 Y.-F. Chien and W.-K. Hon. Cryptographic and physical zero-knowledge proof: From Sudoku to Nonogram. In *Proceedings of the 5th International Conference on Fun with Algorithms (FUN)*, pages 102–112, 2010. doi:10.1007/978-3-642-13122-6_12.
- 4 J.-G. Dumas, P. Lafourcade, D. Miyahara, T. Mizuki, T. Sasaki, and H. Sone. Interactive physical zero-knowledge proof for Norinori. In *Proceedings of the 25th International Computing and Combinatorics Conference (COCOON)*, pages 166–177, 2019. doi:10.1007/978-3-030-26176-4_14.
- 5 O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. *Journal of the ACM*, 38(3):691–729, 1991. doi:10.1145/116825.116852.

- 6 S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. doi:10.1137/0218012.
- 7 Google Play: Shikaku. <https://play.google.com/store/search?q=Shikaku&c=apps>.
- 8 R. Gradwohl, M. Naor, B. Pinkas, and G.N. Rothblum. Cryptographic and physical zero-knowledge proof systems for solutions of sudoku puzzles. *Theory of Computing Systems*, 44(2):245–268, 2009. doi:10.1007/s00224-008-9119-9.
- 9 R. Isuzugawa, D. Miyahara, and T. Mizuki. Zero-knowledge proof protocol for crypt-arithmic using dihedral cards. In *Proceedings of the 19th International Conference on Unconventional Computation and Natural Computation (UCNC)*, pages 51–67, 2021. doi:10.1007/978-3-030-87993-8_4.
- 10 A. Koch and S. Walzer. Foundations for actively secure card-based cryptography. In *Proceedings of the 10th International Conference on Fun with Algorithms (FUN)*, pages 17:1–17:23, 2020. doi:10.4230/LIPIcs.FUN.2021.17.
- 11 P. Lafourcade, D. Miyahara, T. Mizuki, L. Robert, T. Sasaki, and H. Sone. How to construct physical zero-knowledge proofs for puzzles with a “single loop” condition. *Theoretical Computer Science*, 888:41–55, 2021. doi:10.1016/j.tcs.2021.07.019.
- 12 D. Miyahara, L. Robert, P. Lafourcade, S. Takeshige, T. Mizuki, K. Shinagawa, A. Nagao, and H. Sone. Card-based zkp protocols for takuzu and juosan. In *Proceedings of the 10th International Conference on Fun with Algorithms (FUN)*, pages 20:1–20:21, 2020. doi:10.4230/LIPIcs.FUN.2021.20.
- 13 D. Miyahara, T. Sasaki, T. Mizuki, and H. Sone. Card-based physical zero-knowledge proof for Kakuro. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E102.A(9):1072–1078, 2019. doi:10.1587/transfun.E102.A.1072.
- 14 L. Robert, D. Miyahara, P. Lafourcade, and T. Mizuki. Physical zero-knowledge proof for suguru puzzle. In *Proceedings of the 22nd International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 235–247, 2020. doi:10.1007/978-3-030-64348-5_19.
- 15 L. Robert, D. Miyahara, P. Lafourcade, and T. Mizuki. Interactive physical zkp for connectivity: Applications to nurikabe and hitori. In *Proceedings of the 17th Conference on Computability in Europe (CiE)*, pages 373–384, 2021. doi:10.1007/978-3-030-80049-9_37.
- 16 S. Ruangwises. An improved physical zkp for nonogram. In *Proceedings of the 15th Annual International Conference on Combinatorial Optimization and Applications (COCOAA)*, pages 262–272, 2021. doi:10.1007/978-3-030-92681-6_22.
- 17 S. Ruangwises. Two standard decks of playing cards are sufficient for a zkp for sudoku. In *Proceedings of the 27th International Computing and Combinatorics Conference (COCOON)*, pages 631–642, 2021. doi:10.1007/978-3-030-89543-3_52.
- 18 S. Ruangwises and T. Itoh. Physical zero-knowledge proof for numberlink puzzle and k vertex-disjoint paths problem. *New Generation Computin*, 39(1):3–17, 2021. doi:10.1007/s00354-020-00114-y.
- 19 S. Ruangwises and T. Itoh. Physical zero-knowledge proof for ripple effect. *Theoretical Computer Science*, 895:115–123, 2021. doi:10.1016/j.tcs.2021.09.034.
- 20 S. Ruangwises and T. Itoh. Physical zkp for connected spanning subgraph: Applications to bridges puzzle and other problems. In *Proceedings of the 19th International Conference on Unconventional Computation and Natural Computation (UCNC)*, pages 149–163, 2021. doi:10.1007/978-3-030-87993-8_10.
- 21 T. Sasaki, D. Miyahara, T. Mizuki, and H. Sone. Efficient card-based zero-knowledge proof for sudoku. *Theoretical Computer Science*, 839:135–142, 2020. doi:10.1016/j.tcs.2020.05.036.
- 22 K. Shinagawa, T. Mizuki, J.C.N. Schuldt, K. Nuida, N. Kanayama, T. Nishide, G. Hanaoka, and E. Okamoto. Card-based protocols using regular polygon cards. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E100.A(9):1900–1909, 2017. doi:10.1587/transfun.E100.A.1900.

24:12 How to Physically Verify a Rectangle in a Grid: A Physical ZKP for Shikaku

- 23 Y. Takenaga, S. Aoyagi, S. Iwata, and T. Kasai. Shikaku and ripple effect are np-complete. *Congressus Numerantium*, 216:119–127, 2013.
- 24 I. Ueda, D. Miyahara, A. Nishimura, Y. Hayashi, T. Mizuki, and H. Sone. Secure implementations of a random bisection cut. *International Journal of Information Security*, 19(4):445–452, 2020. doi:10.1007/s10207-019-00463-w.