

3rd Symposium on Algorithmic Foundations of Dynamic Networks

SAND 2024, June 5–7, 2024, Patras, Greece

Edited by

Arnaud Casteigts

Fabian Kuhn



Editors

Arnaud Casteigts 

University of Geneva, Switzerland
arnaud.casteigts@u-bordeaux.fr

Fabian Kuhn 

University of Freiburg, Germany
kuhn@cs.uni-freiburg.de

ACM Classification 2012

Theory of computation; Mathematics of computing; Networks

ISBN 978-3-95977-315-7

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/978-3-95977-315-7>.

Publication date

June, 2024

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.SAND.2024.0

ISBN 978-3-95977-315-7

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 (Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- 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 (*Chair*, 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)
- Pierre Senellart (ENS, Université PSL, Paris, FR)

ISSN 1868-8969

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

■ Contents

Preface	
<i>Arnaud Casteigts and Fabian Kuhn</i>	0:vii–0:viii
Authors	
.....	0:ix–0:xi

Invited Talks

Exploration and Rendezvous in Temporal Graphs	
<i>Thomas Erlebach</i>	1:1–1:1
Distributed Computation with Bacteria	
<i>Thomas Nowak</i>	2:1–2:1
Algorithmic Programmable Matter: From Local Markov Chains to “Dumb” Robots	
<i>Andréa Werneck Richa</i>	3:1–3:1

Regular Papers

Harmonious Colourings of Temporal Matchings	
<i>Duncan Adamson</i>	4:1–4:11
Fault-Tolerant Distributed Directories	
<i>Judith Beestermöller, Costas Busch, and Roger Wattenhofer</i>	5:1–5:20
Black Hole Search in Dynamic Tori	
<i>Adri Bhattacharya, Giuseppe F. Italiano, and Partha Sarathi Mandal</i>	6:1–6:16
Online Space-Time Travel Planning in Dynamic Graphs	
<i>Quentin Bramas, Jean-Romain Luttringer, and Sébastien Tixeuil</i>	7:1–7:14
On Inefficiently Connecting Temporal Networks	
<i>Esteban Christiann, Eric Sanlaville, and Jason Schoeters</i>	8:1–8:19
All for One and One for All: An $O(1)$ -Musketeers Universal Transformation for Rotating Robots	
<i>Matthew Connor, Othon Michail, and George Skretas</i>	9:1–9:20
On the Complexity of Temporal Arborescence Reconfiguration	
<i>Riccardo Dondi and Manuel Lafond</i>	10:1–10:15
Partial Temporal Vertex Cover with Bounded Activity Intervals	
<i>Riccardo Dondi, Fabrizio Montecchiani, Giacomo Ortali, Tommaso Piselli, and Alessandra Tappini</i>	11:1–11:14
Parameterized Algorithms for Multi-Label Periodic Temporal Graph Realization	
<i>Thomas Erlebach, Nils Morawietz, and Petra Wolf</i>	12:1–12:16
Computational Power of Opaque Robots	
<i>Caterina Feletti, Lucia Mambretti, Carlo Mereghetti, and Beatrice Palano</i>	13:1–13:19

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn



Leibniz International Proceedings in Informatics

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

Forming Large Patterns with Local Robots in the OBLOT Model <i>Christopher Hahn, Jonas Harbig, and Peter Kling</i>	14:1–14:20
Efficient Shape Formation by 3D Hybrid Programmable Matter: An Algorithm for Low Diameter Intermediate Structures <i>Kristian Hinnenthal, David Liedtke, and Christian Scheideler</i>	15:1–15:20
Temporal Graph Realization from Fastest Paths <i>Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis</i>	16:1–16:18
An Analysis of the Recurrence/Transience of Random Walks on Growing Trees and Hypercubes <i>Shuma Kumamoto, Shuji Kijima, and Tomoyuki Shirai</i>	17:1–17:17
Reconfiguration and Locomotion with Joint Movements in the Amoebot Model <i>Andreas Padalkin, Manish Kumar, and Christian Scheideler</i>	18:1–18:20
Complexity of Boolean Automata Networks Under Block-Parallel Update Modes <i>Kévin Perrot, Sylvain Sené, and Léah Tapin</i>	19:1–19:19
Space and Move-Optimal Arbitrary Pattern Formation on Infinite Rectangular Grid by Oblivious Robot Swarm <i>Avisek Sharma, Satakshi Ghosh, Pritam Goswami, and Buddhadeb Sau</i>	20:1–20:17
Gathering in Carrier Graphs: Meeting via Public Transportation System <i>Haozhi Zheng, Ryota Eguchi, Fukuhito Ooshita, and Michiko Inoue</i>	21:1–21:17

Brief Announcements

Brief Announcement: Collision-Free Robot Scheduling <i>Duncan Adamson, Nathan Flaherty, Igor Potapov, and Paul G. Spirakis</i>	22:1–22:5
Brief Announcement: On the Exponential Growth of Geometric Shapes <i>Nada Almallki, Siddharth Gupta, and Othon Michail</i>	23:1–23:6
Brief Announcement: The Dynamic Steiner Tree Problem: Definitions, Complexity, Algorithms <i>Stefan Balev, Yoann Pigné, Éric Sanlaville, and Mathilde Vernet</i>	24:1–24:6
Brief Announcement: Crash-Tolerant Exploration of Trees by Energy Sharing Mobile Agents <i>Quentin Bramas, Toshimitsu Masuzawa, and Sébastien Tixeuil</i>	25:1–25:5
Brief Announcement: Collision Detection for Modular Robots – It Is Easy to Cause Collisions and Hard to Avoid Them <i>Siddharth Gupta, Marc van Kreveld, Othon Michail, and Andreas Padalkin</i>	26:1–26:5
Brief Announcement: On the Existence of δ -Temporal Cliques in Random Simple Temporal Graphs <i>George B. Mertzios, Sotiris Nikolettseas, Christoforos Raptopoulos, and Paul G. Spirakis</i>	27:1–27:5

■ Preface

This volume contains the papers that were presented at the 3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND), held in Patras, Greece, June 5–7, 2024. SAND is a new conference whose objective is to become the primary venue for original research on fundamental aspects of computing in dynamic networks and computational dynamics, bringing together researchers from computer science and related areas. SAND is seeking important contributions from all viewpoints, including theory and practice, characterized by a marked algorithmic aspect and addressing or being motivated by the role of dynamics in computing. It welcomes both conceptual and technical contributions, as well as novel ideas and new problems that will inspire the community and facilitate the growth of the area.

The program committee of SAND 2024 consisted of:

- Arnaud Casteigts, University of Geneva, Switzerland (chair)
- Fabian Kuhn, University of Freiburg, Germany (chair)
- Karine Altisen, Verimag, France
- Quentin Bramas, University of Strasbourg, France
- Bernadette Charron-Bost, CNRS, ENS Paris PSL, France
- Gianlorenzo D’Angelo, Gran Sasso Science Institute, Italy
- Swan Dubois, Sorbonne Université & Inria, France
- Jessica Enright, University of Glasgow, UK
- Thomas Erlebach, Durham University, UK
- Matthias Függer, CNRS & LMF, ENS Paris-Saclay, France
- Emmanuel Godard, Université Aix-Marseille, France
- Timothy Gomez, Massachusetts Institute of Technology, USA
- Nicolas Hanusse, LaBRI. Bordeaux U., CNRS, France
- Colette Johnen, University of Bordeaux, France
- Spyros Kontogiannis, University of Patras, Greece
- Bernard Mans, Macquarie University, Australia
- Andrea Marino, Università degli Studi di Firenze, Italy
- Yannic Maus, TU Graz, Austria
- Alessia Milani, Aix-Marseille University, France
- Kitty Meeks, University of Glasgow, UK
- George Mertzios, Durham University, UK
- Othon Michail, University of Liverpool, UK
- Hendrik Molter, Ben-Gurion University of the Negev, Israel
- Rotem Oshman, Tel-Aviv University, Israel
- Matthew Patitz, University of Arkansas, USA
- Giuseppe Prencipe, Università di Pisa, Italy
- Michael Raskin, University of Bordeaux, France
- Jason Schoeters, University of Cambridge, UK
- Ana Silva, Universidade Federal do Ceara, Brazil
- Paul Spirakis, University of Liverpool, UK
- Kostas Tsichlas, University of Patras, Greece
- Laurent Viennot, Inria de Paris, France
- Petra Wolf, University of Bergen, Norway
- Viktor Zamaraev, University of Liverpool, UK

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn



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

We are also very grateful to the non-PC-member reviewers who helped us evaluating some of the submissions. Namely, Duncan Adamson, Nada Almkali, Emmanuel Arrighi, Samuel Baguley, Josefran Bastos, Josh Brunner, Daniele Carnevale, Carlos Zapata Carratala, Argyrios Deligkas, Jenny Diomidova, David Doty, Jérôme Féret, Nathan Flaherty, Thomas Gebhart, Daniel Hader, Thekla Hamm, Allen Ibiapina, David Ilcinkas, Ekhine Iruozki, Evangelos Kipouridis, Ralf Klasing, Nina Klobas, Manish Kumar, Patrick Lambein-Monette, Raul Lopes, Nicolas Martins, Kaalkidan Sahele, Frédéric Simard, George Skretas, Frederick Stock, John Sylvester, Kunihiro Wasa, Cai Wood.

SAND 2024 received 43 submissions. The review process was double-blind and each paper was assigned to at least three members of the program committee with relevant expertise and eventually reviewed by them and/or by additional reviewers whenever needed. The program committee accepted 18 papers as regular papers, and 6 as brief announcements. These papers cover a wide range of topics, including dynamic networks and distributed algorithms, mobile computing and robotics, programmable matter, and temporal and dynamic graph algorithms. Keynote talks were given by distinguished researchers, to whom we are grateful: Thomas Erlebach (Durham University, UK), Thomas Nowak (ENS Paris-Saclay, France), and Andréa W. Richa (Arizona State University, USA).


We wish to thank the members of the various committees of SAND as well as its advisory board, for all the hard work that they have put and which has made it possible to set up a new conference. All have been supportive throughout. We are grateful to the program committee members and to the additional reviewers for devoting time and effort in order to come up with a strong conference program. A special thanks goes to the chairs of the organizing committee, Spyros Kontogiannis, Sotiris Nikolettseas, and Kostas Tsihlias. We are also indebted to the chair of the SAND steering committee, Paola Flocchini, for all her support, and to Sotiris Nikolettseas for handling all the financial aspects.

Above all, we thank the authors for submitting their work to SAND 2024. We can assure the reader that in this volume they will find well-presented ideas and results that make substantial contributions to our knowledge on the role of dynamics in computing. We do believe that this volume will inspire further work and will contribute to the further growth of this exciting research area.

June, 2024

Arnaud Casteigts and Fabian Kuhn

■ List of Authors

Duncan Adamson  (4, 22)

Leverhulme Centre for Functional Material Design, University of Liverpool, Liverpool, United Kingdom

Nada Almalki  (23)


Department of Computer Science, University of Liverpool, UK

Stefan Balev (24)

Université Le Havre Normandie, Univ Rouen Normandie, INSA Rouen Normandie, Normandie Univ, LITIS UR 4108, F-76600 Le Havre, France

Judith Beestermöller (5)

ETH Zurich, Switzerland

Adri Bhattacharya  (6)

Indian Institute of Technology Guwahati, Assam, India

Quentin Bramas  (7, 25)

University of Strasbourg, ICUBE, CNRS, Strasbourg, France

Costas Busch (5)


Augusta University, GA, USA

Esteban Christiann (8)

École normale supérieure Paris Saclay, 91190 Gif-sur-Yvette, France

Matthew Connor (9)

Department of Computer Science, University of Liverpool, UK

Riccardo Dondi  (10, 11)

Università degli Studi di Bergamo, Italy

Ryota Eguchi  (21)

Graduate School of Science and Technology, Nara Institute of Science and Technology, Japan

Thomas Erlebach  (1, 12)


Department of Computer Science, Durham University, Durham, UK

Caterina Feletti  (13)


Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy

Nathan Flaherty  (22)


Leverhulme Research Centre for Functional Materials Design, University of Liverpool, UK

Satakshi Ghosh  (20)


Department of Mathematics, Jadavpur University, India

Pritam Goswami  (20)


Department of Mathematics, Jadavpur University, India

Siddharth Gupta  (23, 26)

Department of Computer Science & Information Systems, BITS Pilani Goa Campus, India

Christopher Hahn  (14)


Universität Hamburg, Germany

Jonas Harbig  (14)


Paderborn University, Germany

Kristian Hinnenthal  (15)

Paderborn University, Germany

Michiko Inoue  (21)

Graduate School of Science and Technology, Nara Institute of Science and Technology, Japan

Giuseppe F. Italiano  (6)


Luiss University, Rome, Italy

Shuji Kijima  (17)

Faculty of Data Science, Shiga University, Hikone, Japan

Peter Kling  (14)


Universität Hamburg, Germany

Nina Klobas  (16)


Department of Computer Science, Durham University, UK

Shuma Kumamoto (17)

Graduate School of Mathematical Science, Kyushu University, Fukuoka, Japan

Manish Kumar  (18)


Bar-Ilan University, Israel

Manuel Lafond  (10)


Université de Sherbrooke, Canada

David Liedtke  (15)

Paderborn University, Germany

Jean-Romain Luttringer  (7)

University of Strasbourg, ICUBE, CNRS, Strasbourg, France

Lucia Mambretti  (13)

Milan, Italy

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).


Editors: Arnaud Casteigts and Fabian Kuhn



Leibniz International Proceedings in Informatics


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


- Partha Sarathi Mandal  (6)
Indian Institute of Technology Guwahati, Assam,
India
- Toshimitsu Masuzawa  (25)
Graduate School of Information Science and
Technology, Osaka University, Japan
- Carlo Mereghetti  (13)
Dipartimento di Informatica, Università degli
Studi di Milano, Milan, Italy
- George B. Mertzios  (16, 27)
Department of Computer Science, Durham
University, UK
- Othon Michail  (9, 23, 26)
Department of Computer Science, University of
Liverpool, UK
- Hendrik Molter  (16)
Department of Computer Science, Ben-Gurion
University of the Negev, Beer-Sheva, Israel
- Fabrizio Montecchiani  (11)
Università degli Studi di Perugia, Italy
- Nils Morawietz  (12)
Friedrich Schiller University Jena, Institute of
Computer Science, Jena, Germany
- Sotiris Nikolettseas  (27)
Computer Engineering and Informatics
Department, University of Patras, Greece
- Thomas Nowak  (2)
Université Paris-Saclay, CNRS, ENS
Paris-Saclay, Laboratoire Méthodes Formelles,
France; Institut Universitaire de France, France
- Fukuhito Ooshita  (21)
Faculty of Engineering, Fukui University of
Technology, Japan
- Giacomo Ortali  (11)
Università degli Studi di Perugia, Italy
- Andreas Padalkin  (18, 26)
Paderborn University, Germany
- Beatrice Palano  (13)
Dipartimento di Informatica, Università degli
Studi di Milano, Milan, Italy
- Kévin Perrot (19)
Université publique, Marseille, France;
Aix-Marseille Univ, CNRS, LIS, Marseille,
France
- Yoann Pigné (24)
Université Le Havre Normandie, Univ Rouen
Normandie, INSA Rouen Normandie,
Normandie Univ, LITIS UR 4108, F-76600 Le
Havre, France
- Tommaso Piselli  (11)
Università degli Studi di Perugia, Italy
- Igor Potapov  (22)
Department of Computer Science, University Of
Liverpool, UK
- Christoforos Raptopoulos  (27)
Department of Mathematics, University of
Patras, Greece
- Andréa Werneck Richa  (3)
School of Computing and Augmented
Intelligence, Arizona State University, Tempe,
AZ, USA
- Eric Sanlaville  (8)
Université Le Havre Normandie, Univ Rouen
Normandie, INSA Rouen Normandie,
Normandie Univ, LITIS UR 4108, F-76600 Le
Havre, France
- Éric Sanlaville  (24)
Université Le Havre Normandie, Univ Rouen
Normandie, INSA Rouen Normandie,
Normandie Univ, LITIS UR 4108, F-76600 Le
Havre, France
- Buddhadeb Sau  (20)
Department of Mathematics, Jadavpur
University, India
- Christian Scheideler  (15, 18)
Paderborn University, Germany
- Jason Schoeters  (8)
University of Cambridge, United Kingdom
- Sylvain Sené (19)
Université publique, Marseille, France;
Aix-Marseille Univ, CNRS, LIS, Marseille,
France
- Avisek Sharma  (20)
Department of Mathematics, Jadavpur
University, India
- Tomoyuki Shirai  (17)
Institute of Mathematics for Industry, Kyushu
University, Fukuoka, Japan
- George Skretas  (9)
Hasso Plattner Institute, University of Potsdam,
Germany

Paul G. Spirakis  (16, 22, 27)
Department of Computer Science, University of
Liverpool, UK

Léah Tapin (19)
Aix-Marseille Univ, CNRS, LIS, Marseille,
France

Alessandra Tappini  (11)
Università degli Studi di Perugia, Italy


Sébastien Tixeuil  (7, 25)
Sorbonne University, CNRS, LIP6, Institut
Universitaire de France, Paris, France

Marc van Kreveld  (26)
Utrecht University, The Netherlands

Mathilde Vernet (24)
LIA, Avignon Université, Avignon, France

Roger Wattenhofer (5)
ETH Zurich, Switzerland

Petra Wolf  (12)
LaBRI, CNRS, Université de Bordeaux,
Bordeaux INP, France

Haozhi Zheng  (21)
Graduate School of Science and Technology,
Nara Institute of Science and Technology, Japan

Exploration and Rendezvous in Temporal Graphs

Thomas Erlebach  

Department of Computer Science, Durham University, Durham, UK

Abstract

Given a temporal graph \mathcal{G} and a start vertex v in \mathcal{G} , the temporal exploration problem (TEXP) is the problem of determining a temporal walk that starts at v and visits all vertices of \mathcal{G} , with the objective of minimizing the time when the last unvisited vertex is reached. Studies have investigated the (parameterized) complexity and approximability of TEXP and the worst-case number of time steps required to complete an exploration. While many upper and lower bounds have been obtained for different settings, there are still some large gaps that pose interesting open problems. In this talk, we will give an overview of known results and techniques as well as open problems. Furthermore, we will discuss recent results (from joint work with Konstantinos Dogeas, Frank Kammer, Johannes Meintrup, and William K. Moses Jr) about exploiting symmetries in temporal graphs to get faster exploration. We view the number of automorphism orbits of the temporal graph as a new parameter, termed the orbit number, that may also be useful in other contexts. Finally, we show how a subroutine for quickly exploring a single orbit of the graph can be exploited to solve a certain rendezvous problem with two agents using a near-linear number of time steps in every always-connected temporal graph.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Mathematics of computing \rightarrow Discrete mathematics

Keywords and phrases Dynamic graphs, parameterized algorithms, orbit number

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.1

Category Invited Talk



© Thomas Erlebach;

licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 1; pp. 1:1–1:1

Leibniz International Proceedings in Informatics



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

Distributed Computation with Bacteria

Thomas Nowak   

Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, France
Institut Universitaire de France, France

Abstract

Computing via synthetically engineered bacteria is a vibrant and active field with numerous applications in bio-production, bio-sensing, and medicine. Motivated by the lack of robustness and by resource limitation inside single cells, distributed approaches with communication among bacteria have recently gained in interest. In this talk, we describe the most important distributed approaches to synthetic biology with bacteria and discuss the crucial task of mathematical modeling of these systems. A particular problem is that of population growth happening concurrently, and possibly interfering, with the desired bio-computation. Specifically, we present a fast protocol in systems with continuous population growth for the majority consensus problem and prove that it correctly identifies the initial majority among two inputs with high probability. We also present a fast protocol that correctly computes the NAND of two inputs with high probability. By combining NAND gates with the majority consensus protocol as an amplifier, it is possible to compute arbitrary Boolean functions. The proposed protocols help set the stage for bio-engineered distributed computation that directly addresses continuous stochastic population growth.

Own work presented in this talk is mostly based on joint work with Da-Jung Cho, Matthias Függer, Corbin Hopper, Manish Kushwaha, and Quentin Soubeyran.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Microbiological circuits, chemical reaction networks, birth-death processes

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.2

Category Invited Talk

Funding Part of the research described in this talk has received financial support from the French National Research Agency (ANR) under grant agreements ANR-21-CE48-0003, ANR-23-CE04-0008, and ANR-23-CE45-0013.



© Thomas Nowak;

licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics



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

Algorithmic Programmable Matter: From Local Markov Chains to “Dumb” Robots

Andréa Werneck Richa   

School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, USA

Abstract

Many programmable matter systems have been developed, including modular and swarm robotics, synthetic biology, DNA tiling, and smart materials. We describe programmable matter as an abstract collection of simple computational elements (particles) with limited memory that each execute distributed, local algorithms to self-organize and solve system-wide problems, such as movement, reconfiguration, and coordination. Self-organizing particle systems (SOPS) have many interesting potential applications like coating objects for monitoring and repair purposes, and forming nano-scale devices for surgery and molecular-scale electronic structures.

We describe some of our work on the algorithmic foundations of programmable matter, investigating how macro-scale system behaviors can naturally emerge from local micro-behaviors by individual particles: We utilize tools from statistical physics and Markov chain analysis to translate Markov chains defined at a system level into distributed, local algorithms for SOPS that drive the desired emergent collective behavior for the problems of compression, separation, and foraging, among others. We further establish the notion of algorithmic matter, where we leverage standard binary computation, as well as physical characteristics of the robots and interactions with the environment in order to implement our micro-level algorithms in actual testbeds composed of robots that are not capable of any standard computation. We conclude by addressing full concurrency and asynchrony in SOPS.

This is joint work with Dana Randall and Dan Goldman (Georgia Tech), Michael Strano (MIT), Todd Murphey (Northwestern), Josh Daymude (Arizona State University), Sarah Cannon (Claremont McKenna), Christian Scheideler (University of Paderborn) and their research labs.

2012 ACM Subject Classification Theory of computation → Self-organization; Theory of computation → Distributed computing models; Theory of computation → Random walks and Markov chains

Keywords and phrases Programmable matter, Self-organizing particle systems, Biologically-inspired distributed algorithms, Local Markov chains, Emergent collective behavior

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.3

Category Invited Talk

Funding *Andréa Werneck Richa*: Supported in part by the National Science Foundation (NSF) award CCF-210691739 and by U.S. Army Research Office (ARO) award MURI W911NF-19-1-0233.



© Andréa Werneck Richa;

licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 3; pp. 3:1–3:1

Leibniz International Proceedings in Informatics



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

Harmonious Colourings of Temporal Matchings

Duncan Adamson  

Leverhulme Centre for Functional Material Design,
University of Liverpool, Liverpool, United Kingdom

Abstract

Graph colouring is a fundamental problem in computer science, with a large body of research dedicated to both the general colouring problem and restricted cases. *Harmonious colourings* are one such restriction, where each edge must contain a globally unique pair of colours, i.e. if an edge connects a vertex coloured x with a vertex coloured y , then no other pair of connected vertices can be coloured x and y . Finding such a colouring in the traditional graph setting is known to be NP-hard, even in trees. This paper considers the generalisation of harmonious colourings to *Temporal Graphs*, specifically (k, t) -Temporal matchings, a class of temporal graphs where the underlying graph is a matching (a collection of disconnected components containing pairs of vertices), each edge can appear in at most t timesteps, and each timestep can contain at most k other edges. We provide a complete overview of the complexity landscape of finding *temporal harmonious colourings* for (k, t) -matchings. We show that finding a *Temporal Harmonious Colouring*, a colouring that is harmonious in each timestep, is NP-hard for (k, t) -Temporal Matchings when $k \geq 4, t \geq 2$, or when $k \geq 2$ and $t \geq 3$. We further show that this problem is inapproximable for $t \geq 2$ and an unbounded value of k , and that the problem of determining the temporal harmonious chromatic number of a $(2, 3)$ -temporal matching can be determined in linear time. Finally, we strengthen this result by a set of upper and lower bounds of the temporal harmonious chromatic number both for individual temporal matchings and for the class of (k, t) -temporal matchings.

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

Keywords and phrases Temporal Graphs, Harmonious Colouring, NP-Completeness

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.4

Acknowledgements This work was supported by the Leverhulme Research Centre for Functional Materials Design. The author would like to thank the reviewers for their helpful comments.

1 Introduction

In real-world settings, networks are changing structures with connections between vertices changing at each time step. Temporal graphs provide a natural means of modelling such a network. Formally, a *temporal graph* G is defined by a static collection of vertices V and sequence of edge sets E_1, E_2, \dots, E_T , where T is the *lifetime* of the graph. The underlying graph \mathcal{G} of a temporal graph G is the static graph formed by taking the vertex set V and the union of edge sets $E_1 \cup E_2 \cup \dots \cup E_T$. Temporal graphs have recently become a well-studied object, with a particular focus on reachability [3, 5, 11] and exploration [6, 7, 13].

Graph colouring, despite being a fundamental problem in computer science, has remained relatively unstudied within temporal graphs. The main reason for this is that, for the general problems of vertex and edge colouring, finding such a static colouring on a temporal graph is equivalent to finding a static colouring on the underlying graph. Recent work on graph colouring in the temporal setting can be split into two broad directions. First is the work of Yu, Bar, Basu, and Ramanathan [14] and Ghosal and Ghosh [8], who focused on finding a sequence of colourings for each node, with the twin goals of minimising the total number of colours and the number of changes of the colour of each vertex. Second is the work by Mertzios, Molter, and Zamaraev [12] on *sliding window* colourings, where the goal is to provide a colouring such that each (active) edge is coloured properly at least once within



© Duncan Adamson;

licensed under Creative Commons License CC-BY 4.0

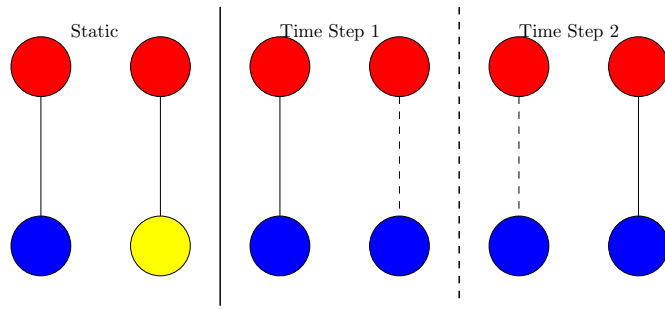
3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 4; pp. 4:1–4:11

Leibniz International Proceedings in Informatics



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



■ **Figure 1** An example of two harmonious colourings. Left is the harmonious colouring for the static underlying graph, while the right provides a simple example of a harmonious temporal colouring. Observe that the static colouring needs three colours in order to avoid having two edges with the same pair of colours. In the temporal example, note that neither edge is active at the same time-step. Therefore, both can have the same colours.

each window of a given size. In [12], the authors prove a series of results on the hardness of this problem, as well as a number of exact and approximation algorithms for several special cases of the problem. The computational aspects of this problem have been further studied by Marino and Silva [10] who, in particular, considered the problem on temporal graphs where each edge is either active for at least t times steps in a row or at least t snapshots over the lifetime of the graph.

In this paper, we are interested in finding a *harmonious colouring* of a temporal graph. Harmonious colourings of static graphs are colourings where the colour pair on each edge is globally unique. In the static setting, this problem is known to be very challenging, with hardness results for a wide variety of otherwise simple graph classes such as interval and permutation graphs [1], bipartite permutation and quasi-threshold graphs [2], and trees [4]. Despite the challenge, the temporal setting offers a slight relaxation of the problem, namely a requirement that the pair of colours on each edge is unique only within the same snapshot. This means that any pair of edges that are not active at the same time may share a colouring. Such a colouring is called a *temporal harmonious colouring*. Figure 1 illustrates that such a colouring can be found using fewer colours than in the underlying graph.

Noting that this problem is trivially hard when any snapshot includes a graph that is known to be hard to harmoniously colour, this paper focuses on the class of *temporal matchings*. In a temporal path, the underlying graph is a matching graph. Beyond the general class, we look at (k, t) -temporal matchings, where at each snapshot the graph contains at most t active edges, and each edge can be active for at most k time steps within the lifetime of the graph. Restricting the graphs in this way allows a more precise understanding of the complexity landscape. Further, showing that this problem remains hard for perhaps the simplest non-trivial graph class highlights the difference between the complexity of problems on static graphs and temporal ones.

Our Contribution

This paper provides two main results. In Section 3, we show that the problem of finding a temporal harmonious colouring is NP-complete for $(2, 4)$ -temporal matchings and $(3, 2)$ -temporal matchings. This shows the problem to be highly challenging even for a relatively simple class of temporal graphs. In addition to the hardness result, we show that the problem of determining the temporal harmonious chromatic number (the minimum number

of colours needed for a temporal harmonious colouring) can not be approximated within polynomial time within a factor of $n^{(1-\epsilon)/2}$ for any positive value ϵ for any $(k, 2)$ -temporal paths when the value of k is unbounded. In Section 4, we show that the temporal chromatic number (the minimum number of colours needed for a temporal harmonious colouring) of $(2, 3)$ -matchings can be determined in linear time. Finally, in Section 5, we provide a series of bounds on the temporal harmonious chromatic number of (k, t) -temporal paths, including a $\sqrt{4 \min(k, t) - 2}$ lower bound and a $t(k - 1) + 2$ upper bound.

2 Notation and Definitions

We use the notation $[n]$ to denote the set $\{1, 2, \dots, n\}$. For a graph G , we denote by $V(G)$ and $E(G)$ its vertex set and edge set respectively. A *temporal graph* \mathcal{G} is an ordered sequence (G_1, G_2, \dots, G_T) of static graphs over the common set V of vertices. The static graphs G_i , $i \in [T]$ are called *snapshots* of \mathcal{G} , and T is called *the lifetime* of the temporal graph. We say that the edges in $E(G_i)$ are *active* at time step i . The *underlying graph* of the temporal graph \mathcal{G} is the graph formed by taking the union of its snapshots, i.e. $(V, \bigcup_{i \in [T]} E(G_i))$.

► **Definition 1.** A temporal graph \mathcal{G} of lifetime T is a (k, t) -temporal graph if every edge of its underlying graph is active in at most t time steps, and every snapshot has at most k edges.

If the underlying graph of \mathcal{G} is a path, we say that \mathcal{G} is a (k, t) -temporal path; similarly, if the underlying graph is a matching (i.e. graph of maximum degree at most 1), we say that \mathcal{G} is a (k, t) -temporal matching.

For a natural number c , a c -colouring of a graph G is a mapping $\psi : V(G) \rightarrow [c]$ such that for any two adjacent vertices u, v we have $\psi(u) \neq \psi(v)$. If G admits a c -colouring we say that G is c -colourable. The *chromatic number* of G is the smallest c such that G is c -colourable. A c -colouring ψ of G is *harmonious* if, for every pair of edges $\{v_1, u_1\}, \{v_2, u_2\} \in E$, $\{\psi(v_1), \psi(u_1)\} \neq \{\psi(v_2), \psi(u_2)\}$. The *harmonious chromatic number* of G is the smallest value c such that G admits a harmonious c -colouring. Given a graph G and a natural number c the Harmonious Colouring problem asks whether G admits a harmonious c -colouring or not. The optimisation variant of this problem asks to find the harmonious chromatic number of G .

► **Definition 2.** For a temporal graph $\mathcal{G} = (G_1, G_2, \dots, G_T)$ over a vertex set V , a c -colouring $\phi : V \rightarrow [c]$ of its underlying graph is a c -temporal harmonious colouring if ψ is a harmonious colouring of every snapshot G_i , $i \in [T]$. The smallest c such that \mathcal{G} admits a c -temporal harmonious colouring is the temporal harmonious chromatic number.

TEMPORAL HARMONIOUS COLOURING (THC)

Input: A temporal graph \mathcal{G} , and an integer c .

Output: YES, if there exists a temporal harmonious c -colouring of \mathcal{G} ; NO otherwise.

As in the static case, the Temporal Harmonious Colouring can be phrased as an optimisation problem, asking for the temporal harmonious chromatic number of the input graph \mathcal{G} .

Since temporal graphs generalise (static) graphs, TEMPORAL HARMONIOUS COLOURING is at least as hard as the Harmonious Colouring problem. In particular, the NP-hardness of the latter problem on trees [4] implies NP-hardness of the former on temporal graphs where snapshots are restricted to trees. On the other hand, the Harmonious Colouring problem is

4:4 Harmonious Colourings of Temporal Matchings

rather simple in matchings. Indeed, if G is a matching with m edges, then its harmonious chromatic number is the smallest c such that $\binom{c}{2} = \frac{c(c-1)}{2} \geq m$, i.e. the smallest number of colours providing at least m unique pairs of different colours.

It turns out that in the temporal setting, the problem becomes much harder, even in the simplest case of temporal graphs whose underlying graph is a matching. In particular, we will show that TEMPORAL HARMONIOUS COLOURING is NP-complete on (k, t) -temporal matchings when $k \geq 4$ and $t \geq 2$ or when $k \geq 2$ and $t \geq 3$. On the other hand, we show that the temporal harmonious chromatic number can be determined in linear time for $(2, 3)$ -temporal matchings. Thus, our results provide computational complexity dichotomy for TEMPORAL HARMONIOUS COLOURING on (k, t) -temporal matchings. We further provide a set of bounds on the chromatic number of (k, t) -temporal paths.

3 Hardness of Harmonious Colourings on (k, t) -Temporal Matching

In this section, we show that the problem of finding a temporal harmonious colouring is NP-hard even for $(2, 4)$ -temporal matchings and $(3, 2)$ -temporal matching. We start by providing a tool for constructing temporal matchings from a static graph. Informally, the goal is to construct a matching with a temporal harmonious chromatic number that can be used to determine the chromatic number of the static graph for low-degree graphs. We strengthen this reduction by showing that each snapshot contains at most 2 edges.

► **Lemma 3.** *Let $G = (V, E)$ be a static graph with a chromatic number χ and maximum degree $\Delta > 1$, then there exists a $(2, \Delta)$ -temporal matching $G' = (V', E')$ such that the temporal harmonious chromatic number of G' is χ' such that χ' is the smallest value for which $\chi \leq \frac{\chi'(\chi'-1)}{2}$. Further, G' has a lifetime of $|E|$.*

Proof. For each vertex $v \in V$, a pair of vertices v_1, v_2 are added to V' . A snapshot is constructed for each edge $(v, u) \in E$, with the edge set $E_{v,u}$ is constructed containing the edges (v_1, v_2) and (u_1, u_2) . Note that as each vertex v has degree at most Δ , the edge (v_1, v_2) appears in at most Δ timesteps. Under the current construction, the graph is a matching rather than a path.

Let ψ be a temporal harmonious colouring of G' . Observe that at each snapshot, there exists exactly 2 edges, with each edge corresponding to a vertex in G and the snapshot corresponding to an edge in G . Therefore, for any colouring to be harmonious, given any edge $(v, u) \in E$, the pairs $(\psi(v_1), \psi(v_2))$ and $(\psi(u_1), \psi(u_2))$ must be distinct. Therefore, there must be a mapping from the set of distinct pairs of colours from $[\psi']$ to some set of colours of size at most $\gamma = \frac{\chi'(\chi'-1)}{2}$. Let ϕ be a γ -colouring of G such that each vertex of G is coloured using the pair of colours given by the χ' colouring of G' .

Assume, for the sake of contradiction, that ϕ is not a valid colouring of G . Then, there must exist some edge $(v, u) \in E$ such that $\phi(v) = \phi(u)$. In this case, in the colouring of G' , $(\psi(v_1), \psi(v_2)) = (\psi(u_1), \psi(u_2))$. However, as there exists some snapshot of G' containing the edges (v_1, v_2) and (u_1, u_2) , this contradicts the assumption that G' has a valid temporal harmonious colouring. Therefore, ϕ must be a valid colouring of G . Further, if $\gamma < \chi$, then there must exist a colouring of G using fewer than χ colours, contradicting the assumption that χ is the chromatic number of G .

In the other direction, let ψ be a χ -colouring of G . A colouring ψ' of G' is constructed via a bijective mapping $\lambda : [\chi] \mapsto \{(x, y) | x, y \in [\chi'], x > y\}$. Assume, for the sake of contradiction, that ψ' is not a valid colouring of G' . Then, there must exist some snapshot at time step i such that for the pair of edges $(v_1, v_2), (u_1, u_2) \in E_i$, $(\psi'(v_1), \psi'(v_2)) = (\psi'(u_1), \psi'(u_2))$.

Following the above construction, such a snapshot must correspond to an edge $(v, u) \in E$. As ψ is a valid χ -colouring of G $\psi(v) \neq \psi(u)$. Therefore, the $\lambda(\psi(v)) \neq \lambda(\psi(u))$, and hence $(\psi'(v_1), \psi'(v_2)) \neq (\psi'(u_1), \psi'(u_2))$, contradicting the assumption that ψ' is not a valid colouring. Further, χ' must be the smallest value such that $\chi \leq \frac{\chi'(\chi'-1)}{2}$. \blacktriangleleft

► **Theorem 4.** *The problem of determining if a given (k, t) -temporal matching \mathcal{G} has a temporal harmonious chromatic number of c is NP-complete for any $k \geq 2$ and $t \geq 4$.*

Proof. Let $G = (V, E)$ be a 3-regular graph. As established by Leven and Galil [9], determining if G has an edge colouring of size 3 is an NP-complete problem. In order to reduce the problem of finding an edge colouring of G to finding a temporal harmonious colouring on a $(2, 4)$ -temporal graph, let H be the edge adjacency graph of G . Note that H has a maximum degree of 4. Using Lemma 3, H can be transformed into a $(2, 4)$ -temporal matching G' .

If H has a chromatic number of 3, then the temporal harmonious chromatic number of G' is exactly 3. Otherwise, if the chromatic number of H is either 4 or 5, the temporal harmonious chromatic number of G' is 4. Therefore, any algorithm to determine the temporal harmonious chromatic number of a $(2, 4)$ -temporal matching can also determine if a 3-regular graph has a 3-edge colouring. Hence finding the temporal harmonious chromatic number of a $(2, 4)$ -temporal matching is NP-hard.

To show that the problem is NP-Complete, note that any colouring can be verified as a temporal harmonious colouring in polynomial time. \blacktriangleleft

► **Corollary 5.** *The problem of finding the temporal harmonious chromatic number of a (k, t) -temporal path is NP-hard for any $t \geq 2, k \geq 4$.*

► **Corollary 6.** *The problem of finding the temporal harmonious chromatic number of a (k, t) -temporal cycle is NP-hard for any $t \geq 2, k \geq 4$.*

Building on the above results, we now show that determining the harmonious chromatic number of a $(3, 2)$ -temporal path is NP-complete.

► **Theorem 7.** *The problem of determining if a given (k, t) -temporal matching \mathcal{G} has a temporal harmonious chromatic number of c is NP-complete for any $k \geq 3$ and $t \geq 2$.*

Proof. This proof follows a similar outline to the proof of Theorem 4. As before, we take a cubic graph $G = (V, E)$ and construct a $(3, 2)$ -temporal matching G' , such that the edge-chromatic number of G is equal to the harmonious temporal harmonious chromatic number of G' . The temporal matching G' is constructed as follows. For each edge $e \in E$, a pair of vertices e_1, e_2 are constructed and connected in the underlying graph of G' . For each vertex $v \in V$, a snapshot is constructed containing the pair of vertices (e_1, e_2) for every edge e incident to v . As in Lemma 3, this matching may be transformed into a path by adding dummy vertices between the pairs.

To show that this problem is NP-hard, first assume that G has an edge chromatic number of 3. For the sake of contradiction, assume further that the temporal harmonious chromatic number of G' is greater than 3. Let ψ be an edge colouring of G using 3 colours, and let

$$f(c) = \begin{cases} (1, 2) & c = 1 \\ (1, 3) & c = 2 \\ (2, 3) & c = 3 \end{cases} \text{ be a function mapping the colours given by } \psi \text{ to pairs of colours. For}$$

each edge $e \in E$, let the colours of e_1 and e_2 be equal to the colours given by $f(\psi(e))$. As the mapping f does not allow any pair of incident vertices to share a colour, this colouring must be valid. Further, for each snapshot in G' , note that the only active edges are those

corresponding to edges incident to a given vertex in V . Therefore, any given snapshot is not harmonious if and only if there is a pair of edges in G sharing a colour, contradicting the assumption that ψ is a valid colouring of G . Hence, G' has a temporal harmonious chromatic number greater than 3 if and only if G does not have an edge colouring of size 3.

In the other direction, assume for the sake of contradiction that the temporal harmonious chromatic number of G' is 3, while the edge chromatic number of G is greater than 3. Let ϕ

be a vertex colouring of G' and $f'(c) = \begin{cases} 1 & (1, 2) \text{ or } (2, 1) \\ 2 & (1, 3) \text{ or } (3, 1) \\ 3 & (2, 3) \text{ or } (3, 2) \end{cases}$ be a function mapping pairs of

vertex colours to a set of 3 colours. For each edge $e \in E$, let e be coloured $f'(\phi(e_1), \phi(e_2))$. For the edge chromatic number of G to be greater than 3, this colouring must not be feasible. Observe that at each snapshot of G' , the colouring is harmonious. Therefore, given any pair of edges $e, h \in E$ such that e and h are incident to the vertex v , the pairs $(\psi(e_1), \psi(e_2))$ and $(\psi(h_1), \psi(h_2))$ must be distinct. Hence, $f(\psi(e_1), \psi(e_2)) \neq f(\psi(h_1), \psi(h_2))$ and by extension the edges coloured using this mapping must be distinct. Therefore using the mapping given by f , either G has a proper edge colouring with 3 colours, or G' does not have a temporal harmonious chromatic number of 3, contradicting the original assumption.

Hence, the temporal harmonious chromatic number of G' is 3 if and only if the edge chromatic number of G is 3. By extension the problem of computing the temporal harmonious chromatic number of a $(3, 2)$ -temporal matching is NP-hard. Further, any colouring can be verified as a temporal harmonious colouring in polynomial time. Therefore the problem of computing the temporal harmonious chromatic number of a $(3, 2)$ -temporal matching is NP-complete. ◀

► **Corollary 8.** *The problem of finding the temporal harmonious chromatic number of a (k, t) -temporal path is NP-hard for any $t \geq 3, k \geq 2$.*

► **Corollary 9.** *The problem of finding the temporal harmonious chromatic number of a (k, t) -temporal cycle is NP-hard for any $t \geq 3, k \geq 2$.*

3.1 Hardness of Approximation

Building on Theorem 4, this section shows that the temporal harmonious colouring problem is hard to approximate even on (k, t) -temporal matchings. This bound utilises the tools of Lemmas 3 as a basis for converting existing results on the inapproximability of colouring problems to the temporal harmonious setting. In this section, we consider the more general class of $(\infty, 2)$ -temporal matchings, where there is no bound on the number of times each edge appears in the graph. By focusing on the restricted case of $(\infty, 2)$ -temporal graphs, we show that the general case is at least as hard, and indeed likely to be much harder.

► **Theorem 10.** *It is NP-hard to approximate the temporal harmonious number of a $(\infty, 2)$ -temporal matching within a factor of $n^{(1-\epsilon)/2}$ for any $\epsilon > 0$, where n is the number of vertices in the graph.*

Proof. From Zuckerman [15], it is known that it is NP-hard to approximate the harmonious number of a graph G within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$. Following the construction given in Lemma 3, given any graph G with maximum degree Δ , and chromatic number χ , a $(2, \Delta)$ -temporal matching can be constructed with a temporal harmonious number γ such that γ is the smallest value satisfying $\chi \leq \frac{\gamma(\gamma-1)}{2} \leq \gamma^2$.

Let α be a polynomial time approximation of χ , and β be a polynomial time approximation of γ . Following [15], $\alpha \geq n$. Similarly, note that any approximation of γ provides an upper bound of χ using the inequality $\gamma^2 \geq \alpha$. Hence β is the smallest value such that $\beta^2 \geq \alpha$ and by extension $\beta \geq \sqrt{\alpha}$. Therefore, any $\sqrt{n^{1-\epsilon}}$ approximation of γ provides a $n^{1-\epsilon}$ approximation of χ . Therefore, γ can not be approximated in polynomial time within $n^{(1-\epsilon)/2}$ for any positive value ϵ for the class $(\infty, 2)$ -temporal matchings unless $P = NP$. ◀

4 Temporal Harmonious Chromatic Number of (2,3)-Temporal Matchings

In this section, we strengthen the hardness result from Section 3 by showing that the temporal harmonious chromatic number of (2,3)-temporal matchings can be determined in linear time. Note that any (2,2)-temporal matching is also a (2,3)-temporal matching, and thus this linear bound also holds. This shows that the hardness bound from Section 3 is “tight”, in the sense that the case of (2,4)-temporal matchings and (3,2)-temporal matchings are the smallest values of k and t for which the problem is hard. Further, this highlights a large gulf in the complexity space, moving from a problem that is solvable in linear time to an NP-complete problem with a relatively small change in the parameters.

We start with the simple case of finding a temporal harmonious colouring of a (1, k)-temporal matching.

► **Lemma 11.** *A temporal harmonious colouring of any (1, k)-temporal matching with n vertices and 2 colours can be found in $O(n)$ time.*

Proof. Observe that in a (1, k)-temporal matching, each snapshot contains at most 1 edge. Therefore, any valid colouring is also a temporal harmonious colouring. As the underlying graph is a matching, a 2-colouring can be found by a greedy algorithm, iterating over the set of edges and colouring one end node colour 1, and the other node colour 2. ◀

We now provide the main result of this section, namely a proof that the temporal harmonious chromatic number of (2,3)-temporal matchings can be determined in linear time. The high-level idea behind this proof is to provide a construction of the *temporal edge adjacency graph* of the temporal matching \mathcal{G} . Informally, such a graph represents edges with vertices and connects them if and only if the corresponding edges are active in the same snapshot. By finding a colouring of this graph, a mapping can be used to connect the colours of the edges to the colours of vertex pairs. As the temporal edge adjacency graph has a maximum degree of 3, the chromatic number of the graph can be determined in linear time using Brooks’ Theorem.

► **Theorem 12.** *The temporal harmonious chromatic number of a (2,3)-temporal matching \mathcal{G} can be determined in linear time.*

Proof. At a high level, this is done by reversing the construction from Lemma 3. We assume, without loss of generality, that \mathcal{G} does not contain any vertices of degree 0. Note that any such vertices may be coloured arbitrarily without conflicting with the temporal harmonious colouring condition. Let $G' = \{V', E'\}$ be the *temporal edge adjacency graph* of \mathcal{G} . Formally, G' is constructed as follows. For every edge $e \in \mathcal{G}$, a vertex is constructed in V' and labelled with the edge e . Given a pair of edges $e_1, e_2 \in \mathcal{G}$, an edge is constructed between v_{e_1} and v_{e_2} if and only if there exists some snapshot of \mathcal{G} in which both e_1 and e_2 are active. Note that for a (2,3)-temporal matching, each edge appears in at most 3 snapshots, and each time step contains at most 2 edges. Therefore, G' has a degree of at most 3.

Let ψ be a colouring of G' using c -colours. Let c' be the smallest value such that $c \leq \frac{c'(c'-1)}{2}$. A temporal harmonious c' -colouring ϕ of \mathcal{G} is constructed from ψ by constructing a mapping f from c to the set $\{\{x, y\} | x, y \in [c], x \neq y\}$. Using this mapping, each edge $e \in \mathcal{G}$ is coloured using $f(v_e)$. We first show that ϕ is a valid c' -colouring. Observe that each pair in $\{\{x, y\} | x, y \in [c], x \neq y\}$ contains two unique colours from $[c']$, and further each vertex belongs to only a single edge. Therefore, by assigning the colours from the pair $f(v_e)$ to the two vertices, ϕ produces a valid c' -colouring of \mathcal{G} . Further, as each pair of edges $e_1, e_2 \in \mathcal{G}$ that are active in the same snapshot are assigned different colours by ψ , the pair of colours on the vertices incident to e_1 and e_2 are distinct. Therefore, ψ is a temporal harmonious c' -colouring. Therefore, given a c -colouring of the edge temporal graph G , a c' colouring of \mathcal{G} can be determined in polynomial time.

As G' is a cubic graph, G' has a chromatic number of 4 if and only if G' contains the complete graph K_4 . As the clique K_4 can be detected in linear time, it is possible to determine if G' has a chromatic number of 4 in polynomial time. By extension, if G' has a chromatic number of 4, then the smallest value c' such that $\frac{c'(c'-1)}{2} \geq 4$ is 4. Hence the temporal harmonious chromatic number of \mathcal{G} is 4 if and only if G' contains the graph K_4 as a subgraph. On the other hand, G' has a chromatic number of 2 if and only if it is bipartite, and further, it is possible to determine this in linear time. If G' is bipartite, then the temporal harmonious chromatic number c' of \mathcal{G} is 3 as 3 is the smallest value such that $\frac{c'(c'-1)}{2} \geq 2$. Further, G' has a chromatic number of 1 if and only if G' contains only disconnected vertices. In this case, no pair of edges in \mathcal{G} are active at the time step. Therefore any proper colouring of \mathcal{G} is also a temporal harmonious colouring. Finally, if G' does not have a chromatic number of 1, 2 or 4, then the chromatic number of G' must be 3, and by extension, the temporal harmonious chromatic number of \mathcal{G} is 3. Therefore, the temporal harmonious chromatic number of \mathcal{G} can be determined in linear time for any $(2, 3)$ -temporal matching. ◀

5 Further Bounds

This section strengthens the results of Section 3 by providing stronger bounds on the temporal harmonious chromatic number of (k, t) -temporal paths. Note that any bounds on (k, t) -temporal paths also apply to (k, t) -matchings. This is done in two ways. First, we provide an upper bound by constructing a linear time greedy algorithm for finding a temporal harmonious colouring using at most $t(k-1) + 2$ colours. Secondly, we provide a series of lower bounds to strengthen the upper bound.

► **Lemma 13.** *Algorithm 1 finds a $(t(k-1) + 2)$ -colouring of any (k, t) -temporal path $G = (V, E_1, E_2, \dots, E_T)$ in $O(n \cdot k \cdot t)$ time, where n is the number of vertices in G .*

Proof. We assume, without loss of generality, that each node in G is labelled from 1 to n such that vertex 1 is a terminal vertex on the path \mathcal{G} and vertex i is incident to $i+1$ for every $i \in [n-1]$. Note that the first vertex can be arbitrarily coloured in the first step without violating the colouring constraint. Similarly, the second vertex can be coloured any colour other than the colour of vertex 1. The remaining vertices are coloured in order from 3 to n . At each step, we treat the vertex as though were the terminal vertex. In doing so, it becomes only necessary to check that the edge $(i-1, i)$ satisfies the harmonious condition and that the colour of i is distinct from $i-1$ to satisfy the colouring condition. Therefore, by an exhaustive search of each previous edge, a list of colours that can be allowed at position i can be determined.

As there are at most t time steps in which the edge $(i-1, i)$ is active, and at most k edges at each snapshot, $(i-1, i)$ can conflict with at most $t(k-1)$ edges. Further, for each edge $(j-1, j)$ appearing at the same timestep as $(i-1, i)$, the colour $col(j-1)$ is removed

■ **Algorithm 1** Greedy Algorithm.

```

1: procedure COLOUR( $G = (V, E_1, E_2, \dots, E_T), c$ )
2:    $col(1) \leftarrow 1$ 
3:    $col(2) \leftarrow 2$ .
4:   for  $i \in V \setminus \{1, 2\}$  do
5:      $Colours \leftarrow \{1, 2, \dots, c\} \setminus \{col(i-1)\}$ 
6:     for  $k \in ActiveTimesteps((i-1, i))$  do
7:       for  $(j-1, j) \in E_k$  do
8:         if  $((j, j+1), (i-1, i)) \in E_k$  then
9:           if  $col(j) = col(i-1)$  then
10:             $Colours \leftarrow Colours \setminus \{col(j+1)\}$ 
11:          end if
12:          if  $col(j+1) = col(i-1)$  then
13:             $Colours \leftarrow Colours \setminus \{col(j)\}$ 
14:          end if
15:        end if
16:      end for
17:    end for
18:     $col(i) \leftarrow \min(Colours)$ 
19:  end for
20: end procedure

```

from the set of potential colours of i if and only if $col(j) = col(i-1)$. Similarly, the colour $col(j)$ is removed from the set of candidate colours of i if and only if $col(j-1) = col(i-1)$. As $col(j) \neq col(j-1)$, at most 1 colour can be removed for each edge that appears in the same timestep as $(i-1, i)$. Hence at most $t(k-1) + 1$ colours are forbidden for i , therefore as long as i has a palette of size at least $t(k-1) + 2$, there must always be at least one colour that i can choose. Therefore, G must have a $(t(k-1) + 2)$ colouring.

To get the time complexity, we assume that each edge is labelled with the time step at which it appears. Note that such a list can be computed by checking the set of active edges for each snapshot in the graph. As there are at most $n-1$ edges, each of which are active for at most t time steps, this will take at most $O(t \cdot n)$ time. For each snapshot, at most k edges need to be checked for each vertex. Therefore, as there are at most k time steps at which each edge is active, the total complexity of this algorithm is $O(n \cdot k \cdot t)$. ◀

Finally, we provide a lower bound on the temporal harmonious chromatic number of the class of (k, t) -temporal matchings of $\sqrt{8(\min(k, t) - 1)}$. In doing so, we provide a clear gap between the upper and lower bounds and leave open the question of the optimal bound for $(2, 2)$ -temporal matchings and $(2, 3)$ temporal matchings. Further, we provide a lower bound of \sqrt{t} on the temporal harmonious chromatic number for any (k, t) -temporal matching.

► **Lemma 14.** *For any $k, t \in \mathbb{N}$, there exists a (k, t) -temporal matching that has a temporal harmonious chromatic number of $\sqrt{4 \min(k, t) - 2}$.*

Proof. This lemma is proven by constructing a $(\min(k, t), \min(k, t))$ -temporal matching $G = (V, E_1, E_2, \dots, E_T)$ with a temporal harmonious chromatic number of $\sqrt{4 \min(k, t) - 2}$. We assume without loss of generality that $k = t$. Note that any $(\min(k, t), \min(k, t))$ -temporal matching is also a (k, t) -temporal matching. The set V is constructed by forming two sets \mathcal{T} and \mathcal{K} . The set \mathcal{T} contains t vertices labelled v_1, v_2, \dots, v_t . The set \mathcal{K} contains k vertices

labelled u_1, u_2, \dots, u_t . For every $i \in [t - 1]$, an edge is constructed between v_i and v_{i+1} . Similarly, for every $j \in [k - 1]$, an edge is constructed between u_j and u_{j+1} . Finally, an edge is constructed between v_t and u_1 .

The first k snapshots are constructed by having every edge of the form (v_i, v_{i+1}) active, as well as exactly one edge of the form (u_j, u_{j+1}) . Formally, for $l \in [k]$, the time step l contains has the active edges $\{(v_1, v_2), (v_2, v_3), \dots, (v_{t-1}, v_t), (u_l, u_{l+1})\}$. Note that each such step contains exactly t members. The final snapshot contains every edge between members of \mathcal{K} and the edge (v_t, u_1) .

To determine the chromatic number of G , observe that by construction, every edge in G requires a different pair of colours. As there are $k + t - 1 = 2k - 1$ edges in G , the temporal harmonious chromatic number of G , χ must be satisfy $2k - 1 \leq \frac{\chi(\chi-1)}{2} \leq \frac{\chi^2}{2}$. Hence $4k - 2 \leq \chi^2$ and by extension $\chi \geq \sqrt{4k - 2}$. In the general case, when $k \neq t$, this can be rewritten as $\sqrt{4 \min(k, t) - 2}$ ◀

► **Lemma 15.** *For any (k, t) -temporal matching G that is not also a $(k - 1, t)$ -temporal matching, the temporal harmonious chromatic number of G is at least $\sqrt{2 \cdot k}$.*

Proof. Note that if G is not a $(k - 1, t)$ -temporal matching, then there must be at least one snapshot containing k edges. Let χ be the temporal harmonious chromatic number of G . Therefore, to colour every edge in this timestep with a unique pair of colours, χ must satisfy $\frac{\chi(\chi-1)}{2} \geq k$. Hence $\chi^2 \geq 2k$ and by extension $\chi \geq \sqrt{2k}$. ◀

References

- 1 Katerina Asdre, Kyriaki Ioannidou, and Stavros D. Nikolopoulos. The harmonious coloring problem is np-complete for interval and permutation graphs. *Discrete Applied Mathematics*, 155(17):2377–2382, 2007. doi:10.1016/j.dam.2007.07.005.
- 2 Katerina Asdre and Stavros D. Nikolopoulos. NP-completeness results for some problems on subclasses of bipartite and chordal graphs. *Theoretical Computer Science*, 381(1):248–259, 2007. doi:10.1016/j.tcs.2007.05.012.
- 3 Argyrios Deligkas and Igor Potapov. Optimizing reachability sets in temporal graphs by delaying. *Inf. Comput.*, 285(Part):104890, 2022. doi:10.1016/j.ic.2022.104890.
- 4 Keith Edwards and Colin McDiarmid. The complexity of harmonious colouring for trees. *Discrete Applied Mathematics*, 58(2-3):133–144, 1995.
- 5 Jessica Enright, Kitty Meeks, George B Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021.
- 6 Thomas Erlebach, Michael Hoffmann, and Michael Kammer. On temporal graph exploration. *Journal of Computer and System Sciences*, 119:1–18, 2021.
- 7 Thomas Erlebach and Jakob T. Spooner. Exploration of k -edge-deficient temporal graphs. In Anna Lubiw and Mohammad Salavatipour, editors, *Algorithms and Data Structures*, pages 371–384, Cham, 2021. Springer International Publishing.
- 8 Subhankar Ghosal and Sasthi C. Ghosh. Channel assignment in mobile networks based on geometric prediction and random coloring. In *Proceedings of the 2015 IEEE 40th Conference on Local Computer Networks (LCN 2015)*, LCN '15, pages 237–240, USA, 2015. IEEE Computer Society. doi:10.1109/LCN.2015.7366315.
- 9 Daniel Leven and Zvi Galil. Np completeness of finding the chromatic index of regular graphs. *Journal of Algorithms*, 4(1):35–44, 1983. doi:10.1016/0196-6774(83)90032-9.
- 10 Andrea Marino and Ana Silva. Coloring temporal graphs. *J. Comput. Syst. Sci.*, 123(C):171–185, February 2022. doi:10.1016/j.jcss.2021.08.004.

- 11 Kitty Meeks. Reducing reachability in temporal graphs: Towards a more realistic model of real-world spreading processes. In *Conference on Computability in Europe*, pages 186–195. Springer, 2022.
- 12 George B. Mertzios, Hendrik Molter, and Viktor Zamaraev. Sliding window temporal graph coloring. *J. Comput. Syst. Sci.*, 120:97–115, 2021. doi:10.1016/j.jcss.2021.03.005.
- 13 Othon Michail and Paul G Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016.
- 14 Feng Yu, Amotz Bar-Noy, Prithwish Basu, and Ram Ramanathan. Algorithms for channel assignment in mobile wireless networks using temporal coloring. In Björn Landfeldt, Mónica Aguilar-Igartua, Ravi Prakash, and Cheng Li, editors, *16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM '13, Barcelona, Spain, November 3-8, 2013*, pages 49–58. ACM, 2013. doi:10.1145/2507924.2507965.
- 15 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 681–690, 2006.

Fault-Tolerant Distributed Directories

Judith Beestermöller ✉

ETH Zurich, Switzerland

Costas Busch ✉

Augusta University, GA, USA

Roger Wattenhofer ✉

ETH Zurich, Switzerland

Abstract

Many fundamental distributed computing problems require coordinated access to a shared resource. A distributed directory is an overlay data structure on an asynchronous graph G that helps to access a shared token t . The directory supports three basic operations: *publish*, to initialize the directory, *lookup*, to read the contents of the token, and *move*, to get exclusive update access to the token. There are known directory schemes that achieve message complexity within polylog factors of the optimal cost with respect to the number of nodes n and the diameter D of G . Motivated by fault-tolerant distributed computing implementations, we consider the impact of edge failures on distributed directories. We give a distributed directory overlay data structure that can tolerate edge failures without disrupting the directory operations. The directory can be repaired concurrently while it processes directory operations. We analyze the impact of the faults on the amortized cost of the three directory operations compared to the optimal cost. We show that f edges failures increase the amortized competitive ratio of the operations by at most factor f . We also analyze the message complexity to repair the overlay structure, in terms of the number of messages that are sent and the maximum distance a message traverses. For an edge failure, the repair mechanism uses messages of size $\mathcal{O}(\log n)$ that traverse distance at most D' , the graph diameter after the fault. To our knowledge, this is the first asymptotic analysis of a fault-tolerant distributed directory.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Shared memory algorithms; Software and its engineering → Software fault tolerance

Keywords and phrases distributed directory, sparse partition, fault tolerance, message complexity, path dilation

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.5

Related Version *Full Version:* <https://arxiv.org/abs/2207.09940>

Funding Costas Busch is supported by grant NSF-CNS-2131538

1 Introduction

Many distributed computing applications require finding and accessing a shared token, where the token represents some shared resource. At all times, only the token owner has exclusive access to the token which grants the owner the ability to modify the content that the token represents. Distributed directories enable other nodes to find the token to read its contents or to get exclusive access. Distributed directories have applications in shared memory and sensor networks. Distributed transactional memory systems use distributed directories to atomically access shared memory objects and execute transactions at the network nodes [10, 21]. Sensor networks use distributed directories to track moving objects [1, 23].

We study distributed directories that facilitate access to a shared token t on an asynchronous weighted graph $G = (V, E, w)$. The directory supports three operations: (i) *publish*, which initializes the directory and announces the initial owner; (ii) *lookup*, which allows a node to read the contents of the t ; (iii) *move*, which moves t to a new owner for exclusive access. These operations may be issued and processed concurrently by the nodes in G .



© Judith Beestermöller, Costas Busch, and Roger Wattenhofer;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 5; pp. 5:1–5:20

Leibniz International Proceedings in Informatics



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

Several directory schemes based on an overlay data structure on graph G have been developed, for which the amortized total distance traversed by the messages in a lookup or move operation is close to optimal, namely within some poly-log factor to the number of nodes n and the diameter of the graph [10, 19, 21, 22]. However, these directory protocols are not fault tolerant. If an edge failure occurs, they are not able to maintain a directory structure that can support publish, lookup, and move operations.

In reality, a directory is implemented on a distributed network, and it is typical to have unreliable networks with link failures between nodes. As processing nodes need to continue to operate correctly during or after the occurrence of failures, designing fault-tolerant distributed algorithms is important. We provide a directory protocol that tolerates edge failures with provable correctness and performance guarantees.

We consider the impact of $f \geq 1$ edge failures. We assume that the edge failures do not disconnect G , as otherwise, the token becomes unreachable in G making the directory unusable. Nevertheless, edge failures may happen at arbitrary moments and concurrently. Given the initial partition hierarchy, our protocol is fully distributed and handles the failures without disrupting concurrent directory operations. We analyze the message complexity of repairing the directory and provide performance bounds for the amortized cost of the operations related to the number of failed edges f .

1.1 Contributions

We present a distributed directory that can handle edge failures without disrupting concurrent directory operations. Our directory is inspired by the Spiral directory protocol [21]. Spiral uses a sparse cover decomposition hierarchy of G that allows clusters at the same level to overlap. Instead, we use a sparse partition hierarchy \mathcal{P} of G that does not allow clusters at the same level to overlap. As we will show, sparse partitions have improved asymptotic performance in the directory operations, and are affected less by edge failures. We consider two kinds of sparse partitions: *weak*, where the diameter of a cluster is with respect to all nodes in G , and *strong*, where the diameter of a cluster is calculated within the cluster. Weak partitions are available for more kinds of graphs than strong partitions [7, 11].

To evaluate the performance of the directory and the repair operations, we analyze their communication cost. For the basic directory operations (publish, move, lookup) that send messages sequentially, the communication cost is the sum of the distances traversed by the sequential messages for each operation. For the repair mechanisms, we often send several messages in parallel, here the communication cost consists of the total number of messages and the maximum distance that any one of these messages traverses.

Table 1 shows the communication costs of directory operations before/after f edge failures:

- **Publish:** A publish operation costs $\mathcal{O}(D \cdot \log n)$. After $f \geq 1$ failures the publish operation costs $\mathcal{O}(D' \cdot \log n)$, where D is the diameter of G before the edge failures and D' is the diameter after the edge failures. Note here that we do not compare with the optimal, as there is really no specific way that optimizes this step.
- **Lookup:** For lookup, the message cost of our algorithm is an $\mathcal{O}(\log^3 n)$ approximation of the optimal cost (compared to the shortest path to the token). This is a $\log n$ factor improvement over Spiral [21]. With f edge failures, the approximation becomes $\mathcal{O}(f \cdot \log^3 n)$ for weak partitions, and $\mathcal{O}(f \cdot \log^2 n + \log^3 n)$ for strong partitions.
- **Move:** For move, the amortized cost of a sequence of move operations is an $\mathcal{O}(\log D \cdot \log^2 n)$ approximation of the optimal. With f edge failures, the approximation factor becomes $\mathcal{O}(f \cdot \log D' \cdot \log^2 n)$ for weak and $\mathcal{O}((f + \log n) \log D' \cdot \log n)$ for strong partitions.

■ **Table 1** Cost of operations for general/special graphs and weak/strong diameter partitions; publish cost is absolute; lookup and move costs are approximation factors compared to the optimal cost; failures are $f \geq 1$; D' is the diameter of G after the f failures; special graphs include constant doubling dimension, constant pathwidth (weak and strong partitions) and also fixed minor-free, chordal (weak partitions only) for which sparse partition schemes with $\sigma, I \in \mathcal{O}(1)$ are known [7].

Graph	Partition	Partition Parameters	Failures	Publish	Lookup	Move
general	any	$(\mathcal{O}(\log n), \mathcal{O}(\log n))$	none	$\mathcal{O}(D \cdot \log n)$	$\mathcal{O}(\log^3 n)$	$\mathcal{O}(\log D \cdot \log^2 n)$
general	weak	$(\mathcal{O}(\log n), \mathcal{O}(\log n))$	f	$\mathcal{O}(D' \cdot \log n)$	$\mathcal{O}(f \cdot \log^3 n)$	$\mathcal{O}(f \cdot \log D' \cdot \log^2 n)$
general	strong	$(\mathcal{O}(\log n), \mathcal{O}(\log n))$	f	$\mathcal{O}(D' \cdot \log n)$	$\mathcal{O}(f \cdot \log^2 n + \log^3 n)$	$\mathcal{O}((f + \log n) \log D' \cdot \log n)$
special	any	$(\mathcal{O}(1), \mathcal{O}(1))$	none	$\mathcal{O}(D)$	$\mathcal{O}(1)$	$\mathcal{O}(\log D)$
special	any	$(\mathcal{O}(1), \mathcal{O}(1))$	f	$\mathcal{O}(D')$	$\mathcal{O}(f)$	$\mathcal{O}(f \cdot \log D')$

■ **Table 2** Cost of repair mechanism for general graphs and strong/weak partitions; σ is a sparse partition parameter, generally of order $\mathcal{O}(\log n)$; ρ is the locality parameter usually a constant; a cluster at level i has diameter at most $\sigma \rho^i$ independent of the number of failures; the hierarchy consists of $\log_\rho D$ levels; D denotes the diameter of G before the edge failure, D' denotes the diameter of G after the edge failure; n denotes the number of nodes, m denotes the number of edges.

Operation	Partition	Size of Message	Number of Messages	Maximum Distance Traversed by Individual Message
Initialize Shortest Path Tree Update	any	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(D)$
Update Shortest Path tree (per tree)	any	$\mathcal{O}(\log n)$	$\mathcal{O}(m)$	$\mathcal{O}(D')$
Splitting a cluster (per level i cluster)	strong	$\mathcal{O}(\log n)$	1	$\sigma \rho^i$
	weak	$\mathcal{O}(\log n)$	2	$\sigma \rho^i$
Informing nodes within cluster of split (per level i cluster)	strong	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\sigma \rho^i$
	weak	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$2\sigma \rho^i$
Informing neighborhood of leader change (per cluster)	any	$\mathcal{O}(\log n)$	$\mathcal{O}(n^2)$	ρ^i
Update Directory Path (per level)	any	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(D')$
Updating the Special Parent	strong	$\mathcal{O}(\log n)$	2	$\sigma \rho^i$
Information of level i cluster	weak	$\mathcal{O}(\log n)$	2	$2\sigma \rho^i$

For special kinds of graphs [7], we get better bounds which are $\mathcal{O}(1)$ (for f failures $\mathcal{O}(f)$) approximation for lookup, and $\mathcal{O}(\log D)$ (resp. $\mathcal{O}(f \cdot \log D')$) approximation for move, while the cost of the publish operation is simply $\mathcal{O}(D)$ (resp. $\mathcal{O}(D')$).

We also analyze the repair communication costs (see Table 2). To maintain the sparse partition, we store a spanning tree within each cluster of \mathcal{P} . If an edge fails in the spanning tree of a cluster X , we split X into two. This requires one message in a strong and two messages in a weak sparse partition of size $\mathcal{O}(\log n)$, traversing a distance of at most $\sigma \rho^i$ (the cluster diameter) in a strong partition and at most $2\sigma \rho^i$ in a weak partition, where σ and ρ are parameters defining the sparse partition hierarchy (cf. Subsection 1.2). There are additional steps, such as informing all nodes within X of the split, requiring $\mathcal{O}(n)$ messages of size $\mathcal{O}(\log n)$ traversing similar distance. More details are given below.

1.2 Techniques

Each level of the hierarchy \mathcal{P} is a partition of $V(G)$ into clusters obtained from a (σ, I) -sparse partition scheme. At level i , each cluster has a diameter of at most $\sigma \rho^i$ (where ρ is a constant), and the ρ^i -neighborhood of a node intersects with at most I of these. The highest level of \mathcal{P} consists of a single cluster, while on the lowest level, each node of G forms its own cluster. We pick a leader in each cluster X of \mathcal{P} . The leader of the highest level is called the “root”.

The distributed directory maintains a directory path ϕ from the root to the current owner of the token t (see Figure 2). On each level of \mathcal{P} , exactly one leader node belongs to ϕ and has pointers to the directory path nodes at the level above and below, forming

a double linked list. All operations are executed through message passing. The directory path is initialized by the first owner of t through a publish operation. A lookup or move operation, issued by a node v , searches for ϕ by checking the level i leaders of the nodes in the ρ^i -neighborhood of v , for all increasing levels i . When the operation discovers the directory path, it follows ϕ toward the token t . A move operation changes the directory path toward the new owner while it searches for ϕ .

To search for the directory path, node v needs to know the leaders of the nodes in its ρ^i -neighborhood for $0 \leq i \leq h$. To avoid double computation, every node pre-computes this information. Edge failures can increase the distance between some nodes, thereby affecting the precomputed ρ^i -neighborhoods. To update the preprocessed information, every node v maintains a shortest path tree $T(v)$. When an edge on $T(v)$ fails we use King's fully dynamic algorithm for maintaining shortest path trees [13] to update it. To initialize the update of the shortest path trees the endpoints of the failed edge inform the nodes whose shortest path tree are affected. This requires at most $\mathcal{O}(n)$ messages of size $\mathcal{O}(\log n)$ that traverse a distance at most $\mathcal{O}(D)$, where D is the diameter of G before the edge failure.

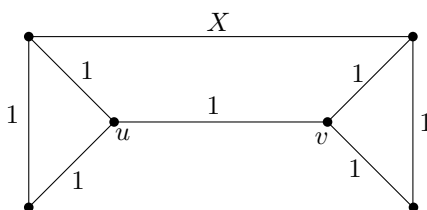
To improve the performance of lookup operations, a leader node $l(X)$ added to ϕ at level i informs its level i' leader $l_{i'}(l(X))$ for $i' = i + \log_{\rho}(c'\sigma)$ for an appropriately chosen constant c' . We call $l_{i'}(l(X))$ the special parent of $l(X)$. When a lookup operation finds the special parent of a node on the directory path, it traverses the directory path from there.

Upon an edge failure, both endpoints detect the failure immediately. In response to the failure, we update \mathcal{P} to maintain the directory's performance. For each cluster X , we store a spanning tree $T(X)$. When an edge e on $T(X)$ fails, we split X into X_1 and X_2 . X_1 has the same leader as X , and X_2 has a node incident to e as its leader. (If in a weak diameter partition e is outside X , $l(X_2)$ is selected appropriately in X_2 .) This mechanism ensures that the diameter of any level i cluster of \mathcal{P} is at most $2\sigma\rho^i$ regardless of the number of failures. The sparse partition scheme ensures that in a strong partition, at most one cluster splits per level of \mathcal{P} , and in a weak sparse partition, at most I clusters split. All these processes are initialized by the two endpoints of the failed edge.

When a cluster X with a leader on the directory path splits, we update the directory path to include the leader of the node that added $l(X)$ to ϕ . This ensures that lookup and move operations find ϕ at a level proportional to the distance between the token owner and the node that issued the operation. To update the directory path, the leader nodes of $l(X_1)$ and $l(X_2)$ need to communicate with each other and with the leader nodes on the directory path at the level below and above X . The whole process requires $\mathcal{O}(1)$ messages of size $\mathcal{O}(\log n)$, traversing a distance of at most $\mathcal{O}(D')$, where D' is the diameter of G after the edge failure.

We update the special parent information and notify nodes in the ρ^i -neighborhood of nodes in X_2 about the leader change as we update \mathcal{P} and ϕ . To update special parents for a level i cluster X , two messages of size $\mathcal{O}(\log n)$ are required, traversing a distance of at most $\sigma\rho^{i'}$ in a strong partition and $2\sigma\rho^{i'}$ in a weak partition, where i' is the level of the special parents. To notify the ρ^i -neighborhood of nodes in X_2 , $\mathcal{O}(n)$ messages of size $\mathcal{O}(\log n)$ are required, traversing a distance of ρ^i .

In unweighted graphs, a failure can at most double the diameter of G [3]. However, as illustrated in Figure 1, the diameter increase in a weighted graph may not be bounded. To accommodate such changes, we ensure that the number of layers in \mathcal{P} equals $\log_{\rho} D'$, where D' represents the current graph diameter. When adding layers to the partition hierarchy, we extend the directory path accordingly.



■ **Figure 1** An example of a graph showing that we cannot bound the stretch in the diameter of the graph. Assuming $X > 1$ the initial graph has diameter 3. If edge $\{u, v\}$ fails, the diameter becomes $2 + X$. Without further assumptions on X this cannot be bounded as a constant multiple of 3.

1.3 Related Work

An alternative way to implement a distributed directory is to use a spanning tree T on G . The edges of T are directed toward the owner node of the token. If node u requests the token, then the move request redirects the edges of the tree toward u (edge reversal). The benefit of the tree is that it can easily handle distributed requests since concurrent move operations are ordered when they intersect on the tree. Several protocols have been proposed based on trees: Arrow [4, 9, 14, 20], Relay [24], Ivy [15], Arvy [12]. The approximation factor of the operations is $\mathcal{O}(\log D_T)$, with respect to the diameter D_T of T . However, by using a tree the performance of the lookup and move operations may be sub-optimal with respect to G , as T may not accurately represent the distances in G . Considering the distance stretch s of the tree the approximation becomes $\mathcal{O}(s \log D_T)$, and s can be as large as the graph G diameter D . Nevertheless, considering an appropriate overlay tree that preserves on average the pairwise node distances of G [6], it is possible to get close to optimal performance on the average case for a set of random source operation requests [8, 18]. Our approach, on the other hand, has guaranteed performance for arbitrary sources of requests (not just random).

Another work [5] considers fault-tolerant routing and labeling schemes. These rely on knowing the destinations of messages. In our case, the destinations are leaders which may not be immediately known after the failures. Hence, we cannot rely on such routing schemes directly to implement the fault-tolerant directory. Another line of research related to edge failures maintains fault-tolerant sparse spanners of G that preserve the stretch (usually poly-log) of the distances in G even after edge or node failures [2, 17].

Outline of the Paper

In Section 2, we give some necessary definitions and define our model. Section 3 presents the basic directory scheme without failures and Section 4 describes our failure response mechanisms and analyzes their costs. A performance analysis of the directory after f failures is given in Section 5. In Section 6, we describe the integration of these mechanisms into the protocol. We conclude in Section 7. Omitted proofs and the pseudocode appear in the appendix. (The cases of concurrent edge failures and handling transient failures are discussed in the full version of the paper.)

2 Definitions and Preliminaries

Let $d_G(u, v)$ denote the length of a shortest path between u and v in G . The r -neighborhood of a node u , denoted $N_{G,u}(r)$, is the set of nodes that are within distance r to u . The diameter of a graph is $\text{diam}(G) = \max_{u,v \in V(G)} d_G(u, v)$. For a set $X \subseteq V$, let $G[X]$ denote

the subgraph of G induced by X . There are two ways to measure the diameter of X : (i) *weak diameter*, $diam_G(X)$, which considers all possible shortest paths in G that may also use nodes outside X ; (ii) *strong diameter*, $diam_{G[X]}(X)$, which considers only paths in X .

A partition of G is a collection of disjoint sets of nodes whose union is V . A *sparse partition* is a partition that restricts both the diameter of each cluster and the number of clusters within a specific distance. There are weak and strong sparse partitions that differ in whether the weak or strong diameter of a cluster is restricted.

A (r, σ, I) -*weak (strong) sparse partition* of G satisfies two properties:

- (i) each cluster has weak (strong) diameter at most $r\sigma$, and
- (ii) the r -neighborhood of each node $u \in V$ intersects at most I clusters.

A (σ, I) -*weak (strong) sparse partition scheme* is a procedure that gives a (r, σ, I) -weak (strong) partition for any $r > 0$. Jia *et al.* [11] give a $(\mathcal{O}(\log n), \mathcal{O}(\log n))$ -weak sparse partition scheme for an arbitrary metric space and general graphs. Filtser [7] gives a $(\mathcal{O}(\log n), \mathcal{O}(\log n))$ -strong partition scheme for general graphs based on the clustering technique by Miller *et al.* [16]. There are $(\mathcal{O}(1), \mathcal{O}(1))$ -partition schemes for special network topologies such as for low doubling-dimension and fixed minor-free graphs [7, 11].

2.1 Model

We model the distributed network as a weighted graph $G = (V, E, w)$ with positive edge weights of at least one. The weight of an edge $e = \{u, v\}$ represents the cost of sending a message over edge e . The cost of an operation is the sum of the edge weights the request traverses. The goal of a distributed directory is to minimize the total communication cost for a request in the worst case. The edge weight represents solely the cost of sending a message but does not indicate the delay or latency of an edge. In particular, for the correctness of our protocol, no message synchronization is needed.

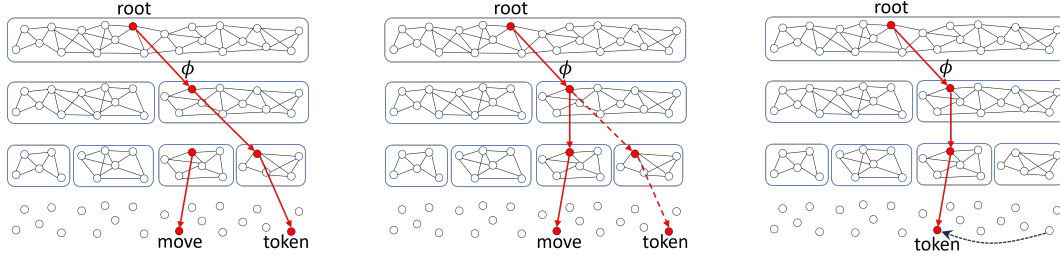
Each node u stores a shortest path tree $T(u)$ with $root(T(u)) = u$. The shortest path trees are built such that the shortest paths are consistent, meaning the path from u to v stored in $T(u)$ is identical (reversed) to the path from v to u stored in $T(v)$.

The communication in our network is asynchronous, and messages sent along the same edge are delivered in the order they are sent. All messages have the same size $\mathcal{O}(\log n)$ and are transmitted along shortest paths.

Our directory is built on a sparse partition hierarchy \mathcal{P} of $G = (V, E, w)$. Our directory works for strong and weak sparse partitions, but we show that they have different performances. We construct \mathcal{P} using any of the aforementioned partition schemes. We use $diam(X)$ to denote the strong or weak diameter of cluster X , depending on the partition type. The partition hierarchy \mathcal{P} comprises $h = \lceil \log_\rho D \rceil$ levels with an exponentially increasing locality parameter ρ at each level. At level i ($0 \leq i \leq h$), we define P_i as a (r_i, σ, I) -sparse partition of G , where $r_i = \min\{D, \rho^i\}$. We define level -1 where each node of V is a cluster by itself ($r_{-1} = 0$). At top level h , P_h comprises of a single cluster that covers the entirety of V .

We select a leader $l(X)$ in each cluster X of \mathcal{P} . At P_{-1} , each node is the leader of its own cluster, while the leader of the single cluster in P_h is called the *root*. Each node $u \in V$ belongs to exactly one cluster in each level of \mathcal{P} , denoted by $C_i(u)$, and its leader is $l_i(u) = l(C_i(u))$. Each node knows the leader of all the clusters to which it belongs.

To maintain the sparse partition in the presence of edge failures, we store a spanning tree $T(X)$ for each cluster X . In a weak sparse partition, $T(X)$ is the shortest path tree of $l(X)$, while in a strong sparse partition, $T(X)$ is a shortest path tree of $G[X]$ with root $l(X)$. The choice of spanning tree ensures that for any node u in X , the path on $T(X)$ connecting u and $l(X)$ is at most $diam(X)$. We denote the subtree of $T(X)$ rooted in u by $T_{\setminus u}(X)$. Every



■ **Figure 2** An example of a move operation. **Left:** a node issues a move request, which starts the bottom-up formation of a new directory path. **Middle:** the new directory path intersects the existing directory path ϕ ; the part of ϕ from the intersection until the old owner will be deleted. **Right:** the token has moved to the new owner and the directory path ϕ has been revised accordingly.

node u on $T(X)$ knows $T(X)$. In a weak sparse partition, u also knows which nodes belong to X . To store this information, a node in a weak sparse partition requires $\mathcal{O}(Ihn)$ memory, and a node in a strong sparse partition requires $\mathcal{O}(hn)$ memory.

3 Directory Scheme

The directory supports three operations: *publish* to build the initial directory path, *lookup* to read the current value of the token, and *move* to request ownership of the token and update the directory path. All three operations are executed through message passing. Nodes can issue lookup and move operations at any moment and simultaneously. Lookup operations simply get a copy of the token from the latest token owner. Concurrent move operations from different nodes are ordered through the directory because there can only be one token owner node at a time. Hence, the directory acts as a distributed queue for the move requests. The queue ordering is implicit by the way the concurrent move operations intersect each other in the directory data structure. The directory ensures that the previous token owner will know which node is the next token owner. In this way, the token is passed from its previous owner to the next in the queue. There is no requirement that the requests are served in a particular order, but every move request has to be served eventually. (The pseudocode of the directory is displayed in Algorithm 1 in Appendix C.)

The token resides at an owner node at the lowest level. There is a virtual *directory path* ϕ that points to the owner (see Figure 2). The path ϕ consists of $h + 2$ leader nodes, one node at every level of \mathcal{P} . Let ϕ_i denote the leader node of ϕ at level i , for $-1 \leq i \leq h$, where ϕ_{-1} is the token owner and ϕ_h is the root. For each level i , $0 \leq i \leq h - 1$, leader ϕ_i has pointers to ϕ_{i-1} and ϕ_{i+1} which form a virtual doubly linked list.

The directory path is created via the *publish* operation. The initial owner u sends a **publish**-message to its leader nodes at every level of \mathcal{P} , namely $\phi_i = l_i(u)$. This operation creates the pointers from ϕ_i to ϕ_{i-1} (both ways) for $0 \leq i \leq h$. Theorem 1 measures the cost of a *publish* operation and the length of the initial directory path. In the next result, $cost(publish)$ is the total distance that the **publish**-message traverses.

► **Theorem 1.** *The cost of the publish operation and the length of the initial directory is $\mathcal{O}(D)$.*

Proof. To build the directory path, node u contacts each of its leader nodes. Therefore, $cost(publish) \leq \sum_{i=0}^h \sigma \rho^i = \frac{\sigma \rho^{h+1} - 1}{\rho - 1}$. Since the nodes on the directory path u 's leaders, the distance between consecutive nodes is at most $d(\phi_i, \phi_{i+1}) = d(l_i(u), l_{i+1}(u)) \leq d(l_i(u), u) + d(u, l_{i+1}(u)) \leq \sigma(\rho^i + \rho^{i+1})$. Summing over the entire directory path gives $length(\phi) \leq \sum_{i=-1}^{h-1} \sigma(\rho_i + \rho_{i+1}) \leq \frac{\sigma(\rho+1)(\rho^{h+1}-1)}{(\rho-1)\rho} = \mathcal{O}(\rho\sigma^h)$. ◀

To locate the token, the issuer of a lookup or move operation first searches for the directory path (Figure 2). Once a leader node of the directory path is found, the token owner is reached by following ϕ . To search for the directory path, a node sends messages to leader nodes near itself at increasing levels of \mathcal{P} : Let $P_i(v)$ be the set of clusters in P_i that intersect $N_{G,v}(\rho^i)$, where $|P_i(v)| \leq I$. Node v checks whether any of the leaders in $P_i(v)$ equals ϕ_i for $0 \leq i \leq h$. The search stops at the lowest level where a directory path leader is found (at the root in the worst case). The next Lemma bounds the cost of searching level i for ϕ .

► **Lemma 2.** *The sum of all distances that messages travel during the search of the directory path at level i in a lookup or move operation issued by node u is $\mathcal{O}(\rho^i \sigma I)$.*

Proof. Node u contacts every leader in $P_i(u)$. By definition, $|P_i(u)| \leq I$. Further, if X is in $P_i(u)$, then there exists a node x in X with $d(u, x) \leq \rho^i$. Therefore, $d(u, l(X)) \leq d(u, x) + d(x, l(X)) \leq \rho^i + \sigma \rho^i$. Summing over all clusters in $P_i(u)$ gives the result. ◀

A move operation by node v modifies the directory path to denote the new ownership at v . The new directory path is formed in a bottom-up way while v searches for the existing directory path (Figure 2). Node v first adds $l_{-1}(v)$ to the directory path. Let ϕ_j , $j \geq 0$, be the first node of ϕ that v discovers. For levels $0 \leq i < j$, node v searches $P_i(v)$ and adds $l_i(v)$ to the directory path when it does not find ϕ_i . At level j node v finds ϕ_j , which will remain then in the directory path but its pointer changes to ϕ'_{j-1} . The move operation then follows the old directory path toward ϕ_{-1} . While going down the move operation deletes the leaders from the old directory path, that is, $\phi_{j-1} \cdots \phi_{-1}$ are removed from ϕ . Hence, the new directory path is $\phi_h \cdots \phi_j \phi'_{j-1} \cdots \phi'_{-1}$.

Concurrent move operations create multiple partial directory paths. However, only the latest directory path includes the root node. To ensure a unique complete directory path (without splits or gaps) from the root to the owner, the upward phase of a move is atomic. Contacting ϕ_i about the directory path triggers an immediate update of its downward pointer to ϕ'_{i-1} , directing subsequent operations to the new path. As sub-paths merge, the distance between consecutive nodes on the directory path can increase, as shown by the next lemma.

► **Lemma 3.** *The distance between two consecutive nodes ϕ_i and ϕ_{i+1} on the directory path for $-1 \leq i < h$ is at most $d(\phi_i, \phi_{i+1}) \leq \sigma(\rho^i + \rho^{i+1}) + \rho^{i+1}$.*

Proof. Consider two consecutive directory path nodes ϕ_i and ϕ_{i+1} . There were either added by the same node v , in which case $d(\phi_i, \phi_{i+1}) \leq d(\phi_i, v) + d(v, \phi_{i+1}) \leq \sigma(\rho^i + \rho^{i+1})$, or some node v added ϕ_i to ϕ and then found ϕ_{i+1} during its search of level $i+1$. In this case, there must be a node w in v 's ρ^{i+1} -neighborhood that belongs to ϕ_{i+1} 's cluster. Hence, $d(\phi_i, \phi_{i+1}) \leq d(\phi_i, v) + d(v, w) + d(w, \phi_{i+1}) \leq \sigma(\rho^i + \rho^{i+1}) + \rho^{i+1}$. ◀

The continuous modifications of the directory path imply that not all leaders on the directory path contain the current token owner w in their cluster. To ensure that a lookup operation issued by node u discovers the directory path at a level proportional to its distance to w , we use the concept of a *special parent* (as in Spiral [21]). Every leader node $l_i(x)$ that is added to ϕ informs leader node $l_{i'}(x)$, where $i' = i + \log_\rho(c'\sigma)$, for an appropriately constant c' . We call $l_{i'}(x)$ the special parent of $l_i(x)$ with respect to level i . When node u searches for the directory path, it asks the leader nodes if they are part of the directory path, or if they are the special parent of a node on the directory path. If it finds a special parent, it takes the link to the node on the directory path and continues the search from there. Like the directory path itself, special parent information is updated during a move operation. If a move removes a node from the directory path while a lookup follows a link from a special parent to that node, the lookup goes back to level i' and continues the search from there.

Using Lemmas 2 and 3, we obtain the following results for the lookup and move costs when there are no failures. (The proofs of these two results appear in the full version of the paper; here we focus on the fault analysis aspects.)

► **Lemma 4.** *A lookup operation finds the token with a cost that is a $O(\sigma^2 \rho I)$ factor from optimal.*

Consider a sequence of move requests $S = s_1, \dots, s_q$, that execute in a sequential manner, so that s_i starts only after s_{i-1} completes, where $i > 0$.

► **Lemma 5.** *The total cost of the move operations in S is a $O(h\rho\sigma(\sigma + I))$ factor from optimal.*

4 Responding to Edge Failures

In case of edge failures, our clustering may no longer satisfy the properties of a sparse partition, and some of the shortest path trees that nodes store may become disconnected. To guarantee the correctness and performance of our algorithm, we update our data structures accordingly. To accomplish this, we modify the operations described in Section 3 as follows:

1. Each node on ϕ remembers the node that added it to ϕ .
2. When w contacts $l(X)$ in a lookup or move operation to find the directory path, it includes a list of the nodes from w 's ρ^i -neighborhood that it believes are part of X .
3. Node w contacts a level i leader node $l(X)$ only if $d(w, l(X)) \leq \rho^i + 2\sigma\rho^i$.

Whenever an edge e fails, our update mechanisms recompute all shortest path trees that contained edge e , split every cluster whose spanning tree contained edge e , and update the directory path accordingly. We discuss each of them below.

4.1 Updating Shortest Path Trees

Each node in our protocol stores a shortest path tree, which we update when an edge $e = \{u, v\}$ on the tree fails. We use King's fully dynamic algorithm to maintain the shortest path trees in the presence of edge failure [13]. Updating a single tree takes $O(md)$ time, where m is the number of edges in $G \setminus \{e\}$, and d is the maximum distance of a node to the root of the tree. To use King's centralized algorithm in a distributed system, we let the root node compute the updated shortest path tree. This causes an additional cost because we need to inform the root node of the available edges. Namely, we need to inform the root of at most $O(m)$ edges with a maximum distance of D' from the root.

The updating of the shortest path trees is initialized by the endpoints of the failed edge $e = \{u, v\}$, which can detect the failure immediately. The consistency assumption we placed on the shortest path trees implies that u and v know if any tree needs to be updated.

► **Observation 6.** *If edge $e = \{u, v\}$ is part of the shortest path tree of a node w , then edge e is also part of the shortest path tree of u and v .*

To initialize the updates, nodes u and v send a broadcast along the remaining parts $T(u)$ and $T(v)$. The next lemma bounds the cost of this operation.

► **Lemma 7.** *To initialize the update of the shortest path tree, $O(n)$ messages of size $O(\log n)$ are required, that travel a maximum message distance of D , where D is the diameter of G before the failure of e .*

4.2 Reclustering

When an edge e fails, we split every cluster X whose spanning tree contains e into two. For the reclustering, we distinguish between the root level and clusters below the root level. For clusters below level h , we define the two new clusters as $X_1 = X \setminus (X \cap T_{\setminus v}(X))$ and $X_2 = X \cap T_{\setminus v}(X)$. The leader node of X_1 is the same as that of X , and for X_2 , either v becomes the new leader (if using strong sparse partition) or the closest node to v in X (if using weak sparse partition). The spanning tree for X_1 is $T(X) \setminus T_{\setminus v}(X)$ and for X_2 is $T_{\setminus v}(X)$ rerooted at $l(X_2)$. The following lemma bounds the diameter and the number of the generated clusters.

► **Lemma 8.** *Let X be a cluster at level i , $-1 \leq i < h$, and suppose at most f edges fail. Then X splits into at most $f + 1$ clusters. Each new cluster X_j , generated from X , has diameter at most $\text{diam}(X_j) \leq 2\sigma\rho^j$.*

Similar to the update of the shortest path tree, the splitting of the clusters is initiated when the endpoints of the failed edge detect the failure. Recall that u and v both know which clusters have e in their spanning tree. For every cluster X that needs to split the path on $T(X)$ from $l(X)$ to one of the endpoints of e , say u , the part between u and $l(X)$ remains unaffected by the failure of e . Therefore, u can inform $l(X)$ of the failure by sending a message along $T(X)$. If node v is not in X (i.e. weak diameter sparse partition), it chooses a node w closest to it on $T(X)$ from $X \cap T_{\setminus v}(X)$ to become the new leader node and sends a message to w to inform it of the reclustering and its new leadership role.

► **Lemma 9.** *Splitting a level i cluster requires one message in a strong and up to two messages in a weak sparse partition. These have size $\log n$ and traverse a distance of at most $\sigma\rho^i$.*

When $l(X_1)$ is informed about the failure and $l(X_2)$ knows whether it is part of the directory tree (see Section 4.3), both broadcast the update to all nodes in their respective cluster so that these can update their knowledge of $T(X)$ and the leader for the nodes in X_2 .

► **Lemma 10.** *To inform all nodes in X_1 and X_2 of the cluster change we need to send $\mathcal{O}(n)$ messages, each of which has size $\mathcal{O}(\log n)$. In a strong partition, the maximum distance a message traverses is $\sigma\rho^i$ and in a weak sparse partition, the maximum distance is $2\sigma\rho^i$.*

When the nodes in X_2 are informed of the cluster change, they forward this information to their ρ^i -neighborhood, so they can update their preprocessing information.

► **Lemma 11.** *To inform the ρ^i -neighborhood of the nodes in X_2 about the new leader requires $\mathcal{O}(n^2)$ messages of size $\mathcal{O}(\log n)$. The maximum distance traversed by any message is ρ^i .*

Initially, \mathcal{P} consists of $\log_\rho D$ layers, where $D = \text{diam}(G)$. To maintain this relationship between the number of layers and the diameter, we add layers to \mathcal{P} as the diameter increases.

Consider the single cluster X at level h . Before the edge failure, we have $d(r, u) \leq \sigma\rho^h$ on $T(X)$ for all nodes $u \in V$. Hence, if the failed edge e does not lie on $T(X)$, then the diameter of G is at most $2\sigma\rho^h$. In this case, we do not modify the root level.

If edge e lies on $T(X)$, then r 's updated shortest path tree tells us if we need to increase the number of layers. Let V_1 and V_2 be the partition of the vertices of G defined by the connected components of $T(X) \setminus \{e\}$, and assume w.l.o.g. that $r \in V_1$. By Lemma 8, $\text{diam}(G[V_1]) \leq 2\sigma\rho^h$ and $\text{diam}(G[V_2]) \leq 2\sigma\rho^h$. Let v be the node in X_2 closest to r before the edge failure. If after the edge failure, the distance from r to v is $d(x, r) > \sigma\rho^{h+1} - 2\sigma\rho^h$, then we increase the number of layers to h' , where $h' = \min_{z \in \mathbb{Z}} \{\sigma\rho^z > d(x, r)\}$ for all $x \in V$.

We split the level h the same way that we split lower level clusters whose spanning tree contains edge e . Levels $h + 1, \dots, h' - 1$ are simply copies of level h , and the directory path goes through the copy of the cluster that is part of the directory path at level h .

When the single cluster at level h splits, both leader nodes of the two generated clusters inform the nodes in their clusters about the additional layers. This information can be included in the usual message $l(X_1)$ and $l(X_2)$ sent to the nodes in their clusters when a failure occurs. This message will cause the directory path nodes at levels $h - (i + \log_\rho(c'\sigma)) + 1, \dots, h' - i + \log_\rho(c'\sigma)$ to send a message to their special parent, so that this information is also extended to the additional layers. Level h' is a single cluster that contains the entire graph. The leader node is r and the spanning tree is $T(r)$.

4.3 Updating the Directory Path and Special Parents

We need to ensure that the directory path and special parent information are maintained during cluster splitting. When a cluster X splits, we check if $l(X)$ is on the directory path. If it is not, then neither $l(X_1)$ nor $l(X_2)$ will be. If it is, then $l(X)$ can determine whether $l(X_1)$ or $l(X_2)$ becomes part of the directory path by checking if the node that added $l(X)$ to ϕ remains in X_1 . Once $l(X)$ knows about $l(X_2)$'s role, it informs v (using $T(l(X))$). If v is not $l(X_2)$, then it forwards this message to $l(X_2)$. If $l(X_2)$ is to become part of the directory path, then the message contains the ids of ϕ_{i-1} , ϕ_{i+1} , and the id of the node that added $l(X)$ to the directory path. With this, $l(X_2)$ sets its pointers to ϕ_{i+1} and ϕ_{i-1} and informs them to update their pointers too. When they receive $l(X_2)$'s message ϕ_{i-1} and ϕ_{i+1} send a message to $l(X)$ to remove its outdated directory path pointers. If $l(X_2)$ is not on the directory path, then $l(X)$'s messages simply informs $l(X_2)$ that it is not part of ϕ .

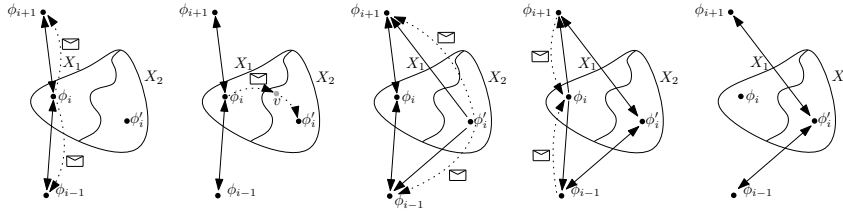


Figure 3 The steps of updating the directory path at level i : 1) Node ϕ_i sends a message to ϕ_{i-1} and ϕ_{i+1} to inform them about the update. 2) Node ϕ_i informs ϕ'_i to join the directory path. If we are using a weak sparse partition, node v acts as an intermediate in the message transfer. 3) ϕ'_i sets pointers to ϕ_{i-1} and ϕ_{i+1} and sends them a message so they update their pointers too. 4) ϕ_{i-1} and ϕ_{i+1} change their pointer to ϕ'_i and send a message to ϕ_i to remove its pointers. 5) ϕ_i removes its pointers to ϕ_{i+1} and ϕ_{i-1} .

For simplicity, we prevent consecutive nodes on the directory path to update concurrently (when the modification is due to an edge failure). Therefore, if $\phi_i = l(X)$ needs to be replaced by $l(X_2)$, then ϕ_i will first contact ϕ_{i-1} and ϕ_{i+1} to inform them of the update, before messaging $l(X_2)$. Neither of them will be able to initialize an update on their level until the update at level i is complete. In case two subsequent nodes attempt to initialize an update of the directory path simultaneously, then the node with the lower id will be allowed to update first. The process of updating the directory path is displayed in Figure 3. In the next lemma, we bound the cost of updating the directory path.

► **Lemma 12.** *To update the directory path we send $\mathcal{O}(1)$ messages of size $\mathcal{O}(\log n)$ which travel a distance at most $\mathcal{O}(D')$ where $D' = \text{diam}(G \setminus \{e\})$.*

At the same time that we modify the directory path, we also update the special parent information of $l(X_1)$ and $l(X_2)$. When $l(X_1)$ and $l(X_2)$ update ϕ , they also send a message to their special parent instructing them to update their pointers accordingly.

► **Lemma 13.** *To update the special parent information we send two messages of constant size that traverse a distance of at most $\sigma\rho^i$ in a strong sparse partition and at most $2\sigma\rho^{i'}$ in a weak sparse partition, where $i' = i + \log_\rho(c'\sigma)$.*

5 Analysis of Algorithm

We examine our protocol's performance with up to f faults. Edge failures may stretch the directory path, leading to delayed updates of special parent information. Consequently, operations that occur during or immediately after a failure may experience additional costs, referred to as *transient operations*. Once the directory path is rebuilt by a publish or move operation and the special parent information is updated, operations are considered *normal operations*. Here we analyze the cost of normal operations. (We discuss transient operations in the full version of the paper.)

We first bound the length of the directory path after failures.

► **Lemma 14.** *Suppose that since the last edge failure, the directory path has been rebuilt up to level i . Then the length of the directory path up to level i is at most $\mathcal{O}(\sigma\rho^i)$.*

Before we analyze the cost of lookup and move operations, we bound the number of clusters affected by an edge failure.

► **Lemma 15.** *A failure of edge $e = \{u, v\}$ splits at most h clusters in a strong, and Ih clusters in a weak sparse partition.*

In addition, if an edge failure occurs, we may add extra layers to the directory to accommodate the increased diameter. As explained in Section 4.2, the number of added layers is proportional to the diameter increase.

We now analyze the cost of lookup and move operations when up to f failures occur.

► **Theorem 16.** *Suppose we are using a strong sparse partition and that f edge failures have occurred. After updating our data structures, a lookup operation finds the token with cost that is $\mathcal{O}(\sigma^2(I + f)\rho)$ factor from optimal.*

Proof. Suppose node u issues a lookup request while the current owner is node v , where $u \neq v$, and $\rho^{i-1} \leq d_G(u, v) \leq \rho^i$. This implies the optimal cost is at least $d_G(u, v) \geq \rho^{i-1}$.

Let w be the directory path node at level i . By Lemma 14, the segment of ϕ from v to w is at most $c_1\sigma\rho^i$ for some constant c_1 . Hence, $d_G(v, w) \leq c_1\sigma\rho^i$. Therefore, $d_G(u, w) \leq d_G(u, v) + d_G(v, w) \leq c_2\sigma\rho^i$, for some constant $c_2 \geq 1 + c_1$.

Let s_w be the special parent of w , which is the leader of the cluster that includes w at level $i' = i + \log_\rho(c'\sigma)$, for a constant $c' \geq c_2$. Since $\rho^{i'} = c'\sigma\rho^i$, node s_w is in the $\rho^{i'}$ -neighborhood of u . Therefore, the lookup operation is guaranteed to discover s_w at level i' .

We sum the cost of the search up to level i' . On each level, node u contacts at most $I + f$ leaders by Lemma 15. When node u contacts a cluster leader node $l(X)$ at level i , then there must be a node x in X such that $d(u, x) \leq \rho^i$. Hence, the distance between u and $l(X)$ is at most $d(u, l(X)) \leq \rho^i + 2\sigma\rho^i \leq c\sigma\rho^i$ for some $c \geq 1 + 2\sigma$. Therefore,

$$\text{cost upward phase} \leq \sum_{j=0}^{i'} (I + f)c\sigma\rho^j = \mathcal{O}(\sigma^2(I + f)\rho^i).$$

The cost of the downward phase is given by Lemma 14. Thus, for strong sparse partitions, the total cost of a lookup is $\mathcal{O}(\sigma^2(I + f)\rho^i)$ which is $\mathcal{O}(\sigma^2(I + f)\rho)$ factor from optimal. ◀

► **Theorem 17.** *Suppose we are using a weak sparse partition and that f edge failures have occurred. After updating our data structures, a lookup operation finds the token with cost that is $\mathcal{O}(\sigma^2 f I \rho)$ factor from optimal.*

Proof. The proof is identical to Theorem 16, except that after f edge failures $P_i(u) \leq fI$, according to Lemma 15. Thus, the lookup operation visits up to fI clusters on each level. ◀

► **Theorem 18.** *Consider a sequence S of move requests $S = s_1, \dots, s_q$ that are all issued after the f^{th} edge failure and which are executed sequentially. The total cost of the move operations in S is a $\mathcal{O}(h' \sigma \rho((I + f) + \sigma))$ factor from optimal in a strong sparse partition (for sufficiently large S).*

Proof. Let $S_i = s_{i_1}, s_{i_2}, \dots, s_{i_z}$, $0 \leq i \leq h'$, be the sub-sequence of S that reach level i in their upward phase, where h' is the highest level of \mathcal{P} after the f failures. And let u_{i_j} be the issuer of s_{i_j} . Define s_{i_0} to be the last move operation prior to S that reached level i . If no such operation exists, s_{i_0} is the initial publish operation.

Operation s_{i_j} forms a new directory path p_{i_j} that links the leaders of u_{i_j} up to level $i - 1$. At level i , p_{i_j} links to ϕ_i , which is the level i leader of a node in u_{i_j} 's ρ^i -neighborhood.

We show that $d(u_{i_{j-1}}, u_{i_j}) > \rho^{i-1}$, for $j > 0$. Between $s_{i_{j-1}}$ and s_{i_j} no operation modified ϕ_{i-1} . Since s_{i_j} reaches level i , it does not discover $\phi_{i-1} = p_{i_{j-1}}$ at level $i - 1$. This implies that $u_{i_{j-1}}$ is not in the ρ^{i-1} -neighborhood of u_{i_j} .

Let $C^*(S_i)$ denote the optimal cost of the operations in S_i and C^* be the optimal cost of all operations. Since the distance between any two consecutive nodes in S_i is more than ρ^{i-1} , we have that $C^*(S_i) > |S_i| \rho^{i-1}$, which implies that

$$C^*(S) \geq \max_{0 \leq i \leq h'} C^*(S_i) \geq \frac{\sum_{i=0}^{h'} C^*(S_i)}{h' + 1} > \frac{\sum_{i=0}^{h'} |S_i| \rho^{i-1}}{h' + 1}. \quad (1)$$

The cost of searching for the directory path up to level i is, the same for a move and a lookup operation. Hence, $\text{cost}(S_i \text{ upward phase}) \leq |S_i| c \sigma^2 (I + f) \rho^i$ for some constant c .

For the downward phase, we need to be more careful because s_{i_1} could be a transient operation for $0 \leq i \leq h'$, that encounters two consecutive nodes ϕ_k and ϕ_{k-1} with distance up to $d(\phi_k, \phi_{k-1}) = D'$, where D' is the diameter of G after the f edge failures. However, for each sub-sequence S_i , where $0 \leq i \leq h'$, only s_{i_1} can be a transient operation as subsequent operations will traverse the updated directory path. For normal move operations, we can bound the downward phase by the length of the upward phase. Hence, we have

$$C(S) \leq h' D' + 2 \sum_{i=0}^{h'} |S_i| c \sigma^2 (I + f) \rho^i. \quad (2)$$

From Equations 1 and 2, we get the competitive ratio for the move operations in S . For a strong sparse partition we have

$$\frac{C(S)}{C^*(S)} \leq \frac{(h' + 1)(h' D' + 2 \sum_{i=1}^{h'} |S_i| c \sigma^2 (I + f) \rho^i)}{\sum_{i=0}^{h'} |S_i| \rho^{i-1}} = \mathcal{O}(h' \sigma \rho((I + f) + \sigma)),$$

where we assume the second term to be the dominating one, which holds for a sufficiently large set of move operations S (namely, $|S| = \Omega(h'^2 D')$). ◀

► **Theorem 19.** *The total cost of the move operations in S is a $\mathcal{O}(h' \sigma \rho(fI + \sigma))$ factor from optimal in a weak sparse partition (for sufficiently large S).*

Proof. The proof is identical to Theorem 18, except that to search a single layer for the directory path, node u needs to contact $|P_k(u_{i_j})| \leq fI$ clusters in a weak sparse partition. ◀

6 Adding Fault Tolerance to the Directory

In this Section, we explain how to integrate the fault-tolerance mechanisms into our directory. Here, we consider failures of one edge at a time. (Concurrent edge failures are discussed in the full version of the paper.) We first explain why we modified the directory operations as described at the beginning of Section 4: The first modification lets us determine if we need to update the directory path due to an edge failure. The second and third modifications ensure correctness and performance during the upward phase of a lookup or move operation.

When node w searches for the directory path its preprocessing information determines which nodes it contacts. If the leader of a node in w 's ρ^i -neighborhood changes due to a cluster split, then w might contact the wrong leader if it does not get informed of the update in time. By including a list of all nodes that w believes to be part of the cluster when contacting a leader about ϕ , the leader can inform w if a node is no longer part of the cluster. In this case, w knows that it needs to wait for a cluster update to contact all leaders.

The distance to a leader $l(X)$ of a node x in w 's ρ^i -neighborhood is at most $d(w, l(X)) \leq d(w, x) + d(x, l(X)) \leq \rho^i + 2\sigma\rho^i$ (by Lemma 8). If the distance between w and the node whom it believes to be x 's level i leader is larger, then a cluster split must have occurred. Therefore, instead of paying a too high cost, w waits for a cluster update info by node x .

We discuss the cases of edge failures on the shortest path tree and during move operations. (The case of publish and lookup operations are covered in Appendix B.)

6.1 Edge Failure on the Shortest Path Tree

When an edge on $T(w)$ fails, w updates $T(w)$ immediately upon being informed of the failure, regardless of whether w was in the middle of a directory operation. All directory operations rely on $T(w)$: publish uses it to contact the leaders of w at the lowest cost possible, move and lookup further use on it to determine whose leaders to contact at each level.

If w is notified of the failure while performing an operation, it stops the operation, updates $T(w)$, and then resumes the operation. For a publish operation, w simply continues. For lookup and move operations, node w takes into account the updated shortest path tree: Suppose the edge failure increases the distance from x to w from d to d' , where $\rho^{j-1} < d \leq \rho^j$ and $\rho^{k-1} < d' \leq \rho^k$, while w searches level i for the directory path. If $j > i$ and $k > j$, then the first time w contacts x 's leader node is at level k (unless w contacts x 's leader due to a different node in the cluster). If $j \leq i$ and $k > i$, then w does not contact $C_i(x)$, unless it already did so before being informed about the edge failure, or because there is a different node in w 's ρ^i -neighborhood that belongs to cluster $C_i(x)$. The first time it contacts x 's leader node is at level k . If $j \leq i$ and $k \leq i$, then w will contact cluster $C_i(x)$ during its level i search and at each subsequent level until it finds the directory path. The downward phase of a lookup or move operation is not affected by the edge failure on $T(w)$.

6.2 Edge Failure during Move Operation

While Searching for the Directory Path

The search phase of a move issued by w can only be affected by the edge failure if w needs to contact the leader of a cluster that splits due to the edge failure. Suppose that the level i leader of a node x in w 's ρ^i -neighborhood changes due to a split of a cluster X .

There are two cases to consider depending on whether w is informed about the change before or after contacting $l(X)$. If w is informed before, then there is no issue. Otherwise, two sub-cases arise. If $l(X)$ is already aware of the failure, it informs w that it is not x 's

leader, causing w to wait for the cluster update message from x . If not, $l(X)$ responds to w 's message as though x was still part of the cluster, and w does not need to contact x 's new level i cluster since the information received from $l(X)$ is valid for w 's new cluster.

The new directory path built during the search is unaffected by the failure of edge e .

While Following the Directory Path Downward

Assume the edge failure occurs while w 's move follows the directory path down. If the downward phase of the move operation does not encounter any clusters that split, an update at level i completes before the move reaches level i , or if the directory path remains unchanged, then the failure does not affect this phase.

Suppose that the split of cluster X results in a modification to the directory path at level i , and $l(X)$ realizes the need for the modification before the move operation reaches ϕ_{i+1} . Two cases arise: If $l(X)$ has already initialized the modification at level i , ϕ_i sends a message to ϕ_{i+1} to inform it of the update. In this case, ϕ_{i+1} does not forward the move message until the modification is complete. If cluster X is waiting for a modification of the directory path at a level above or below i , the move operation either halts before reaching level i if the modification occurs above i . Or, if the modification occurs below i , the move operation traverses the old pointers up to the modified level and removes them, preventing $l(X)$ from initializing a directory path modification.

7 Conclusions

We presented a fault-tolerant directory scheme based on sparse partitions that tolerates edge failures. We showed that the performance of the directory is linearly affected by the number of failures f . We showed how to adjust the clusters due to failures to transform the σ and I parameters, such that σ simply doubles while I is affected by either a f factor (weak diameter clusters), or f additive term (strong diameter clusters).

There are a few open questions that remain to be studied. One is to handle partitions of G due to failures. The connected component that contains the token can still function and respond to operation requests. A related problem is examining the impact of node failures. If G has bounded-degree d a node failure corresponds to at most d edge failures, then the techniques we developed could be adapted to analyze node failures.

Another line of research related to preserving distances is building fault-tolerant sparse spanners. A sparse spanner of G is a subgraph H such that the pairwise distances on G are stretched by a small factor on H . There exist fault-tolerant sparse spanners that maintain the stretch of the distances even after edge or node failures [2, 17]. Inspired by this, a future research direction is to design failure-oblivious sparse partitions with appropriate multiple pre-selected leaders in each cluster. Such leaders would be able to handle the failures without the need for cluster restructuring.

References

- 1 Baruch Awerbuch and David Peleg. Concurrent online tracking of mobile users. In *Proceedings of the Conference on Communications Architecture & Protocols, SIGCOMM '91*, pages 221–233, New York, NY, USA, 1991. Association for Computing Machinery. doi:10.1145/115992.116013.
- 2 Greg Bodwin, Michael Dinitz, Merav Parter, and Virginia Vassilevska Williams. Optimal vertex fault tolerant spanners (for fixed stretch). In Artur Czumaj, editor, *Proceedings of the Twenty-*

- Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1884–1900. SIAM, 2018. doi:10.1137/1.9781611975031.123.
- 3 FRK Chung and MR Garey. Diameter bounds for altered graphs. *Journal of graph theory*, 8(4):511–534, 1984.
 - 4 Michael J. Demmer and Maurice Herlihy. The arrow distributed directory protocol. In Shay Kutten, editor, *Distributed Computing, 12th International Symposium, DISC '98, Andros, Greece, September 24-26, 1998, Proceedings*, volume 1499 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 1998. doi:10.1007/BFb0056478.
 - 5 Michal Dory and Merav Parter. Fault-tolerant labeling and compact routing schemes. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 445–455. ACM, 2021. doi:10.1145/3465084.3467929.
 - 6 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004. doi:10.1016/j.jcss.2004.04.011.
 - 7 Arnold Filtser. Scattering and Sparse Partitions, and Their Applications. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:20, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.47.
 - 8 Abdolhamid Ghodselahe and Fabian Kuhn. Dynamic analysis of the arrow distributed directory protocol in general networks. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 22:1–22:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.DISC.2017.22.
 - 9 Maurice Herlihy, Fabian Kuhn, Srikanta Tirthapura, and Roger Wattenhofer. Dynamic analysis of the arrow distributed protocol. *Theory Comput. Syst.*, 39(6):875–901, 2006. doi:10.1007/s00224-006-1251-9.
 - 10 Maurice Herlihy and Ye Sun. Distributed transactional memory for metric-space networks. *Distributed Comput.*, 20(3):195–208, 2007. doi:10.1007/s00446-007-0037-x.
 - 11 Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for tsp, steiner tree, and set cover. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, STOC '05*, pages 386–395, New York, NY, USA, 2005. Association for Computing Machinery. doi:10.1145/1060590.1060649.
 - 12 Pankaj Khanchandani and Roger Wattenhofer. The arvy distributed directory protocol. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 225–235. ACM, 2019. doi:10.1145/3323165.3323181.
 - 13 Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 81–89. IEEE, 1999.
 - 14 Fabian Kuhn and Roger Wattenhofer. Dynamic analysis of the arrow distributed protocol. In Phillip B. Gibbons and Micah Adler, editors, *SPAA 2004: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 27-30, 2004, Barcelona, Spain*, pages 294–301. ACM, 2004. doi:10.1145/1007912.1007962.
 - 15 Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Trans. Comput. Syst.*, 7(4):321–359, 1989. doi:10.1145/75104.75105.
 - 16 Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures, SPAA '13*, pages 196–203, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2486159.2486180.

- 17 Merav Parter. Nearly optimal vertex fault-tolerant spanners in optimal time: sequential, distributed, and parallel. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1080–1092. ACM, 2022. doi:10.1145/3519935.3520047.
- 18 David Peleg and Eilon Reshef. A variant of the arrow distributed directory with low average complexity. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming, ICALP '99*, pages 615–624, Berlin, Heidelberg, 1999. Springer-Verlag.
- 19 Shishir Rai, Gokarna Sharma, Costas Busch, and Maurice Herlihy. Load balanced distributed directories. *Information and Computation*, 285(A), 2022. doi:10.1016/j.ic.2021.104700.
- 20 Kerry Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989. doi:10.1145/58564.59295.
- 21 Gokarna Sharma and Costas Busch. Distributed transactional memory for general networks. *Distributed computing*, 27(5):329–362, 2014.
- 22 Gokarna Sharma and Costas Busch. An analysis framework for distributed hierarchical directories. *Algorithmica*, 71(2):377–408, 2015. doi:10.1007/s00453-013-9803-2.
- 23 Gokarna Sharma, Hari Krishnan, Costas Busch, and Steven R. Brandt. Near-optimal location tracking using sensor networks. *International Journal of Networking and Computing*, 5(1):122–158, 2015. URL: <http://www.ijnc.org/index.php/ijnc/article/view/100>.
- 24 Bo Zhang and Binoy Ravindran. Dynamic analysis of the relay cache-coherence protocol for distributed transactional memory. In *24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010, Atlanta, Georgia, USA, 19-23 April 2010 - Conference Proceedings*, pages 1–11. IEEE, 2010. doi:10.1109/IPDPS.2010.5470393.

A Omitted Proofs

► **Observation 6.** *If edge $e = \{u, v\}$ is part of the shortest path tree of a node w , then edge e is also part of the shortest path tree of u and v .*

Proof. If this was not the case, then the shortest paths from w to u and v stored in $T(w)$ cannot be consistent with the shortest paths from v and u to w stored in $T(v)$ and $T(u)$. ◀

► **Lemma 7.** *To initialize the update of the shortest path tree, $\mathcal{O}(n)$ messages of size $\mathcal{O}(\log n)$ are required, that travel a maximum message distance of D , where D is the diameter of G before the failure of e .*

Proof. Broadcasting the id of the failed edge allows any node to detect if it needs to update its shortest path tree. The remainders of $T(u)$ and $T(v)$ contain at most n nodes and the maximum distance from any node to the root is at most D . ◀

► **Lemma 8.** *Let X be a cluster at level i , $-1 \leq i < h$, and suppose at most f edges fail. Then X splits into at most $f + 1$ clusters. Each new cluster X_j , generated from X , has diameter at most $\text{diam}(X_j) \leq 2\sigma\rho^j$.*

Proof. The connected components of $T(X) \setminus F$, where F is the set of failed edges, form the final clusters. Since $T(X) \cap F \subseteq F$, we remove at most f edges from $T(X)$, which means $T(X) \setminus F$ has at most $f + 1$ connected components.

Let X_j be a cluster generated through the splitting of the initial cluster X . By construction, the maximal distance of any node on $T(X)$ to $l(X)$ is at most $\sigma\rho^j$ (with respect to $T(X)$). Let u_j be the node in X_j that was closest to $l(X)$ on $T(X)$ and a and b be any two nodes in X_j . Then $d(a, b) \leq d(a, u_j) + d(u_j, b) \leq 2\sigma\rho^j$, because u_j must have been on the path from a , respectively b to $l(X)$ on $T(X)$. ◀

► **Lemma 9.** *Splitting a level i cluster requires one message in a strong and up to two messages in a weak sparse partition. These have size $\log n$ and traverse a distance of at most $\sigma\rho^i$.*

Proof. In any partition, node u sends a message to $l(X)$ to inform it about the failure. As $l(X)$ knows $T(X)$ it suffices to send u 's id. If in a weak sparse partition node v is not in X_2 , it selects a node w closest to it in X_2 to become $l(X_2)$. To inform w of its new leadership role, v sends a message with e 's id along $T(X)$ to w , so w can update its knowledge on $T(X)$. ◀

► **Lemma 10.** *To inform all nodes in X_1 and X_2 of the cluster change we need to send $\mathcal{O}(n)$ messages, each of which has size $\mathcal{O}(\log n)$. In a strong partition, the maximum distance a message traverses is $\sigma\rho^i$ and in a weak sparse partition, the maximum distance is $2\sigma\rho^i$.*

Proof. Node $l(X_1)$ broadcasts the id of u and $l(X_2)$ broadcasts its own id and the id of v . This suffices to inform each node of its leader and to update $T(X)$. As $|X_1 \cup X_2| = \mathcal{O}(n)$, we send at most $\mathcal{O}(n)$ messages. Our mechanisms ensure that in a strong sparse partition, a node's distance to its leader is at most the distance it had to $l(X)$ before the failure, which is $\sigma\rho^i$. In a weak sparse partition, the new diameter is at most $2\sigma\rho^i$, according to Lemma 8. ◀

► **Lemma 11.** *To inform the ρ^i -neighborhood of the nodes in X_2 about the new leader requires $\mathcal{O}(n^2)$ messages of size $\mathcal{O}(\log n)$. The maximum distance traversed by any message is ρ^i .*

Proof. In our algorithm, each node in X_2 sends the id of the new leader to every node in its ρ^i -neighborhood using its shortest path tree. In the worst case, $|X_2| = \mathcal{O}(n)$, and the ρ^i -neighborhood of every node in X_2 has size $\mathcal{O}(n)$. ◀

► **Lemma 12.** *To update the directory path we send $\mathcal{O}(1)$ messages of size $\mathcal{O}(\log n)$ which travel a distance at most $\mathcal{O}(D')$ where $D' = \text{diam}(G \setminus \{e\})$.*

Proof. One message is sent from $l(X_1)$ to v and forwarded to $l(X_2)$ to inform $l(X_2)$ whether it is part of the directory path. The distance from $l(X)$ to v is at most D' and the distance between v and $l(X_2)$ can be bounded by the diameter of X_2 , that is $d(l(X_2), v) \leq 2\sigma\rho^i$.

When $l(X_2)$ is part of the directory path, then $l(X)$ contacts ϕ_{i+1} and ϕ_{i-1} about the upcoming update. When $l(X_2)$ receives $l(X_1)$'s message, it also sends a message to ϕ_{i+1} and ϕ_{i-1} . When they receive this message, they again a message to $l(X)$. None of these messages need to travel further than D' , because all messages are sent along shortest path trees. ◀

► **Lemma 13.** *To update the special parent information we send two messages of constant size that traverse a distance of at most $\sigma\rho^i$ in a strong sparse partition and at most $2\sigma\rho^{i'}$ in a weak sparse partition, where $i' = i + \log_\rho(c'\sigma)$.*

Proof. Nodes $l(X_1)$ and $l(X_2)$ both send a message to their respective special parent. Both $l(X)$ and $l_{i'}(l(X))$ are in $C_{i'}(l(X))$ and both $l(X_2)$ and $l_{i'}(l(X_2))$ are in $C_{i'}(l(X_2))$, thus these messages can be sent along the spanning trees of $C_{i'}(l(X))$ and $C_{i'}(l(X_2))$. In a strong sparse partition, the initial spanning tree of the clusters guarantee that the distance from any node to the leader along the spanning tree is at most $\sigma\rho^{i'}$. This property is maintained even if clusters split. In a weak sparse partition, Lemma 8 tells us that the diameter of the spanning tree of any cluster is at most $2\sigma\rho^i$. ◀

► **Lemma 14.** *Suppose that since the last edge failure, the directory path has been rebuilt up to level i . Then the length of the directory path up to level i is at most $\mathcal{O}(\sigma\rho^i)$.*

Proof. Suppose the last time ϕ was modified at level i was by node u and the last time ϕ was modified at level $i + 1$ was by node w . Then the modification at level i must have been due to a move operation issued by u which found the directory path level $i + 1$. We thus know that $d(\phi_i, \phi_{i+1}) \leq d(\phi_i, u) + d(u, w) + d(w, \phi_{i+1})$. Because u found ϕ at level $i + 1$, we have $d(u, w) \leq \rho^{i+1}$. After the repair of the data structure, the diameter of a cluster X at level i is at most $\text{diam}(X) \leq 2\sigma\rho^i$. Thus, we obtain

$$\text{length}(\phi_1, \dots, \phi_i) = \sum_{j=-1}^{i-1} d(\phi_j, \phi_{j+1}) \leq \sum_{j=-1}^{i-1} 2\sigma(\rho^j + \rho^{j+1}) + \rho^{i+1} = \mathcal{O}(\sigma\rho^i). \quad \blacktriangleleft$$

► **Lemma 15.** *A failure of edge $e = \{u, v\}$ splits at most h clusters in a strong, and Ih clusters in a weak sparse partition.*

Proof. A cluster X splits if $T(X)$ contains e . In a strong sparse partition $T(X)$ contains only nodes from X . As a node belongs to only one cluster, this implies that in a strong sparse partition at most cluster per level splits. In weak sparse partition, $T(X)$ may contain nodes not in X , but a node's ρ^i -neighborhood intersects at most I clusters of \mathcal{P}_i . Since a cluster whose spanning tree contains e also contains u and v , at most I clusters on a level need to be reclustered due to e failing. ◀

B Adding Fault Tolerance to the Directory (Cont.)

B.1 Edge Failure during Publish Operation

Suppose w issues a publish operation and an edge failure occurs before the publish operation reaches level h . The publish operation is only affected if at some level i the leader of w changes. If w is informed of the change before adding $l(X)$ to the directory path, then w will add its new level i leader to the directory path. If the directory path has already been built to level i , then our failure mechanisms will update the directory path.

B.2 Edge Failure during Lookup Operation

While Searching for the Directory Path

The search for the directory path of the lookup operation is similar to that of the move operation, but the lookup operation also uses the information provided by special parents and it does not modify the directory path. We thus only discuss the impact of an edge failure on the special parent information: When w 's lookup finds a special parent $l(X)$ of a node $l(X')$ on the directory path the lookup follows the link from $l(X)$ to $l(X')$. If cluster X' splits and the directory path updates before the lookup reaches $l(X')$, then w 's lookup will go back to X and continue its search for the directory path there.

While Following the Directory Path Downward

The downward phase of a lookup is not affected by modifications to the directory path. Suppose that the directory path gets modified at level i due to the edge failure. If the directory path is updated before the lookup operation reaches level i , then the lookup follows the updated path. Otherwise, the lookup operation follows the pointers from ϕ_{i+1} to ϕ_i , from ϕ_i to ϕ_{i-1} as these are still intact.

C

 Pseudocode of Basic Directory Algorithm

■ **Algorithm 1** Directory Operations Issued by Node v .

Graph G has partition hierarchy \mathcal{P} with topmost level $h = \lceil \log_\rho D \rceil$, for constant $\rho > 1$;
 Directory path $\phi = \phi_{-1}, \phi_0, \dots, \phi_h$ points toward the current owner of token t ;

// Publish Operation

for level i from 0 to h **do**

$\phi_i \leftarrow l_i(v)$; // ϕ_i is set to be the leader of v at level i
 Add bidirectional links between ϕ_i and ϕ_{i-1} ;

// Lookup Operation

$i \leftarrow 0$; // start level of upward phase

while none of the leaders of clusters in $P_i(v)$ know about ϕ **do**

$i++$; // upward phase to discover ϕ

If a special parent pointer toward $\phi_{i'}$ ($i' < i$) was discovered at level i , then adjust $i \leftarrow i'$;

// downward phase toward token

for level $k \leftarrow i$ down to 0 **do**

Follow the downward pointer of ϕ_k ;

Return value of token t from owner node ϕ_{-1} ;

// Move Operation

$\phi_{-1} \leftarrow v$; // start forming new ϕ toward v

$i \leftarrow 0$; // upward phase discovers previous ϕ

while none of the leaders of clusters in $P_i(v)$ are ϕ_i **do**

$\phi_i \leftarrow l_i(v)$; // form new path ϕ
 Add bidirectional links between ϕ_i and ϕ_{i-1} ;
 Inform special parent $l_{i'}$ at level $i' = i + \log_\rho(c'\sigma)$ about ϕ_i ;
 $i++$;

$old \leftarrow$ level $i - 1$ node pointed downwards by ϕ_i ;

Add bidirectional links between ϕ_i and $l_{i-1}(v)$; // adjust topmost node

Delete upward link of old and information at special parent of old ;

// downward phase deletes old ϕ

while level of old is not -1 **do**



$w \leftarrow$ node pointed by downward link of old ;
 Delete links between w and old and information at special parent of w ;
 $old \leftarrow w$;

Transfer token t from old to v ; // v is new owner



Black Hole Search in Dynamic Tori

Adri Bhattacharya  

Indian Institute of Technology Guwahati, Assam, India

Giuseppe F. Italiano  

LuiSS University, Rome, Italy

Partha Sarathi Mandal  

Indian Institute of Technology Guwahati, Assam, India

Abstract

We investigate the black hole search problem using a set of mobile agents in a dynamic torus. A black hole is defined as a dangerous stationary node that has the capability to destroy any number of incoming agents without leaving any trace of its existence. A torus of size $n \times m$ ($3 \leq n \leq m$) is a collection of n row rings and m column rings, and the dynamicity is such that each ring is considered to be 1-interval connected, i.e., in other words at most one edge can be missing from each ring at any round. The parameters which define the efficiency of any black hole search algorithm are: the number of agents and the number of rounds (or *time*) for termination. We consider two initial configurations of mobile agents: first, the agents are co-located, second, the agents are scattered. In each case, we establish lower and upper bounds on the number of agents and on the amount of time required to solve the black hole search problem.

2012 ACM Subject Classification Theory of computation \rightarrow Distributed algorithms

Keywords and phrases Black Hole Search, Time Varying Graphs, Dynamic Torus, Distributed Algorithms, Mobile Agents

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.6

Related Version *Full Version*: <https://arxiv.org/abs/2402.04746> [3]

Funding *Adri Bhattacharya*: Supported by CSIR, Govt. of India, Grant Number: 09/731(0178)/2020-EMR-I.

Acknowledgements This work was done while Partha Sarathi Mandal was in the position of Visiting Professor at LuiSS University, Rome, Italy.

1 Introduction

Given a network and a set of mobile agents, the black hole search problem (also termed as BHS problem) consists of locating a malicious stationary node that has the power to eliminate any number of incoming agents without leaving any trace of its existence. This problem is not new, and it readily has many real-life implications. For example, the black hole may be a node infected with a virus in a computer network, and in order to make the network safe, the infected node should be located for further action. The first task for any set of mobile agents ought to be to locate the black hole. To accomplish this task, at least one agent needs to visit the node; we aim at an efficient BHS algorithm, where the minimum number of agents gets consumed by the black hole so that at least one agent must remain alive in order to locate the black hole within finite time. This problem has been extensively studied in networks which are static, see, e.g., [1, 6, 8, 9, 14, 15, 17, 18, 20]. Recently, research on black hole search problem has been mainly focused on dynamic networks; in particular, the most relevant dynamic networks studied are *time-varying graphs*. These networks work on temporal domains, which are mainly considered to be discrete time steps. More precisely, the network is a collection of static graphs, in which some edges may disappear or reappear



© Adri Bhattacharya, Giuseppe F. Italiano, and Partha Sarathi Mandal;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 6; pp. 6:1–6:16

Leibniz International Proceedings in Informatics



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

at each discrete time step, while the vertex set is fixed, with the additional constraint that at each time step the underlying graph remains connected (also termed as *1-interval connected*). Presently, apart from the black hole search on a dynamic ring [11] and on a dynamic cactus [2], nothing much is known about the black hole search problem on dynamic networks.

In this paper, we investigate this problem on a dynamic torus of size $n \times m$ (where each ring is 1-interval connected and without loss of generality $3 \leq n \leq m$), where a set of agents synchronously perform the same execution, with the goal of locating the black hole. Moreover, we consider that each node in the underlying torus has a whiteboard, used by the agents to write/read some information used by the agents while executing a certain black hole search algorithm.

We study two types of initial configurations of agents, under the assumption that each agent executing the black hole search algorithm, can communicate among themselves when they are at the same node, and they can also use the whiteboard present at each node of the underlying graph in order to communicate with other agents. In the first configuration, all agents are initially located at the same node; in the second configuration, the agents are scattered along the nodes of the underlying network. In both configurations, all the nodes where agents are initially located are not *dangerous*, i.e., they do not contain the black hole (they are *safe*). Our primary objective is to design an efficient BHS algorithm such that: (a) within a finite time, at least one agent survives, and (b) it gains knowledge of the black hole location.

2 Related Works and Our Contribution

2.1 Related work

Network exploration by mobile agents is one of the fundamental problems in this domain, and it was first introduced by Shannon [24]. After his pioneering work, this problem has been extensively studied in various topologies such as rings [23], trees [10], general graphs [7] under different models of communication (particularly, pebbles [12] and whiteboard [26]), synchrony (synchronous [7], semi-synchronous [4] and asynchronous [19]) and both in static [7] as well as dynamic networks (tori [22] and general graphs [21]).

The black hole search (BHS) problem is a special version of the exploration problem, where, in the worst case the underlying network needs to be explored in order to locate the black hole position. This problem was first introduced by Dobrev et al. [15], and after that has received a lot of attention: indeed, it has been studied for directed [8] as well as undirected graphs [6], and for different underlying networks, such as rings [6], tori [5], trees [9] and general graphs [15]. In addition, different communication models have been considered for this problem, including ‘Enhanced Token’ [13], “Pure Token” [20] and whiteboard [16]. Moreover, this problem has also been explored for different initial agent configurations. In particular, Shi et al. [25] showed that, when the agents are co-located, a minimum of 2 co-located agents communicating via tokens can solve the BHS problem in hypercube, torus and complete network with $\Theta(n)$ moves, whereas in the case where k agents ($k > 3$) are scattered, then with only 1 token per agent it is shown that BHS can be solved in $O(k^2 n^2)$ moves. All these above papers discuss black hole search in a static network, and very little is known about the problem in dynamic networks. Di Luna et al. [11] first investigated this problem in a dynamic ring, and they showed that when the agents are co-located, then in face-to-face communication with 3 agents there is an optimal algorithm that works in $\Theta(n^2)$ moves and $\Theta(n^2)$ rounds (where n is the size of the ring). Next, with whiteboard communication, they reduced the complexity to $\Theta(n^{1.5})$ rounds and $\Theta(n^{1.5})$ moves. Lastly,

when the agents are initially scattered and each node has a whiteboard, then again with 3 agents they showed that at least $\Theta(n^2)$ moves and $\Theta(n^2)$ rounds are required for any BHS algorithm. In each case, they gave an optimal algorithm. Next, Bhattacharya et al. [2] studied the BHS problem in a dynamic cactus graph, and proposed an agent optimal algorithm when at most one edge can be dynamic; in the case when at most k (> 1) edges can be dynamic, they proposed a lower bound of $k + 2$ and an upper bound of $2k + 3$ agents.

In this paper, we further investigate the BHS problem in a dynamic torus, with the aim of providing an efficient BHS algorithm. To the best of our knowledge, this is the first work where the BHS problem is explored in the case of a dynamic torus. Previously, Gotoh et al. [22] studied the exploration problem under link presence detection and no link presence detection in dynamic tori, whereas Chalopin et al. [5] studied the BHS problem in a static torus and gave tight bounds on the number of agents and tokens when the agents are initially scattered.

2.2 Our Contribution

We investigate the BHS problem in a dynamic torus for two initial configurations: first, when the set of agents are initially co-located, and next, when the agents can be initially scattered in different nodes. When the agents are initially co-located, we provide the following results.

- We establish the impossibility of correctly locating the black hole with $n + 1$ agents.
 - We show that with $n + c$ (where $c \geq 2$ and $c \in \mathbb{Z}^+$) co-located agents, any BHS algorithm requires at least $\Omega(m \log n)$ rounds.
 - With $n + 3$ agents we present a BHS algorithm that works in $O(nm^{1.5})$ rounds.
 - With $n + 4$ agents we present an improved BHS algorithm that works in $O(mn)$ rounds.
- The following results are obtained when the agents are initially scattered.
- We establish the impossibility of correctly locating the black hole with $n + 2$ agents.
 - We show that with $k = n + c$ (where $c \geq 3$ and $c \in \mathbb{Z}^+$) scattered agents, any BHS algorithm requires $\Omega(mn)$ rounds.
 - With $n + 6$ agents we present a BHS algorithm that works in $O(nm^{1.5})$ rounds.
 - With $n + 7$ agents we present a $O(mn)$ round optimal BHS algorithm.

The list of results are summarised in the following Table 1.

■ **Table 1** Summary of Results where LB, UB and IC represent lower bound, upper bound and initial configuration of the agents, respectively.

IC	Bound	# Agents	Rounds	Results
Colocated	LB	$n + 2$	$\Omega(m \log n)$	Cor 1 & Thm 3
	UB	$n + 3$	$O(nm^{1.5})$	Thm 7
	UB	$n + 4$	$O(nm)$	Thm 8
Scattered	LB	$n + 3$	$\Omega(nm)$	Cor 3 & Thm 5
	UB	$n + 6$	$O(nm^{1.5})$	Thm 9
	UB	$n + 7$	$O(nm)$	Thm 10

Organisation. The remainder of the paper is organised as follows. In Sections 3 and 4, we explain the model and prove the lower bound results. Next, in Section 5, we discuss some preliminary notation and basic subroutines which will be used by our algorithms. Further, in Sections 6 and 7, we present and analyse our algorithms for the co-located and scattered case. Finally, we list some concluding remarks in Section 8.

Due to the restrictions in the page limit, the pseudocodes of the algorithms, proofs of the theorems and lemmas are omitted and can be found in the full version of this paper [3], which also includes a detailed explanation of the states and predicates used in our algorithms.

3 Model and Problem Definition

Graph Model. The dynamic graph is modelled as an undirected time-varying graph (or formally known as *temporal graph*) $\mathcal{G} = (G, V, E, \mathbb{T}, \rho)$, where V is the set of vertices (or nodes), E is the set of edges in G , \mathbb{T} is defined to be the *temporal domain*, which is defined to be \mathbb{Z}^+ as in this model we consider discrete time steps, also $\rho : E \times \mathbb{T} \rightarrow \{0, 1\}$ is defined as the *presence* function, which indicates the presence of an edge at a given time. The graph $G = (V, E)$ is the underlying static graph of the dynamic graph \mathcal{G} , also termed as *footprint* of \mathcal{G} . More specifically, the footprint $G = (V, E)$ is a torus of size $n \times m$, where n represents the number of rows and m represents the number of columns, we define $V = \{v_{i,j} \mid 0 \leq i \leq n-1, 0 \leq j \leq m-1\}$ and E is the set of edges, where the horizontal and vertical edges are $\{(v_{i,j}, v_{i,j+1 \bmod m})\}$ and $\{(v_{i,j}, v_{i+1 \bmod n,j})\}$, respectively. By the node $v_{i,j}$ we invariably mean $v_{i \bmod n, j \bmod m}$ and these modulus functions are ignored further in this paper. In order to restrict self-loop or multiple edges without loss of generality, we assume $3 \leq n \leq m$. A row ring R_i (resp, a column ring C_j) is the subgraph of G induced by the set of vertices $\{v_{i,j} \mid 0 \leq j \leq m-1\}$ (resp, $\{v_{i,j} \mid 0 \leq i \leq n-1\}$). The adversary has the ability to make an edge reappear or disappear at any particular time step with the added constraint that, irrespective of how many edges disappear or reappear, each row and column ring at any time step must be connected; in other words, each row and column ring in \mathcal{G} is *1-interval connected* (so at any time, the adversary can make at most one edge disappear from each row and column ring, in order to maintain the 1-interval connectivity property). A disappeared edge is termed as a *missing edge* in this paper.

Every node $v_{i,j} \in G$ is labelled by a unique Id (i, j) , whereas each node in G has 4 ports adjacent to it, where the ports corresponding to the edges $(v_{i,j}, v_{i,j-1})$, $(v_{i,j}, v_{i,j+1})$, $(v_{i,j}, v_{i-1,j})$, $(v_{i,j}, v_{i+1,j})$, are denoted by *west*, *east*, *south*, *north*, respectively. In addition, corresponding to each port of a node $v_{i,j}$ of G a *whiteboard* of storage of $O(1)$ -bits is placed. The purpose of the whiteboard is to store and maintain certain information, such as the node Id or agent Id or the agent's course of traversal (depending on the amount of storage the whiteboard can store). Any incoming agent can read the existing information or write any new information corresponding to a port along which it travels to the next node. Fair mutual exclusion to all incoming agents restricts concurrent access to the whiteboard. In this paper, we consider our temporal graph \mathcal{G} to be an oriented dynamic torus, i.e., each row and column ring in \mathcal{G} has an orientation. In other words, the nodes of a row (or a column) ring are marked in an increasing order along a counter-clockwise direction. The network G has a malicious node or unsafe node also termed as a *black hole*, which vanishes any incoming agent without leaving any of its trace. The remaining nodes in G are not malicious, hence they are termed as *safe nodes*.

Agent Model. A set of k agents $A = \{a_1, a_2, \dots, a_k\}$ are assigned the task to locate the black hole in \mathcal{G} . We consider two initial configurations in this paper: first, the set of A agents are initially *co-located* at a safe node (the node in G at which they are co-located is termed as *home*), second, the agents are initially *scattered* along safe nodes in \mathcal{G} . Each agent in A has a distinct Id of size $\lceil \log k \rceil$ bits taken from the set $[1, k]$, and every agent has computational capabilities so that it can communicate with other agents when they are at

the same node at the same time. Each agent has knowledge of the torus size, i.e., both n and m are known to the agents, and also, each agent has an internal memory of $O(\log m)$ bits. An agent moves from one node to another using the edges at each round; furthermore, any number of agents can concurrently move along an edge at any round. These actions are atomic in nature, so an agent cannot recognise the other agents concurrently passing through the same edge at the same round, but it can see and communicate with all the other agents present at the current node at the same round. These agents operate in *synchronous* rounds, so in each round, every agent becomes active and takes a local snapshot of its current node. The snapshot includes the presence of the ports of its current node at the current round, the agent's local memory (which contains the amount of information gathered by the agent while communicating with other agents), the set of agents present at the current node, and the contents of the whiteboard. Now, based on this information, the agent performs the following actions:

- *Look*: In this step, the agent takes the *snapshot* of the current node. This snapshot helps the agent gather the information about the Ids of other agents residing at the same node, the edges that currently exist at the current round and also the whiteboard information at the current node.
- *Compute*: On the basis of its earlier snapshot and local memory, the agent decides to stay at the current node or move to another node. The direction of its movement is also calculated in this step.
- *Move*: In this step, if the agent decides to move along a specific direction and if the corresponding edge is present, then it moves along this edge while updating the whiteboard (if required, based on the algorithm) to the next node in the subsequent round.

Since, the agents operate in *synchronous* rounds, so each agent gets activated at each round and performs the LCM cycle. So, the time taken by any algorithm is calculated in terms of the *rounds*.

Configuration. A configuration C_r at a round r is defined to be the amalgamation of the presence of the number of agents at a node, the local memory of each agent and contents of the whiteboard at the start of round r . The transformation from C_{r-1} to C_r depends on multiple factors: first, the execution of the algorithm; second, the adversarial choices of edges disappeared and reappeared in round $r - 1$. C_0 is the initial configuration, where, in the co-located case, the initial safe node is chosen by the adversary, whereas in the scattered case, the adversary arbitrarily places the agents along the safe nodes.

► **Definition 1** (Black Hole Search). *Given a dynamic torus \mathcal{G} of size $n \times m$, an algorithm \mathcal{A} for a set of k agents solves the BHS problem if at least one agent survives and terminates. The terminating agent must know the exact position of the black hole in the footprint of \mathcal{G} .*

The measures of the complexity for the BHS problem are as follows: the number of agents or *size*, required to successfully execute \mathcal{A} , the *time* or the number of rounds required to execute \mathcal{A} . Note that in this paper, we have assumed the fact that whenever an agent correctly locates the black hole, the algorithm terminates, so all the other agents executing any action get terminated immediately.

4 Lower Bound Results

In this section, we present the lower bound results on the number of agents and number of rounds in both scenarios when the agents are either initially co-located or scattered.

4.1 Co-located Agents

The next theorem gives impossibility result on the number of agents when they are co-located.

► **Theorem 1.** *Given a dynamic torus \mathcal{G} of size $n \times m$, there does not exist a BHS algorithm which correctly locates the black hole with $k = n + 1$ co-located agents and each node in \mathcal{G} contains a whiteboard of $O(1)$ bits.*

► **Corollary 1.** *Any BHS algorithm on a dynamic torus \mathcal{G} of size $n \times m$ requires at least $k = n + 2$ co-located agents to correctly locate the black hole when each node in \mathcal{G} has a whiteboard of $O(1)$ bits.*

Next lemma gives a lower bound on the round complexity for any exploration algorithm on a dynamic ring, where at least 2 agents are initially co-located.

► **Lemma 1.** *In a dynamic ring of size $n > 3$ in presence of whiteboard, any exploration algorithm with l ($l \geq 2$) co-located agents require at least $\Omega(n)$ rounds to explore such a ring.*

► **Theorem 2** ([11]). *In a dynamic ring of size $n > 3$, any BHS algorithm with 3 co-located agents in presence of whiteboard requires $\Omega(n^{1.5})$ rounds, even if the agents have distinct Ids.*

The following corollary follows from Lemma 1 and Theorem 2.

► **Corollary 2.** *In a dynamic ring of size $n > 3$, any BHS algorithm with at least 4 co-located agents in presence of whiteboard requires $\Omega(n)$ rounds, even if the agents have distinct Ids.*

The next theorem gives a lower bound on the round complexity for any BHS algorithm operating on a dynamic torus with k co-located agents.

► **Theorem 3.** *Any BHS algorithm with $k = n + c$ co-located agents, where $c \in \mathbb{Z}^+$ and $c \geq 2$, on a $n \times m$ dynamic torus requires at least $\Omega(m \log n)$ rounds.*

Proof. Given a dynamic torus of size $n \times m$ (with $3 \leq n \leq m$) and $k = n + c$ agents are initially co-located at a safe node, observe by Corollary 2, l (where $l \geq 4$) agents can perform BHS in presence of whiteboard along a row ring of size m in at least $\Omega(m)$ rounds. Now, let us consider there exists an algorithm \mathcal{H} which is tasked to perform BHS along the dynamic torus \mathcal{G} , so concurrently exploring a set of rings by a set of l agents is always a better strategy rather than exploring a ring one at a time by a set of agents. Hence, we consider \mathcal{H} instructs a set of l agents to explore a set of rings concurrently. So, if t (where $t \leq \frac{k}{l}$) rings are concurrently explored by the set of k agents, then as each ring in \mathcal{G} is 1-interval connected, so the adversary has the ability to block an agent each in every t such rings. This means the remaining agents left to explore for the next concurrent exploration is at least $k - \frac{k}{l}$, where each of these concurrent exploration requires $\Omega(m)$ rounds and the number of rings till now explored is $\frac{k}{l}$. In the next concurrent exploration, at least $\frac{k - \frac{k}{l}}{l}$ row rings can be explored in $\Omega(m)$ rounds, which further blocks this many agents, and the remaining agents left to explore remaining graph is $k - \frac{k}{l} - \frac{k - \frac{k}{l}}{l}$, whereas the total number of row rings explored yet is $\frac{k}{l} + \frac{k - \frac{k}{l}}{l}$. Continuing this way, we can define a recursion relation on the remaining number of agents, $T(\alpha) = T(\alpha - 1)(1 - \frac{1}{l})$ and $T(1) = k - \frac{k}{l}$, where $T(\alpha)$ resembles that at the α -th iteration this many agents are left to explore the remaining part of \mathcal{G} and each such concurrent exploration for black hole requires $\Omega(m)$ rounds. So, for α many iterations \mathcal{H} requires $\alpha\Omega(m) = \Omega(\alpha m)$ rounds. Now, we try to approximate the value of α . Observe, when $T(\alpha) \leq 7$, then either the whole torus is explored in the worst case for the black hole or there is no further concurrency possible because in order to concurrently explore at least two

rings in $\Omega(m)$ rounds, a minimum of 8 agents (as 4 agents are at least required to explore a ring in $\Omega(m)$ rounds) are required to be left available, so if at most 7 agents are remaining that means no concurrency is possible for any BHS algorithm. Hence, for $T(\alpha) \leq 7$, we approximate the value of α .

$$\begin{aligned} T(\alpha) \leq 7 &\implies T(\alpha - 1) \left(1 - \frac{1}{l}\right) \leq 7 \implies T(\alpha - 1) \leq \frac{7l}{l-1} \\ &\implies T(\alpha - 2) \left(1 - \frac{1}{l}\right) \leq \frac{7l}{l-1} \leq 7 \left(\frac{l}{l-1}\right)^2 \dots \implies T(1) \leq 7 \left(\frac{l}{l-1}\right)^{\alpha-1} \\ &\implies k \left(1 - \frac{1}{l}\right) \leq 7 \left(\frac{l}{l-1}\right)^{\alpha-1} \implies k \leq 7 \left(\frac{l}{l-1}\right)^\alpha \implies \frac{\log k - \log 7}{\log \left(\frac{l}{l-1}\right)} \leq \alpha \end{aligned}$$

This implies $\alpha \approx \log n$, as $k = n + c$ and $l \geq 4$. Hence, this means that for any algorithm \mathcal{H} , in order to either explore the whole dynamic torus for a black hole or to stop concurrent exploration, at least $\alpha \approx \log n$ many concurrent exploration needs to be performed, where each iteration takes $\Omega(m)$ rounds. This concludes that the total number of rounds at least required by any algorithm with $k = n + c$, co-located agents is $\Omega(m \log n)$. ◀

4.2 Scattered Agents

Next theorem shows the impossibility of BHS with $k = n + 2$ scattered agents.

► **Theorem 4.** *Given a dynamic torus \mathcal{G} of size $n \times m$, there does not exist any BHS algorithm which can correctly locate the black hole with $k = n + 2$ scattered agents, the result holds as well even if the nodes in \mathcal{G} has a whiteboard.*

► **Corollary 3.** *Any BHS algorithm on a dynamic torus \mathcal{G} of size $n \times m$ requires at least $k = n + 3$ scattered agents to correctly locate the black hole when each node in \mathcal{G} has a whiteboard of $O(1)$ bits.*

Following theorem is inspired from Theorem 4.2 in [22], which gives the lower bound on the round complexity for any BHS algorithm with k scattered agents along \mathcal{G} .

► **Theorem 5.** *Any BHS algorithm with $k = n + c$ scattered agents, where $c \in \mathbb{Z}^+$ and $c \geq 3$, on a $n \times m$ dynamic torus \mathcal{G} requires at least $\Omega(mn)$ rounds.*

5 Preliminaries

In this section, we explain all the subroutines, definitions and ideas used in our BHS algorithm, but first, we explain the contents maintained by the agents on the whiteboard.

Whiteboard. The following data is stored and maintained in the whiteboard by the agents. For each $dir \in \{east, west, north, south\}$ with respect to each $v_{i,j} \in \mathcal{G}$ we define the function $f : \{east, west, north, south\} \rightarrow \{\perp, 0, 1\}$,

$$f(dir) = \begin{cases} \perp, & \text{if an agent is yet to visit the port } dir \\ 0, & \text{if no agent has marked the port } dir \text{ as safe} \\ 1, & \text{if the port } dir \text{ is marked safe, i.e., the node along } dir \text{ is not black hole} \end{cases}$$

Cautious Walk. This is a fundamental movement strategy used in a network with a black hole, and it is used as a building block of all our algorithms. In this strategy, if two agents are together, then this strategy ensures that only one among them enters the black hole while the other survives. On the contrary, if only a single agent is present, then whenever it visits a new node, it leaves some mark behind on the whiteboard so that whenever another agent tries to visit this node along the same edge, it finds the mark and does not enter the black hole.

This walk is performed in three rounds, where if an agent a_1 (say) is alone (resp, with another agent a_2 , say) then in the first round a_1 decides to move one step along $e = (u, v)$ from u to v by marking $f(e) = 0$ (while a_2 waits) and if it is safe, i.e., does not contain the black hole, then in the next round a_1 returns to u and marks the edge e safe by writing $f(e) = 1$, then in the third step a_1 (resp, a_2) moves to v . This strategy ensures that no two agents enter the black hole along the edge e .

Stuck. An agent a_1 is defined to be *stuck* while exploring a 1-interval connected ring for two reasons.

- First, if while performing *cautious* walk along an edge $e = (u, v)$, a_1 at round r marks $f(e) = 0$ at u and moves to v , while v is safe and a_1 tries to return to u at round $r + 1$ to mark $f(e) = 1$, finds e to be missing, in this situation a_1 is *stuck* at v until e reappears.
- Second, if while moving along dir , a_1 finds e to be missing. In this situation, if more than one agent is simultaneously trying to move along dir at the same round and if a_1 is the lowest Id among them, then a_1 is *stuck* until e reappears, or, if a_1 is alone, then in that case also a_1 is *stuck* until e reappears.

5.1 Subroutines

In this section, we will discuss the sub-routines used as a building block in our BHS algorithms for both the co-located and scattered initial configurations. We have followed some of the pseudocode convention from the papers [11] and [22]. In this paper, we use three kinds of MOVE procedure in our algorithms, first, $\text{MOVE}(dir \mid p_1 : s_1; p_2 : s_2; \dots; p_k : s_k)$, second, $\text{MOVE}(dir \rightarrow f(dir) \mid p_1 : s_1; p_2 : s_2; \dots; p_k : s_k)$, and lastly, $\text{MOVE}(dir \rightarrow f(dir) \rightarrow s_i \mid p_1 : s_1; p_2 : s_2; \dots; p_k : s_k)$, where p_i is the predicate corresponding to the state s_i and $f(dir)$ represents the value with respect to dir (where $dir \in \{east, west, north, south\}$) in the whiteboard, so depending on the algorithm we use either of these MOVE procedures. The agent at each round, first takes a snapshot at its current location, and thereafter checks the predicates p_1, \dots, p_k one after another. If no predicate is satisfied, then in the first MOVE procedure, the agent moves along the direction dir , in the second MOVE procedure the agent moves along dir while updating the whiteboard of the current node along dir to $f(dir)$, and lastly, in the third MOVE procedure, in addition to moving along dir and updating the whiteboard, it also moves directly in to the state s_i . On the other hand, if some predicates are satisfied, then the agent chooses the first satisfied predicate (say) p_i , and the procedure stops, and the agent moves into state s_i corresponding to p_i . The predicate and state of the form $p_j : time + i \rightarrow s_j$ indicates that if p_j is satisfied then the agent enters the state s_j after $time + i$ rounds, whereas the predicate and the state of the form $p_j : f(dir) \rightarrow s_j$, indicates that if p_j is satisfied then the agent performs the action $f(dir)$ and then moves to the state s_j . Further, these procedures are again executed in the subsequent rounds. In the following part, we define the algorithm $\text{CAUTIOUS-WAITMOVEWEST}()$.

CAUTIOUS-WAITMOVEWEST(j, l): This algorithm works on 1-interval connected ring R_i (say), where the main purpose is to make a certain number of agents reach the node $v_{i,j}$ along the C_j -th column from any initial configuration. Further, whenever an agent reaches the desired node, and it is not stuck, it waits at that node until further instruction is provided.

The algorithm works as follows: for the first $4(l-1)m$ rounds, if an agent a_1 is instructed to perform CAUTIOUS-WAITMOVEWEST(j, l) along R_i , then it starts the following procedure, if the agent a_1 (say) is initially with another agent a_2 (say) and since a_1 is the lowest Id among them, a_1 starts cautious walk along *west* until it either gets stuck or has reached the desired node. On the other hand the task of a_2 is to follow a_1 until a_1 is stuck. While a_1 is stuck, a_2 performs the following action. If a_1 is stuck due to a missing edge along *west*, then a_2 instead of waiting reverses its direction to *east* and continues to perform *cautious* walk. Otherwise, if a_1 is stuck while returning back to mark a port safe along *west* which it has in the last round marked unsafe while exploring and, then a_2 waits for at most $3m$ rounds since the round it encountered this situation, and then reverses its direction and continues to perform *cautious* walk. On the other hand, if a_1 is alone, then it performs *cautious* walk until it either reaches the desired node or it is stuck. If a_1 catches another agent stuck, and if it is not the lowest Id among them, then it performs a similar action, as explained earlier in case of a_2 . After $4(l-1)m$ rounds have passed, the agents enter the state *Return*, in which each agent not stuck due to a missing edge tries to reach the node $v_{i,j}$.

► **Observation 1** ([11]). *Given a dynamic ring R and a cut U , where $|U| > 1$, if its footprint is connected by edges e_c and e_{cc} (where e_c and e_{cc} are the clockwise and counter-clockwise edges, respectively) to nodes in $V \setminus U$ (where V is the set of vertices not in U). If all the agents at a round r are at U , and do not try to cross along e_c , whereas there exists an agent which tries to cross along e_{cc} , then the adversary has the ability to prevent any agent from crossing U .*

CAUTIOUS-WAITMOVEWEST() ensures that this situation does not arise, as when an agent is stuck on e_c (or e_{cc}), another agent after finding this situation waits for at most $3m$ rounds (depending on the fact that whether the earlier agent is stuck while backtracking or it is stuck because it has encountered a missing edge along *west*), and then reverses its movement towards e_{cc} (or e_c), while the other agent remains stuck. Hence, there exists a round r where an agent each is trying to cross e_c , and another agent is trying to cross e_{cc} .

The following lemma ensures the correctness of our algorithm in detecting the black hole when the algorithm terminates.

► **Lemma 2.** *If an agent executing CAUTIOUS-WAITMOVEWEST() terminates while moving along a specific direction, then it correctly locates the black hole.*

Next corollary states that in the worst case at most two agents can enter the black hole while executing CAUTIOUS-WAITMOVEWEST().

► **Corollary 4.** *CAUTIOUS-WAITMOVEWEST() ensures that at most two agents enters the black hole.*

The following lemma shows the complexity and correctness of reaching the desired node while executing CAUTIOUS-WAITMOVEWEST().

► **Lemma 3.** *If two agents along a safe row ring R_i of size m ($m \geq 3$) execute our algorithm, then at least one agent reaches the desired node within $7m$ rounds.*

► **Lemma 4.** *If three agents are executing CAUTIOUS-WAITMOVEWEST($j, 3$) along R_i and $v_{i,j}$ is the black hole, then at most 2 agents enter the black hole whereas the adversary has the ability to stop the third agent from detecting the black hole location.*

The next corollary states that if the black hole is located at the current ring, then within $15m$ (or $15n$) rounds, the black hole is detected by a set of 4 agents.

► **Corollary 5.** *A set of 4 agents, executing CAUTIOUS-WAITMOVEWEST(j, l) (where $l \geq 4$) along R_i can correctly locate the black hole in at most $15m$ rounds, where $v_{i,j}$ is the black hole node.*

The following corollary states that while executing CAUTIOUS-WAITMOVEWEST(j, l) the worst situation happens with exactly 2 agents entering the black hole, which is when the desired node to reach is $v_{i,j}$ and it is also the black hole node.

► **Corollary 6.** *CAUTIOUS-WAITMOVEWEST(j, l) ensures that exactly 2 agents can be consumed by the black hole when the desired node $v_{i,j}$ is also the black hole node.*

The next lemma states that if at any round after the first $4m$ rounds since the start CAUTIOUS-WAITMOVEWEST(j, l) along R_i , if there still exists at least 3 agents yet to reach the desired node $v_{i,j}$, then our algorithm ensures that in a period of $4m$ rounds since the last agent has reached the desired node, at least one among these set of agents yet to reach the desired node, reaches $v_{i,j}$.

► **Lemma 5.** *After the first $4m$ rounds have elapsed executing CAUTIOUS-WAITMOVEWEST(j, l) (where $l > 2$), if at least 3 agents are still present along R_i then it takes at most $4m$ rounds for an agent among them to reach the desired node, since the last agent has reached the desired node.*

The following corollary follows from Lemmas 3 and 5.

► **Corollary 7.** *Our algorithm ensures that among l agents operating along R_i at least $l - 2$ agents reach the desired node within $4(l - 1)m$ rounds.*

► **Lemma 6.** *Among the remaining two agents which enter state Return after $4(l - 1)m$ rounds has elapsed, at least one among them reaches the desired node.*

The following theorem follows from Corollary 7 and Lemma 6.

► **Theorem 6.** *If l agents ($l \geq 2$) agents are in a safe ring R_i and they perform CAUTIOUS-WAITMOVEWEST(j, l), then at least $l - 1$ agents reach and stay on $v_{i,j}$ within $4(l - 1)m + 3m$ rounds, since the start of execution of our algorithm.*

CAUTIOUS-MOVE($west, j$): This algorithm is a special version of our earlier algorithm, it has two stages, and requires at least 2 agents. The first stage is exploration and works for $3m$ rounds, and the second stage is Exit. The algorithm works as follows: the lowest Id agent becomes the *Leader*, whereas the other agents become the *Follower*. The *Leader* explores new nodes in first stage and *Follower* follows the *Leader* until it either finds the *Leader* to be stuck or *Leader* stops reporting either due to a missing edge or it has entered the black hole. Whenever, the *Follower* finds the edge is missing and *Leader* is not reporting and $time < 3m$ it waits until the missing edge reappears or till $time = 3m$, whereas if it finds that the edge exists and *Leader* is not reporting then it terminates the algorithm, by declaring the node in which *Leader* has explored is the black hole node. On the other hand, whenever the *Leader* is also stuck due to a missing edge along its moving direction, then

both *Leader* and *Follower* wait until $time = 3m$. Whenever $time > 3m$, both *Leader* and *Follower* enter the second stage, i.e., state *Exit*, in which, irrespective of the fact that they are stuck or not, they try to return to their desired node, i.e., the node along C_j -th column, while returning each agent irrespective of *Leader* or *Follower* is instructed to mark the port of each node along their movement to 1 if not already marked so. Whenever the agents, while returning back, encounter a missing edge, the lowest Id agent waits, and other agents change direction.

The following two lemmas ensures that if l agents start executing CAUTIOUS-MOVE(), then among them eventually $l - 1$ agents reach the desired node within at most $3lm$ rounds, since the start of its execution.

► **Lemma 7.** *If l ($l \geq 2$) agents execute CAUTIOUS-MOVE(*west*, j) along a safe ring R_i , then at least $l - 1$ agents reach $v_{i,j}$ within $3lm$ rounds.*

► **Corollary 8.** *If l agents enter the state *Exit*, then at least $l - 1$ agents reach $v_{i,j}$ by at most $3(l - 1)m$ rounds.*

CAUTIOUSDOUBLEOSCILLATION[11] We have used this BHS ring exploration algorithm as a sub-routine in our BHS Torus exploration algorithm. This algorithm uses 3 agents to explore the ring and successfully detect the black hole, if the black hole node is along that ring, otherwise, it explores the ring. Among these three agents, one agent is recognised as the *Leader*, whereas the remaining two agents are known to be as AVANGUARD and RETROGUARD, respectively. The only difference is that both AVANGUARD and RETROGUARD while exploring a new node marks the corresponding ports safe or unsafe in whiteboard (where a port is safe implies that the adjacent node along a that port does not contain the black hole), so this means if RETROGUARD enters the black hole while exploring a sector of \sqrt{m} nodes along R_i (or \sqrt{n} nodes along C_j) then using the whiteboard instead of a pebble, LEADER can detect its location. As stated in [11], three agents executing CAUTIOUSDOUBLEOSCILLATION requires $O(m^{1.5})$ rounds to detect the black hole along a 1-interval connected ring of size m .

6 Co-located Agents

In this section, we propose two BHS algorithms on $n \times m$ dynamic torus. First algorithm requires $n + 3$ agents and works in $O(nm^{1.5})$ rounds, whereas the second algorithm requires $n + 4$ agents and works in $O(nm)$ rounds.

6.1 BHS with $n + 3$ agents

The set of $n + 3$ agents, $A = \{a_1, a_2, \dots, a_{n+3}\}$ are initially located at a safe node $v_{i,j}$, also termed as *home*. Initially from *home*, a_1 and a_2 executes the algorithm CAUTIOUS-MOVE(*north*, i), whereas a_3 and a_4 executes CAUTIOUS-MOVE(*south*, i). Once $12n$ rounds have passed, at least 3 out of these 4 agents return to $v_{i,j}$ (refer corollary 8). Whenever 3 among 4 agents return back to *home*, the first three lowest Id agents become LEADER, AVANGUARD and RETROGUARD and they are instructed to perform CAUTIOUSDOUBLEOSCILLATION along R_i . Now, as per Lemmas 14 and 15 in [11] it takes at most $T = 25m^{1.5} + 7(m + \sqrt{m})$ rounds to locate the black hole along R_i . So, after T rounds since the start of CAUTIOUSDOUBLEOSCILLATION, if the algorithm hasn't terminated (or the black hole is not detected) then these agents are instructed to return to $v_{i,j}$ which is the desired node, irrespective of the fact, whether they are stuck or not. While returning,

if an agent encounters a missing edge, then the lowest Id agent waits, whereas the other agent changes direction. Using this strategy, in at most $6m$ rounds, at least 2 among 3 agents return to $v_{i,j}$ (as this is similar to state *Exit* in algorithm CAUTIOUS-MOVE(), hence by corollary 8 this bound holds). After which they all together start executing CAUTIOUS-WAITMOVESOUTH($i - 1, 4$), which enables at least 3 among $n + 3$ agents reach the node $v_{i-1,j}$ and continue the same process. This process iterates for each R_i , where $0 \leq i \leq n - 1$.

► **Lemma 8.** *Our BHS algorithm ensures that there always exist 3 agents to perform CAUTIOUSDOUBLEOSCILLATION along R_i , where $0 \leq i \leq n - 1$.*

An iteration of our BHS algorithm is defined to be the collection of steps the set of agents perform from the node $v_{t,j}$ (where suppose $v_{i,j}$ be the initial starting node) to reach the node $v_{t+1,j}$ (where $t > 0 \ t \in \mathbb{N}^+$). More precisely, the agents at $v_{t,j}$, first perform CAUTIOUSDOUBLEOSCILLATION along R_t , then they try to return back to $v_{t,j}$, after which each agent along C_j try to reach the node $v_{t+1,j}$ while executing CAUTIOUS-WAITMOVESOUTH($t - 1, 4$), this whole process is defined to be one iteration. Now, the next lemma gives the number of rounds required by the set of $n + 3$ while executing our BHS algorithm to perform one iteration.

► **Lemma 9.** *It takes at most $T + 6m + 15n$ rounds to perform one iteration of the BHS algorithm with $n + 3$ agents.*

The following lemma and theorem gives the correctness and complexity of our BHS algorithm.

► **Lemma 10.** *Our algorithm correctly locates the black hole.*

► **Theorem 7.** *A group of $n + 3$ agents executing the BHS algorithm along a dynamic torus of size $n \times m$ correctly locates the black hole in $O(nm^{1.5})$ rounds.*

6.2 BHS with $n + 4$ agents

In this case the set of $n + 4$ agents, $A = \{a_1, a_2, \dots, a_{n+4}\}$ agents are initially co-located at *home* = $v_{i,j}$, say. The algorithm in this case is similar to the earlier BHS algorithm with $n + 3$ agents; the only difference is that here in order to explore R_t , instead of 3, 4 agents are used, where the lowest and second lowest Id agents at $v_{t,j}$ perform CAUTIOUS-MOVE(*west*, j) and the third lowest and fourth lowest Id agents are instructed to perform CAUTIOUS-MOVE(*east*, j), instead of CAUTIOUSDOUBLEOSCILLATION. The pseudocode is explained in Algorithm 1.

■ **Algorithm 1** BHS with $n + 4$ agents.

```

1 Instruct  $a_1$  and  $a_2$  to perform CAUTIOUS-MOVE(north,  $i$ ).
2 Instruct  $a_3$  and  $a_4$  to perform CAUTIOUS-MOVE(south,  $i$ ).
3 if time >  $12n$  then
4   for  $t = i; t \leq i - 1; t --$  do
5     Instruct the lowest and second lowest Id agents at  $v_{t,j}$  to perform
6       CAUTIOUS-MOVE(west,  $j$ ).
7     Instruct the third lowest and fourth lowest Id agents at  $v_{t,j}$  to perform
8       CAUTIOUS-MOVE(east,  $j$ ).
9        $\triangleright$  time1 is defined as the time since the last call of CAUTIOUS-MOVE.
10      if time1 >  $12m$  then
11        Perform CAUTIOUS-WAITMOVESOUTH( $t - 1, 5$ ).

```

► **Lemma 11.** *At least 3 among 4 agents executing CAUTIOUS-MOVE(*west*, *j*) and CAUTIOUS-MOVE(*east*, *j*) along R_t at some i -th iteration of the for loop of Algorithm 1, reach $v_{t,j}$ within $12m$ rounds since the start of CAUTIOUS-MOVE() in the current iteration, where $0 \leq t \leq n-1$.*

► **Lemma 12.** *Our BHS algorithm, with $n+4$ agents, ensures that in every iteration of the for loop of our algorithm, there always exists 4 agents to perform CAUTIOUS-MOVE(*west*, *j*) and CAUTIOUS-MOVE(*east*, *j*).*

The following lemma and theorem gives the correctness and complexity of our algorithm.

► **Lemma 13.** *A set of $n+4$ agents executing Algorithm 1, correctly locates the black hole.*

► **Theorem 8.** *A group of $n+4$ agents executing Algorithm 1 along a dynamic torus of size $n \times m$ correctly locates the black hole in $O(nm)$ rounds.*

7 Scattered Agents

This section proposes two BHS algorithms on an $n \times m$ dynamic torus. Our first algorithm works with $n+6$ agents and requires $O(nm^{1.5})$ rounds, whereas our second algorithm works with $n+7$ agents and requires $O(nm)$ rounds.

7.1 BHS with $n+6$ agents

A set of $n+6$ agents, $A = \{a_1, a_2, \dots, a_{n+6}\}$ are initially scattered along different nodes of the torus \mathcal{G} , i.e., the agents are arbitrarily placed, where there may be more than one agent at a node or there can be single agent at each $n+6$ nodes in \mathcal{G} .

At the first step, each agent performs CAUTIOUS-WAITMOVEWEST(0, 6) from any initial configuration, so after $time1 = 23m (g(6)+3m = 20m+3m)$ rounds has elapsed since the start of CAUTIOUS-WAITMOVEWEST(0, 6), each agent currently along C_0 is further instructed to perform CAUTIOUS-WAITMOVESOUTH(0, 6), so after $time2 = 23n (g(6) + 3n = 20n + 3n)$ rounds has elapsed since CAUTIOUS-WAITMOVESOUTH(0, 6), if at least 3 agents have reached the node $v_{0,0}$, then 3 lowest Id agents at $v_{0,0}$ become LEADER, AVANGUARD and RETROGUARD, respectively and are then instructed to perform CAUTIOUSDOUBLEOSCILLATION along R_0 . Hence, within T rounds from the start of CAUTIOUSDOUBLEOSCILLATION either the black hole is detected and the algorithm terminates or the ring R_0 is explored. After T rounds since the start of CAUTIOUSDOUBLEOSCILLATION, these 3 agents are instructed to return to $v_{0,0}$ by marking each node along their movement till $v_{0,0}$ to 1 (as the ring is explored and there is no black hole in this ring, so an agent can mark each port as safe, if not already marked so). So, by corollary 8 in at most $6m$ rounds, at least 2 among these 3 agents return to $v_{0,0}$, after which, each agent in G are instructed to perform CAUTIOUS-WAITMOVEWEST(0, 6).

On the other hand, if two agents have reached $v_{0,0}$ after $23n$ rounds have elapsed since CAUTIOUS-WAITMOVESOUTH(0, 6), then the lowest Id agent *cautiously* walks along *west* whereas the other agent *cautiously* walks along *east*. If along their movement they *catches* another agent trying to move along the same direction, then they together perform CAUTIOUS-MOVE() in the same direction. After $3m$ rounds has passed since they have started *cautious* walk, these agents along R_0 are instructed to return to $v_{0,0}$ by marking each port along their movement to 1. So, within $6m$ rounds, if 3 agents are along R_0 , then at least 2 among them returns, or if 2 agents are along R_0 , then at least 1 among them returns, further each agent along G is again instructed to perform CAUTIOUS-WAITMOVEWEST(0, 6). This process iterates for each R_i ring (where $0 \leq i \leq n-1$), depending on whether 2 or 3 agents have reached the node $v_{i,0}$.

The following lemma states that if 2 agents eventually reach the node $v_{i,0}$ at some i -th iteration, then this implies that among the $n + 6$ agents, already 3 agents have entered the black hole from three different directions without the black hole getting detected.

► **Lemma 14.** *If 2 agents reach $v_{i,0}$ at the i -th iteration after the execution of CAUTIOUS-WAITMOVESOUTH($i, 6$) when $time2 > 23n$, and the algorithm has not terminated yet, then this implies exactly 3 agents has entered black hole from three different directions.*

This corollary gives the possible directions along which 3 agents might have entered the black hole without still detecting it, while executing our algorithm.

► **Corollary 9.** *If 3 agents enter black hole from 3 directions without detecting it, then 2 among these 3 directions are east and west, whereas the 3rd is either north or south.*

The following lemma states that if eventually 2 agents reach the node $v_{i,0}$ at some i -th iteration while executing our BHS algorithm, then this implies that there must be another agent stuck somewhere at a node along R_i other than the fact that 3 agents have already entered the black hole and an agent each is already stuck along the remaining $n - 1$ row rings. Otherwise only 2 agents must not have reached the node $v_{i,0}$.

► **Lemma 15.** *Our BHS algorithm with $n + 6$ agents, ensures that if at the i -th iteration after $time2 > 23n$ only 2 agents are present at $v_{i,0}$, then there exists another agent stuck somewhere along R_i .*

The following lemma and theorem gives the correctness and complexity of our algorithm.

► **Lemma 16.** *Our BHS algorithm with $n + 6$ agents correctly locates the black hole.*

► **Theorem 9.** *A group of $n + 6$ agents executing the BHS algorithm along a dynamic torus of size $n \times m$ correctly locates the black hole in $O(nm^{1.5})$ rounds.*

7.2 BHS with $n + 7$ agents

In this case the set of $n + 7$ agents, $A = \{a_1, a_2, \dots, a_{n+7}\}$ are scattered along the nodes of \mathcal{G} . The BHS algorithm with $n + 7$ agents is almost similar to the earlier BHS algorithm with $n + 6$ agents. The differences are as follows: at each iteration the agents are instructed to perform CAUTIOUS-WAITMOVEWEST($0, 7$) instead of CAUTIOUS-WAITMOVEWEST($0, 6$). Next, while exploring a ring R_i at the i -th iteration at least 3 agents reach $v_{i,0}$ within $time2 > 27m$ ($g(7) + 3m = 24m + 3m$), whereas in earlier BHS algorithm with $n + 6$ agents at least 2 agents reach $v_{i,0}$, within $time2 > 23m$. Next, if 3 agents reach $v_{i,0}$, then our earlier algorithm, 2 agent scenario is similar to 3 agent scenario in this case. In our BHS algorithm with $n + 6$ agents, both agents are instructed to walk *cautiously* along *west* and *east*, respectively, but now as we have one more agent, the two lowest Id agents among them perform CAUTIOUS-MOVE(*west*, i), while the other walks *cautiously* along *east*. Otherwise, if 4 agents reach $v_{i,0}$, then this case is again similar to our 3 agent scenario in our earlier BHS algorithm with $n + 6$ agents, in which these 3 agents perform CAUTIOUSDOUBLEOSCILLATION whereas in this algorithm as we have one more agent, so two lowest Id agents perform CAUTIOUS-MOVE(*west*, 0) and the other two agents (i.e., 3rd lowest and 4th lowest Id agents) perform CAUTIOUS-MOVE(*east*, 0), and all these process iterates for each row ring.

► **Lemma 17.** *If 3 agents reach $v_{i,0}$ when $time2 > 27n$, and the algorithm has not terminated, then 3 agents have entered the black hole from three different directions.*

► **Lemma 18.** *Our BHS algorithm with $n + 7$ agents ensures that if at the i -th iteration after $time2 > 27n$ only 3 agents are present at $v_{i,0}$, then there exists another agent stuck somewhere along R_i .*

Lemmas 17 and 18 are just a consequence of Lemmas 14 and 15. Also, Corollary 9 holds for this algorithm as well.

► **Lemma 19.** *Our BHS algorithm with $n + 7$ agents correctly locates the black hole.*

► **Theorem 10.** *A group of $n + 7$ agents executing our algorithm along a dynamic torus of size $n \times m$ correctly locates the black hole in $O(nm)$ rounds.*

8 Conclusion

In this paper, we have considered the black hole search problem on a dynamic torus, in which each row and column are 1-interval connected. We have considered two types of initial configuration of the agents, and in each case, gave the bounds (both upper and lower bound) on number of agents and complexity in order to locate the black hole. To be specific, when the agents are initially co-located, first, we give lower bounds of $n + 2$ and $\Omega(m \log n)$ on number of agents and rounds, respectively. Next, with $n + 3$ agents, we design a BHS algorithm that works in $O(nm^{1.5})$ rounds, whereas with $n + 4$ agents, we propose an improved algorithm that works in $O(nm)$ rounds.

When the agents are scattered, we give a lower bound of $n + 3$ and $\Omega(mn)$ on a number of agents and rounds, respectively. Next, we propose two BHS algorithms, first, works with $n + 6$ agents in $O(nm^{1.5})$ rounds and second, works with $n + 7$ agents in $O(nm)$ rounds (round optimal algorithm).

Moreover, in this paper we have considered that each node in the dynamic torus is labeled. A possible future work in this direction can be first to remove this assumption and give a BHS algorithm and check if the bounds get changed. Secondly, for both these cases, finding an agent optimal algorithm is another possible direction which can be pondered in to.

References

- 1 Balasingham Balamohan, Paola Flocchini, Ali Miri, and Nicola Santoro. Time optimal algorithms for black hole search in rings. *Discrete Mathematics, Algorithms and Applications*, 3(04):457–471, 2011.
- 2 Adri Bhattacharya, Giuseppe F Italiano, and Partha Sarathi Mandal. Black hole search in dynamic cactus graph. In *International Conference and Workshops on Algorithms and Computation, WALCOM 2024*, pages 288–303. Springer, 2024.
- 3 Adri Bhattacharya, Giuseppe F Italiano, and Partha Sarathi Mandal. Black hole search in dynamic tori. *arXiv preprint arXiv:2402.04746*, 2024.
- 4 Sebastian Brandt, Jara Uitto, and Roger Wattenhofer. A tight lower bound for semi-synchronous collaborative grid exploration. *Distributed Computing*, 33:471–484, 2020.
- 5 Jérémie Chalopin, Shantanu Das, Arnaud Labourel, and Euripides Markou. Black hole search with finite automata scattered in a synchronous torus. In *Distributed Computing: 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings 25*, pages 432–446. Springer, 2011.
- 6 Jérémie Chalopin, Shantanu Das, Arnaud Labourel, and Euripides Markou. Tight bounds for black hole search with scattered agents in synchronous rings. *Theoretical Computer Science*, 509:70–85, 2013.
- 7 Reuven Cohen, Pierre Fraigniaud, David Ilcinkas, Amos Korman, and David Peleg. Label-guided graph exploration by a finite automaton. *ACM Transactions on Algorithms (TALG)*, 4(4):1–18, 2008.
- 8 Jurek Czyzowicz, Stefan Dobrev, Rastislav Kráľovič, Stanislav Miklík, and Dana Pardubská. Black hole search in directed graphs. In *Structural Information and Communication Complexity*:

- 16th International Colloquium, SIROCCO 2009, Piran, Slovenia, May 25-27, 2009, Revised Selected Papers 16*, pages 182–194. Springer, 2010.
- 9 Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability and Computing*, 16(4):595–619, 2007.
 - 10 Shantanu Das, Dariusz Dereniowski, and Christina Karousatou. Collaborative exploration of trees by energy-constrained mobile robots. *Theory of Computing Systems*, 62:1223–1240, 2018.
 - 11 Giuseppe Antonio Di Luna, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Tight bounds for black hole search in dynamic rings. *arXiv preprint arXiv:2005.07453*, 2020.
 - 12 Yann Disser, Jan Hackfeld, and Max Klimm. Tight bounds for undirected graph exploration with pebbles and multiple agents. *Journal of the ACM (JACM)*, 66(6):1–41, 2019.
 - 13 S Dobrev, P Flocchini, R Kralovic, and N Santoro. Exploring a dangerous unknown graph using tokens. In *Proceedings of 5th IFIP International Conference on Theoretical Computer Science*, pages 131–150, 2006.
 - 14 Stefan Dobrev, Paola Flocchini, Rastislav Kráľovič, P Ružička, Giuseppe Prencipe, and Nicola Santoro. Black hole search in common interconnection networks. *Networks: An International Journal*, 47(2):61–71, 2006.
 - 15 Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. *Distributed Computing*, 19:1–99999, 2006.
 - 16 Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 48:67–90, 2007.
 - 17 Stefan Dobrev, Rastislav Kráľovič, Nicola Santoro, and Wei Shi. Black hole search in asynchronous rings using tokens. In *Algorithms and Complexity: 6th Italian Conference, CIAC 2006, Rome, Italy, May 29-31, 2006. Proceedings 6*, pages 139–150. Springer, 2006.
 - 18 Stefan Dobrev, Nicola Santoro, and Wei Shi. Using scattered mobile agents to locate a black hole in an un-oriented ring with tokens. *International Journal of Foundations of Computer Science*, 19(06):1355–1372, 2008.
 - 19 Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. *Algorithmica*, 65:562–583, 2013.
 - 20 Paola Flocchini, David Ilcinkas, and Nicola Santoro. Ping pong in dangerous graphs: Optimal black hole search with pebbles. *Algorithmica*, 62:1006–1033, 2012.
 - 21 Tsuyoshi Gotoh, Paola Flocchini, Toshimitsu Masuzawa, and Nicola Santoro. Exploration of dynamic networks: tight bounds on the number of agents. *Journal of Computer and System Sciences*, 122:1–18, 2021.
 - 22 Tsuyoshi Gotoh, Yuichi Sudo, Fukuhito Ooshita, Hirotugu Kakugawa, and Toshimitsu Masuzawa. Exploration of dynamic tori by multiple agents. *Theoretical Computer Science*, 850:202–220, 2021.
 - 23 Shota Nagahama, Fukuhito Ooshita, and Michiko Inoue. Ring exploration of myopic luminous robots with visibility more than one. *Information and Computation*, 292:105036, 2023.
 - 24 Claude E Shannon. Presentation of a maze-solving machine. *Claude Elwood Shannon Collected Papers*, pages 681–687, 1993.
 - 25 Wei Shi, Joaquin Garcia-Alfaro, and Jean-Pierre Corriveau. Searching for a black hole in interconnected networks using mobile agents and tokens. *Journal of Parallel and Distributed Computing*, 74(1):1945–1958, 2014.
 - 26 Yuichi Sudo, Daisuke Baba, Junya Nakamura, Fukuhito Ooshita, Hirotugu Kakugawa, and Toshimitsu Masuzawa. A single agent exploration in unknown undirected graphs with whiteboards. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 98(10):2117–2128, 2015.

Online Space-Time Travel Planning in Dynamic Graphs

Quentin Bramas  

University of Strasbourg, ICUBE, CNRS, Strasbourg, France

Jean-Romain Luttringer  

University of Strasbourg, ICUBE, CNRS, Strasbourg, France

Sébastien Tixeuil  

Sorbonne University, CNRS, LIP6, Institut Universitaire de France, Paris, France

Abstract

We study the problem of traveling in an unknown dynamic graph, to reach a destination with minimum latency. At each step of the execution, an agent can decide to move to a neighboring node if an edge exists at this time instant, wait at the current node in the hope that other links will appear in the future, or move backward in time using an expensive time travel device. A travel that makes use of backward time travel is called a space-time travel. Our aim is to arrive at the destination with zero delay, which always requires the use of backward time travel if no path exists to the destination during the first time instant.

Finding an optimal space-time travel is polynomial when the agent knows the entire dynamic graph (including the future edges), even with additional constraints. However, we consider in this paper that the agent discovers the dynamic graph while it is exploring it, in an online manner.

In this paper, we propose two models that define how an agent learns new knowledge about the dynamic graph during the execution of its protocol: the T-online model, where the agent reaching time t learns about the entire past of the network until t (even nodes not yet visited), and the S-online model, where the agent learns about the past and future about the current node he is located at. We present an algorithm with an optimal competitive ratio of 2 for the T-online model. In the S-online model, we prove a lower bound of $2/3n - 7/4$ and an upper bound of $2n - 3$ on the optimal competitive ratio when the cost function is linear.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Dynamic graphs, online algorithm, space-time travel, treasure hunt

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.7

1 Introduction

We consider the problem of an agent moving in both space and time in a dynamic graph representing a transportation network. The goal of the agent is to reach a destination node in the aforementioned graph with a delay of zero, thanks to backward time-travels. As the dynamic graph evolves, its edges may appear and disappear over time. The agent can wait at a given node for an adjacent edge to appear (thus moving forward in time). However, conversely to most known models, we consider that the agent can also go back in time, to cross an adjacent edge that previously appeared in the past. However, moving backward in time involves a cost that the agent seeks to minimize.

It has been shown by Bramas et al. [4] that finding optimal-delay optimal-costs travels can be computed with a polynomial offline algorithm, even when assuming an upper constraint on the cost. However, the offline setting considered by Bramas et al. [4] implies that the agent knows the entire dynamic graph. For example, an agent at time t may only be aware of the evolution of the dynamic graph up to time t (e.g., if this dynamic graph represents a transportation network, unforeseen problems may arise in the future, while past availability



© Quentin Bramas, Jean-Romain Luttringer, and Sébastien Tixeuil;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 7; pp. 7:1–7:14

Leibniz International Proceedings in Informatics



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

periods have been tracked). Also, the dynamic graph itself may be infinite, which may cause storage issues before running the offline algorithm. Thus, it is important to consider settings where the agent that plans its space-time travel only has limited knowledge.

In this paper, we focus on *online* settings, where agents possess limited initial knowledge of the underlying dynamic graph. We define two settings, referred to as *T-Online* and *S-Online*. In the T-online setting, the agent does *not* know the future of the dynamic graph, but learns everything about the temporal graph up to its current time instant (even the existence of time-edges between nodes that were not yet visited). Thus, the agent has complete spatial knowledge up to its current time instant, but still navigates *Time* in an *online* fashion. Conversely, in the S-online settings, the agent knows the entire past and future of the nodes it has visited, but has no knowledge about yet unvisited nodes, and thus navigates *Space* in an *online* fashion. The knowledge acquired by the agent in both settings is illustrated in Fig. 3 and 4.

Related Work. Space-Time routing has been studied, mostly assuming forward time travel, i.e., waiting, is available. Many studies (see *e.g.* Casteigts et al.[7] and references herein) recently revisited popular problems previously studied in static graphs [6, 9, 19] in a dynamic context.

Casteigts et al [8] studied the possibility of discovering a *restless* temporal path between two nodes in a dynamic network with a waiting time constraint: at each step, the traveling agent cannot wait more than c time instants, where c is a given constant. It turns out that computing such paths is NP-Hard. Perhaps surprisingly, Villacis-Llobet et al [20] showed that if one allows going several times through the same node, the obtained restless temporal walk can arrive earlier, and finding it can be done in linear time. As previously mentioned, this line of work only considers *forward* time travel (a temporal path cannot go back in time), and focuses on offline settings.

Multi-criteria path computation problems have been extensively studied within computer networks [10, 16, 17]. In this context, each edge is characterized by a weight vector, comprising both cost and delay. Path computation algorithms thus have to maintain and explore all non-comparable paths, whose number may grow exponentially with respect to the size of the network, leading to the use of approximation schemes or heuristics. However, these works always focus on static graphs and offline settings.

As aforementioned, Bramas et al. [4] have proposed path computation algorithms on dynamic graphs with both forward and backward time-travel (assuming costly backward time-travel). They demonstrated the polynomial solvability of finding the path with minimum delay, even when constraining (or optimizing) the cost. Note that, conversely to us, such travels may not always allow for a delay of 0, if the constraint on the cost is too stringent. However, their study exclusively focuses on offline settings.

Related online problems in graphs include graph exploration and treasure hunting. Online graph exploration has been extensively studied in the literature in models that are similar to our S-online model, i.e., when visiting a node, the agent learns about the identifiers of the neighboring nodes. Algorithms with optimal competitive ratios were found in various classes of graphs such as cycles, tadpole graphs [5], trees [15], and arbitrary graphs [2], in undirected and directed [14] graphs. The case where more than one agent explores the graph has also been investigated [11, 12].

The treasure-hunting problem is equivalent to the problem of reaching a destination node with minimum latency, if considering the destination as the node where the treasure is located. Previous work on treasure hunting only considers static graph [1, 3], usually

considering a different model where the agent sees outgoing edges, but lacks visibility of the neighboring nodes identifiers. In [18], the authors considered a model similar to ours, where the agent sees the neighboring nodes, and show that the optimal competitive ratio is $\Theta(n)$. This result implies the same asymptotic bounds in our model when assuming a linear cost function, but the exact bound remains unknown. Moreover, an exact bound in their setting cannot be generalized to our setting as an edge with a given cost requires several nodes to be emulated in our model.

To the best of our knowledge, our paper is the first to consider a problem similar to the online graph exploration and treasure-hunting problem in dynamic graphs.

Contributions. In this paper, we provide the following contributions:

- We introduce the problem of online space-time travel in dynamic networks and formally define several settings. In particular (a) in the T-online setting, an agent learns the past of the entire network when reaching a particular moment in time, and (b) in the S-online setting, an agent learns the past and future interactions involving the node where it is currently located.
- We present a T-online algorithm with an optimal competitive ratio able to compute a space-time travel with lowest delay and having a cost of at most two times the optimal cost.
- We present a lower bound of $2n/3 - 7/3$ for the competitive ratio of S-online algorithms, even if the cost function is the identity. In contrast, we provide a $2n - 3$ competitive S-online algorithm assuming a linear cost function. This algorithm is at most n^2 competitive for arbitrary (but feasible) cost functions.

Our work opens several problems, for instance, how to close the gap between our lower and upper bound regarding the competitive ratio of S-online algorithms.

2 Model

In this section, we define the models and notations used throughout this paper, before formalizing the aforementioned problems.

We represent the dynamic graph as an evolving graph, as introduced by Ferreira [13]: a graph-centric view of the network that maps a dynamic graph as a sequence of static graphs. The *footprint* of the dynamic graph (that includes all nodes and edges that appear at least once during the lifetime of the dynamic graph), is fixed. Furthermore, we assume that the set of nodes is fixed over time, while the set of edges evolves.

More precisely, an evolving graph G is a pair $(V, (E_t)_{t \in \mathbb{N}})$, where V denotes the set of vertices, \mathbb{N} is the set of time instants, and for each $t \in \mathbb{N}$, E_t denotes the set of edges that appears at time t . The *snapshot* of G at time t is the static graph $G(t) = (V, E_t)$, which corresponds to the state, supposedly fixed, of the network in the time interval $t, t + 1$). The *footprint* $\mathcal{F}(G)$ of G is the static graph corresponding the union of all its snapshots, $\mathcal{F}(G) = (V, \bigcup_{t \in \mathbb{N}} E_t)$. We say $(\{u, v\}, t)$ is a temporal edge of graph G if $\{u, v\} \in E_t$. We say that an evolving graph is *connected* if its footprint is connected.

Space-time Travel. We assume that at each time instant, an agent can travel along any number of adjacent consecutive communication links. However, the graph may not be connected at each time instant, hence it may be that the only way to reach a particular destination node is to travel forward (i.e., wait) or backward in time, to reach a time instant where an adjacent communication link exists. In more detail, an agent travels from a node s to a node d using a *space-time travel* (or simply travel when it is clear from the context).

7:4 Online Space-Time Travel Planning in Dynamic Graphs

► **Definition 1.** A space-time travel of length k is a sequence $((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$ such that

- $\forall i \in \{0, \dots, k\}$, $u_i \in V$ is a node and $t_i \in \mathbb{N}$ is a time instant,
- $\forall i \in \{0, \dots, k-1\}$, if $u_i \neq u_{i+1}$, then $t_i = t_{i+1}$ and $\{u_i, u_{i+1}\} \in E_{t_i}$ i.e., there is a temporal edge between u_i and u_{i+1} at time t_i .

By extension, the *footprint* of a travel is the static graph containing all edges (and their adjacent nodes) appearing in the travel. Now, the *itinerary* of a travel $((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$ is its projection (u_0, u_1, \dots, u_k) on nodes, while its *schedule* is its projection (t_0, t_1, \dots, t_k) on time instants. Let $\mathcal{T}_G((u, t), (v, t'))$ denote the set of travels in G starting from node u at time t , and arriving at node v at time t' .

► **Definition 2.** A travel $((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$ is simple if for all $i \in \{2, \dots, k\}$ and $j \in \{0, \dots, i-2\}$, we have $u_i \neq u_j$.

Intuitively, a travel is simple if its footprint is a line (i.e., a simple path) and contains at most one time-travel per node (as a consequence, no node appears three times consecutively in a simple travel).

► **Definition 3.** The delay of a travel $T = ((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$, denoted $\text{delay}(T)$ is defined as $t_k - t_0$.

The Backward cost of a travel.

► **Definition 4.** The backward-cost is the cost of going to the past. The backward-cost function $\mathfrak{f} : \mathbb{N}^* \rightarrow \mathbb{R}^+$ returns, for each $\delta \in \mathbb{N}$, the backward-cost $\mathfrak{f}(\delta)$ of traveling δ time instants to the past. As we assume that there is no cost associated to forward time travel (that is, waiting), we extend \mathfrak{f} to \mathbb{Z} by setting $\mathfrak{f}(-\delta) = 0$, for all $\delta \in \mathbb{N}$. In particular, the backward-cost of traveling 0 time instants in the past is zero. When it is clear from context, the backward-cost function is simply called the cost function.

► **Definition 5.** The backward-cost (or simply cost) of a travel $T = ((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$, denoted $\text{cost}(T)$ is defined as follows:

$$\text{cost}(T) = \sum_{i=0}^{k-1} \mathfrak{f}(t_i - t_{i+1})$$

► **Definition 6.** Let $T_1 = ((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$ and $T_2 = ((u'_0, t'_0), (u'_1, t'_1), \dots, (u'_{k'}, t'_{k'}))$ be two travels. If $(u_k, t_k) = (u'_0, t'_0)$, then the concatenated travel $T_1 \oplus T_2$ is defined as follows:

$$T_1 \oplus T_2 = ((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k), (u'_1, t'_1), \dots, (u'_{k'}, t'_{k'}))$$

► **Remark 7.** One can easily prove that $\text{cost}(T_1 \oplus T_2) = \text{cost}(T_1) + \text{cost}(T_2)$. In the following, we sometimes decompose a travel highlighting an intermediate node: $T = T_1 \oplus ((u_i, t_i)) \oplus T_2$. Following the definition, this means that T_1 ends with (u_i, t_i) , and T_2 starts with (u_i, t_i) , so we also have $T = T_1 \oplus T_2$ and $\text{cost}(T) = \text{cost}(T_1) + \text{cost}(T_2)$.

Our notion of space-time travel differs from the classical notion of *journey* found in the literature related to dynamic graphs [13] as we do *not* assume time instants monotonically increase along a travel. As a consequence, some evolving graphs may not allow a journey from A to B yet allow one or several travels from A to B .

We say a travel is cost-optimal, if there does not exist a travel with the same departure and arrival node and times as T having a smaller cost. One can easily prove the following Property.

► **Property 1.** *Let T be a cost-optimal travel from node u to node v arriving at time t , and T' a sub-travel of T i.e., a travel such that $T = T_1 \oplus T' \oplus T_2$. Then T' is also cost-optimal.*

In the remaining of this paper, we consider a given evolving graph $G = (V, (E_t)_{t \in \mathbb{N}})$, a given cost function f , a source nodes s and a destination node d in V .

Problem specification. We consider an agent that travels in the evolving graph, starting from a node s at time 0. When at a node u at time t , the agent can either wait, go back in time, or traverse an edge to a neighboring node v if the temporal edge $(\{u, v\}, t)$ exists. If it waits, it stays in the same node, but the time is incremented by one. If it goes back in time, the new time can be any t' , $0 \leq t' < t$, and the cost of this operation is $f(t - t')$. Note that when traveling in space the agent can traverse several edges during a single time instant. When time-traveling, the agent may travel back or forward in time to any time instant but will remain on the same node.

The goal of the agent is to reach the destination d with minimal delay, i.e., arriving at time 0. Notice that a backward time-travel is always necessary if no path exists to the destination during the first time instant, and that reaching d at time 0 is always possible if the footprint is connected.

This problem is trivial when the agent knows the entire evolving graph [4] (even when the cost is constrained, and when the backward time travels are limited in amplitude). In this paper, we consider that the agent has a limited initial knowledge of the evolving graph, and it can learn more information by moving in the graph (moving in the topological or temporal sense).

Online Algorithms. An online algorithm A is a function that takes as input tuple (G', t, u) where G' is a sub-graph of G representing the partial information of the agent, t the current time, and u a node where the agent is located. The algorithm outputs the action performed by the agent: wait, go back at time $t' < t$ or traverse an edge. For simplicity, we can consider without loss of generality that the output is a space-time travel T that exists in G' . By doing so, the agent may learn new information about the traversed nodes or they can wait to learn new information about the future. A single action (wait, go back in time, or traverse an edge) can be seen as an elementary space-time travel.

We consider only deterministic algorithms so that executing an online algorithm on a given evolving graph G makes the agent follow a single space-time travel (maybe of infinite length if the agent loops for infinity). On a given evolving graph G , the cost obtained by an online algorithm A is denoted $Cost(A, G)$ and is the cost of the space-time travel performed by the agent on this graph. For comparison, we denote by $Cost(opt, G)$ the optimal cost given by an optimal offline algorithm.

Our goal is to find an algorithm that minimizes the competitive ratio defined as follows.

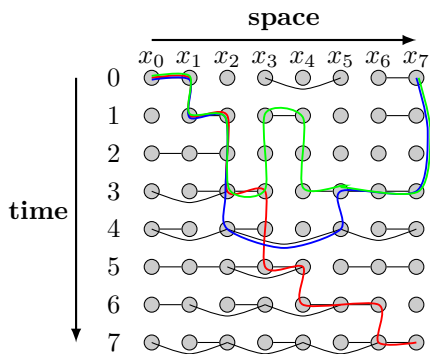
► **Definition 8.** *An online algorithm has competitive ratio ρ if in any evolving graph G , we have*

$$\frac{Cost(A, G)}{Cost(opt, G)} \leq \rho$$

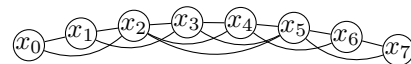
Acquiring new Knowledge. The way the agent learns about the evolving graph depends on the model. We consider two cases. First, in the *T-online* model, when an agent reaches time t , they learn about all the temporal edges $(\{u, v\}, t') \in E$ with $t' \leq t$, for all $u, v \in V$. In other words, they learn about the entire graph up to this time. Second, in the *S-online* setting,

when an agent reaches a node u , it learns about all the temporal edges $(\{u, v\}, t') \in E$, for all $t' \geq 0$ and for all $v \in V$. In other words, it learns all the information, past and future, concerning the current node.

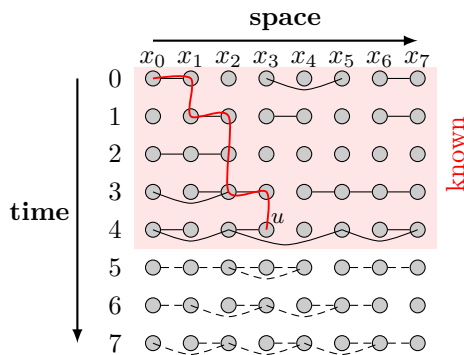
Observe that the definition of competitive ratio does not depend on the setting, as the offline optimal algorithm gives the same solution regardless of the setting. The setting just impacts the knowledge of the agent, so in practice, different knowledge should give different algorithms, and it might not be possible to obtain the same competitive ratio in two different settings. Hence, we are also interested in finding lower bounds for the competitive ratio in each setting.



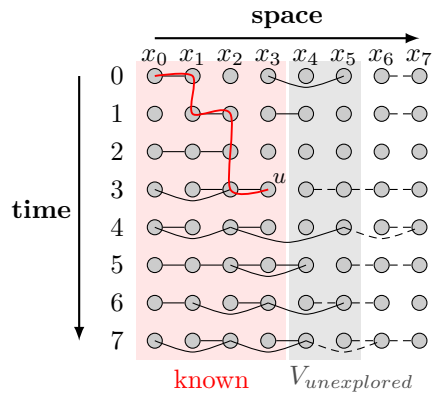
■ **Figure 1** Possible representation of an evolving graph. Possible travels from x_0 to x_7 are shown in red, green, and blue. Note that the blue and green travels require sending an agent to the past (to a previous time instant).



■ **Figure 2** Footprint of the evolving graph represented in Figure 1.



■ **Figure 3** Example of the state of an agent during a T-online travel. The agent at position u does not know about the dashed edges



■ **Figure 4** Example of the state of an agent during an S-online travel. The agent at position u does not know about the dashed edges, nor the unknown nodes outside $V_{unexplored}$.

Visual representation of online space-time travels. To help visualize the problem, consider a set of $n + 1$ nodes denoted $x_0, x_1, x_2, \dots, x_n$. Then, the associated evolving graph can be seen as a vertical sequence of graphs mentioning for each time instant which edges are present. A possible visual representation of an evolving graph can be seen in Fig. 1. One can see the evolution of the topology (consisting of the nodes x_0 to x_7) over time through eight snapshots performed from time instants 0 to 7. Several possible travels are shown in red,

green, and blue. The red travel only makes use of forward time travel (that is, waiting) but is the earliest arriving travel in this class (arriving at time 7, while it is possible to arrive at time 4). The green and blue travels both make use of backward time travel and arrive at time 0, so they have minimal travel delay. Similarly, the red travel concatenated with $((x_7, 7), (x_7, 0))$ (i.e., a backward travel to reach x_7 at time 0) also has minimal travel delay. However, if we assume that the cost function is the identity ($f : d \mapsto d$) then the green travel has a backward cost of 5, the blue travel has a backward cost of 4, and the concatenated red travel has a backward cost of 7.

The main challenge arises when an agent explores the graph in an online manner, i.e., learns about the graph while it is exploring it. Figure 3 and 4 illustrate the current knowledge of the agent after it traversed the red travel and is currently at node u . In Figure 3, the agent is T-online and knows about the entire past of the graph, i.e., it knows about all the edges that occurred at time 4 or before, regardless of the nodes involved. The agent knows about a possible travel to reach destination x_7 , but does not know if it is cost-optimal depending on the cost function (with a cost function $f : x \mapsto x$, it knows that the blue travel is optimal).

In contrast, in Figure 4, the agent is S-online and knows about the past and the future of all the visited nodes x_0, \dots, x_3 . In this case, the agent does not know a travel to the destination yet, but it is challenging to decide what node to explore first to minimize the cost.

3 Backward-cost Function Classes

The cost function f represents the cost of going back to the past. It has been shown by Bramas et al. [4] that it is necessary for f to be non-negative and that it attains its minimum (not just converge to it) on every interval that includes infinity, for an optimal-delay optimal cost travel to exist. These conditions were also shown to be sufficient for an offline algorithm to find an optimal solution.

► **Definition 9.** *A cost function f is user optimizable if it is non-negative, and it attains its minimum when restricted to any interval $[C, \infty)$, with $C > 0$. Let \mathcal{UO} be the set of user optimizable cost functions.*

For simplicity, in this paper, we only consider *user friendly* cost functions as defined by Bramas et al. [4]:

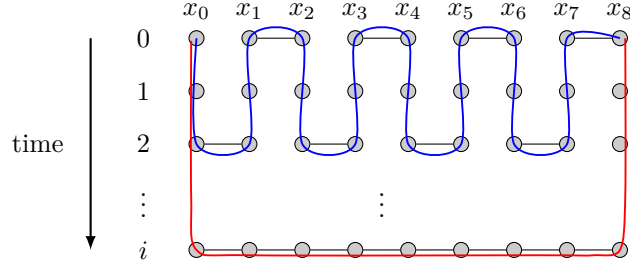
► **Definition 10.** *A cost function f is user friendly if it is user optimizable, non-decreasing, and sub-additive¹. Let \mathcal{UF} be the set of user friendly cost functions.*

Indeed, following the methodology by Bramas et al. [4], the optimal output of an algorithm using a user friendly function can be transformed into an optimal solution assuming the cost function is only user optimizable.

4 T-Online Algorithm with Optimal Competitive Ratio

In this section, we consider the T-online setting. In other words, the future of the evolving graph is unknown to the algorithm: at a time t , only the snapshots at time instants $t' \leq t$ are known. We first prove that there exists no algorithm with a competitive ratio smaller than 2, even if the cost function is the identity. Then we present our T-online Algorithm 1 and we show that it has an optimal competitive ratio.

¹ sub-additive means that for all $a, b \in \mathbb{N}$, $f(a + b) \leq f(a) + f(b)$



■ **Figure 5** Definition of the evolving graphs G^i , with 9 nodes ($n = 8$). The blue travel T_8 has a backward-cost of 8. The red travel T_i has a backward cost of i .

► **Theorem 11.** Assuming $f : x \mapsto x$, if the future is unknown, there exists no T -online algorithm with competitive ratio $2 - \varepsilon$, with $\varepsilon > 0$.

Proof. Assume for the sake of contradiction that algorithm A is a T -online algorithm and has a competitive ratio of $2 - \varepsilon$, with $\varepsilon > 0$. Let n be an even integer greater than $\frac{5}{\varepsilon}$. For any $i > 3$, let G^i be an evolving graph whose footprint is a line with $n + 1$ nodes x_0, x_1, \dots, x_n defined in the following way:

- $G^i(0)$ is the graph where half of the edges are present:

$$E^i(0) = \{\{x_k, x_{k+1}\} \mid k \in [0, n] \wedge k \equiv 1 \pmod{2}\}.$$

- $G^i(2)$ is the graph where the other half of the edges are present:

$$E^i(2) = \{\{x_k, x_{k+1}\} \mid k \in [0, n] \wedge k \equiv 0 \pmod{2}\}.$$

- $G^i(i)$ is a line graph : $E^i(i) = \{\{x_k, x_{k+1}\} \mid k \in [0, n - 1]\}$.

- for all $j \notin \{0, 2, i\}$, $G^i(j)$ is a graph with no edge : $E^i(j) = \emptyset$.

It is clear that, in all such graphs G^i , there exists a travel from x_0 to x_n , denoted by T_n , with backward-cost n , using the edges present at time 0 and 2 (the blue travel in Figure 5). In addition, there exists a travel, denoted T_i , of backward-cost i in the evolving graph G^i (the red travel in Figure 5).

If $i > n$, the optimal travel is T_n , and if $i < n$ the optimal travel is T_i .

Let us now run Algorithm A on the evolving graph G^{2n} , with source being x_0 and destination x_n . Clearly, the algorithm cannot wait until time instant $2n$ otherwise the backward cost would be at least $2n$, which is two times more than the backward cost of the optimal path T_n . This implies that Algorithm A cannot distinguish between (hence runs exactly in the same way in) graphs G^i , with $i \geq 2n$. Let t_{\max} be the maximum time instant reached by Algorithm A in G^{2n} . Then we can even say that A cannot distinguish between graphs G^i with $i > t_{\max}$.

Claim 1: In G^i , with $i > t_{\max}$, Algorithm A outputs a travel with a backward-cost of at least $n + t_{\max} - 2$

Proof of the Claim: The travel $T = A(G^i)$ that A outputs must contain the same temporal edges as T_n because those are the only edges that exist before time i (recall that $t_{\max} < i$). Let t_j be the time instant reached by Algorithm A at node x_j , for all $j = 0, \dots, n$. Since at each node the travel either arrives at time 2 or leaves at time 2, then $\forall j \in [0..n], t_j \geq 2$.

To move from node x_j to node x_{j+1} the travel T includes a backward trip of cost t_j , if $j \equiv 1 \pmod{2}$, and of cost $t_j - 2$, otherwise. Let $t_{j_{\max}} = t_{\max} = \max(t_j)$, we have that

$$\text{cost}(T) = \sum_{j \in \text{Odd}(n)} t_j + \sum_{j \in \text{Even}(n)} t_j - 2 \geq \begin{cases} 2|\text{Odd}(n)| + t_{\max} - 2 & \text{if } j_{\max} \text{ is odd} \\ 2(|\text{Odd}(n)| - 1) + t_{\max} & \text{if } j_{\max} \text{ is even} \end{cases}$$

Where $\text{Even}(n)$, resp. $\text{Odd}(n)$, denotes the set of even, resp. odd, numbers smaller or equal to n . Since $|\text{Odd}(n)| = n/2$, we obtain in both case $\text{cost}(J) \geq n + t_{\max} - 2$

Claim 2: $t_{\max} \leq n - 4$

Proof of the Claim: Since algorithm A has a competitive ratio of $2 - \varepsilon$, then, if it runs in the evolving graph G^{2n} , it must return a path of backward-cost at most

$$(2 - \varepsilon)\text{Cost}(\text{opt}, G^{2n}) = (2 - \varepsilon)n < 2n - 5$$

(recall that $n\varepsilon > 5$), so it cannot reach time instant $n - 3$. Indeed, if the algorithm waits until time instant $t_{\max} \geq n - 3$, then, using the previous claim, the backward-cost of the travel would be at least $n + n - 5$.

Now we run Algorithm A on graph $G^{t_{\max}+1}$. Using Claim 1, we know that A returns a travel of cost at least $n + t_{\max} - 2$. However, in $G^{t_{\max}+1}$, since $t_{\max} + 4 \leq n$ (Claim 2), the optimal travel is $T_{t_{\max}+1}$ having a cost of $t_{\max} + 1$. We obtain the following inequality:

$$\begin{aligned} \text{cost}(A(G^{t_{\max}+1})) &\geq n + t_{\max} - 2 \geq t_{\max} + 4 + t_{\max} - 2 \geq 2(t_{\max} + 1) \\ &\geq 2\text{Cost}(\text{opt}, G^{t_{\max}+1}) \end{aligned}$$

This contradicts the fact that A has a competitive ratio of $2 - \varepsilon$. ◀

Our Algorithm 1 works as follows: the agent remains in the initial node and waits to learn more about the network until it finds a space-time travel to the destination. Since it does not know whether this is an optimal travel, it waits until it is sure that this is the case, and then goes back in time and enjoys its trip. Computing the best space-time travel given a sub-graph is possible in polynomial time using the existing offline algorithm [4]. Our algorithm assumes that the cost function tends to infinity when the input goes to infinity. It is easy to see that this assumption is necessary to achieve a constant competitive ratio. For instance, with a constant cost function that is equal to 1, one can create two indistinguishable (up to time n) dynamic graphs where a travel with cost $n - 2$ exists before time $t = n$. Then, the first graph contains no other travel, and the second contains another travel with cost 1 but requires a path available at time $t \gg n$ arbitrarily large. Not knowing which graph it is put in (at time n , the agent's knowledge about the two graphs is identical), the agent either waits indefinitely to see if the second travel exists, or follows the first travel with a competitive ratio of $n - 2$. A similar argument can be constructed when the cost function is upper bounded by some number C when its input goes to infinity.

► **Theorem 12.** *For any $f \in \mathcal{UF}$ that diverges to infinity, Algorithm 1 is a T -online algorithm with a competitive ratio of 2.*

Proof. Let T_{\max} be the final value of the variable in our algorithm, so it is the space-time travel used by the agent after the agent returns back at time 0. Let

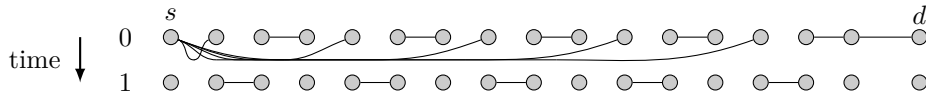
$$t_{\max} = \max\{t \mid f(t+1) < \text{cost}(T_{\max})\}.$$

■ **Algorithm 1** T-Online Algorithm.

Input :
 G' : the known evolving graph
 $time$: the current time
 u : the current node (here u is always the starting node s)

Let T_{max} be an optimal space-time travel in G' , if it exists, starting at time 0, from node s to node d .

if T_{max} does not exist **or** $f(time + 1) < cost(T_{max})$ **then**
 | wait 1 time instant;
else
 | go back at time 0, then follow the travel T_{max} ;



■ **Figure 6** Evolving graph G used to prove that an S-online algorithm has a competitive ratio of at least $2n/3 - 7/3$.

First, we prove that T_{max} is an optimal offline travel. Indeed, the algorithm reached time t_{max} so all the other travels that are not discovered by our algorithm require temporal edges appearing after time t_{max} , so their backward-costs are at least $f(t_{max} + 1)$, which is at least $cost(T_{max})$ by definition. Hence $cost(T_{max})$ is the optimal backward-cost.

When the algorithm terminates, the travel T that is returned is $((s, 0), (s, t_{max}), (s, 0)) \oplus T_{max}$. It has a backward-cost of $f(t_{max}) + cost(T_{max})$. Since $f(t_{max}) \leq cost(T_{max})$, the Lemma is proved. ◀

■ **5 S-Online Algorithm**

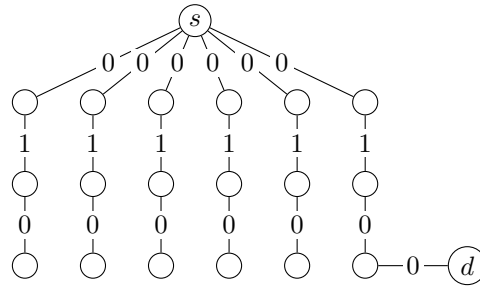
In this section, we study the S-online setting, where an agent knows only about the nodes they have explored. In this case, we show a lower bound of $2n/3 - 7/3$ for the competitive ratio, even when the cost function is the identity. We then present an algorithm that has a competitive ratio of $2n - 3$ when the cost function is linear.

► **Theorem 13.** *Assuming a cost function is $f : x \mapsto x$, an S-online algorithm cannot have a competitive ratio smaller than $2n/3 - 7/3$, where n denotes the number of nodes in the graph.*

Proof. Assume for the purpose of contradiction that there exists an S-online algorithm A with a competitive ratio c .

Consider an evolving graph G consisting of k paths of length 3 having one node s in common and append a node d to one of the paths, as illustrated in Figures 6 and 7. Links connecting nodes at hop-distance 1 from s and nodes at hop-distance 2 from s appear at time 1, and all the other links appear at time 0. The number of nodes is $n = 3k + 2$.

An agent traveling in this graph initially knows about all the neighbors of s does not know their neighbors. Since all the edges connected to s appear at the same time 0, they are indistinguishable. We can consider without loss of generality that the branches are explored from left to right, direction according to Figure 7 (a branch is explored when the node at hop-distance 3 is visited). Visiting a branch costs $f(1)$, and going back to s also costs $f(1)$, so when the agent finally visits the last branch that is connected to d , they have paid $2(k - 1) + 1$.



■ **Figure 7** The same evolving graph G shown in Figure 6 as a static graph where the label represents the time where the edge is present.

In the end the travel costs

$$2(k - 1) + 1 = 2k - 1 = \frac{2(n - 2)}{3} - 1 = 2n/3 - 7/3$$

while the optimal travel costs 1. So the competitive ratio is $2n/3 - 7/3$. ◀

One important question is whether or not the lower bound is higher when assuming a cost function f that is not linear. Interestingly, one can observe that, to create a worst-case using non-linear functions, one must use longer paths because the travel cost of a two-hop travel is the same in one way and the other, so longer travels are required to create travels having higher cost. Moreover, following k smaller backward travels costs at most k times the equivalent single but larger backward travel. So we conjecture that the lower bound is the same for any cost function in \mathcal{UC} .

We now present our Algorithm 2. At a given step, the agent is located at a node u at time t and knows a subgraph G' of G . Among the known nodes, some are not yet explored, called $V_{unexplored}$. For a node v in $V_{unexplored}$, the agent does not know its entire neighborhood. In particular, it does not know if it is connected to the destination d . Indeed, d is either unknown or unexplored (otherwise the agent has already reached the destination). The goal of the agent is to find a travel towards the destination that is not too expensive. A possible travel to d is either in G' or goes through a node in $V_{unexplored}$. A travel in G to d that is not in G' must go through a node $v \in V_{unexplored}$, so its cost is at least the cost of a travel towards v at time 0. So the main idea of the algorithm is to explore nodes one by one starting from the one that could potentially be an optimal travel to the destination d i.e., the agent visits first a node $v \in V_{unexplored}$ whose travel from s to v at time 0 is minimal.

To illustrate a step of the algorithm, consider Figure 4. The agent at position u does not know any travel towards d , but it knows that a travel must either go through x_4 or x_5 . In the best case, if it goes through x_4 , a travel to d costs at least 3 (it costs 2 to reach x_4 using the edge (x_3, x_4) at time 1 plus at best a backward travel to 0 if d is directly connected to x_4 . If it goes through x_5 , a travel to d costs at least 4. So in this situation, the agent travels towards x_4 . When it reaches x_4 , the agent realizes that it is not connected to d and that a travel to d passing through x_4 costs at least 5 (the green travel in Figure 1) so the agent decides to explore next x_5 . Then it explores x_6 and finally $d = x_7$.

► **Theorem 14.** *Assuming the cost function is linear, Algorithm 2 is an S -online algorithm with a competitive ratio of $2n - 3$, where n denotes the number of nodes. Assuming non-linear $f \in \mathcal{UF}$, the competitive ratio of Algorithm 2 is at most n^2 .*

Proof. Consider first that the cost function is linear. After each iteration of the algorithm, at least one new node v is explored.

■ **Algorithm 2** S-Online Algorithm.

Input :
 G' : the known evolving graph
 $time$: the current time
 u : the current node
Let $V_{unexplored}$ be the set of nodes known but unexplored;
Let $H = \{T_v = \mathcal{T}_{G'}((s, 0), (v, 0)) | v \in V_{unexplored}\}$;
/* Recall that $\mathcal{T}_{G'}((s, 0), (v, 0))$ is the set of travels from s to v arriving and departing at time 0 */
Let T_v be a space-time travel in H with minimum cost
Follow an optimal space-time travel towards $(v, 0)$ in G' , from the current location u .

The cost of the travel from s to v arriving at time 0 is at most the cost $Cost(opt, G)$ of the optimal travel from s to d arriving at time 0. Indeed, if d is unknown, all travels goes through at least one unexplored node, meaning that the cost to reach d is higher or equal than the cost to reach v . If d is known, the travels towards d are included in the set H .

Note that when traveling back to s at time 0 from v at time 0, backward time jumps become waiting, and vice-versa. Hence, an outbound trip requiring several, small waits and a large backward-time jump translate in a return trip with a large wait and several small backward-time jumps, which, intuitively, may lead to different costs.

However, because we first assume the cost function to be linear (i.e. $f(a+b) = f(a) + f(b)$), the cost of the outbound trip is the same as the cost of the return trip. Thus, to visit the next unexplored node v' , the agent has to, in the worst case, go back to s and then travel to v' , incurring a cost of $2Cost(opt, G)$. In the worst case, the destination d is the last visited node. The total cost is at most $2(n-2)Cost(opt, G)$ for the first $n-2$ nodes (all except s and d) plus the last travel towards d , i.e., at most $2(n-2)Cost(opt, G) + Cost(opt, G)$ and the competitive ratio is $2n-3$.

Now, suppose the cost function $f \in \mathcal{UF}$ is non-linear. In this case, the costs of the outbound and return trip may be different.

Let T be a k -hop travel from s to a node v , arriving and departing at time 0. Let T_r be the associated returned trip. Observe that the sum of the amplitude (denoted Δ in the following) of all backward-time jumps is the same in both directions (i.e., for T and T_r). Also, it is clear that each backward jump performed in T_r has amplitude smaller than Δ and since f is non-decreasing, each of these jumps costs at most $f(\Delta)$. Since there are at most $k \leq n$ backwards jumps in T_r , we know that, $cost(T_r) \leq k \times f(\Delta)$. As f is also sub-additive, we know that $f(\Delta) \leq cost(T)$. Thus, we have $cost(T_r) \leq k \times cost(T)$.

Hence, using the same proof as in the previous case, we obtain in the worst case, a cost of at most $n \times Cost(opt, G)$ to explore the next node v , which results in a total cost of at most $n^2 Cost(opt, G)$. ◀

6 Discussion and Open Problems

One may notice that the strategies used in Algorithm 2 are similar to some works related to online static graph exploration and treasure hunting. These similarities raise interesting questions regarding possible relations between these problems and possible applications of existing works to our novel model.

Studying this question is challenging due to the many different existing models. For instance, many papers about treasure hunting in static graphs assume that the agent located at a node does not learn about the identifier of the neighboring nodes, but only about the outgoing edges [3, 1]. This results in bounds based on the number of edges, while our algorithm performance only depends on the number of nodes.

One paper by Komm et al. [18] considers that an agent learns about the identifier of its neighbors (a model dubbed *fixed graph*). Interestingly, our models align with theirs under the assumption of a linear cost function. We, however, give a more refined bound about the competitive ratio, implying perhaps a more precise bound for the model given by Komm et al. [18] (which briefly mentions an asymptotically linear competitive ratio before focusing on the impact of advice fed to the agent).

When assuming a *non-linear* cost function, our problem seems to exhibit similarities with the treasure-hunting problem in directed graphs when agents see the neighboring nodes (a problem that, to our knowledge, is unexplored in the literature).

Our work in the S-online setting can thus be seen as the first generalization of the treasure-hunting problem in dynamic graphs. It is interesting to see that such generalizations bear similarities to their static counterparts. However, these outcomes depend on the agent's ability to engage in time travel. In the absence of such capabilities, certain assumptions about the graph may be needed to make up for the absence of backward time travel. For instance, one might consider assumptions such as periodicity, or the presence of bounded-recurrent edges, allowing for the traversal of disappearing edges by waiting for their reappearance, instead of traveling back in time. The study of relations between the settings studied in this paper and more general ones, as well as the study of associated complexity results, are open.

7 Conclusion

We presented the first online solutions to the delay-optimal cost-optimal space-time travel problem in dynamic networks.

We first showed that, when the future is unknown, even assuming an identity cost function, no online algorithm can exhibit a competitive ratio of less than two, and we present a very simple online algorithm with a competitive ratio of two, for a larger class of cost functions.

Then, when the graph itself is unknown and has to be explored to gain connectivity knowledge, we showed that there exists a linear (in the size of the graph) lower bound on the competitive ratio, even when the cost function is the identity, and we present an algorithm with a linear (in the size of the graph) competitive ratio assuming any linear cost function. Refining the constants between our lower and upper bound is left for future work.

References

- 1 Adri Bhattacharya, Barun Gorain, and Partha Sarathi Mandal. Treasure hunt in graph using pebbles. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pages 99–113. Springer, 2022.
- 2 Alexander Birx, Yann Disser, Alexander V Hopp, and Christina Karousatou. An improved lower bound for competitive graph exploration. *Theoretical Computer Science*, 868:65–86, 2021.
- 3 Sébastien Bouchard, Yoann Dieudonné, Arnaud Labourel, and Andrzej Pelc. Almost-optimal deterministic treasure hunt in unweighted graphs. *ACM Transactions on Algorithms*, 19(3):1–32, 2023.
- 4 Quentin Bramas, Jean-Romain Luttringer, and Sébastien Tixeuil. Offline constrained backward time travel planning. In Shlomi Dolev and Baruch Schieber, editors, *Stabilization, Safety, and*

- Security of Distributed Systems - 25th International Symposium, SSS 2023, Jersey City, NJ, USA, October 2-4, 2023, Proceedings*, volume 14310 of *Lecture Notes in Computer Science*, pages 466–480. Springer, 2023. doi:10.1007/978-3-031-44274-2_35.
- 5 Sebastian Brandt, Klaus-Tycho Foerster, Jonathan Maurer, and Roger Wattenhofer. Online graph exploration on a restricted graph class: Optimal solutions for tadpole graphs. *Theoretical Computer Science*, 839:176–185, 2020.
 - 6 Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. Shortest, fastest, and foremost broadcast in dynamic networks. *Int. J. Found. Comput. Sci.*, 26(4):499–522, 2015. doi:10.1142/S0129054115500288.
 - 7 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *Int. J. Parallel Emergent Distributed Syst.*, 27(5):387–408, 2012. doi:10.1080/17445760.2012.668546.
 - 8 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021. doi:10.1007/s00453-021-00831-w.
 - 9 Arnaud Casteigts, Joseph G. Peters, and Jason Schoeters. Temporal cliques admit sparse spanners. *J. Comput. Syst. Sci.*, 121:1–17, 2021. doi:10.1016/j.jcss.2021.04.004.
 - 10 Shigang Chen and Klara Nahrstedt. An overview of qos routing for the next generation high-speed networks: Problems and solutions. *Network, IEEE*, 12:64–79, December 1998. doi:10.1109/65.752646.
 - 11 Dariusz Dereniowski, Yann Disser, Adrian Kosowski, Dominik Pająk, and Przemysław Uznański. Fast collaborative graph exploration. *Information and Computation*, 243:37–49, 2015.
 - 12 Yann Disser, Frank Mousset, Andreas Noever, Nemanja Škorić, and Angelika Steger. A general lower bound for collaborative tree exploration. *Theoretical Computer Science*, 811:70–78, 2020.
 - 13 Afonso Ferreira. On models and algorithms for dynamic communication networks: The case for evolving graphs. In *Quatrièmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (ALGOTEL 2002)*, pages 155–161, Mèze, France, May 2002. INRIA Press.
 - 14 Klaus-Tycho Foerster and Roger Wattenhofer. Lower and upper competitive bounds for online directed graph exploration. *Theoretical Computer Science*, 655:15–29, 2016.
 - 15 Robin Fritsch. Online graph exploration on trees, unicyclic graphs and cactus graphs. *Information Processing Letters*, 168:106096, 2021.
 - 16 Rosario G. Garroppo, Stefano Giordano, and Luca Tavanti. A survey on multi-constrained optimal path computation: Exact and approximate algorithms. *Computer Networks*, 54(17):3081–3107, December 2010. doi:10.1016/j.comnet.2010.05.017.
 - 17 Jochen W. Guck, Amaury Van Bemten, Martin Reisslein, and Wolfgang Kellerer. Unicast qos routing algorithms for sdn: A comprehensive survey and performance evaluation. *IEEE Communications Surveys & Tutorials*, 20(1):388–415, 2018. doi:10.1109/COMST.2017.2749760.
 - 18 Dennis Komm, Rastislav Královič, Richard Královič, and Jasmin Smula. Treasure hunt with advice. In Christian Scheideler, editor, *Structural Information and Communication Complexity*, pages 328–341, Cham, 2015. Springer International Publishing.
 - 19 Giuseppe Antonio Di Luna, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Black hole search in dynamic rings. In *41st IEEE International Conference on Distributed Computing Systems, ICDCS 2021, Washington DC, USA, July 7-10, 2021*, pages 987–997. IEEE, 2021. doi:10.1109/ICDCS51616.2021.00098.
 - 20 Juan Villacis-Llobet, Binh-Minh Bui-Xuan, and Maria Potop-Butucaru. Foremost non-stop journey arrival in linear time. In Merav Parter, editor, *Structural Information and Communication Complexity - 29th International Colloquium, SIROCCO 2022, Paderborn, Germany, June 27-29, 2022, Proceedings*, volume 13298 of *Lecture Notes in Computer Science*, pages 283–301. Springer, 2022. doi:10.1007/978-3-031-09993-9_16.

On Inefficiently Connecting Temporal Networks

Esteban Christiann ✉

École normale supérieure Paris Saclay, 91190 Gif-sur-Yvette, France

Eric Sanlaville ✉ 🏠 

Université Le Havre Normandie, Univ Rouen Normandie, INSA Rouen Normandie, Normandie Univ, LITIS UR 4108, F-76600 Le Havre, France

Jason Schoeters ✉ 🏠 

University of Cambridge, United Kingdom

Abstract

A temporal graph can be represented by a graph with an edge labelling, such that an edge is present in the network if and only if the edge is assigned the corresponding time label. A journey is a labelled path in a temporal graph such that labels on successive edges of the path are increasing, and if all vertices admit journeys to all other vertices, the temporal graph is temporally connected. A temporal spanner is a sublabelling of the temporal graph such that temporal connectivity is maintained. The study of temporal spanners has raised interest since the early 2000's. Essentially two types of studies have been conducted: the positive side where families of temporal graphs are shown to (deterministically or stochastically) admit sparse temporal spanners, and the negative side where constructions of temporal graphs with no sparse spanners are of importance. Often such studies considered temporal graphs with happy or simple labellings, which associate exactly one label per edge. In this paper, we focus on the negative side and consider proper labellings, where multiple labels per edge are allowed. More precisely, we aim to construct dense temporally connected graphs such that all labels are necessary for temporal connectivity. Our contributions are multiple: we present exact or asymptotically tight results for basic graph families, which are then extended to larger graph families; an extension of an efficient temporal graph labelling generator; and overall denser labellings than previous work, whether it be global or local density.

2012 ACM Subject Classification Networks

Keywords and phrases Network design principles, Network dynamics, Paths and connectivity problems, Branch-and-bound

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.8

Related Version *Full Version*: <https://arxiv.org/abs/2312.07117>

Supplementary Material *Software (Source Code)*: <https://gitlab.com/echrstnn/PTGen>
archived at `swh:1:dir:a40ecef5b1d1554ed3ee9c6aed8fad4bbdf43a9`

Funding *Eric Sanlaville*: project ANR-22-CE48-0001 (TEMPOGRAL)

Jason Schoeters: DyNet RIN Tremplin Région Normandie 2020-2022 & Leverhulme Trust International Professorship in Neuroeconomics

Acknowledgements We thank the referees for their careful reading and constructive comments which significantly improved the presentation of these results.

1 Introduction

A temporal graph is a graph which can evolve over time, through the appearing and/or disappearing of edges. Numerous classical graph problems and parameters have been extended to temporal graphs, such as colouring, connected components, maximum matchings, and independent sets [19, 28, 30, 33]. In temporal graphs, connectivity may become very poor when considering the graph at every distinct time step, but the graph may still



© Esteban Christiann, Eric Sanlaville, Jason Schoeters;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 8; pp. 8:1–8:19

Leibniz International Proceedings in Informatics



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

be connected when considering connectivity over time. Indeed, temporal connectivity is motivated through many contexts in which temporal graphs naturally arise, most notably the context of swarms of mobile entities with distance-based communication capabilities (drone networks, insect colonies, and, particularly useful during the COVID-19 pandemic: people) [12, 14, 15, 21]. This temporal connectivity has since redefined classical connectivity problems, such as (temporal) dominating sets, and (temporally) connected components, and particularly interesting concerning this paper: (temporal) spanners [4, 6, 10, 24].

After presenting a wide range of interesting changes and results concerning typical graph problems with temporal paths instead of paths, Kempe, Kleinberg, and Kumar discuss further interesting questions [20]. One of these is whether a temporally connected graph can always be sparsified (that is, if labels can be removed) so as to obtain a “sparse” remaining structure maintaining temporal connectivity. Such a structure is later called a temporal spanner. Note that the static graph analogue would be asking whether a connected graph always admits a spanning tree, which is of course always the case. They follow up with a preliminary negative result, stating that some temporal graphs do not admit a linear size spanner (hypercube graphs with each edge labelled with the corresponding dimension). The real question then became whether dense temporal graphs could always admit a sparse spanner, the intuition being that there exists many more ways to potentially sparsify a dense graph. The question remained open for many years, until Axiotis and Fotakis answered in the negative: they construct a non-trivial dense temporal graph in which some labels may be removed but prove that a dense part has to remain to ensure temporal connectivity. [2] A couple of years afterwards, a complementing positive result was presented by Casteigts, Peters, and Schoeters: any temporal complete graph always admits a sparse spanner [10]. Following these, more papers surfaced related to temporal spanners: sharp thresholds on the density of random temporal graphs to asymptotically almost surely admit particular sparse spanners; positive and negative results regarding spanners which have a limited stretch, as well as on temporal spanners which are blackout-resistant [5, 6, 11].

Another topic of interest in temporal graph theory is that of temporal network design, where instead of analysing a given temporal graph, one would like to design a temporal graph with some desired property or decide such a temporal graph does not exist. In most works on temporal network design, the graph itself is given and a corresponding labelling needs to be constructed. One of the earliest such design problems was to create a gossip protocol, that is, a schedule of pairwise communications between n agents, each having some piece of information which can be transferred over successive communications, such that at the end of the schedule, all agents are up to date with all the information. It is natural to minimise the number of communications (*e.g.* the total cost of phone calls), and thus some tight results arise with protocols using $2n - 3$ communications, with the idea being to gather all information to some agent and then broadcast the information out again, designing essentially a temporal in-tree and a temporal out-tree *resp.* For more information, see survey [18]. More recently in [29], Mertzios *et al.* reconsider and extend this work as a temporal graph design problem. Direct results from gossiping apply, but more importantly, they include other restrictions on the labelling, such as a maximum lifetime *i.e.* the labels cannot be greater than some value, which was further investigated in [22]. Also, two measures of density, both of interest for this paper, are defined regarding a temporal graph: the *temporal cost*, being the total amount of labels; and the *temporality*, being the maximum amount of labels on some edge. The former is a global density measure, and the latter a local one.

In this paper, we combine the study of temporal spanners and of temporal graph design, by designing dense temporal graphs such that each label is necessary for temporal connectivity. As opposed to most previous work we will not restrict ourselves to happy or simple labellings

Measure \ Graph class	Any class	Trees (Connected)	Cycles (Hamiltonian)	Cacti (Circumference c)
Maximum temporal cost T^+	$\leq n^2 - n - 1$ Theorem 2	$2n - 3$ Theorem 5	$\geq \frac{1}{4}n^2 + \frac{3}{4}$ Theorem 7	$\geq \frac{1}{4}c^2 + 2(n - c)$ omitted (see [13])
Maximum temporality τ^+	$\leq n - 1$ Theorem 2	2 Theorem 4	$\geq \lceil \frac{1}{2}n \rceil$ Theorem 8	$\geq \lceil \frac{1}{2}c \rceil$ omitted (see [13])

■ **Figure 1** Main results of our density measures on specific classes of graphs: \leq indicates an upper bound, \geq indicates a lower bound. Results for **Trees**, and by extension **Connected** graphs, are tight.

(one label per edge) but instead extend to consider proper labellings (multiple labels per edge allowed). This is a double-edged sword: on the one hand this intuitively may allow for much denser labellings, but on the other hand, a combinatorial explosion on the amount of possible labellings occurs implying algorithmics may be more difficult in this setting. In a sense, we are interested in designing the most inefficient temporal networks possible. Outside of the already established applications of temporal spanners in related work, the negative results in particular can have direct implications concerning adversarial behaviour in temporal network game theory and the potential waste of temporal and structural resources [27, 32]. Lastly, a slowly temporally connected network may allow for time to detect any anomalies/viruses before the whole network is infected, while not hindering the supposedly essential connectivity of the network, and may have applications for fraud detection in financial transactions [31].

In short, throughout the paper, we will steadily answer both of the following questions:

- What is the densest temporal graph overall?
- Given a graph class, what is the densest labelling all graphs can attain?

1.1 Contributions

First, in Section 2, we give standard graph theory and temporal graph theory notation and define our setting as well as a global and a local density measure: maximum temporal cost T^+ and maximum temporality τ^+ respectively. Lower bounds from the literature and upper bounds through analysis are presented. Then, in Section 3, we focus on tree graphs for which we obtain tight results through an argument on bridge edges. These results do not beat aforementioned lower bounds however. In Section 4, we focus on cycles, partly due to a lower bound being a labelling on cycles, which shows promise for obtaining even denser labellings. For this, we decide to extend labelling generator STGen from [7] so as to fit to our setting and specifically to cycles. After executing it on small cycles, we obtain the intuition for a complex labelling which beats the lower bounds for local density by a factor of 1.5, and for global density by exactly 1 label. A non-trivial proof is given to show that all labels of this labelling are indeed necessary and that the resulting temporal graph is temporally connected, using a representation of temporal graphs called link streams and by reasoning on journeys which are necessary. A summary of our results is presented in Figure 1, where n is the number of vertices of a graph. We discuss and extend results in Section 5, and conclude.

Due to the page limit, technical proofs (marked with a \star) were moved to the appendix and other parts were omitted. For these, the reader is referred to the full version on arXiv [13].

2 Preliminaries

In this paper, all graphs are simple and undirected (except for the reachability graph defined below). A temporal graph is a tuple (G, λ) with graph $G = (V, E)$, called the footprint or underlying graph, and edge labelling $\lambda : E \rightarrow 2^{\mathbb{N}}$. The labels correspond to when the edges

8:4 On Inefficiently Connecting Temporal Networks

are present over the lifetime of the temporal graph. A pair (e, ℓ) with $e \in E$ and $\ell \in \lambda(e)$ is called a time edge. Reachability in temporal graphs is defined through temporal paths, also called journeys, which are adjacent time edges $j = (c_1, c_2, \dots, c_k)$ such that for all $c_i = (e_i, \ell_i)$ with $i \in [2, k]$ we have that $\ell_i > \ell_{i-1}$. We say u can reach v , or v can be reached by u , if there exists a journey from u to v . A journey \mathcal{J} is said to cover a set of vertices V' if for all vertices v in V' , v is part of some time edge of \mathcal{J} . A temporal graph $\mathcal{G}' = (G', \lambda')$ is a temporal subgraph of $\mathcal{G} = (G, \lambda)$ if G' is a subgraph of G and λ' a sublabelling of λ . For a label ℓ of temporal graph \mathcal{G} , $\mathcal{G}^{-\ell}$ corresponds to the temporal subgraph of \mathcal{G} without label ℓ (if other labels exist in \mathcal{G} with the same value, then these remain).

A temporal branching $\mathcal{B} = (T, \lambda)$ with root r is a tree T with $|\lambda| = n - 1$ such that vertex r can reach all vertices. A temporal branching $\mathcal{B} = (T, \lambda')$ with root v of a temporal graph $\mathcal{G} = (G, \lambda)$ is a temporal subgraph of \mathcal{G} which is a temporal branching, and it is spanning if $V(T) = V(G)$. The reachability graph $R(\mathcal{G})$ is defined on the same vertex set as \mathcal{G} and an arc exists from u to v if and only if u can reach v in \mathcal{G} . If all vertices can reach all other vertices in \mathcal{G} , we say \mathcal{G} is temporally connected. Note that a temporal graph is temporally connected if and only if the corresponding reachability graph is complete (with arcs in both directions). From [8], two temporal graphs \mathcal{G}_1 and \mathcal{G}_2 are reachability-equivalent¹ if reachability graphs $R(\mathcal{G}_1)$ and $R(\mathcal{G}_2)$ are isomorphic, denoted $\mathcal{G}_1 \stackrel{R}{\simeq} \mathcal{G}_2$.

A labelling is proper when no incident edges share a same label. In the rest of the paper, all labellings are supposed proper, unless specifically stated otherwise. Using terms from [1], a label ℓ in a temporal graph \mathcal{G} is redundant if and only if it can be removed from \mathcal{G} without reducing reachability, *i.e.* $\mathcal{G} \stackrel{R}{\simeq} \mathcal{G}^{-\ell}$. Conversely, a label ℓ of \mathcal{G} is necessary if and only if $\mathcal{G} \not\stackrel{R}{\simeq} \mathcal{G}^{-\ell}$. If a labelling contains only necessary labels, we call it a minimal labelling. We call a temporal graph with a proper (*resp.* minimal) labelling a proper (*resp.* minimal) temporal graph.

In [29], the authors defined two measures of density for a temporal graph \mathcal{G} : the temporal cost $T(\mathcal{G})$, which is the total amount of labels in \mathcal{G} ; and the temporality $\tau(\mathcal{G})$ which is the maximum amount of labels on an edge, among all edges. The former is intended as a global density measure, whereas the latter is more of a local one, potentially of interest for example in distributed or parallel computing. We adapt temporal cost in the following manner. The three types of maximum temporality are defined analogously.

- Let $T^+(G)$ be the maximum temporal cost of graph G , *i.e.* the maximum temporal cost $T(\mathcal{G} = (G, \lambda))$ of all proper minimal labellings λ such that \mathcal{G} is temporally connected;
- Let $T^+(\text{Class})$ be the maximum temporal cost of graph class **Class**, *i.e.* the maximum value x such that for all graphs G of **Class**, $T^+(G) \geq x$;
- Let T^+ be the maximum temporal cost, *i.e.* the maximum temporal cost $T^+(G)$ among all graphs G on n vertices.

We study three simple graph classes in this work, being **Trees** (Section 3), **Cycles** (Section 4), and **Cacti** (omitted, see [13]), and in Section 5 superclasses are discussed.

2.1 Upper and lower bounds on T^+ and τ^+

The upper bounds are both obtained through the following fact:

► **Lemma 1.** *A minimal temporally connected graph \mathcal{G} equals the union of any n spanning temporal branchings with distinct roots of \mathcal{G} .*

¹ Originally *closure-equivalent*, but changed to reachability-equivalent for journal version (private message).

Proof. By contradiction, suppose that the union of some n spanning temporal branchings with distinct roots of \mathcal{G} does not equal \mathcal{G} . This implies at least some label of \mathcal{G} isn't part of the branchings, which means it can be removed without reducing reachability. However, \mathcal{G} is minimal so no redundant labels exist which is a contradiction. ◀

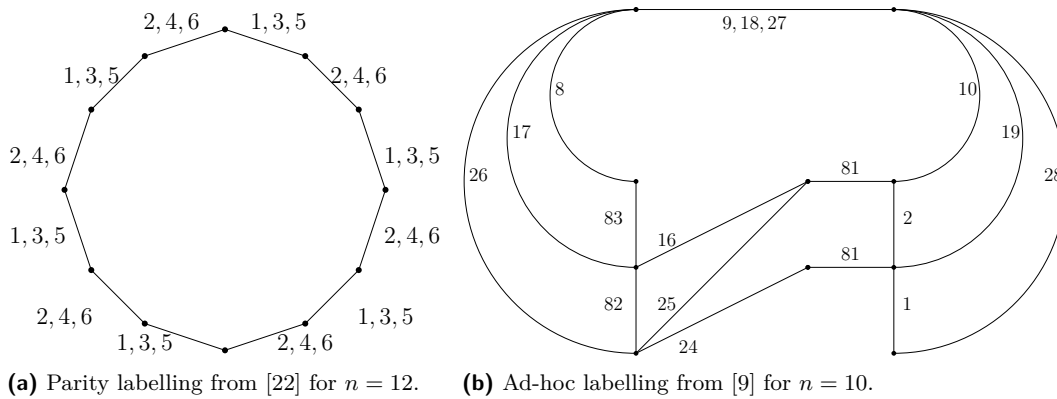
Lemma 1 allows us to reason on minimal temporally connected graphs through the corresponding spanning temporal branchings: for the largest possible maximum temporal cost T^+ , consider the branchings to all be using distinct labels; whereas the branchings all using some same edge with distinct labels would result in the largest maximum temporality τ^+ .

► **Theorem 2** (*). *The maximum temporal cost $T^+ \leq n^2 - n - 1$ and the maximum temporality $\tau^+ \leq n - 1$.*

Observe that the idea of considering n root-distinct temporal branchings is used in Observation 3 in [22] as well. We believe that their result could be improved slightly from $n(n - 1)$ to $n(n - 1) - 1$ in a similar fashion as we do for the maximum temporal cost T^+ .

Also in [22], Klobas et al. construct some minimal labellings which are strict, meaning journeys are allowed to traverse at most one edge per time step. Among these labellings, the one given in Lemma 4 happens to be proper, giving lower bounds $T^+ \geq \frac{1}{4}n^2$ and $\tau^+ \geq \frac{1}{4}n$. The idea of the labelling is to label every other edge of an even cycle graph with all even labels up to $n/2$, and all other edges with all odd labels up to $n/2$. Let us refer to this labelling as the parity labelling (see Figure 2).

During the open question session of a Dagstuhl seminar on temporal graphs (see [9] for the report), some preliminary results of this work were presented, including a labelling of an ad-hoc graph giving lower bounds $T^+ \geq \frac{1}{18}n^2 + O(n)$ and $\tau^+ \geq \lfloor \frac{1}{5}n \rfloor$. The idea is to force journeys to go through the top edge using distinct labels. Let us refer to this labelling as the ad-hoc labelling (see Figure 2).



■ **Figure 2** Minimal labellings from the literature giving lower bounds on maximum temporal cost T^+ and maximum temporality τ^+ .

Note that the parity labelling is denser regarding the temporal cost, but the ad-hoc labelling is denser regarding temporality. Some natural questions arise from these bounds. It seems that very sparse graphs (such as **Cycles**) can admit dense labellings, does that mean that other sparse graphs, such as **Trees**, can admit dense labellings as well? Can one do better than these labellings, and more specifically better in **Cycles**? These questions are answered in the following sections.

3 Tree graphs

In this section we prove the following labelling is densest possible for tree graphs, considering maximum temporal cost T^+ and maximum temporality τ^+ . The labelling originates from gossiping strategies from e.g. [3, 17] and has been used in temporal graph theory papers such as in Theorem 2 in [1] and the pivot technique in [10]. We refer to it as the pivot labelling, and define it as follows (see also Figure 3). Select an arbitrary pivot vertex p and construct journeys from all other vertices towards p , using the reverse breadth-first search order. Then, using the breadth-first search order, add journeys from p to all other vertices. The earliest label of the second BFS is removed.

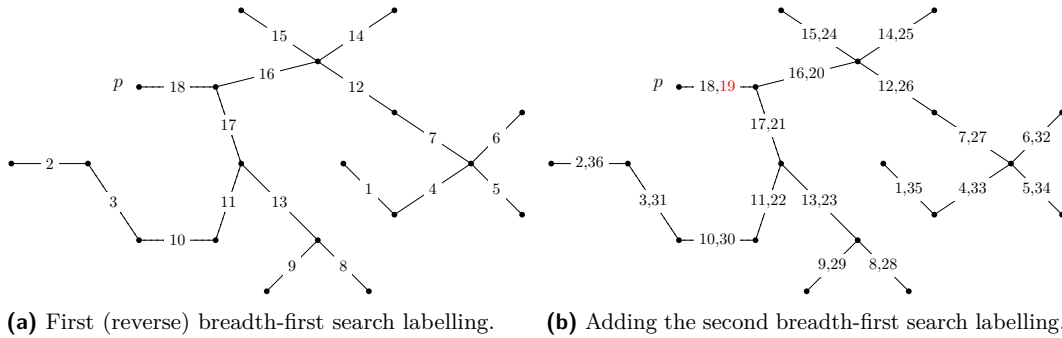


Figure 3 The pivot labelling of an example tree graph. A first labelling converging to pivot vertex p is shown, which is then complemented by a second broadcasting labelling from p . Label 19 (shown in red) is redundant and removed.

The resulting temporal graph is temporally connected since by design all vertices can reach pivot vertex p at time $n - 1$, and starting at time $n - 1$, vertex p can reach all vertices. It is also a minimal labelling since removing any label $\leq n - 1$ on a path from a leaf vertex f to p reduces the reachability of f , and removing any label $> n - 1$ makes it so f cannot be reached by some other leaf vertex.

As stated by Theorem 2(a) in [1], the pivot labelling thus trivially gives the lower bound $T^+(\text{Trees}) \geq 2n - 3$. Also, it trivially gives the lower bound of $\tau^+(\text{Trees}) \geq 2$. To prove these are tight, *i.e.* no denser labellings exist, we first present a lemma focusing on bridge edges (edges which disconnect the graph if removed), and then apply it on tree graphs.

► **Lemma 3.** *For any bridge edge e of graph G and any minimal labelling λ such that $\mathcal{G} = (G, \lambda)$ is temporally connected, λ can assign at most two labels to e .*

Proof. Consider a temporally connected graph \mathcal{G} with bridge edge $e = \{u, v\}$, separating \mathcal{G} into two temporal subgraphs \mathcal{G}_1 (with vertex u) and \mathcal{G}_2 (with v). Suppose by contradiction that the labelling λ of \mathcal{G} assigns more than two labels to edge e , say $k > 2$ labels $\ell_1, \ell_2, \dots, \ell_k$, and that this labelling is minimal. Define t_u^- to be the earliest time at which all vertices in \mathcal{G}_1 are able to reach u . Similarly, define t_v^+ to be the latest time at which all vertices in \mathcal{G}_2 can be reached by v . Since \mathcal{G} is temporally connected, there exists some label ℓ_i of e such that $t_u^- < \ell_i < t_v^+$. Keeping ℓ_i is thus sufficient for maintaining reachability from all vertices in \mathcal{G}_1 to all vertices in \mathcal{G}_2 . A symmetrical argument can be used to find label ℓ_j which is sufficient for maintaining reachability from all vertices in \mathcal{G}_2 to all vertices in \mathcal{G}_1 . Together, ℓ_i and ℓ_j are thus sufficient for reachability concerning journeys using edge e , and edge e can trivially be ignored for reachability between vertices in \mathcal{G}_1 (*resp.* \mathcal{G}_2). This results in all other labels on edge e being redundant, which is a contradiction since it is minimal. ◀

► **Theorem 4.** $\tau^+(\text{Trees}) = 2$.

Proof. Tree graphs contain only bridge edges, so by Lemma 3 no edge of a tree graph can have more than two labels in a minimal labelling, and this is attained by the pivot labelling. ◀

Note that Theorem 4 implies that $T^+(\text{Trees}) \leq 2n - 2$, which is only off by 1 from the lower bound. We finish this section with proving the latter to be tight.

► **Theorem 5** (*). $T^+(\text{Trees}) = 2n - 3$.

As a side note, we remark that the maximum temporal cost (*resp.* temporality) of tree graphs corresponds to the minimum temporal cost (*resp.* temporality) of tree graphs. In other words, all minimal temporally connected labellings of tree graphs contain exactly $2n - 3$ labels and exactly 2 labels on all edges except one, independently of whether one tries to minimise or maximise the density. For proofs of these minimisation costs, we refer the reader to [18] for the original proofs in the gossiping context, or to Theorem 2 in [22] and Corollary 3 in [29] in the temporal graph context.

For trees, the results are mixed: on one hand we exactly determined both maximum temporal cost and maximum temporality of tree graphs $T^+(\text{Trees})$ and $\tau^+(\text{Trees})$, but on the other hand both are very sparse and do not improve upon lower bounds of maximum temporal cost T^+ and maximum temporality τ^+ .

4 Cycle graphs

We focus here on finding dense labellings of cycle graphs, which we know exist thanks to the parity construction. For this, we decided to adapt temporal graph generator STGen from [7] (the description and adaptation of which is omitted in this version), which ultimately leads us to the densest labelling in this paper, the generator labelling. The main idea is to distribute even labels to an arbitrary edge, odd labels to the incident non-labelled edges, and so forth for the next non-labelled edges switching between even and odd labels with some additional complex rules.

Proving this labelling results in a minimal and temporally connected graph for any order n cycle graph is complex due to the inherent unreadability of the multiple journeys in these temporal graphs. For this reason, we introduce a different type of representation which is very similar to the so-called link stream representation used in various temporal graph theory papers [25, 26, 34], and thus by slight misuse of terminology, we simply refer to it as the link stream representation. Link streams intuitively focus more on the time edge aspect of a temporal graph, and less on the structure of the underlying graph. Since we already know the underlying structure of the graph (being a cycle graph), link streams are perfect to represent our generator labelling. As an illustrative example, the link stream representation of Figure 4 is given in Figure 5, where all edges of the cycle are represented in one dimension, the horizontal dimension, and time is represented in the other, the vertical dimension. A “label” is thus represented as a time edge at the intersection of the corresponding edge and time value.

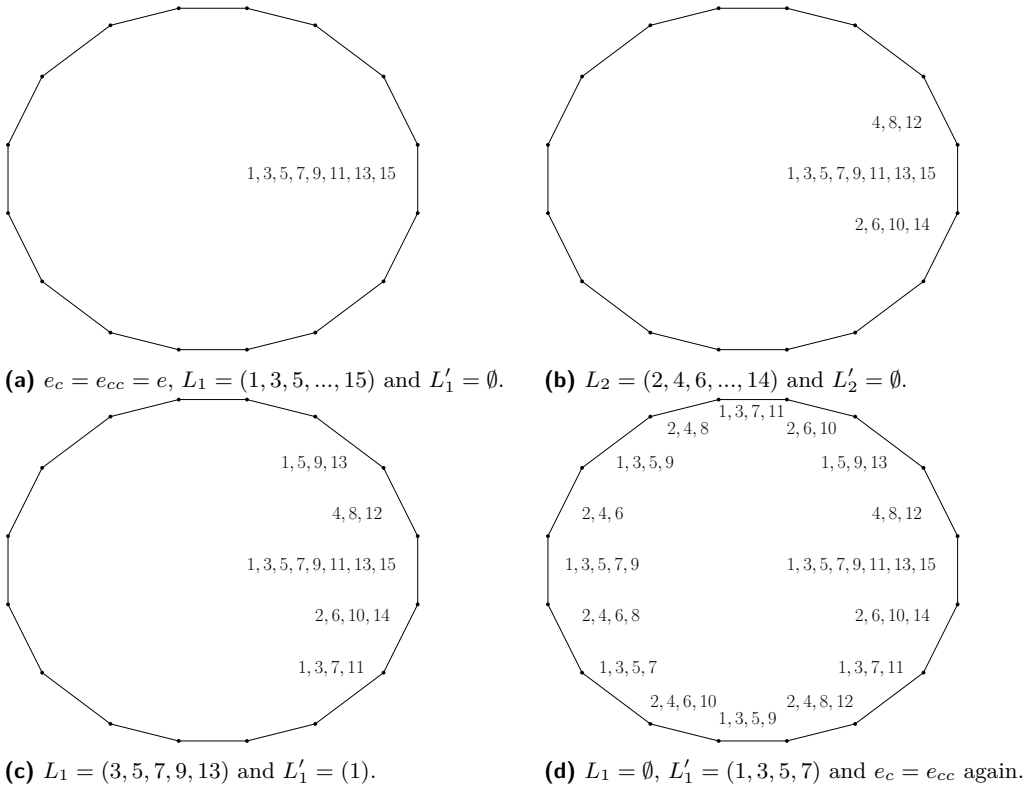
In this representation, a journey informally corresponds to a (possibly steep) “staircase” of time edges which, obeying the flow of time, cannot go down. We now remind and define some concepts concerning journeys in this setting, all of which are illustrated in Figure 5. A prefix of a journey $(v_1, \ell_1, v_2, \ell_2, \dots, \ell_{k-1}, v_k)$ is a part of the journey cut back from the arrival vertex, *i.e.* $(v_1, \ell_1, v_2, \ell_2, \dots, \ell_{i-1}, v_i)$ for some $i \leq k$, and a suffix of a journey is a part

■ **Algorithm 1** Generator labelling (see also Figure 4).

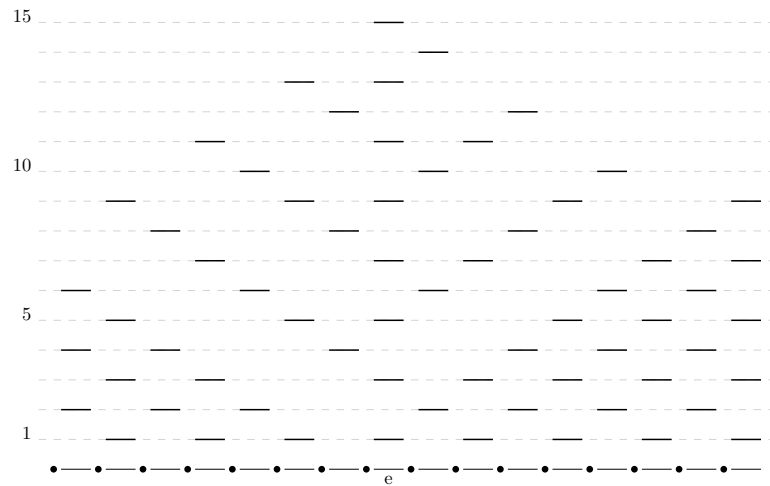
```

input : even cycle graph  $G$  of order  $n$ , edge  $e$  of  $G$ 
output : temporal graph  $\mathcal{G} = (G, \lambda)$  with generator labelling  $\lambda$ 
 $L, L_1 \leftarrow \text{oddNumbersBetween}(1, n - 1)$  /* ascending order */
 $L_2 \leftarrow \text{evenNumbersBetween}(1, n - 1)$  /* ascending order */
 $L', L'_1, L'_2 \leftarrow \emptyset$ 
 $e_c, e_{cc} \leftarrow e$ 
while  $e_c = e$  or  $e_c \neq e_{cc}$  do
    addLabels( $L', e_c$ )
    addLabels( $L', e_{cc}$ )
     $e' \leftarrow e_c$ 
    for  $\ell$  in  $L$  do
        addLabel( $\ell, e'$ )
        if  $e' = e_c$  then  $e' \leftarrow e_{cc}$  else  $e' \leftarrow e_c$ 
    moveSmallestLabelFromTo( $L, L'$ )
    removeLargestLabelFrom( $L$ )
     $e_c \leftarrow \text{nextClockwiseEdge}(e_c)$ 
     $e_{cc} \leftarrow \text{nextCounterClockwiseEdge}(e_{cc})$ 
    if  $L = L_1$  then  $L \leftarrow L_2$  else  $L \leftarrow L_1$ 
    if  $L' = L'_1$  then  $L' \leftarrow L'_2$  else  $L' \leftarrow L'_1$ 
addLabel( $\text{largestLabel}(e_c) + 2, e_c$ )

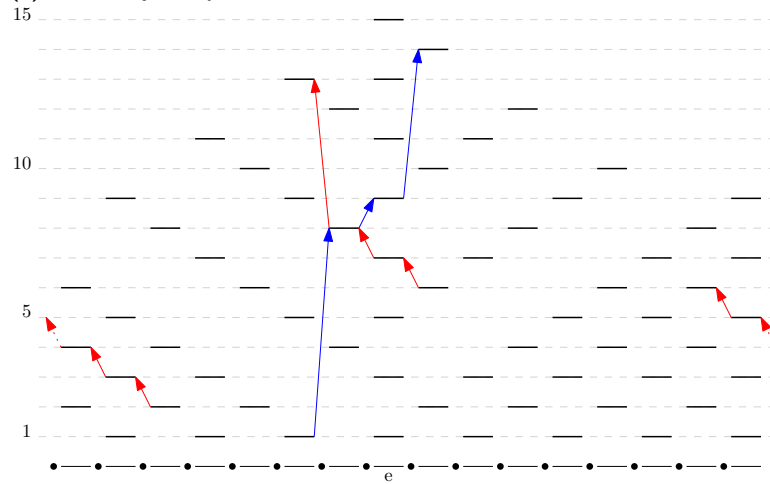
```



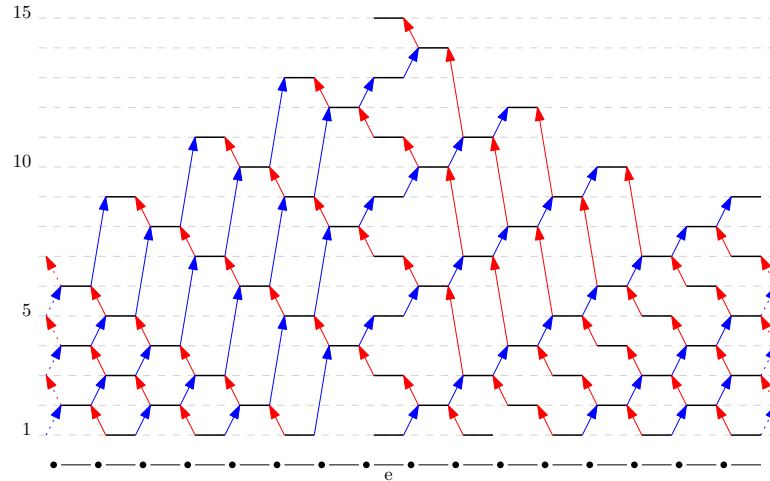
■ **Figure 4** The generator labelling for $n = 16$, starting on rightmost edge e , and repeating **while** loop with lists in captions. After the **while** loop finishes, label 9 is added on the last (leftmost) edge.



(a) Without journeys.



(b) With two clockwise journeys in red and one counter-clockwise journey in blue. Only the outermost red journey (which goes around) is prefix-foremost.



(c) With all dominating journeys.

■ **Figure 5** The generator labelling for $n = 16$ in the link stream representation, with edge e shown in the middle. The rightmost edge connects the outermost vertices, allowing journeys to go around.

8:10 On Inefficiently Connecting Temporal Networks

of the journey cut back from the starting vertex, *i.e.* $(v_j, \ell_j, v_{j+1}, \ell_{j+1}, \dots, \ell_{k-1}, v_k)$ again for some $j \leq k$. A foremost journey from vertex u to vertex v is a journey which arrives at the earliest time among all journeys from u to v . A prefix-f foremost journey from u to v is a journey which prefixes are all foremost, In other words, a prefix-f foremost journey always takes the earliest edges possible on its journey. We define a clockwise journey as a journey which takes only time edges to the left, and a counter-clockwise journey is composed of only time-edges going to the right. Finally, we define dominating journeys as clockwise (*resp.* counter-clockwise) journeys such that no other clockwise (*resp.* counter-clockwise) journey exists which covers all its vertices or more.

We show in a very technical proof by induction that the generator labelling essentially “grows” domination journeys and creates new ones, as it is applied to larger and larger cycles graphs, and that these dominating journeys are necessary and ensure temporal connectivity.

► **Theorem 6** (★). *The generator labelling yields a minimal temporally connected graph.*

The temporal cost of the generator labelling is the largest presented in this paper, and with it also comes the largest temporality, namely on edge e . In the full version of this work, we show that the generator labelling works on all even cycles, and also give an adaptation for odd cycles. Analysing both gives us the main results.

► **Theorem 7.** $T^+(\text{Cycles}) \geq \frac{1}{4}n^2 + \frac{3}{4}$.

► **Theorem 8.** $\tau^+(\text{Cycles}) \geq \lceil \frac{1}{2}n \rceil$.

5 Conclusion

In conclusion, we provided some general upper bounds and proved tight or lower bound results for some basic graph classes on how dense a labelling they can admit. Also, our proposed generator labelling beats the previously densest labellings, in both temporal cost and temporality.

Since we allow a labelling to assign no labels to any edge, a density result for class \mathcal{C} translates as a lower bound for any class \mathcal{C}' if for all graphs $G' \in \mathcal{C}'$, there exists $G \in \mathcal{C}$ such that G is an edge-deleted subgraph of G' . Conversely, a density result for class \mathcal{C} implies an upper bound for any superclasses of \mathcal{C} . Together, this means that our (tight and lower bound) results for **Trees**, **Cycles**, and **Cacti**, thus respectively transfer for **Connected**, the class of connected graphs, **Hamiltonian**, the class of Hamiltonian graphs, and **Circumference c** , the class of graphs of circumference c .

We omitted in this version (again, see full version on arXiv [13]) the section on cactus graphs, where essentially the pivot labelling is used to gather information towards a largest cycle in the graph, instead of towards a pivot vertex, then apply the generator labelling on that cycle, and finally broadcast information outwards again with the pivot labelling, leading to the density results presented in Figure 1.

In terms of future work, one clear option is to consider other types of labellings. There are already some lower bounds known, such as Axiotis and Fotakis’ construction from [2] for happy labellings, although we beat this construction slightly by adapting the ad-hoc labelling to become a happy labelling (omitted in this version). A lower bound for strict labellings comes from [23], in which Klobas et al. label all edges of an odd cycle with all integers up to $\lfloor \frac{n}{2} \rfloor$; another labelling attaining the same lower bound is the complete graph with label 1 on all edges. Another direction for future work is considering computational complexity of associated problems. The corresponding minimisation problems are all polynomial-time

solvable, with the exception of deciding whether the minimum temporality of a graph is 1, which is NP-complete [16]. Our results only prove polynomial-time solvability for trees, and containment in APX for cycles and cacti of large circumference.

References

- 1 Eleni C Akrida, Leszek Gaşieniec, George B Mertzios, and Paul G Spirakis. The complexity of optimal design of temporally connected graphs. *Theory of Computing Systems*, 61(3), 2017.
- 2 Kyriakos Axiotis and Dimitris Fotakis. On the size and the approximability of minimum temporally connected subgraphs. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- 3 Brenda Baker and Robert Shostak. Gossips and telephones. *Discrete Mathematics*, 2:191–193, 1972.
- 4 Stefan Balev, Yoann Pigné, Eric Sanlaville, and Jason Schoeters. Temporally connected components. Available at SSRN 4590651, 2023.
- 5 Davide Bilò, Gianlorenzo D’Angelo, Luciano Gualà, Stefano Leucci, and Mirko Rossi. Sparse temporal spanners with low stretch. In *30th Annual European Symposium on Algorithms (ESA 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.
- 6 Davide Bilò, Gianlorenzo D’Angelo, Luciano Gualà, Stefano Leucci, and Mirko Rossi. Blackout-tolerant temporal spanners. In *International Symposium on Algorithms and Experiments for Wireless Sensor Networks*, pages 31–44. Springer, 2022.
- 7 Arnaud Casteigts. Efficient generation of simple temporal graphs up to isomorphism. *Github repository*, 2020. URL: <https://github.com/acasteigts/STGen>.
- 8 Arnaud Casteigts, Timothée Corsini, and Writika Sarkar. Simple, strict, proper, happy: A study of reachability in temporal graphs. *Theoretical Computer Science*, page 114434, 2024.
- 9 Arnaud Casteigts, Kitty Meeks, George B. Mertzios, and Rolf Niedermeier. Temporal Graphs: Structure, Algorithms, Applications (Dagstuhl Seminar 21171). *Dagstuhl Reports*, 11(3):16–46, 2021. doi:10.4230/DagRep.11.3.16.
- 10 Arnaud Casteigts, Joseph G Peters, and Jason Schoeters. Temporal cliques admit sparse spanners. *Journal of Computer and System Sciences*, 121:1–17, 2021.
- 11 Arnaud Casteigts, Michael Raskin, Malte Renken, and Viktor Zamaraev. Sharp thresholds in random simple temporal graphs. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–326. IEEE, 2022.
- 12 Daniel Charbonneau, Benjamin Blonder, and Anna Dornhaus. Social insects: a model system for network dynamics. *Temporal networks*, pages 217–244, 2013.
- 13 Esteban Christiann, Eric Sanlaville, and Jason Schoeters. On inefficiently connecting temporal networks, 2023. arXiv:2312.07117.
- 14 Nicholas S DiBrita, Khoulood Eledlebi, Hanno Hildmann, Lucas Culley, and Abdel F Isakovic. Temporal graphs and temporal network characteristics for bio-inspired networks during optimization. *Applied Sciences*, 12(3):1315, 2022.
- 15 Jessica Enright, Kitty Meeks, George B Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. In *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- 16 F. Göbel, J.Orestes Cerdeira, and H.J. Veldman. Label-Connected Graphs and the Gossip Problem. *Discrete Mathematics*, 87(1):29–40, January 1991. doi:10.1016/0012-365X(91)90068-D.
- 17 András Hajnal, Eric C Milner, and Endre Szemerédi. A cure for the telephone disease. *Canadian Mathematical Bulletin*, 15(3):447–450, 1972.
- 18 Hovhannes Harutyunyan, Arthur Liestman, Joseph Peters, and Dana Richards. Broadcasting and Gossiping. In Ping Zhang, editor, *Handbook of Graph Theory, Second Edition*, volume

- 20134658, pages 1477–1494. Chapman and Hall/CRC, December 2013. Series Title: Discrete Mathematics and Its Applications. doi:10.1201/b16132-87.
- 19 Danny Hermelin, Yuval Itzhaki, Hendrik Molter, and Rolf Niedermeier. Temporal unit interval independent sets. In *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.
 - 20 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.
 - 21 Arindam Khanda, Federico Corò, Francesco Betti Sorbelli, Cristina M Pinotti, and Sajal K Das. Efficient route selection for drone-based delivery under time-varying dynamics. In *2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, pages 437–445. IEEE, 2021.
 - 22 Nina Klobas, George B Mertzios, Hendrik Molter, and Paul G Spirakis. The complexity of computing optimum labelings for temporal connectivity. In *47th International Symposium on Mathematical Foundations of Computer Science*, 2022.
 - 23 Nina Klobas, George B Mertzios, Hendrik Molter, and Paul G Spirakis. Realizing temporal graphs from fastest travel times. *arXiv preprint arXiv:2302.08860*, 2023.
 - 24 David C Kutner and Laura Larios-Jones. Temporal reachability dominating sets: contagion in temporal graphs. *arXiv preprint arXiv:2306.06999*, 2023.
 - 25 Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining*, 8:1–29, 2018.
 - 26 Jundong Li, Kewei Cheng, Liang Wu, and Huan Liu. Streaming link prediction on dynamic attributed networks. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 369–377, 2018.
 - 27 Fuqiang Liu, Jingbo Tian, Luis Miranda-Moreno, and Lijun Sun. Adversarial danger identification on temporally dynamic graphs. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.
 - 28 Andrea Marino and Ana Silva. Coloring temporal graphs. *Journal of Computer and System Sciences*, 123:171–185, 2022.
 - 29 George B Mertzios, Othon Michail, and Paul G Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019.
 - 30 George B Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. Computing maximum matchings in temporal graphs. In *37th International Symposium on Theoretical Aspects of Computer Science*, 2020.
 - 31 U Rajeshwari and B Sathish Babu. Real-time credit card fraud detection using streaming analytics. In *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, pages 439–444. IEEE, 2016.
 - 32 Lichao Sun, Yingtong Dou, Carl Yang, Kai Zhang, Ji Wang, S Yu Philip, Lifang He, and Bo Li. Adversarial attack and defense on graph data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2022.
 - 33 Mathilde Vernet, Yoann Pigné, and Eric Sanlaville. A study of connectivity on dynamic graphs: computing persistent connected components. *4OR*, pages 1–29, 2022.
 - 34 Peixiang Zhao, Charu Aggarwal, and Gewen He. Link prediction in graph streams. In *2016 IEEE 32nd international conference on data engineering (ICDE)*, pages 553–564. IEEE, 2016.

A Proof(s) of Section 2

► **Theorem 2.** *The maximum temporal cost $T^+ \leq n^2 - n - 1$ and the maximum temporality $\tau^+ \leq n - 1$.*

Proof. By Lemma 1, a minimal temporally connected graph equals the union of any of its n distinct-root spanning temporal branchings.

Concerning maximum temporal cost, the worst-case scenario for the total number of labels in such a graph is when these temporal branchings are all disjoint. This results in a labelling using $n - 1$ labels for each branching (as they are spanning), of which there are n , resulting in a total of $n^2 - n$ labels.

Consider however the smallest label ℓ^- used in the graph, say on edge $e = \{u, v\}$. This label can only be part of the spanning temporal branching of root u , denoted \mathcal{B}_u , or of the spanning temporal branching of root v , denoted \mathcal{B}_v , since it's unreachable from any other vertex. Suppose *w.l.o.g.* ℓ^- is part of \mathcal{B}_u . We know v must reach u through some journey in \mathcal{B}_v arriving at some time ℓ . Note that ℓ can be removed from \mathcal{B}_v , and ℓ^- added. Indeed, for all w , any journey $v \rightsquigarrow w$ in \mathcal{B}_v is either maintained by the swap, or passes through u earlier with ℓ^- , meaning \mathcal{B}_v remains a spanning temporal branching. Thus, label ℓ^- can be considered part of two spanning temporal branchings, decrementing the total amount of labels to $n^2 - n - 1$.

Concerning maximum temporality, the worst-case number of labels on an edge in such a graph is when the spanning temporal branchings are all label-disjoint and all use one same edge $e = \{u, v\}$, resulting in an edge having 1 label for each branching, of which there are n , resulting in a total of n labels.

Note however that the label from the branching corresponding to root u , and the label from branching corresponding to root v , are necessarily the same label, since otherwise the later of the two would be redundant, as both branchings can use the earlier label. Thus edge e would have $n - 1$ labels. ◀

B Proof(s) of Section 3

We first need the following lemma concerning dense path graphs.

► **Lemma 9.** $T^+(\text{Paths}) = 2n - 3$

Proof. The pivot labelling and Lemma 3 apply to path graphs, implying that $2n - 3 \leq T^+(\text{Paths}) \leq 2n - 2$.

Suppose *w.l.o.g.* the vertices of a path graph to be, from one leaf vertex to the other, v_1, v_2 etc. up to v_n . Note that for a path graph to be temporally connected, we only need to ensure that both extremities, v_1 and v_n , can reach each other, via a journey in one direction, and a journey in the other. Indeed, any other pair of vertices can use these journeys to reach each other. Thanks to this, we can reason on temporally connected path graphs and only need to worry about the reachability between these two leaf vertices, instead of between all vertices.

Suppose by contradiction that a minimal temporally connected path graph G exists with $2n - 2$ labels. By our previous argument, we have that any label which is not part of either the journey from v_1 to v_n , or of the journey from v_n to v_1 , is redundant. Each of these two journeys is composed of $n - 1$ labels, meaning that to obtain $2n - 2$ necessary labels from only these two journeys, we must have that all labels on the two journeys must be distinct. There must exist one edge e_i such that the corresponding labels have the smallest difference among all edges of path G . By the temporal nature of journeys, there cannot be more than one such edge as the difference must necessarily increase on edges further away from e_i . Consider the labels on edge e_i and incident edges e_{i-1} and e_{i+1} , denoted $\ell_i^{\rightarrow}, \ell_i^{\leftarrow}, \ell_{i-1}^{\rightarrow}, \ell_{i-1}^{\leftarrow}, \ell_{i+1}^{\rightarrow}$, and ℓ_{i+1}^{\leftarrow} . (If edge e_i only has one incident edge, then ignore the following concerning the non-existent other edge and labels.) Consider labels $\ell_{i-1}^{\rightarrow} < \ell_i^{\rightarrow} < \ell_{i+1}^{\rightarrow}$ to be part of $v_1 \rightsquigarrow v_n$, and $\ell_{i-1}^{\leftarrow} > \ell_i^{\leftarrow} > \ell_{i+1}^{\leftarrow}$ to be part of $v_n \rightsquigarrow v_1$. Suppose *w.l.o.g.* that $\ell_i^{\rightarrow} > \ell_i^{\leftarrow}$. Thus

8:14 On Inefficiently Connecting Temporal Networks

we have that $\ell_{i+1}^{\rightarrow} > \ell_i^{\rightarrow} > \ell_i^{\leftarrow}$ and also $\ell_{i+1}^{\leftarrow} < \ell_i^{\leftarrow} < \ell_i^{\rightarrow}$. On edge e_{i-1} however, two cases are possible:

- $\ell_{i-1}^{\rightarrow} < \ell_i^{\leftarrow}$: this means label ℓ_i^{\rightarrow} is redundant as $\ell_{i-1}^{\rightarrow} < \ell_i^{\leftarrow} < \ell_{i+1}^{\rightarrow}$;
- $\ell_{i-1}^{\leftarrow} > \ell_i^{\rightarrow}$: this means label ℓ_i^{\leftarrow} is redundant as $\ell_{i+1}^{\leftarrow} < \ell_i^{\rightarrow} < \ell_{i-1}^{\leftarrow}$.

Due to the inequalities presented, and the fact that the difference between ℓ_{i-1}^{\leftarrow} and ℓ_{i-1}^{\rightarrow} must be larger than the difference between ℓ_i^{\leftarrow} and ℓ_i^{\rightarrow} , at least one of the previous cases must be present, meaning at least one label must be redundant which is a contradiction. ◀

► **Theorem 5.** $T^+(\text{Trees}) = 2n - 3$.

Proof. Since path graphs are tree graphs, Lemma 9 gives an upper bound of $2n - 3$, which is attained by the pivot labelling. ◀

C Proof(s) of Section 4

The following technical lemmas are needed to then prove minimality of the generator labelling.

► **Lemma 10.** *In a cycle graph $G = (V, E)$ without any journey covering V , a clockwise journey $\mathcal{J} = ((\{v_j, v_{j-1}\}, t_1), (\{v_{j-1}, v_{j-2}\}, t_2), \dots, (\{v_i, v_{i-1}\}, t_k))$ is dominating if and only if:*

- *it starts at the earliest date possible, i.e. there exists no time edge $(\{v_j, v_{j-1}\}, t)$ nor $(\{v_{j+1}, v_j\}, t)$ with $t < t_1$;*
- *it ends at the latest date possible, i.e. there exists no time edge $(\{v_i, v_{i-1}\}, t)$ nor $(\{v_{i-1}, v_{i-2}\}, t)$ with $t > t_k$;*
- *no other time edges exist between successive time edges, i.e. for all successive pairs of time edges $(\{v_a, v_{a-1}\}, t_b)$ and $(\{v_{a-1}, v_{a-2}\}, t_c > t_b)$ of \mathcal{J} , there exists no time edge $(\{v_a, v_{a-1}\}, t)$ or $(\{v_{a-1}, v_{a-2}\}, t)$ with $t_b < t < t_c$.*

A symmetric characterisation holds for counter-clockwise journeys.

Proof. Let us focus on clockwise journeys, the proof being symmetric for counter-clockwise journeys. Suppose by contradiction that a journey \mathcal{J} obeys the three criteria, but is not dominating, meaning there exists some other distinct clockwise journey \mathcal{J}' covering the same vertex set (or more). A case analysis follows depending on which vertex \mathcal{J}' starts.

If journey \mathcal{J}' starts from any vertex that \mathcal{J} covers, except for v_j , then to ensure \mathcal{J}' covers the vertices of \mathcal{J} , it needs to go all the way around the cycle graph and thus cover V , which is explicitly excluded in this lemma.

If \mathcal{J}' starts at vertex v_j , and it contains some earlier time edge than the corresponding time edge in \mathcal{J} , then \mathcal{J} doesn't respect criterion three (as this earlier time edge exists between successive time edges of \mathcal{J}). If instead it contains a later time edge, then \mathcal{J}' must rejoin or cross \mathcal{J} at some point (since \mathcal{J} uses the latest date of edge $\{v_i, v_{i-1}\}$ by criterion two), implying \mathcal{J} again does not respect criterion three. Of course, if \mathcal{J}' does not contain any earlier or later time edge than \mathcal{J} , then it will end in the same manner as \mathcal{J} without any way of continuing by criterion two, meaning it is identical to \mathcal{J} .

Lastly, if \mathcal{J}' starts on any other vertex, then since \mathcal{J} respects criterion one, \mathcal{J}' must arrive later than t_1 on edge $\{v_j, v_{j-1}\}$. By the same argument as before concerning \mathcal{J}' having a later time edge than \mathcal{J} , the former must rejoin or cross the latter at some point, implying \mathcal{J} does not respect criterion three.

Since all cases end in some contradiction, being either \mathcal{J} breaking one of the criteria or \mathcal{J}' being identical to \mathcal{J} , we can thus conclude that \mathcal{J} is dominating. ◀

► **Lemma 11.** *In a cycle graph $G = (V, E)$ without any journey covering V , a pair of clockwise and counter-clockwise journeys is necessary if:*

- *both start at some same vertex v ;*
- *both are prefix-foremost;*
- *both are a suffix of a dominating journey;*
- *and they do not cross (except on vertex v).*

Proof. We prove that such a pair of clockwise and counter-clockwise journeys, say journey \mathcal{J}_v^w which clockwise goes up to vertex w , and journey \mathcal{J}_v^u which counter-clockwise goes up to vertex u , is necessary for reachability from v to w and from v to u respectively. *W.l.o.g.* we give the proof for the former only, the proof for the latter being symmetric. We first prove that no counter-clockwise journey can reach vertex w , and then that the only clockwise journey that can reach w is journey \mathcal{J}_v^w , from which it follows that this journey is necessary.

First note that vertex v cannot reach further than u in a counter-clockwise manner. Indeed, if by contradiction we suppose there is some counter-clockwise journey $\mathcal{J}_v^{u'}$ from v to vertex u' such that u' is positioned further than u , then *w.l.o.g.* we may consider $\mathcal{J}_v^{u'}$ to be prefix-foremost (if it is not, then we can make it so by changing its time edges for the earliest possible). Since \mathcal{J}_v^u and $\mathcal{J}_v^{u'}$ are both prefix-foremost clockwise journeys, we know that \mathcal{J}_v^u must be a prefix of journey $\mathcal{J}_v^{u'}$, *i.e.* $\mathcal{J}_v^{u'}$ is the concatenation of journeys \mathcal{J}_v^u and say $\mathcal{J}_u^{u'}$. Journey \mathcal{J}_v^u is a suffix of a dominating journey \mathcal{J}_d , meaning no other counter-clockwise journey covers the vertices of \mathcal{J}_d or more, but now we obtain our contradiction: the concatenation of \mathcal{J}_d and $\mathcal{J}_u^{u'}$ covers more vertices (the only case where this wouldn't be true is if \mathcal{J}_d covered V which is explicitly excluded from the lemma statement). Since \mathcal{J}_v^w and \mathcal{J}_v^u don't cross (except on vertex v), we now know that v cannot reach w through a counter-clockwise journey.

To finish the proof, we show \mathcal{J}_v^w is the only clockwise journey that can reach w , meaning all its edges are necessary. Suppose by contradiction another clockwise journey \mathcal{J} exists from v to w . It cannot be a prefix-foremost journey, as by definition this would be journey \mathcal{J}_v^w . Since \mathcal{J} is not prefix-foremost, it uses some edge e with label l' whereas \mathcal{J}_v^w uses edge e with some label $l < l'$. However, we remind the reader that \mathcal{J}_v^w is a suffix of a dominating journey \mathcal{J}_d . Altogether, this means another journey exists covering the same vertices as \mathcal{J}_d , being the concatenation of the prefix of \mathcal{J}_d up to vertex v , and \mathcal{J} . By definition of dominating journeys, this is a contradiction. ◀

We note that if the pair of journeys from Lemma 11 collectively covers V , then vertex v can reach all vertices through these journeys.

► **Theorem 6.** *The generator labelling yields a minimal temporally connected graph.*

Proof. The proof is by induction. Consider cycle graph C_8 as our base case. Apply the generator labelling starting on some edge e . Let e be composed of vertices v_{-2} and v_{-1} , and let vertices v_i be the vertices in the clockwise direction of e , with i the (clockwise) hop distance between e and v_i , and similarly, let vertices $v_{-(i+2)}$ be the vertices in the counter-clockwise direction of e , with i the (counter-clockwise) hop distance between e and $v_{-(i+2)}$. Compute the dominating journeys, see Figure 6. Let us define some specific sets of time edges as follows. Let the five earliest time edges on edges $\{v_{-2}, v_{-1}\}$, $\{v_{-1}, v_1\}$ and $\{v_1, v_2\}$ be referred to as the seed, and the three latest time edges as the trunk. Let the latest two time edges on edges $\{v_{-4}, v_{-3}\}$ and $\{v_{-3}, v_{-2}\}$ be referred to as a branch, as well as the ones on edges $\{v_2, v_3\}$ and $\{v_3, v_4\}$. More specifically, let the former be branch B_2 , as the dominating clockwise journey starting at vertex v_2 ends in these edges, and the latter

B_{-2} as the dominating counter-clockwise journey starting at vertex v_{-2} ends here. Finally, let the other time edges be referred to as the base. Note that all time edges are part of some dominating journey, and that all dominating journeys start on a time edge with time 1 in the base (except for the two dominating journeys starting in the seed) and that all dominating journeys end in the latest time edges of branches (except for two dominating journeys ending in the trunk).

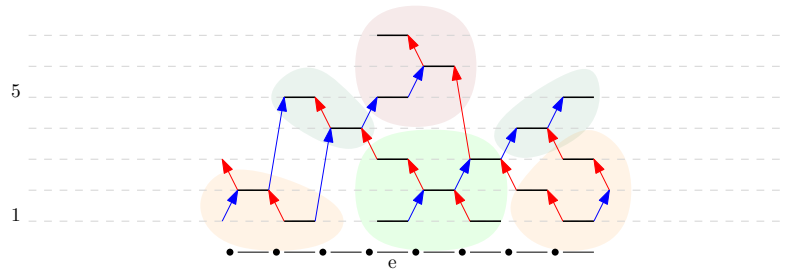
It is possible to claim minimality and temporal connectivity for this small temporal graph, although we specifically point out that Lemma 11 can be used on all vertices (except for those of the seed) to prove necessity of all dominating journeys starting on these vertices, and reachability of all these vertices. Note that now only the time edges of the base remain to be proven necessary. The time edges of the counter-clockwise dominating journey can be proven necessary by applying Lemma 11 on vertex v_{-2} , but it cannot be applied to vertex v_2 to prove the remaining time edges necessary, as its counter-clockwise prefix-foremost journey is not dominating. However, they are proven necessary through the ad hoc argument: without these edges v_2 cannot reach v_{-4} , as any clockwise journey by definition cannot reach it except for its clockwise dominating journey which relied on these time edges, and any counter-clockwise journey reaches at most vertex v_4 . Concerning reachability of the vertices of the seed, it is possible for them to use the dominating journeys starting in the seed to go to any other vertex (note that these journeys do not cross outside of in the seed, but do cover V).

Now, in the inductive step, this structure of seed, base, trunk and branches remains or gets extended when growing the generator labelling for some C_n to some C_{n+4} . More precisely, the seed remains as is, the base gets extended with so-called roots, the trunk with a so-called apex, and the branches with leaves. Also, two new branches are created in every inductive step, which sprout from the top of the trunk. Underlying all this, we prove that in the inductive step, the dominating journeys get extended slightly, are modified, or created, in a precise manner which ultimately allows us to again use Lemma 11 to prove minimality and temporal connectivity, in a very similar manner as how we did for C_8 .

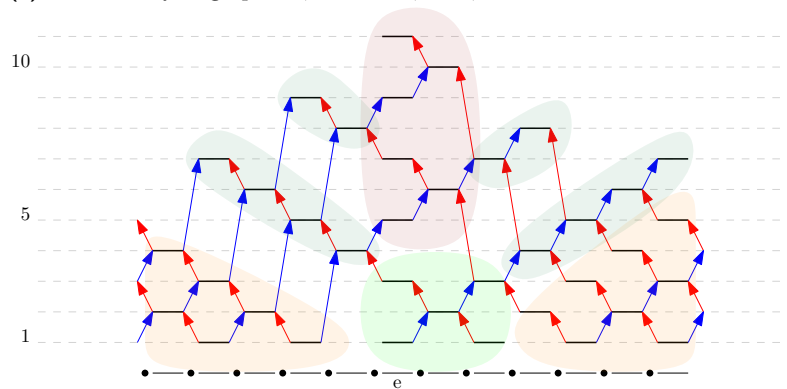
Now, suppose we have a cycle graph C_{4k} with the generator labelling which has been proven minimal and temporally connected, specifically through applying Lemma 11 on all vertices except for the seed. Add vertices v_{2k+1} , v_{2k+2} , v_{-2k-1} , and v_{-2k-2} and the corresponding edges to the link stream representation so as to obtain cycle C_{4k+4} . See Figure 6. This effectively breaks dominating journeys which previously used edge $\{v_{-2k}, v_{2k}\}$, whose time edges now belong to edge $\{v_{-2k}, v_{-2k-1}\}$. We will patch these halves of dominating journeys back together in what follows, although not exactly with their original half. Note that now the generator labelling for C_{4k+4} is exactly this labelling, with some additional time edges which are all later, *i.e.* for all additional time edges (e, t) , there exists no already present time edge $(e, t' > t)$. Let the three additional time edges extending the trunk be referred to as the apex, let the leaves be the pairs of additional time edges extending the branches (as well as creating branches B_{2k} and B_{-2k} from the trunk), and let the remaining additional time edges be the roots, which extend the base.

Let us start by proving these additional edges are all part of some dominating journey.

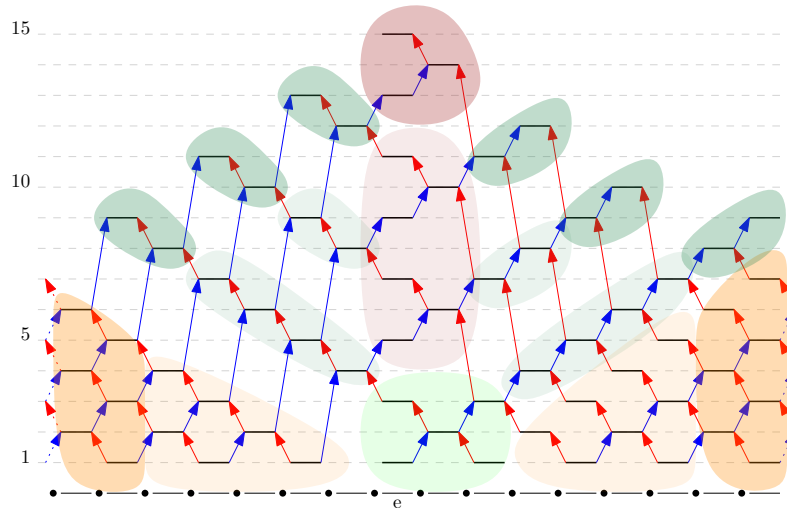
Leaves extending branches B_i extend the corresponding dominating journey starting from vertex v_i . The three conditions from Lemma 10 hold for this extended journey, as it still starts at time 1, no additional time edges have been added in between the time edges it uses, and it ends at the latest time possible at the top of its respective leaves. Regarding the leaves that create a new branch B_{2k} and B_{-2k} , these extend dominating journeys from vertices v_{2k} and v_{-2k} which ended at the top of the trunk before (we can observe two of these exist



(a) Base case cycle graph C_8 , with seed, base, trunk and two branches.



(b) Induction step cycle graph C_n (here $n = 12$) before extending to C_{n+4} .



(c) Apex, leaves, and roots extending C_n to obtain C_{n+4} , adding two branches.

■ **Figure 6** Illustration of the proof by induction for Theorem 6, with the seed (light green), trunk (brown), branches (green) and base (light brown). At the induction step, the apex, leaves, and roots are shown in the same but less transparent colour as the structures they extend.

in C_8 , and below we prove that in every inductive step two new such journeys are created). These extended journeys remain dominating by the same argument as for other branches. All leaves are thus part of a dominating journey. Note that this extends (by exactly two time edges) basically half of all previously existing dominating journeys.

The other half of previous dominating journeys are broken up through the addition of the four new vertices and edges. The roots serve to patch these journeys back together. Note that before, all these dominating journeys started in the base, cycled around, and climbed through the branches to finish at the top of some branch. More specifically, such a dominating journey starting from a vertex v_i finished at the top of branch B_{i-1} for $i > 0$ and at the top of branch B_{i+1} otherwise (an exception being the journey starting from v_{-3} which ends at the second largest time edge of the branch B_{-2}). We show this remains true after the inductive step. The reconstructed dominating journey, suppose from vertex v_i for $i > 0$ (the explanation being symmetric for $i < 0$), starts off with the same time edges it had before in the base until it reaches the roots. This means this part of the journey respects two of the conditions of Lemma 10, being it starts at time 1, and no time edges exist in between its time edges as this journey was dominating before and the additional time edges are all later. Now the earliest four time edges possible are taken to continue this journey in the roots, cycling around to the other side of the link stream. This also respects the condition of having no time edges in between these four time edges, as the roots are densely packed by definition. The journey is now four time steps too late to reconnect with the other half it had before, connect it instead with the half of the journey which previously started from vertex v_{i-4} for which it arrived through the roots perfectly on time. This latter half also respects the condition of having no time edges in between its time edges due to part of a dominating journey before, and no additional time edges have been added in between these time edges. Since our journey now follows the part of the dominating journey which previously started at v_{i-4} , it arrives at branch B_{i-5} , but can now continue through the leaves of B_{i-3} and finally B_{i-1} to end at the latest edge. By construction, this continuation through the leaves respects the conditions of Lemma 10 since there are no time edges in between, and it ends at the latest time possible. There are two exceptions to this: the reconstruction of the dominating journey starting from vertex v_3 only uses three time edges from the roots, before directly ending in the leaves of branch B_2 , and the one starting from vertex v_5 directly goes up through the leaves of B_2 and B_4 after the roots. Both reconstructed journeys are dominating as well. Note that now all dominating journeys starting at vertices v_i with $-2k \leq i \leq 2k$ have been extended (the ones cycling around have been broken apart and refitted first but in terms of length have been extended as well) by exactly two time edges compared to their previous length in C_{4k} .

Observe that some of the earliest roots have not been shown to be part of a dominating journey yet, and also that some halves of previous dominating journeys have not been refitted together yet. We show another four dominating journeys exist which start from the four new vertices, use these earliest roots, as well as the remaining parts of previous dominating journeys, and two of these journeys use the time edges of the apex. Proving these four journeys are dominating is done through again applying Lemma 10 with the arguments already explained for the other dominating journeys, and thus we decide to forgo doing this again four more times. The dominating journey starting at v_{-2k-1} goes clockwise, starting at time 1 and the four earliest roots, then it continues with the part of the previous dominating journey ending in branch B_{-2k+4} , and goes up through the leaves of branches B_{-2k+2} and B_{-2k} , finishing at the latest time edge of the latter. Starting at vertex v_{-2k-2} , we have a counter-clockwise dominating journey, starting at time 1 using only one root, linking up with

part of a previous dominating journey which finished at branch B_{2k-2} , which is extended further through branch B_{2k} and the apex to end on the second latest time edge. We note that the last two dominating journeys do not cross, except on the first edge, and cover V . Continuing, we have a clockwise dominating journey starting at v_{2k+2} and time 1, using two roots before using part of a previous dominating journey leading up to branch B_{-2k-2} , continuing through the leaves of B_{-2k} and ending in the apex on the largest time edge. Finally, there is a counter-clockwise dominating journey from vertex v_{2k+1} using three roots cycling around the link stream, pairing up with part of a previous dominating journey leading up to branch B_{2k-4} which then continues through leaves of B_{2k+2} and B_{2k} to end on the largest time edge of that branch. Again, these two journeys do not cross, except on the first edge, and cover V .

Thus, we have that all time edges are part of a dominating journey, and that basically the same journeys from C_{4k} remain in C_{4k+4} (albeit some of them recombined differently) and were extended by exactly two time edges. Since for all vertices but those of the seed, Lemma 11 was used to prove necessity of the corresponding dominating journeys, this lemma can be used again for these vertices to prove necessity of their corresponding dominating journeys, as well as reachability of these vertices to all others. For the time edges and vertices of the seed, the argument used for C_8 can be generalized to prove necessity and reachability as well. Finally, the last four dominating journeys which start on the four new vertices, can use Lemma 11 as well, since their clockwise and counter-clockwise prefix-foremost journeys are dominating and can collectively cover V .

In conclusion, we have proven that the base case, being the generator labelling for C_8 , is minimal and temporally connected. Then, for any inductive step from C_{4k} to C_{4k+4} , minimality and temporal connectivity are conserved in this labelling. Thus, the generator labelling produces a minimal and temporally connected graph for any size $4k$. (The generator labelling works for C_4 as well, it is easy to check, but the structure of the labelling was easier to explain with C_8 , as for C_4 the labelling is composed of only the seed.) ◀

All for One and One for All: An $O(1)$ -Musketees Universal Transformation for Rotating Robots

Matthew Connor ✉

Department of Computer Science, University of Liverpool, UK

Othon Michail ✉ 

Department of Computer Science, University of Liverpool, UK

George Skretas ✉ 

Hasso Plattner Institute, University of Potsdam, Germany

Abstract

In this paper, we settle the main open question of [Michail, Skretas, Spirakis, ICALP'17], asking what is the family of two-dimensional geometric shapes that can be transformed into each other by a sequence of rotation operations, none of which disconnects the shape. The model represents programmable matter systems consisting of interconnected modules that perform the minimal mechanical operation of 90° rotations around each other. The goal is to transform an initial shape of modules A into a target shape B . Under the necessary assumptions that the given shapes are connected and have identical colourings on a checkered colouring of the grid, and using a seed of only constant size, we prove that any pair of such shapes can be transformed into each other within an optimal $O(n^2)$ rotation operations none of which disconnects the shape.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases programmable matter, universal transformation, reconfigurable robotics, shape formation, centralised algorithms

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.9

1 Introduction

Programmable matter refers to matter that can change its physical properties algorithmically. It is envisioned as a collection of modules connected to each other to form a *shape*. Due to size and other constraints of the individual modules, limited actuation and sensing capabilities are available to them, which a program uses to enable the interaction of the material with its surroundings and to control its structural dynamics. The relevant theoretical literature has almost exclusively focused on designing algorithms (either centralised or distributed) for the task of transforming a given initial shape A into a given target shape B and characterising the families of shapes that can be transformed into each other within a given programmable matter model. Transformations should additionally be efficient, which for sequential transformations is measured by the total number of individual actuation operations.

In [20] (and its journal version [21]), Michail et al. studied a model of programmable matter in which modules are represented by nodes drawn within the cells of a two-dimensional square grid. Nodes are connected to any nodes orthogonally adjacent to them (their *neighbours*) to form a shape. The collection of nodes can be reconfigured between shapes through minimal types of movements. One of the considered movements was *rotation*: a node can rotate 90° around a neighbour provided that the rotating node's trajectory is free from other nodes. The authors introduced the problem of characterising which families of connected shapes can be transformed into each other via rotation movements. If global connectivity need not be preserved, they proved that the a decision version of the problem (called ROT-



© Matthew Connor, Othon Michail, and George Skretas;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 9; pp. 9:1–9:20

Leibniz International Proceedings in Informatics



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

TRANSFORMABILITY) is in \mathbf{P}^1 . They proved this through a constructive exact characterisation of the shapes that can be transformed into each other. For the ROTC-TRANSFORMABILITY version of the problem, in which global connectivity must be preserved after every movement, they proved inclusion in \mathbf{PSPACE} and highlighted that surprisingly small *seeds* can enable transformations that are otherwise infeasible. The main problem they left open was asking if there exists a universal centralised transformation for ROTC-TRANSFORMABILITY under the assumption of a constant-size seed.

A more general version of the model is one that combines rotation and sliding, where a node can additionally slide over pairs of consecutive nodes. For this version, Dumitrescu et al. [14] had studied distributed transformations and had conjectured that universal connectivity-preserving transformation is possible. This was proven to be correct in [13] and independently in [20]. Both the rotation and the combined rotation and sliding models aim to represent minimal and, thus, cost- and energy-efficient mechanical operations and to show that real implementations of programmable matter using them could hope to have universal reconfiguration capabilities. The focus on minimal operations is also justified by the engineering objective to minimise the size of individual modules in order to improve the granularity of the material without sacrificing its global actuation capabilities.

However, with rotation alone not all pairs of connected shapes of the same number of nodes can be transformed into each other. For example, there are shapes, like a rhombus, that are completely blocked and other shapes, like a line, that are blocked within a small final strongly connected component of the shape-reachability graph if connectivity is to be preserved. Because of this, universal transformation by rotation cannot be achieved without additional assumptions. Such an assumption, introduced in [20], is to use a small additional set of nodes, called a *seed* (or *d*-seed, where *d* is the number of nodes of the seed), that when placed appropriately on the perimeter of the shape can trigger the otherwise infeasible transformation. The question posed was: *Is there a reasonably small seed (i.e., of constant size), whose availability enables universal connectivity-preserving transformation when the only available movement is rotation?*

Direct progress on this open question was made by Connor et al. [6] and Connor and Michail [5]. In [6], a 4-seed was shown to be sufficient for solving the problem on a restricted family of shapes, called *nice shapes*. These were first defined in [3] as all shapes S having a central line L , where, for all nodes $u \in S$, either $u \in L$ or u is connected to L by a line of nodes perpendicular to L . The first breakthrough towards universality was achieved in [5], where the problem was shown to be solvable for all orthogonally convex shapes by using a minimal 3-seed². The family of nice shapes and that of orthogonally convex shapes are not directly comparable as each contains at least one shape that does not belong to the other. Nevertheless, orthogonally convex shapes have appeared to be much richer in structure and harder to transform. We extend the techniques developed in [5] to obtain an $O(1)$ -seed universal transformation, that is, one that works for all pairs of connected *colour-consistent* shapes. Two shapes are colour-consistent if they have identical colourings on a checkered colouring of the grid: as a node cannot change colour by rotating, any shapes that can be transformed into each other must be colour-consistent.

¹ Polynomial time in this context refers to the worst-case time complexity of an algorithm that decides if two given shapes A and B can be transformed into each other. It should not be confused with the efficiency of a transformation between the shapes, which is measured in total movements for sequential transformations and in time-steps of parallel movements for parallel transformations.

² A shape S is *orthogonally convex* if for any two nodes u, v in a horizontal or vertical line of the grid, all cells between u and v are occupied by S .

There is a lower bound on the number of worst-case movements required by a transformation, which is quadratic in the number of nodes. It is based on a measure of “distance” between the initial and the target shape, and applies to all models in which every movement reduces the total distance by at most a constant, also affecting solutions that do not preserve connectivity. Because of this, optimal sequential transformations perform $O(n^2)$ movements.

In another model which bares some similarities to the rotation and the combined rotation and sliding models, Akitaya et al. [1] studied a different type of movement, called *pivoting*. Pivoting allows a square-shaped node to emulate sliding and rotation. Both these operations happen by fixing an arm on a shared corner between two squares and rotating one of the squares along that arm. The rotation movement we consider here is not directly comparable to pivoting. Pivoting requires more empty space around the moving node than rotation. On the other hand, it can “slide” a node to an orthogonally adjacent cell, thus, in contrast to what holds for rotation, pivoting nodes have no *a priori* unreachable locations. Akitaya et al. accomplished universal transformation in $O(n^2)$ pivoting movements using a “bridging” procedure assisted by at most 5 seed-nodes, which they called *musketeers*.

2 Contribution and Approach

We study ROTC-TRANSFORMABILITY, the problem of characterising the families of connected shapes that can be transformed into each other via rotation movements without breaking connectivity. As our focus is on the feasibility and complexity of transformations, our approach is naturally based on structural characterisations and centralised procedures. Structural and algorithmic progress is expected to facilitate more applied future developments, such as distributed implementations.

When rotation is combined with sliding, the algorithmic strategy to establish universality [13, 20] is quite intuitive. The goal is to show that any two connected shapes of the same number of nodes can be transformed into each other. Due to reversibility of these movements, it is sufficient to show that any connected shape S of n nodes can be transformed into a straight line of length n . The line is the *canonical shape* of this strategy and all its transformations will go through it. The strategy is based on the observation that, by combining rotation and sliding, a node can traverse the perimeter of S . To transform S into a line, a position on the perimeter of S from which the line can grow to its full length is fixed. It can then be shown that there is always a node to remove from the perimeter without disconnecting the shape. The algorithm moves the node along the perimeter until it reaches the line and places it in the empty cell adjacent to the furthest endpoint of the line. It then repeats by removing another node from the perimeter.

Rotation alone is also quite powerful if connectivity need not be preserved. As there is an infinite family of shapes which are completely blocked under rotation (the rhombi), one cannot hope to achieve universality. Nevertheless, there is a strategy that works for all the remaining shapes [20]. The canonical shape of this strategy is the *line-with-leaves*, a family of shapes with maximal colour capacity. The transformation removes nodes from the shape in pairs and transports them to the line-with-leaves, which can be constructed anywhere on the grid, not necessarily being connected to the original shape.

When connectivity must be preserved, transformations by rotation alone can be notoriously difficult. There are two main sources of this difficulty. We still cannot hope to achieve universal transformation free from additional assumptions due to blocked shapes, whose class is now increased by the requirement to preserve connectivity. Moreover, as it cannot change colour in a checkered colouring of the grid, a node cannot in general traverse the perimeter of a shape. It was known from [20] that this remains impossible for up to 4 nodes working together: 4 or less nodes cannot traverse the perimeter of a straight line.

The transformations of [6] and [5] are based on the approach of using a small set of nodes (called a *robot*) that work together to traverse the perimeter of the shape and transport other nodes, thus simulating the single-node traversal of the combined rotation and sliding model. The robot of [6] consisted of 4 nodes and the robot of [5] of 6 nodes. The reason that the 4-robot of [6] does not violate the lower bound of [20] on the number of nodes needed to traverse the perimeter, is that, due to their special structure, nice shapes can be transformed through partial traversals of their perimeter. Both papers used seeds of 4 and 3 nodes, respectively, to enable the initial formation of the robot and deal with blocked shapes.

The result of [5] is that, under the assumption of a 3-seed, any pair of colour-consistent orthogonally convex shapes can be transformed into each other. The 3-seed is optimal: there are blocked shapes for which non-trivial transformations cannot be enabled by a smaller seed. The transformation uses the 3-seed to remove a 6-robot. It proceeds in phases to transform the given shape into a canonical shape, which is an orthogonally convex variant of the line-with-leaves. In each phase, the 6-robot removes the next node from the perimeter according to an elimination sequence, transports it around the perimeter until it reaches the canonical shape, and places the node in the next available cell of the canonical shape.

A key difficulty in generalising the approach of [5] to any shape is that the perimeter of an arbitrary shape can be a lot harder to traverse. Though, as in [5], there is always a node on the perimeter that can be removed, a robot of nodes might not have enough space to reach that node and it is not even clear if it can successfully traverse the perimeter with or without carrying the node. For example, all removable nodes might be concealed within pockets formed by the perimeter that are too narrow for the robot to access and there can be concave parts of the perimeter to which the traversal of [5] does not readily transfer. As a result, both the elimination sequence and the robot traversal must be carefully redesigned.

We overcome these difficulties and show how to transform any shape S into a variant of a line-with-leaves by a sequence of $O(n^2)$ rotations. Reversibility of rotation then implies a universal transformation between pairs of shapes going through the line-with-leaves. We first argue that there is a placement of an $O(1)$ -seed on S from which a “good” initial configuration for the transformation can be obtained, having a 6-robot and an adequate initialisation of the line-with-leaves on the *reachable* part of the perimeter of S . We show how to compute an elimination sequence of the nodes of S that guarantees a generation sequence of the line-with-leaves that never exceed its colour capacity. As long as there are reachable nodes to be removed as required by the elimination sequence, the 6-robot picks a reachable node, transports (as a 7-robot) the node to the line-with-leaves, and places the node in an appropriate cell adjacent to the line-with-leaves, as specified by the corresponding generation sequence. This entails showing that both a 6-robot and a 7-robot can traverse the whole reachable perimeter of S , by traversing reachable parts and bypassing unreachable parts. If there are no reachable nodes to be removed, we show how closing a “bottleneck lid” and compressing the shape (if needed) allows the elimination sequence to keep making progress.

In Section 3, we discuss other related work. In Section 4, we formally define the model used in this paper. Section 5 presents the universal transformation. In Section 6, we conclude and state some open problems.

3 Other Related Work

As the development of these systems continues, it becomes increasingly necessary to develop theoretical models which are capable of describing and explaining the emergent properties, possibilities and limitations of such systems in an abstract and fundamental manner. To this

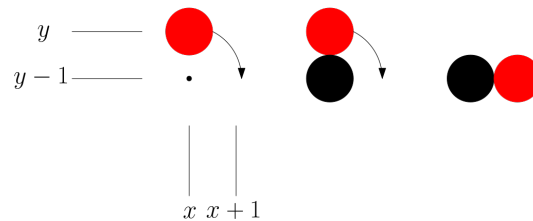
end, models have been developed for programmable matter. For example, algorithmic self-assembly [11, 23, 24] focuses on programming molecules like DNA to grow in a controllable way, and the Abstract Tile Assembly Model [25, 30], the Kilobot model [26], the Robot Pebbles system [17], and the nubot model [31], have all been developed for this area. Network Constructors [22] is an extension of population protocols [4] that allows for network formation and reconfiguration. The latter model is formally equivalent to a restricted version of chemical reaction networks, which “are widely used to describe information processing occurring in natural cellular regulatory networks” [27, 12]. The CATOMS system [28, 29, 15] is a further implementation which constructs 3D shapes by first creating a “scaffolding structure” as a basis for construction. Finally, there is extensive research into the amoebot model [8, 7, 10, 9], where finite automata on a triangular lattice follow a distributed algorithm to achieve a desired goal, including a recent extension [16] to a circuit-based model.

Almalki and Michail [2], building on the insertion operations of [31] and the growth processes on graphs by Mertzios et al. [19], investigated what families of shapes can be grown in time polylogarithmic in their size by using only growth operations.

4 Model

We consider the case of programmable matter on a two-dimensional square grid, with each cell of the grid being uniquely referred to by its (x, y) coordinates. Such a system consists of a set V of n nodes. Each node is viewed as a spherical module fitting inside a cell of the grid. At any point, each node occupies a cell, with the positioning of the nodes defining a shape, and two nodes may not occupy the same cell. It also defines an undirected *neighbouring relation* $E \subset V \times V$, where $uv \in E$ iff $o_x(u) = o_x(v)$ and $|o_y(u) - o_y(v)| = 1$ or $o_y(u) = o_y(v)$ and $|o_x(u) - o_x(v)| = 1$, that is, if u and v occupy *horizontally* or *vertically* adjacent cells of the grid. We use $N(u)$ to denote the set of neighbours of a node u in a given shape. A shape is *connected* if the graph induced by its neighbouring relation is a connected graph.

In general, shapes can *transform* to other shapes via a sequence of one or more mechanical operations, which we refer to as movements. We consider only one type of movement: *rotation*. In this movement, a single node moves relative to one or more neighbouring nodes. A single rotation movement of a node u is a 90° rotation of u around one of its neighbours. Let (x, y) be the current position of u and let its neighbour be v occupying the cell $(x, y - 1)$. Then, u can rotate 90° clockwise (counterclockwise) around v iff the cells $(x + 1, y)$ and $(x + 1, y - 1)$ ($(x - 1, y)$ and $(x - 1, y - 1)$, respectively) are both empty. By rotating the whole system 90° , 180° , and 270° , all possible rotation movements can be defined. See Figure 1 for an example.



■ **Figure 1** An example of rotation movement. A node on the black dot (in the row $y - 1$) and empty cells at positions $(x + 1, y)$ and $(x + 1, y - 1)$ are required for this movement.

Let A and B be two connected shapes. We say that A transforms to B via a rotation r , denoted $A \xrightarrow{r} B$, if there is a node u in A such that if u applies r , then the shape resulting after the rotation is B . We say that A transforms in one step to B (or that B is reachable in

one step from A), denoted $A \rightarrow B$, if $A \xrightarrow{r} B$ for some rotation r . We say that A transforms to B (or that B is reachable from A) if there is a sequence of shapes $A = S_1, S_2, \dots, S_t = B$, such that $S_i \rightarrow S_{i+1}$ for all $1 \leq i \leq t-1$. Rotation is a reversible movement, a fact that we use in our results. As a condition of the problem we consider, all shapes S_1, S_2, \dots, S_t must be connected shapes.

At the start of each transformation, we will be assuming the existence of a *seed*: a small connected shape M placed on the perimeter of the given shape S to trigger the transformation. This is essential because under the constraints of a model with rotation-only movement, there are shapes S that are κ -blocked, meaning that at most κ movements can be made before a configuration is repeated.

For the sake of providing clarity to our transformations, we say that every cell in the two-dimensional grid has a colour from $\{red, black\}$ in such a way that the cells form a black and red checkered colouring of the grid, similar to the colouring of a chessboard. This represents a property of the rotation movement, which is that any given node in a coloured cell can only enter cells of the same colour. We define $c(u) \in \{black, red\}$ as the colour of the cell occupied by the node u for a given chessboard colouring of the grid. We represent this in our figures by colouring the nodes red or black.

Any shape S consists of $b(S)$ black and $r(S)$ red nodes. Two shapes A and B are *colour-consistent* if $b(A) = b(B)$ and $r(A) = r(B)$. For any shape S of n nodes, the *parity* of S is the colour of the majority of nodes in S . If there is no strict majority, we pick any as the parity colour.

We use σ and variants to denote sequences of nodes. A k -subsequence σ' of a sequence σ is any subsequence of σ where $|\sigma'| = k$. For a given colouring of the grid, the *colour sequence* $c(\sigma)$ of a sequence of nodes $\sigma = (u_1, u_2, \dots, u_n)$ is defined as $c(\sigma) = (c(u_1), c(u_2), \dots, c(u_n))$. A sequence σ' is *colour-order preserving with respect to σ* if $c(\sigma') = c(\sigma)$.

The *perimeter* of a connected shape S is the minimum-area polygon that completely encloses S in its interior, existence of an interior and exterior directly following from the Jordan curve theorem [18]. The *cell perimeter* of S consists of every cell of the grid not occupied by S that contributes at least one of its edges to the perimeter of S . The *external surface* of S consists of all nodes $u \in S$ such that u occupies a cell defining at least one of the edges of the perimeter of S . The *extended external surface* of S is defined by adding to the external surface all nodes of S whose cell shares a corner with the perimeter of S .

The orthogonal convex hull $H(S)$ of a connected shape S is defined as the intersection of all orthogonally convex shapes of which S is a subshape. Given a connected shape S , a *pocket* is a maximal connected set of empty cells exterior to the shape and interior to its orthogonal convex hull $H(S)$. The *boundary* of a pocket consists of a line segment which is a subchain of the perimeter of $H(S)$, called the *pocket lid*, and a subchain of the perimeter of the shape, called the *pocket subchain*.

Place a $c \times c$ square K on a subchain of the perimeter of S which is exterior to its convex hull and shift it around. A $c \times c$ -narrow pocket is a maximal subset of a pocket of S which can never overlap with K . The *bottleneck* of a $c \times c$ -narrow pocket is the subchain which separates the cells reachable by K from the cells within the pocket which are unreachable. The $c \times c$ -reachable boundary of S is defined as the areas of the perimeter that are reachable by K , plus the bottlenecks. A shape S has a $c \times c$ -wide exterior if it has no $c \times c$ -narrow pockets. Throughout the paper, a *bottleneck lid* is the lid of a 4×4 -narrow pocket.

A black parity (similarly for red parity) *line-with-leaves* [21] is a straight line with one or more black leaves attached to its red nodes. A *double-line-with-leaves* L is a shape obtained by joining together the endpoints of two lines-with-leaves L_1 and L_2 which have opposite

colour parities and the same orientation. Additionally, both endpoints of the red parity line-with-leaves must be black and both endpoints of the black parity line-with-leaves must be red. We call the straight line formed by joining the straight lines of L_1 and L_2 the *core line* of L .

5 The Universal Transformation

In this section, we give the technical details of the universal transformation. Given any two connected colour-consistent shapes S and S' , our goal is to transform S into S' . We assume a seed M of $d = O(1)$ nodes, meaning that we are free to place any connected shape of d nodes on the perimeter of the shape. The transformation will establish the following theorem.

► **Theorem 26.** *Let S and S' be any two connected colour-consistent shapes. Then, there is a connected shape M of $d = O(1)$ nodes and a placement of M on the perimeter of S , such that $S \cup M$ can be transformed into S' via $O(n^2)$ rotation movements.*

To prove Theorem 26 it is sufficient to show that, for any shape S and some placement of a d -seed M on the perimeter of S , $S \cup M$ can be transformed into a double-line-with-leaves. By reversibility of rotation and the fact that we can transform any pair of colour-consistent double-lines-with-leaves into each other, it follows that any S can be transformed into any S' via the double-line-with-leaves canonical shape.

The following is an intuitive description of our strategy. We will show that there is a placement of the d -seed on the perimeter of S from which a “good” starting configuration for the transformation can be obtained. A 6-robot formed by the d -seed sets up the shape to be in a configuration having the following structures on the perimeter of the shape: (i) a “ladder” on which the double-line-with-leaves will be built, (ii) a reservoir of 7 nodes to be used for the compression subroutine, and (iii) a 6-robot. Then, as long as there are reachable nodes to be removed, the 6-robot picks a removable node, transports it (as a 7-robot), and places it on the double-line-with-leaves. This involves proving that the 6-robot (7-robot) can traverse the 4×4 -reachable boundary (5×5 , respectively) of S , by traversing reachable parts and bypassing unreachable parts, and that there is an order of removing nodes from the perimeter of S , until S is eliminated. This order, called an elimination sequence, removes small clusters of nodes such that no removal of a node disconnects the shape and nodes are only removed from the perimeter of the shape. Whenever there is no cluster of nodes that the 6-robot can reach, we show how to reconfigure S into another shape that has a reachable cluster of nodes. In particular, if no cluster of nodes exists on the perimeter that is also reachable, then there exists a pocket lid that we can close with auxiliary nodes, such that a cycle C on the perimeter is created. We can then compress C in a way that a cluster of nodes becomes reachable and, after removing the cluster, there exists a cycle C' on the perimeter of the shape. The 6-robot transports, one by one, the nodes of the cluster to the double-line-with-leaves, and removes any auxiliary nodes used to close a lid, before moving on to the next cluster.

5.1 Perimeter Traversal

We begin by showing that the 6-robot and the 7-robot can both traverse the perimeter of the shape, by visiting reachable parts and bypassing unreachable parts.

In [5], it was shown that a 6-robot and a 7-robot can both traverse the perimeter of an orthogonally convex shape. We say that a shape S , which is not orthogonally convex, is *traversable by orthogonally convex movement*, if the movements from [5] can be used to traverse its perimeter.

► **Theorem 1** ([5]). *For any orthogonally convex connected shape S , a 6-robot and a 7-robot are both capable of traversing the perimeter of S .*

As in [5], a 6-robot is a 3×2 group of connected nodes used to transport nodes around the shape. By using rotation movements, the 6-robot can slide across and climb lines of the perimeter. These movements of the robot are the outcome of a sequence of rotations, and should not be confused with the sliding of individual nodes of [14, 13, 20].

We prove that a 6-robot can traverse the 4×4 -reachable boundary and a 7-robot can traverse the 5×5 -reachable boundary of any connected shape S by orthogonally convex movement. Beginning with the 6-robot, our strategy is to show that shapes with 4×4 -wide exteriors are traversable by orthogonally convex movement. We then show that for shapes with 4×4 -narrow pockets, the robot can avoid entering those pockets by crossing them.

We assume that a 6-robot, which we will try to move around the perimeter of S , is given on the 4×4 -reachable boundary as a 3×2 rectangle. For the 7-robot, we consider the 5×5 -reachable boundary as the 7-robot requires more space to move. We first prove that shapes with a 4×4 -wide exterior (5×5) have sufficient space for the 6-robot (7-robot, respectively) to perform the climbing and sliding movements of [5], through which the robot can traverse the perimeter of the shape.

► **Lemma 2.** *The perimeter of a connected shape S with a 4×4 -wide exterior (5×5 -wide exterior) can be traversed by orthogonally convex movement by a 6-robot (7-robot, respectively).*

Next, we show that it is possible to traverse small pockets by crossing them. We use the term *gap* to refer to the part of the pocket as well as lines of nodes neighbouring it, which are relevant for crossing operations. We first give our representation of the cases which we consider for these operations, as well as the variables we will be using to describe them.

Assume without loss of generality that the movement of the robot when crossing the gap is to the right, and if necessary, upwards. We have five coordinates: x_l, x_r, y_d, y_u and y_m (see Figure 2 in the Appendix). These coordinates in turn are used to calculate the three variables we use: *size* is equal to $|x_r - x_l|$, *depth* = $|y_m - y_d|$ and *incline* = $|y_u - y_m|$. Intuitively, *size* represents the horizontal distance between the first and last nodes of the gap, *depth* represents the vertical distance between the first node of the gap and the bottom of the gap, and *incline* represents the vertical distance between the first and last nodes of the gap. We say that a gap is *level* if *incline* = 0.

We first show that gaps of size ≤ 3 can be crossed by the 6-robot, by considering a set of cases which cover every possible situation. We assume the robot is above the gap in both the initial and final locations. This is to ensure that the movements do not need to make assumptions about the structure of the rest of the shape. We then prove that the robot can always reach this desired starting location, even in edge cases. It follows from this and from Theorem 1 that it is possible for the 6-robot to traverse the 4×4 -reachable boundary of any shape. See Figure 3 in the Appendix for the main cases we consider in our proof, and Figures 4 and 5 for two examples of crossing a gap.

► **Lemma 3.** *The 6-robot can cross any gap of size ≤ 3 and the 7-robot any gap of size ≤ 4 .*

► **Lemma 4.** *The 6-robot and the 7-robot can position themselves at the start of any gap.*

► **Theorem 5.** *The 4×4 -reachable boundary (5×5 -reachable boundary) of any connected shape S can be traversed by the 6-robot (7-robot, respectively).*

5.2 Elimination Sequence

In this section, we present the sequence in which the 6-robot transports the nodes of the shape S . The elimination sequence of [5] for orthogonally convex shapes, was designed to preserve connectivity and respect the colour capacity of the line-with-leaves, which was the canonical shape. For general connected shapes, we must additionally ensure that removed nodes should lie in a position that the 6-robot can reach. We first give an elimination sequence for the relaxed case in which nodes can be removed from anywhere on the extended external surface of the shape. We will then add the requirement that removed nodes must be reachable by a 6-robot. Apart from simplifying exposition, the relaxed elimination sequence could be useful to any future transformations that would somehow circumvent the reachability issue.

Let S be a connected shape. An *elimination sequence* $\sigma = (u_1, u_2, \dots, u_n)$ of a shape S is a permutation of the nodes of S satisfying the following properties. Let $S_t = S_{t-1} \setminus \{u_t\}$, where $1 \leq t \leq n$ and $S_0 = S$. Observe that S_n is always the empty shape. The first property is that, for all $1 \leq t \leq n-1$, S_t must be a connected shape. Moreover, for all $1 \leq t \leq n$, u_t must be a node on the extended external surface of S_{t-1} . Essentially, σ defines a sequence $S = S_0[u_1]S_1[u_2]S_2[u_3] \dots S_{n-1}[u_n]S_n = \emptyset$, where, for all $1 \leq t \leq n$, the connected shape S_t is obtained by removing node u_t from the extended external surface of shape S_{t-1} .

Our strategy for the elimination sequence is to remove nodes in small *clusters* that have some “nice” properties. Each such cluster will contains 2 to 4 nodes, and is either a pair of neighbouring nodes, or a node together with a subset of its neighbours, which must be leaves. We want to show that removing any of these clusters does not disconnect the shape, and that the clusters have specific colour properties. We first give some necessary definitions.

► **Definition 6.** Let $S = (V, E)$ be a connected shape. Node $u \in V$ is a *separator node* if $S' = (V \setminus \{u\}, E')$, where $E' = E \setminus \{uv \in E \mid v \in N(u)\}$, is a disconnected shape.

► **Definition 7.** Let $S = (V, E)$ be a connected shape. Node u is a *local separator node*, if u is a separator node and there exists a subset $N'(u)$ of $N(u)$, such that $S' = (V', E')$, where $V' = V \setminus N'(u)$ and $E' = E \setminus \{uv \in E \mid v \in N'(u)\}$, is a connected shape and u is not a separator node in S' .

► **Definition 8.** We define a *cluster of nodes* C , to be a set of nodes on the extended external surface of a shape S that satisfies one of the following two properties:

1. C contains two neighbouring nodes u, v such that removing u and then v , or v and then u does not disconnect the shape.
2. C contains a local separator node u , and every neighbour of u that is a leaf in S .

Intuitively, our algorithm computes an elimination sequence as follows. Operating in phases until the whole shape is eliminated, it marks the extended external surface of the shape and repeatedly finds and removes clusters of marked nodes. The order in which the nodes of a cluster are added to the elimination sequence is the order in which the nodes will be transported by the 6-robot. When there is no cluster of marked nodes, the algorithm moves on to the next phase, marking the new extended external surface and searching for clusters of marked nodes. The pseudocode is given in Algorithm 1 in the Appendix. Connectivity-preservation is guaranteed by the separator properties of the clusters.

► **Lemma 9.** Let S be a connected shape on which we execute Algorithm 1 and let σ be the elimination sequence produced by the algorithm. For all $1 \leq t \leq n-1$, S_t is a connected shape.

We also need to show that the algorithm terminates. We do this by making use of the fact that the extended external surface of a connected shape defines a cactus graph.

► **Definition 10.** *Given a connected shape $S = (V, E)$, we define a graph $S' = (V', E')$ on its external extended surface as follows. V' contains every node of the extended external surface and $uv \in E'$ iff u and v are consecutively visited in a clockwise walk on the perimeter of S .*

► **Lemma 11.** *The graph of Definition 10 is a cactus graph.*

► **Lemma 12.** *Let S be a connected shape on which we execute Algorithm 1. Algorithm 1 terminates and outputs a sequence $\sigma = (u_1, u_2, \dots, u_n)$ that is a permutation of the nodes of S , where for all $1 \leq t \leq n$, u_t is a node on the extended external surface of S_{t-1} .*

► **Theorem 13.** *When executed on any connected shape S , Algorithm 1 terminates giving as output an elimination sequence of S .*

Proof. Follows from Lemmas 9 and 12. ◀

The following lemma will be later used to show that the order of colours of the elimination sequence produced by Algorithm 1 respects the colour capacity of a double-line-with-leaves.

► **Lemma 14.** *Consider a connected shape S on which we execute Algorithm 1 and let σ be the elimination sequence produced by the algorithm. There exists a way to split σ into consecutive subsequences $\sigma_1\sigma_2 \dots \sigma_k$ such that every subsequence contains consecutive nodes from σ and has one of the following colour sequences: $bbbr, bbr, br, rrrb, rrb, rb$.*

5.3 Adding Reachability

In the previous section, we showed that, in principle, there exists an elimination sequence. However, in the actual transformation the 6-robot must be able to reach the nodes to be removed. In this section, we show how to restructure a shape that has no cluster of nodes on the reachable part of the extended external surface, so that such a cluster becomes available.

► **Observation 15.** *There exist shapes such that no cluster of nodes on their extended external surface is reachable by the 6-robot. For example, trees concealing their endpoints within narrow spirals.*

In contrast to what holds for orthogonally convex shapes [5], we cannot hope to eliminate a general shape only by directly removing nodes from its extended external surface. As a consequence, further reconfiguration is needed and the elimination sequence of Algorithm 1 must be modified accordingly. First, we extend the definition of the extended external surface to account for nodes that are reachable by the 6-robot.

We say a node u in shape S is reachable if node u resides on the $c \times c$ -reachable boundary of S . The *reachable external surface* of a connected shape A is a shape B , not necessarily connected, consisting of all nodes $u \in A$ such that u occupies a cell defining at least one of the line segments of A 's $c \times c$ -reachable boundary. The *reachable extended external surface* of a connected shape A , is defined by adding to A 's reachable external surface all nodes of A whose cells share a corner with A 's reachable boundary.

Whenever we have a shape S , where every cluster of nodes is not reachable by the 6-robot, we employ a restructuring strategy, where the 6-robot moves nodes around on shape S , until a cluster of nodes can be reached by the 6-robot. First, we modify Algorithm 1 so that the algorithm marks every node of the reachable extended external surface (see Algorithm 2 in the Appendix). This guarantees that every node of the elimination sequence is reachable. Additionally, whenever there exists no cluster of nodes on the reachable extended external

surface, we call a restructuring subroutine. After restructuring is finished, we mark every node of the reachable extended external surface and continue removing clusters. The above two steps are repeated until the shape is empty.

Our strategy for restructuring is based on compression (see Algorithm 3 in the Appendix). This involves the process of creating a cycle on the reachable extended external surface of the shape by adding some auxiliary nodes. Once this is achieved, we show how we can move some nodes on the extended reachable external surface of the shape such that (i) we “compress” the cycle on the extended external surface by removing some nodes from the cycle and making the cycle smaller and (ii) the removed nodes form a cluster of nodes that will reside on the reachable extended external surface of the shape. We show that we can always add auxiliary nodes to the shape, such that we create a cycle C on the extended external surface, where one of the *concave corner* nodes of C is reachable, and either is not a separator node or it is a local separator node.

► **Definition 16.** *Let C be a cycle on the extended external surface of a connected shape S . We say that a node u is a concave corner of cycle C if $u \in C$, u has two neighbouring nodes v, w , where $v, w \in C$, and the cell adjacent to both v and w is part of the interior of the shape.*

► **Definition 17.** *Consider a connected shape S that contains a pocket P . Closing a lid L of pocket P is the process of placing nodes in the empty cells of the pocket P that are adjacent to the pocket lid L .*

► **Lemma 18.** *The number of nodes needed to close a bottleneck lid is at most 7.*

► **Lemma 19.** *Consider any connected shape $S = (V, E)$, where no cluster of nodes resides on the reachable extended external surface. Then, there exists one pocket lid that can be closed such that the new shape S' has a cycle C on the extended external surface, where one of the concave corner nodes of C is both reachable and is either a local separator node or it is not a separator node.*

Using Lemma 19, we can show that after closing a lid, a cluster of nodes can be removed. However, since the number of auxiliary nodes needed to close a lid can be larger than the size of a cluster, this strategy is not guaranteed to succeed. To circumvent this, we use a compression technique starting from concave corner of the shape. After compressing the shape, we still have a cycle C on the extended external surface, where one of the concave corner nodes of C is both reachable and is either a local separator node or it is not a separator node. Additionally, after the compression, we will have a cluster of nodes on the reachable extended external surface that is not part of C . This allows us to compress at least once using the same auxiliary nodes, and then we can remove the auxiliary nodes that were used to close the lid.

► **Lemma 20.** *Consider any connected shape $S = (V, E)$ that has no cluster of nodes on its reachable extended external surface, where we close a lid with the auxiliary node set V_A that creates a cycle C on the extended external surface, where one of the corner nodes u_1 of C is reachable and is a local separator node. There exists a way to compress the shape such that the extended external surface contains a cycle C' , where every auxiliary node is in C' . Additionally, the reachable extended external surface contains a cluster of nodes that is not part of C' .*

► **Theorem 21.** *Let S be a connected shape where no cluster of nodes on the extended external surface is reachable by the 6-robot. Executing Algorithm 3 on S reconfigures S into a connected shape S' that has a reachable cluster of nodes on the extended external surface.*

Proof. Algorithm 3 adds auxiliary nodes to a pocket lid in order to create a cycle C on the extended external surface of S . Lemma 19 guarantees that we can find pocket lid to close with auxiliary nodes, that also creates a reachable concave corner node of C . Then, Lemma 20 guarantees that we can use this node in order to compress the cycle C into a cycle C' , such that a cluster of nodes is reachable by the 6-robot, and that cluster of nodes does not contain any node in C' . Since C' is a cycle, we can remove the auxiliary nodes without disconnecting the shape. ◀

5.4 Generation Sequence

Given a connected shape S of n nodes, a *generation sequence* $\sigma = (u_1, u_2, \dots, u_n)$ of shape S is a permutation of the nodes of S satisfying the following properties. Let $S_t = S_{t-1} \cup \{u_t\}$, where $1 \leq t \leq n$ and $S_0 = \emptyset$. Observe that $S_n = S$. Any shape generation sequence also satisfies the following properties, which it shares with the shape elimination sequence. The first property is that, for all $1 \leq t \leq n - 1$, S_t must be a connected shape. Moreover, for all $1 \leq t \leq n$, u_t must be placed in the cell perimeter of S_{t-1} . Essentially, σ defines a sequence $\emptyset = S_0[u_1]S_1[u_2]S_2[u_3] \dots S_{n-1}[u_n]S_n = S$, where, for all $1 \leq t \leq n$, a connected shape S_t is obtained by adding the node u_t to the cell perimeter of S_{t-1} . The generation sequence that we are going to compute, constructs a double-line-with-leaves.

Given as input an elimination sequence by Algorithm 2, Algorithm 4 will return a generation sequence that constructs a double-line-with-leaves. The algorithm, first constructs the unique bi-coloured pair of the core line and then extends it by placing nodes on both sides of the line.

The algorithm constructs a straight double-line-with-leaves, expects the first two nodes to be a bi-coloured pair and every subsequence to arrive afterwards to have one of the colour sequences as described in Lemma 14. The algorithm positions the first bi-coloured pair horizontally with the red coloured node on the left and the black coloured node on the right. Let (x_l, y_0) be the position of the leftmost node on the core line of the double-line-with-leaves and (x_r, y_0) be the position of the rightmost node on the core line of the double-line-with-leaves.

Every subsequence has size 2, 3 or 4 and has colours br, bbr, bbb or rb, rrb, rrr . If a subsequence arrives starting with a black node (possible subsequences are br, bbr, bbb), the first black node is placed at position (x_{l-1}, y_0) , the last node (which must be red), is placed at (x_{l-2}, y_0) , and any possible other black nodes are placed at positions $(x_l, y_1), (x_l, y_{-1})$. If a subsequence arrives starting with a red node (possible subsequences are rb, rrb, rrr), the first red node is placed at position (x_{r+1}, y_0) , the last node (which must be black), is placed at (x_{r+2}, y_0) , and any possible other red nodes are placed at positions $(x_r, y_1), (x_r, y_{-1})$. The algorithm preserves the invariant that after the placement of the first bi-coloured pair and after the placement of every subsequent subsequence that arrives, the leftmost and rightmost positions of the line, called (x_l, y_0) and (x_r, y_0) , have red and black nodes, respectively, and positions $(x_r, y_1), (x_r, y_{-1}), (x_l, y_1), (x_l, y_{-1})$ are empty. See Algorithm 4 in the Appendix for the pseudocode.

► **Lemma 22.** *Let σ be a bicoloured sequence of nodes that fulfils all the following conditions:*

- *The set of the first two nodes in σ is bi-coloured.*
- *σ can be split into consecutive subsequences $\sigma_1\sigma_2 \dots \sigma_k$ such that every subsequence contains consecutive nodes from σ and also has one of the following colour sequences: $bbb, bbr, br, rrr, rrb, rb$.*

Then there is a double-line-with-leaves generation sequence $\sigma' = (u'_1, u'_2, \dots, u'_n)$ which is colour-order preserving with respect to σ .

► **Theorem 23.** *Given a shape elimination sequence σ computed by Algorithm 2, Algorithm 4 returns a generation sequence which is colour-order-preserving with respect to σ and which constructs a double-line-with-leaves.*

Proof. Follows from Lemmas 14 and 22. ◀

5.5 Wrapping up

To complete the result, we show how the transformation is initialised, including where the double-line-with-leaves will be constructed, and argue that picking the next node in the elimination sequence and placing it on the double-line-with-leaves is always possible.

At the beginning of the transformation, the 6-robot transports 5 nodes from the original seed, and places them as a straight vertical path of length 5, called a *ladder* atop one of the topmost nodes of the initial shape. Then, the first bi-coloured pair that arrives to the elimination sequence is placed perpendicular to the ladder and the double-line-with-leaves extends perpendicular to the ladder. This ladder of 5 nodes guarantees that the construction of the double-line-with-leaves will never create any narrow pockets and this implies that the 6-robot and 7-robot can reach any part of the double-line-with-leaves.

We now show that it is possible to remove nodes from a shape S in the order of a shape elimination sequence generated by Algorithm 2, and place them on the double-line-with-leaves L in the order of a double-line-with-leaves generation sequence created by Algorithm 4, crossing the ladder in the process.

► **Lemma 24.** *Given a shape elimination sequence σ generated by Algorithm 2 for a shape S which starts with the node u , and a double-line-with-leaves generation sequence σ' generated by Algorithm 4 for a double-line-with-leaves L , the 6-robot is able to traverse the 4×4 reachable boundary of $S \cup L$, pick u up, and become a 7-robot. It can then traverse the 5×5 reachable boundary of $(S \setminus \{u\}) \cup L$ and place u on L .*

► **Theorem 25.** *Let σ be the shape elimination sequence generated by Algorithm 2 for a shape S , and a double-line-with-leaves generation sequence σ' which is colour-order preserving with respect to σ (as generated by Algorithm 4), the 6-robot can remove nodes from S according to σ and construct the double-line-with-leaves according to σ' .*

Putting everything together, including the fact that a 6-robot can be used to transform any pair of colour-consistent double-lines-with-leaves into each other, we get:

► **Theorem 26.** *Let S and S' be any connected colour-consistent shapes. Then, there is a connected shape M of $d = O(1)$ nodes and a placement of M on the perimeter of S , such that $S \cup M$ can be transformed into S' via $O(n^2)$ rotation movements.*

6 Conclusions

We have shown that by using a seed of constant size, it is possible to transform any pair of connected shapes A and B on a two-dimensional square grid into each other in an optimal $O(n^2)$ time. This leaves a few open problems to be addressed. First, the issue of creating a distributed version of the algorithm. This will not only make the algorithm more immediately applicable to real-world programmable matter scenarios, which usually assume that each module acts independently, but also opens up the possibility of a “pipelined” or parallel version which may be able to perform the transformation in only $O(n)$ parallel time movements.

Another issue is the size of the seed. It may be possible to reduce the seed's size to as little as 3 nodes, which would be equivalent to [5]. However, if all removable nodes are concealed then this might not be enough, unless a new approach, possibly by “drilling” into boundaries of the shape, is adopted.

Third, a potential comparison could be made to the pivoting model result of [1], to compare the similarities and differences of each approach to universal transformation. Finally, it may be possible to extend the results from shapes on a two-dimensional square grid, to those in a three-dimensional environment. Universal transformation in a three-dimensional environment which does not disconnect the shape is another challenging goal with interesting potential applications.

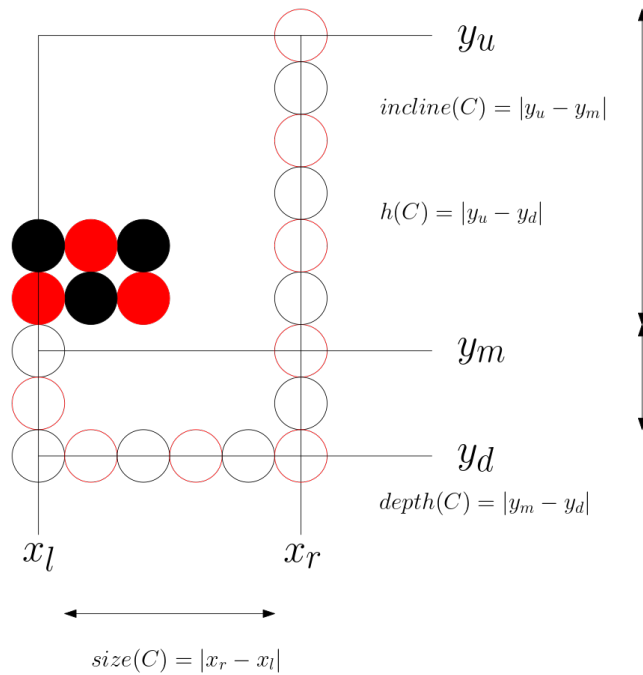
References

- 1 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmović, Robin Flatland, Matias Korman, Belen Palop, Irene Parada, André van Renssen, and Vera Sacristán. Universal Reconfiguration of Facet-Connected Modular Robots by Pivots: The $O(1)$ Musketeers. *Algorithmica*, 83(5):1316–1351, May 2021. doi:10.1007/s00453-020-00784-6.
- 2 Nada Almalki and Othon Michail. On geometric shape construction via growth operations. *Theoretical Computer Science*, 984:114324, February 2024. doi:10.1016/j.tcs.2023.114324.
- 3 Abdullah Almethen, Othon Michail, and Igor Potapov. Pushing lines helps: Efficient Universal Centralised Transformations for Programmable Matter. *Theoretical Computer Science*, 830-831:43–59, August 2020. doi:10.1016/j.tcs.2020.04.026.
- 4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in Networks of Passively Mobile Finite-State Sensors. *Distributed Computing*, 18(4):235–253, March 2006. doi:10.1007/s00446-005-0138-3.
- 5 Matthew Connor and Othon Michail. Centralised connectivity-preserving transformations by rotation: 3 musketeers for all orthogonal convex shapes. In *Proceedings of the 18th International Symposium on Algorithmics of Wireless Networks (ALGOSENSORS)*, pages 60–76, 2022.
- 6 Matthew Connor, Othon Michail, and Igor Potapov. Centralised Connectivity-Preserving Transformations for Programmable Matter: A Minimal Seed Approach. *Theoretical Computer Science*, November 2022. doi:10.1016/j.tcs.2022.09.016.
- 7 Joshua J. Daymude, Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. On the Runtime of Universal Coating for Programmable Matter. *Natural Computing*, 17(1):81–96, March 2018. doi:10.1007/s11047-017-9658-6.
- 8 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Amoebot - a new model for programmable matter. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 220–222, 2014. doi:10.1145/2612669.2612712.
- 9 Zahra Derakhshandeh, Robert Gmyr, Andrea W. Richa, Christian Scheideler, and Thim Strothmann. Universal Shape Formation for Programmable Matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–299, 2016. doi:10.1145/2935764.2935784.
- 10 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. An Algorithmic Framework for Shape Formation Problems in Self-Organizing Particle Systems. In *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication (NANOCOM)*, pages 1–2, 2015. doi:10.1145/2800795.2800829.
- 11 David Doty. Theory of Algorithmic Self-Assembly. *Communications of the ACM*, 55(12):78–88, December 2012. doi:10.1145/2380656.2380675.
- 12 David Doty. Timing in Chemical Reaction Networks. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 772–784. December 2013. doi:10.1137/1.9781611973402.57.

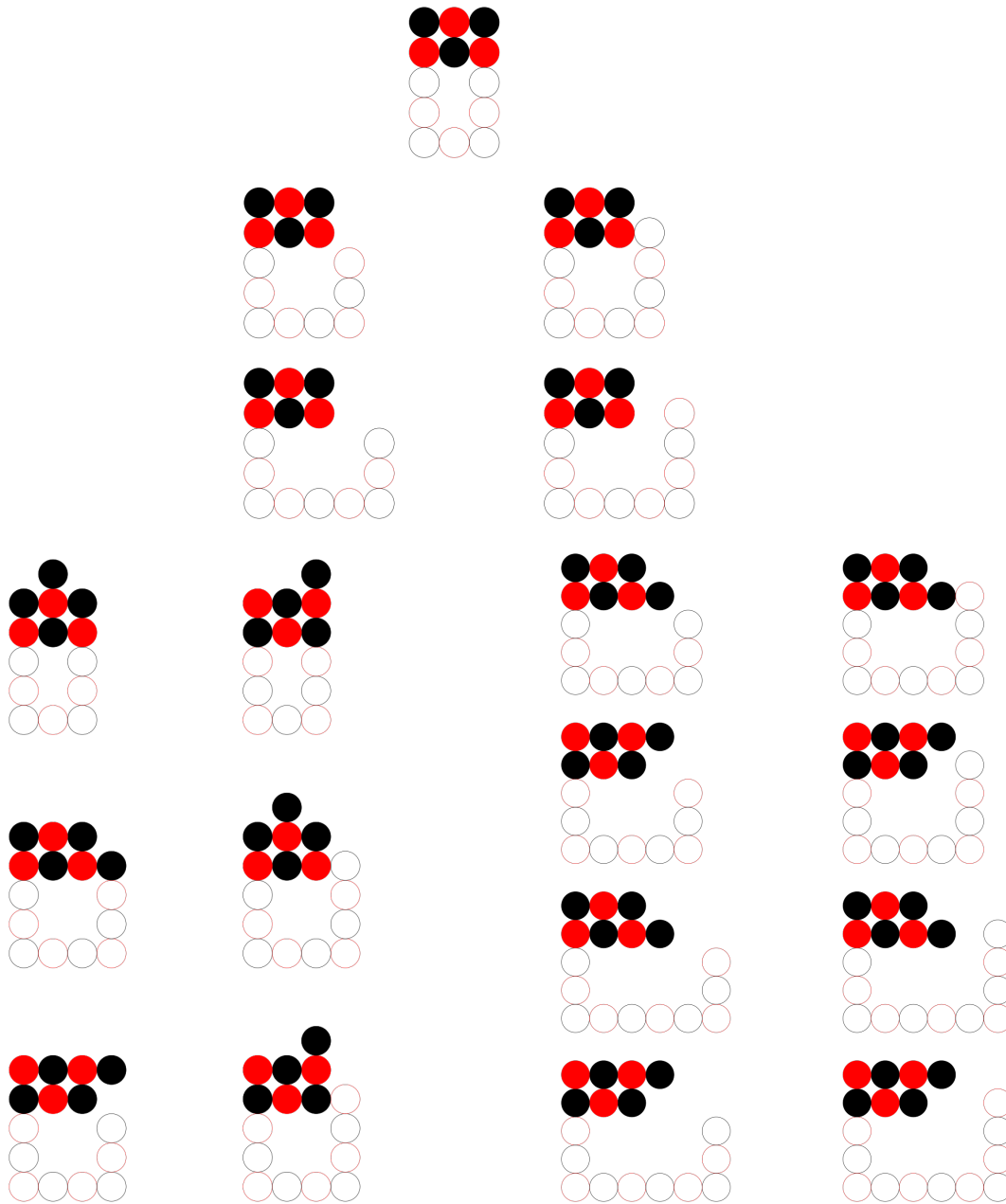
- 13 Adrian Dumitrescu and János Pach. Pushing squares around. In *Proceedings of the Twentieth ACM Annual Symposium on Computational Geometry (SCG)*, pages 116–123, 2004.
- 14 Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Motion planning for metamorphic systems: Feasibility, decidability, and distributed reconfiguration. *IEEE Transactions on Robotics and Automation*, 20(3):409–418, 2004.
- 15 Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. Connected coordinated motion planning with bounded stretch. *Autonomous Agents and Multi-Agent Systems*, 37(2):43, October 2023.
- 16 Michael Feldmann, Andreas Padalkin, Christian Scheideler, and Shlomi Dolev. Coordinating Amoebots via Reconfigurable Circuits. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, 29(4):317–343, April 2022. doi:10.1089/cmb.2021.0363.
- 17 Kyle Gilpin, Ara Knaian, and Daniela Rus. Robot Pebbles: One Centimeter Modules for Programmable Matter through Self-Disassembly. In *2010 IEEE International Conference on Robotics and Automation*, pages 2485–2492, 2010. doi:10.1109/ROBOT.2010.5509817.
- 18 Camille Jordan. *Cours d'analyse de l'École polytechnique*, volume 1. Gauthier-Villars et fils, 1893.
- 19 George B. Mertzios, Othon Michail, George Skretas, Paul G. Spirakis, and Michail Theofilatos. The complexity of growing a graph. In *18th International Symposium on Algorithmics of Wireless Networks (ALGOSENSORS)*, pages 123–137, 2022.
- 20 Othon Michail, George Skretas, and Paul G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. In *44th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 136:1–136:15, 2017. doi:10.4230/LIPIcs.ICALP.2017.136.
- 21 Othon Michail, George Skretas, and Paul G. Spirakis. On The Transformation Capability of Feasible Mechanisms for Programmable Matter. *Journal of Computer and System Sciences*, 102:18–39, June 2019. doi:10.1016/j.jcss.2018.12.001.
- 22 Othon Michail and Paul G. Spirakis. Simple and Efficient Local Codes for Distributed Stable Network Construction. *Distributed Computing*, 29(3):207–237, June 2016. doi:10.1007/s00446-015-0257-4.
- 23 Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13:195–224, June 2014.
- 24 Paul W. K. Rothemund. Folding DNA to Create Nanoscale Shapes and Patterns. *Nature*, 440(7082):297–302, March 2006. doi:10.1038/nature04586.
- 25 Paul W. K. Rothemund and Erik Winfree. The Program-size Complexity of Self-Assembled Squares (extended abstract). In *Proceedings of the thirty-second Annual ACM Symposium on Theory of Computing (STOC)*, pages 459–468, May 2000. doi:10.1145/335305.335358.
- 26 Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable Self-assembly in a Thousand-Robot Swarm. *Science*, 345(6198):795–799, August 2014. doi:10.1126/science.1254295.
- 27 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with Finite Stochastic Chemical Reaction Networks. *Natural Computing*, 7(4):615–633, December 2008. doi:10.1007/s11047-008-9067-y.
- 28 Pierre Thalamy, Benoit Piranda, and Julien Bourgeois. Distributed Self-Reconfiguration using a Deterministic Autonomous Scaffolding Structure. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 140–148, 2019.
- 29 Pierre Thalamy, Benoit Piranda, and Julien Bourgeois. 3D Coating Self-Assembly for Modular Robotic Scaffolds. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11688–11695, 2020. ISSN: 2153-0866. doi:10.1109/IR0S45743.2020.9341324.
- 30 Erik Winfree. *Algorithmic Self-assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

- 31 Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active Self-Assembly of Algorithmic Shapes and Patterns in Polylogarithmic Time. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science (ITCS)*, pages 353–354, 2013. doi:10.1145/2422436.2422476.

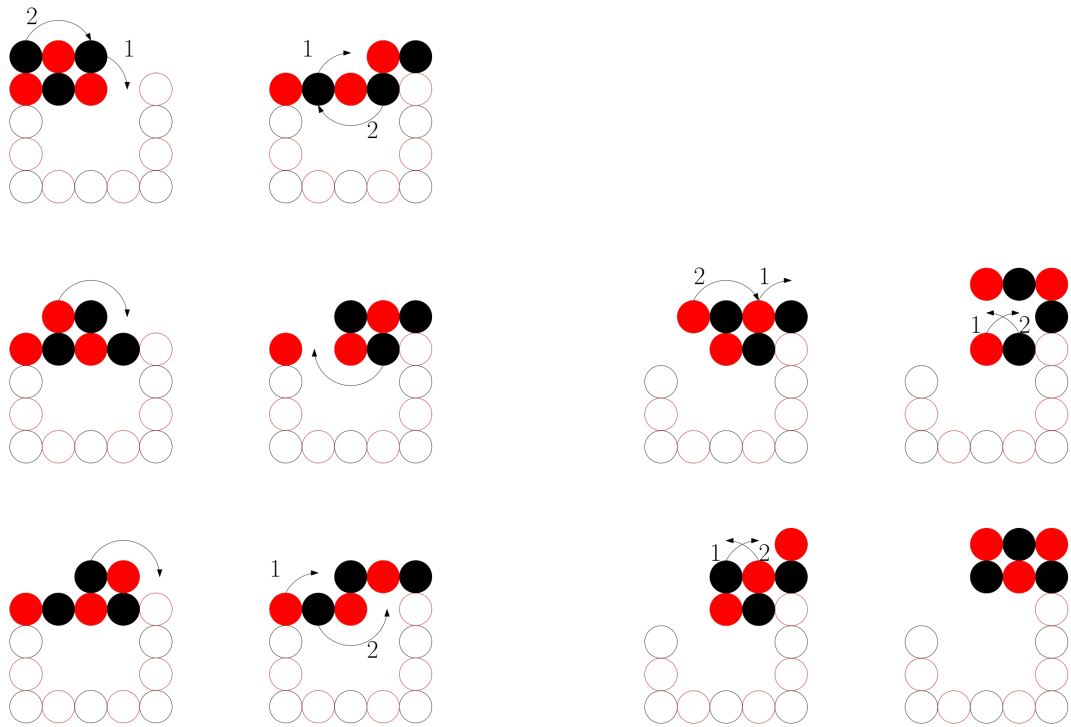
A Appendix



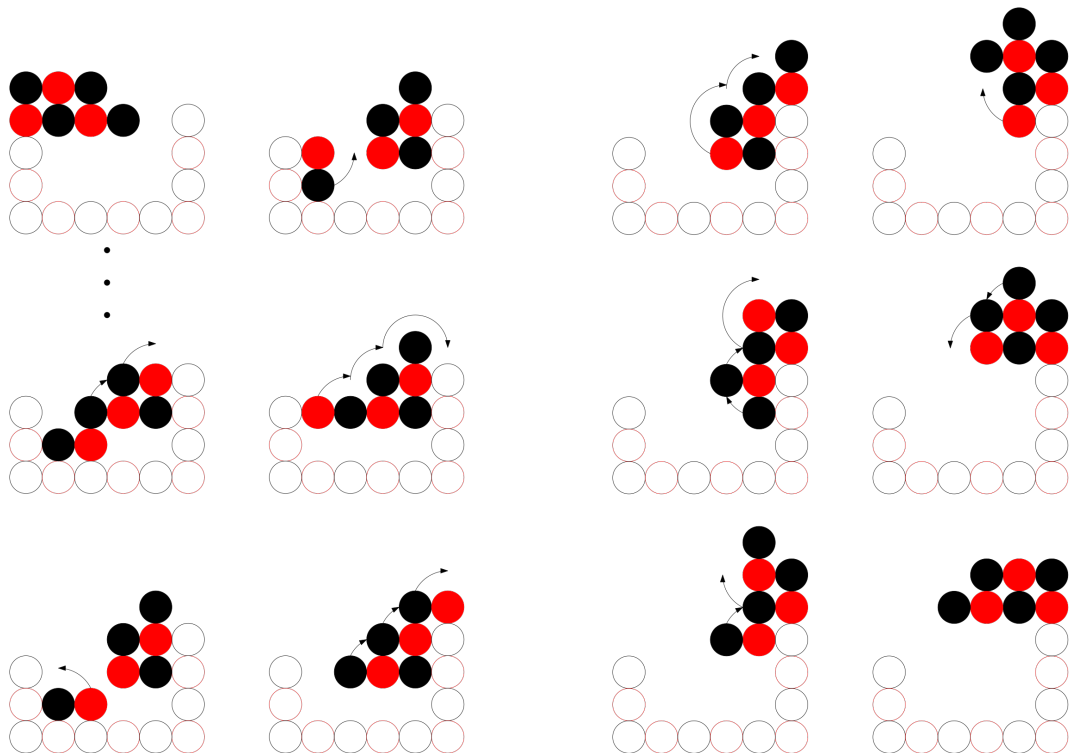
■ **Figure 2** A visual representation of a gap, with the variables we use in our proof.



■ **Figure 3** The main cases we consider in our proof. For all cases where the *incline* > 1 , we only need to show that the robot can reach the other side of the gap, all movement afterwards is equivalent to climbing movements from [5]. All cases with *incline* < 1 are mirrored versions of cases with *incline* > 1 . Our movements generally do not depend on connectivity with the bottom, so *depth* is mostly irrelevant, with the exception of a single edge case. All gaps with a size greater than those considered here are part of the 4×4 or 5×5 -reachable boundary, for the 6-robot and 7-robot respectively, and per Lemma 2 can be traversed.



■ **Figure 4** 3-node inclined pocket traversal for $incline = 1$. Figures are read in columns, top-down.



■ **Figure 5** 4-node pocket traversal for the 7-robot with the load in the bottom position with $incline = 1$.

■ **Algorithm 1** Algorithm that computes a shape elimination sequence which does not require the nodes to be reachable by the 6-robot.

Input: Connected shape $S = (V, E)$
Output: Elimination sequence σ

```

1 while ( $S$  is not empty) do
2   Mark every node  $u$  that belongs to the extended external surface of  $S$ ;
3   while (there exists a marked pair of nodes  $u, v$  satisfying property 1 of Definition 8)
4     do
5       Remove  $u, v$  from  $S$  and append them to  $\sigma$  in the order specified by
6       Definition 8;
7     end
8     while (there exists a marked node  $u$  satisfying property 2 of Definition 8) do
9       Remove the leaf neighbours of  $u$  from  $S$  and append them to  $\sigma$ ;
10      Remove  $u$  from  $S$  and append it to  $\sigma$ ;
11    end
12  Unmark every marked node of  $S$ ;
13 end
14 return  $\sigma$ 

```

■ **Algorithm 2** Algorithm that computes a shape elimination sequence.

Input: Connected shape $S = (V, E)$, $k = 0$
Output: Elimination sequence σ

```

1 while ( $S \neq \emptyset$ ) do
2   Mark every node that belongs to the reachable extended external surface of  $S$ ;
3    $k = 0$ ;
4   while (there exists a marked pair of nodes  $u, v$  satisfying property 1 of Definition 8)
5     do
6        $k++$ ;
7       Remove  $u, v$  from  $S$  and append them to  $\sigma$  in the order specified by
8       Definition 8;
9     end
10    while (there exists a marked node  $u \in S$  satisfying property 2 of Definition 8) do
11       $k++$ ;
12      Remove the leaf neighbours of  $u$  from  $S$  and append them to  $\sigma$ ;
13      Remove  $u$  from  $S$  and append it to  $\sigma$ ;
14    end
15    Unmark every marked node  $u \in S$ ;
16    if  $k = 0$  then
17      Call Algorithm 3 with input  $S = (V, E)$ ;
18    end
19  end
20 return  $\sigma$ 

```

■ **Algorithm 3** Algorithm that reconfigures a connected shape to have a cluster of nodes on the reachable extended external surface.

Input: Connected shape $S = (V, E)$ with no cluster of nodes on the reachable extended external surface

Output: Connected shape $S' = (V, E')$ with a cluster of nodes on the reachable extended external surface

- 1 Close a lid using up to 7 auxiliary nodes to create a cycle C with the properties of Lemma 19;
- 2 Compress the cycle C ;
- 3 Remove the auxiliary nodes used to close the lid;
- 4 **return** S'

■ **Algorithm 4** Algorithm that constructs a double-line-with-leaves generation sequence.

Input: Elimination Sequence σ split into subsequences

Output: Double-line-with-leaves generation sequence $\sigma' = (u'_1, u'_2, \dots, u'_n)$ which is colour-order preserving with respect to σ

- 1 **if** $c(u_1) == \text{red}$ and $c(u_2) == \text{black}$ **then**
- 2 | $u'_1 = (x_l, y_0), u'_2 = (x_{l+1}, y_0)$;
- 3 **else**
- 4 | $u'_1 = (x_{l+1}, y_0), u'_2 = (x_l, y_0)$;
- 5 **end**
- 6 Remove u_1, u_2 from σ ;
- 7 $i = 3, b = 0, r = 0$;
- 8 **while** $\sigma \neq \emptyset$ **do**
- 9 | Remove the next subsequence σ_j from σ ;
- 10 | **if** $c(u_i) == \text{black}$ **then**
- 11 | | $u'_i = (x_{l-b-1}, y_0)$;
- 12 | | **if** $|\sigma_j| == 2$ **then**
- 13 | | | $u'_{i+1} = (x_{l-b-2}, y_0)$;
- 14 | | **else if** $|\sigma_j| == 3$ **then**
- 15 | | | $u'_{i+1} = (x_{l-b}, y_1), u'_{i+2} = (x_{l-b-2}, y_0)$;
- 16 | | **else if** $|\sigma_j| == 4$ **then**
- 17 | | | $u'_{i+1} = (x_{l-b}, y_1), u'_{i+2} = (x_{l-b}, y-1), u'_{i+3} = (x_{l-b-2}, y_0)$;
- 18 | | $b = b + 2, i = i + |\sigma_j|$;
- 19 | **else if** $c(u_i) == \text{red}$ **then**
- 20 | | $u'_i = (x_{l+1+r+1}, y_0)$;
- 21 | | **if** $|\sigma_j| == 2$ **then**
- 22 | | | $u'_{i+1} = (x_{l+1+r+2}, y_0)$;
- 23 | | **else if** $|\sigma_j| == 3$ **then**
- 24 | | | $u'_{i+1} = (x_{l+1+r}, y_1), u'_{i+2} = (x_{l+1+r+2}, y_0)$;
- 25 | | **else if** $|\sigma_j| == 4$ **then**
- 26 | | | $u'_{i+1} = (x_{l+1+r}, y_1), u'_{i+2} = (x_{l+1+r}, y-1), u'_{i+3} = (x_{l+1+r+2}, y_0)$;
- 27 | | $r = r + 2, i = i + |\sigma_j|$;
- 28 | **end**
- 29 **end**

On the Complexity of Temporal Arborescence Reconfiguration

Riccardo Dondi  

Università degli Studi di Bergamo, Italy

Manuel Lafond  

Université de Sherbrooke, Canada

Abstract

We analyze the complexity of ARBORESCENCE RECONFIGURATION on temporal digraphs (TEMPORAL ARBORESCENCE RECONFIGURATION). The problem, given two temporal arborescences in a temporal digraph, asks for the minimum number of arc flips, i.e. arc exchanges, that result in a sequence of temporal arborescences that transforms one into the other. We analyze the complexity of the problem, taking into account also its approximation and parameterized complexity, even in restricted cases. First, we solve an open problem showing that TEMPORAL ARBORESCENCE RECONFIGURATION is NP-hard for two timestamps. Then we show that even if the two temporal arborescences differ only by two arcs, then the problem is not approximable within factor $b \ln |V(D)|$, for any constant $0 < b < 1$, where $V(D)$ is the set of vertices of the temporal arborescences. Finally, we prove that TEMPORAL ARBORESCENCE RECONFIGURATION is W[1]-hard when parameterized by the number of arc flips needed to transform one temporal arborescence into the other.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory

Keywords and phrases Arborescence, Temporal Graphs, Graph Algorithms, Parameterized Complexity, Approximation Complexity

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.10

Acknowledgements The authors thank the anonymous reviewers for their comments on the paper. The authors also thank Shun-ichi Maezawa for introducing the problem to the authors.

1 Introduction

Arborescences, also called branchings, have been deeply studied in theoretical computer science. Given a digraph (a directed graph) and a special vertex, called the root, an arborescence is a directed rooted tree in the digraph that connects the root to every vertex of the digraph. The computation of arborescences of a given digraph finds several applications, for example in communications networks, where the goal is to compute a shortest way to reach some devices [18], to analyze information flow in social networks [3], or in computational biology to analyze mass spectrometry data [7] and reconstruct tumor evolutionary trees [8].

Arborescences have been recently considered also in the temporal graph setting [15, 11, 4, 13], where they can model urban mobility or information dissemination in social networks. Temporal graphs have been studied to model the dynamic evolution of network relations (edges or arcs), that are observed only at certain time instants [17, 9, 19, 10, 1]. In our model of a temporal digraph $D = (V, A)$, the arcs are triples (u, v, t) , where u and v are vertices and t is a positive integer, representing that the arc from u to v is seen at timestamp t . A *temporal arborescence* T in D is a rooted tree, whose arcs are directed away from the root, that contains every vertex of D and such that every path in T is *time-respecting*, that is the timestamps on the arcs of every path are non-decreasing.



© Riccardo Dondi and Manuel Lafond;

licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 10; pp. 10:1–10:15

Leibniz International Proceedings in Informatics



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

In this contribution, we consider temporal arborescences through the lens of *combinatorial reconfiguration* [12, 21]. Given two feasible solutions of a problem (in our case being temporal arborescences of a temporal digraph), combinatorial reconfigurations explore the space of feasible solutions and the distance between the two given solutions. Two feasible solutions are adjacent if they can be transformed one into the other by means of a local operation (such as exchanging two arcs). The goal of combinatorial reconfiguration is to study the reachability of two elements of the space of feasible solutions, that is the possibility of transforming the first solution into the second one by means of sequences of local operations, and possibly obtaining a comparative metric by minimizing the number of such operations.

Given two temporal arborescences T_1 and T_2 in D , a reconfiguration of T_1 into T_2 is a transformation of T_1 into T_2 with a sequence of modifications, one at a time, called *arc flips*, where each modification exchanges two arcs. Note that an arc flip may exchange any two arcs of D , the only constraint is that, by applying arc flips, a reconfiguration may compute intermediate subgraphs, that must all be temporal arborescences in D .

We consider a problem related to the reconfiguration of temporal arborescences, called TEMPORAL ARBORESCENCE RECONFIGURATION, introduced in [13]. Given a digraph D , and two arborescences T_1 and T_2 in D , TEMPORAL ARBORESCENCE RECONFIGURATION asks to compute a reconfiguration of T_1 into T_2 consisting of the minimum number of operations. The problem is known to be NP-hard when the temporal graph is defined over 3 timestamps or more [13], and polynomial-time solvable when the number of timestamps is 1, since in this case the digraph is static and for this case TEMPORAL ARBORESCENCE RECONFIGURATION can be solved in polynomial time [14]. The case of 2 timestamps remained open [13].

An interesting property shown in [13], is that the complexity of TEMPORAL ARBORESCENCE RECONFIGURATION depends on whether the two input temporal arborescences have the same root or not. In the former case, the problem is solvable in polynomial-time, while in the latter the problem is NP-hard, as discussed before.

A decision problem related to TEMPORAL ARBORESCENCE RECONFIGURATION studied in the literature is the reachability of two feasible solutions, that is whether, given two temporal arborescences, one can be transformed into the other (without the requirement of minimizing the number of arc flips). This decision problem is solvable in polynomial time [13] and always admits a positive answer in static directed graphs [14] and when the two arborescences have the same root [13].

Our Results. In this paper we further analyze the complexity of TEMPORAL ARBORESCENCE RECONFIGURATION, considering additional restrictions in the approximation and parameterized complexity frameworks. Note that we consider the temporal graph model of [13], which is a restricted model where each timestamp of an arc specifies its activation time and the arc is present for all times after the activation time. The hardness results we present hold also in this restricted model.

First, we solve the open problem in [13] for the case of two timestamps, and we show in Section 3 that this restriction of TEMPORAL ARBORESCENCE RECONFIGURATION is NP-hard.

Then we consider the case when the two input temporal arborescences are very similar, that is they differ only for a limited number of arcs. We show in Section 4 that if the two temporal arborescences differ by two arc pairs, then the problem is not only NP-hard, but also inapproximable within factor $b \ln |V(D)|$, for any constant $0 < b < 1$, where $V(D)$ is the set of vertices of the arborescences. We also observe that if the two temporal arborescences

differ for one pair of arcs, then the problem is easily solvable in polynomial time. Note that the result can be easily extended to the case where two temporal arborescences differ by more than two arc pairs. For example, we can replicate the construction of Fig. 2 and Fig. 3 by adding many copies of the subtree rooted at y and of the subtrees rooted at $v_{i,j}$. Each copy has to be reconfigured independently, thus the inapproximation ratio is the same as in our result.

Finally, we consider the parameterized complexity of the problem, where the parameter is the number of arc flips required by a reconfiguration. We prove in Section 5 that the problem is $W[1]$ -hard for this parameter (it is, in fact, $W[1]$ -hard in the parameter “number of arc flips plus maximum timestamp”), indicating that a fixed-parameter algorithm is unlikely. We conclude the paper with Section 6 with some open problems. Note that some of the proofs are not included due to page limit.

2 Preliminaries

A *temporal digraph* $D = (V, A)$ is a pair where V is the set of vertices and $A \subseteq V \times V \times \mathbb{N}$ is a set of (temporal) arcs. Note that an arc in a temporal graph is denoted by a triple (u, v, t) , where $u \in V$ is the tail of the arc, $v \in V$ is the head of the arc, and $t \in \mathbb{N}$ is called a timestamp. In our version of a temporal graph, an arc (u, v, t) remains *active* from this timestamp t , that is, once it is activated it exists in the temporal digraph from time t and onwards. We may write $V(D)$ and $A(D)$ for the vertex and arc set of D , respectively. Note that we allow multiple arcs between two vertices u and v , but they must be at different timestamps.

For a triple $e = (u, v, t)$, $D - e$ (resp. $D + e$) is the temporal digraph obtained by removing the arc e , if present (resp. adding the arc e , if absent).

An *arborescence* is a digraph in which there is a vertex u , called the *root*, such that there is a unique directed path from u to any vertex. In other words, an arborescence is a tree in which arcs are oriented away from the root. Let $D = (V, A)$ be a digraph. A subgraph T of D is a *spanning arborescence* of D if $V(T) = V(D)$ and T is an arborescence. Unless stated otherwise, all arborescences are spanning, and we may simply call T an arborescence of D .

Given a temporal graph D , a *temporal arborescence* T of D is an arborescence of D , such that T is *time-respecting*, that is for any pair of arcs $(u, v, t), (v, w, t') \in A(T)$ that are consecutive on some path of T , we have $t \leq t'$.

An *arc flip* on a temporal arborescence T of D is an operation that removes an arc $(u, v, t) \in A(T)$ and inserts an arc $(x, y, t') \in A(D) \setminus A(T)$, such that $T - (u, v, t) + (x, y, t')$ is a temporal arborescence of D (hence spanning and time-respecting).

A *reconfiguration* of a temporal arborescence T_1 of D is a sequence of arc flips, each one producing a temporal arborescence. A *reconfiguration* from T_1 to T_2 is a reconfiguration that transforms T_1 into T_2 . A *reconfiguration sequence* $\mathcal{R} = (R_1, R_2, \dots, R_l)$ from T_1 to T_2 is a sequence of temporal arborescences, where $R_1 = T_1$ and $R_l = T_2$ such that each R_i , with $i \in [l]$, is a temporal arborescence of D and each R_j , $j \in \{2, \dots, l\}$, can be obtained from R_{j-1} with an arc flip.

Now, we are ready to define the problem we are interested into.

► **Problem 1.** (TEMPORAL ARBORESCENCE RECONFIGURATION)

Input: a temporal digraph D , two temporal arborescences T_1, T_2 of D , and an integer $p \geq 1$.

Question: Does there exist a reconfiguration from T_1 to T_2 of at most p arc flips?

In the optimization version of TEMPORAL ARBORESCENCE RECONFIGURATION, we aim to minimize the number of arc flips.

3 NP-Hardness for Two Timestamps

We show that the TEMPORAL ARBORESCENCE RECONFIGURATION problem is NP-hard even on two timestamps (i.e. each arc has timestamp in $\{1, 2\}$) via a reduction from the SET COVER problem. Let (S, U, k) be an instance of SET COVER, where U is the universe, S is a collection of subsets of U , and k is an integer. The question is whether there exists a subcollection $S^* \subseteq S$ of at most k sets of S such that for each $u \in U$ there exists at least one set of S^* that contains u .

We denote $U = \{u_1, \dots, u_n\}$ and $S = \{S_1, \dots, S_m\}$, and we define a corresponding instance $(D = (V, A), T_1, T_2, p)$ of TEMPORAL ARBORESCENCE RECONFIGURATION. First let $S' = \{S'_i : S_i \in S\}$ be a copy of S and let $U' = \{u'_i : u_i \in U\}$ be a copy of U . We let

$$V = \{r_1, r_2, r_3\} \cup S \cup S' \cup U \cup U'.$$

We then add to A the following sets of arcs (we strongly recommend referring to Figure 1):

- $A_r = \{(r_1, r_2, 1), (r_2, r_1, 2), (r_1, r_3, 2), (r_3, r_2, 1)\}$;
- $A_{r_1, U} = \{(r_1, u_i, 1) : u_i \in U\}$;
- $A_{U, U'} = \{(u_i, u'_i, 1) : u_i \in U\}$;
- $A_{r_2, S} = \{(r_2, S_i, 2) : S_i \in S\}$;
- $A_{r_2, S'} = \{(r_2, S'_i, 1) : S_i \in S\}$;
- $A_{r_2, U} = \{(r_2, u_i, 2) : u_i \in S\}$;
- $A_{S, S'} = \{(S_i, S'_i, 2) : S_i \in S\}$;
- $A_{S', U'} = \{(S'_i, u'_j, 1) : S_i \in S \wedge u_j \in S_i\}$;
- $A_{r_3, S'} = \{(r_3, S'_i, 1) : S_i \in S\}$;
- $A_{r_3, U'} = \{(r_3, u'_i, 1) : u_i \in U\}$.

Note that $A_{S', U'}$ is the main set of arcs used to model the set cover instance into D . Finally, we define the input temporal arborescences T_1 (rooted at r_1) and T_2 (rooted at r_3) by specifying their arcs (illustrated in Figure 1, top-right and bottom-right, respectively):

$$\begin{aligned} A(T_1) &= \{(r_1, r_2, 1), (r_1, r_3, 2)\} \cup A_{r_1, U} \cup A_{U, U'} \cup A_{r_2, S} \cup A_{S, S'} \\ A(T_2) &= \{(r_3, r_2, 1), (r_2, r_1, 2)\} \cup A_{r_2, U} \cup A_{r_2, S} \cup A_{r_3, S'} \cup A_{r_3, U'} \end{aligned}$$

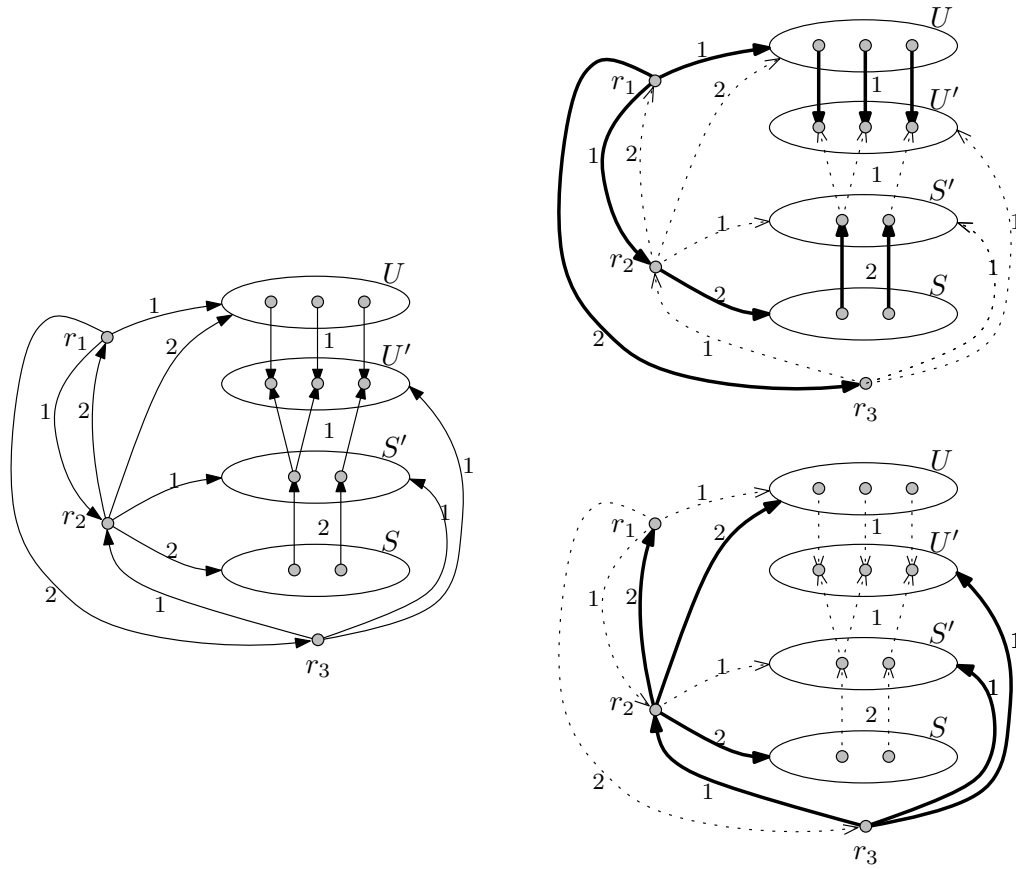
One can verify that T_1 and T_2 are temporal arborescences using Figure 1.

► **Theorem 1.** *The TEMPORAL ARBORESCENCE problem is NP-hard even when the maximum timestamp of an arc is 2.*

Proof. Using the construction described above, we show that there exists $S^* \subseteq S$ of size at most k that covers U if and only if T_1 can be transformed into T_2 using at most $3n + m + 2 + k$ arc flips.

Suppose that there exists $S^* \subseteq S$ of size at most k that covers U . We reconfigure T_1 into T_2 as follows (we say that an arc flip is correct if, after applying it, the resulting subgraph is a temporal arborescence, hence time-respecting).

1. For each $S_i \in S^*$ in an arbitrary order, remove $(S_i, S'_i, 2)$ and add $(r_2, S'_i, 1)$.
Each such arc flip is correct, since r_1 can reach S'_i through $(r_1, r_2, 1), (r_2, S'_i, 1)$.
2. For each $u'_j \in U'$ in an arbitrary order, let S_i be a set of S^* that contains u_j . Remove $(u_j, u'_j, 1)$ and add $(S'_i, u'_j, 1)$, which exists by construction.
Each arc flip is correct since r_1 can reach u'_j through the path $r_1 \rightarrow r_2 \rightarrow S'_i \rightarrow u'_j$ using arcs of timestamp 1 only. Note that at this stage, r_2 reaches the vertices in S, S' , and U' without going through r_1 .



■ **Figure 1** Left: the temporal digraph D obtained from a set cover instance, with $U = \{u_1, u_2, u_3\}$ and $S = \{S_1, S_2\}$, $S_1 = \{u_1, u_2\}$ and $S_2 = \{u_3\}$. Arcs pointing on ellipses indicate that all possible arcs are present (e.g. r_1 has every element of U in its out-neighborhood). The arborescences T_1 and T_2 are shown in thick arcs, top-right and bottom-right, respectively.

3. For each $u_j \in U$ in an arbitrary order, remove $(r_1, u_j, 1)$ and add $(r_2, u_j, 2)$.
Each arc flip is correct since r_1 can reach u_j using the time-respecting path $r_1 \rightarrow r_2 \rightarrow u_j$. At this stage, r_2 also reaches the vertices of U without going through r_1 .
4. Reroot to r_2 by removing $(r_1, r_2, 1)$ and adding $(r_2, r_1, 2)$.
This arc flip is correct since before the arc flip, r_2 was already able to reach each element of U, U', S', S without r_1 , and can now reach r_1 and r_3 through the time-respecting path $r_2 \rightarrow r_1 \rightarrow r_3$.
5. Reroot to r_3 by removing $(r_1, r_3, 2)$ and adding $(r_3, r_2, 1)$.
This arc flip is correct since r_3 reaches r_2 at time 1, and thus r_3 can reach r_1, U, U', S', S through r_2 with a time-respecting path.
6. For $u'_j \in U'$ in an arbitrary order, remove the incoming arc incident to u'_j and add $(r_3, u'_j, 1)$. This is easily seen to be correct since U' vertices are leaves before (and after) the arc flips.
7. For $S'_i \in S'$ in an arbitrary order, remove the incoming arc incident to S'_i and add $(r_3, S'_i, 1)$. This is easily seen to be correct since, because of the previous step, the S' vertices are leaves before (and after) the arc flips.

One can check that this sequence of flips yields T_2 . As for the number of arc flips, by summing the number of arc flips required for each of the above steps, we see that we require at most $|S^*| + |U'| + |U| + 1 + 1 + |U'| + |S'| \leq k + 3n + m + 2$, as desired.

10:6 On the Complexity of Temporal Arborecence Reconfiguration

In the converse direction, suppose that there exists a reconfiguration sequence $\mathcal{R} = (R_1, R_2, \dots, R_l)$ from T_1 to T_2 with $l - 1 \leq 3n + m + 2 + k$, where $T_1 = R_1$ and $T_2 = R_l$, and each R_i can be obtained from R_{i-1} with an arc flip, for $i \in \{2, \dots, l\}$. We gather a set of facts to prove that U can be covered by at most k sets of S .

► **Fact 1.** *For each $i \in [l]$, the root of R_i is one of r_1, r_2 , or r_3 .*

Fact 1 holds because only r_1, r_2 , and r_3 can reach r_1 in D .

► **Fact 2.** *If r_1 is the root of R_i for some $i \in [l]$, then $(r_1, r_3, 2) \in A(R_i)$ and $(r_1, r_2, 1) \in A(R_i)$.*

Fact 2 is true because $(r_1, r_3, 2)$ is the only incoming arc of r_3 and must thus be in R_i . This prevents using $(r_3, r_2, 1)$ because of the time-respecting condition. The only other incoming arc of r_2 is $(r_1, r_2, 1)$ and it must thus be in R_i as well.

► **Fact 3.** *If r_1 is the root of R_i for some $i \in [l - 1]$, then r_3 is not the root of R_{i+1} .*

To see that Fact 3 holds, we know by Fact 2 that $(r_1, r_3, 2), (r_1, r_2, 1) \in A(R_i)$. To make r_3 the root in R_{i+1} we have to remove $(r_1, r_3, 2)$, and add some outgoing arc of r_3 . But adding $(r_3, r_2, 1)$ makes r_2 of in-degree 2, and adding an arc from r_3 to some element of $S' \cup U'$ makes it impossible to reach r_1 from r_3 . Therefore, the root of R_{i+1} is either r_1 or r_2 .

We now proceed with the construction of a set cover. Let $a \in [l]$ be the minimum index such that r_2 is the root of R_a (note that there must exist such a R_a since the root of T_2 is r_3 and by Fact 3 the re-rooting from r_1 to r_3 cannot be done with an arc flip). By Fact 1 and Fact 3, we know that r_1 is the root of R_{a-1} , so that R_a is the first time the root is switched. By Fact 2, $(r_1, r_2, 1) \in A(R_{a-1})$ and, because $(r_2, r_1, 2)$ is the only incoming arc of r_1 , the only way to switch the root from r_1 to r_2 is by removing $(r_1, r_2, 1)$ and adding $(r_2, r_1, 2)$. This means that in R_{a-1} , there cannot be an arc from r_1 to U , as otherwise $(r_2, r_1, 2)$ followed by such an arc would not be time-respecting. This implies that in R_{a-1} , all arcs from r_2 to U are present, since these are the only other incoming arcs of the U vertices. This in turn implies that in R_{a-1} , there cannot be an arc from U to U' because of the time-respecting condition. Also, by Fact 2, $(r_1, r_3, 2) \in A(R_{a-1})$ and the arcs from r_3 to U' cannot be used because of the time-respecting condition. Therefore, all in-neighbors of U' vertices are in S' . In fact by construction, for each $u'_j \in U'$, the in-neighbor of u'_j in R_{a-1} is some $S'_i \in S'$ such that $u_j \in S_i$. Since every $e \in A_{S', U'}$ is active at timestamp 1, every path from r_1 to a U' vertex in R_{a-1} only uses arcs of timestamps 1. Such a path cannot use an arc in which r_3 is the tail, again because of the $(r_1, r_3, 2)$ arc. Thus such a path must use an arc of $A_{r_2, S'}$. Let

$$S^* = \{S_i : (r_2, S'_i, 1) \in A(R_{a-1})\}.$$

Note that because each $u'_j \in U'$ has an S' in-neighbor such that the corresponding S set contains u_j , S^* is a set cover. It remains to argue that $|S^*| \leq k$.

Observe that $A(R_{a-1}) \setminus A(T_1)$ contains at least $|U| + |U'| + |S^*| = 2n + |S^*|$ arcs, since it has all arcs of $A_{r_2, U}$, the arcs from S' to U' , and the arcs from r_2 to $\{S'_i : S_i \in S^*\}$. Thus at least $2n + |S^*| + 1$ arc flips are needed to get to R_a . Then, $A(T_2) \setminus A(R_a)$ contains at least $1 + |S'| + |U| = 1 + n + m$ arcs, namely $(r_3, r_2, 1)$ and the arcs from $A_{r_3, S'}$ and $A_{r_3, U'}$ (which are not in R_{a-1} , and thus not in R_a , because $(r_1, r_3, 2) \in A(R_{a-1})$ by Fact 2). Therefore, the number of arc flips required from T_1 to T_2 is at least $3n + m + 2 + |S^*|$, from which it follows that $|S^*| \leq k$.

Since SET COVER is known to be NP-hard [16], the reduction we have described implies that also TEMPORAL ARBORESCENCE RECONFIGURATION for two timestamps is NP-hard. ◀

4 Inapproximability for Distance Two

In this section we show that, unless $P = NP$, TEMPORAL ARBORESCENCE RECONFIGURATION is not approximable within factor $b \ln |V(D)|$, for any constant $0 < b < 1$, even if the two input temporal arborescences have distance two, that is that is the number of arcs in $A(T_1) \setminus A(T_2)$ and the number of arcs in $A(T_2) \setminus A(T_1)$ is equal to two. We prove the result via an approximation preserving reduction from the SET COVER problem. Let (S, U) be an instance of SET COVER¹, where $U = \{u_1, \dots, u_n\}$ and $S = \{S_1, \dots, S_m\}$. Construct $(D = (V, A), T_1, T_2)$, an instance of TEMPORAL ARBORESCENCE RECONFIGURATION associated with (U, S) , as follows (refer to Fig. 2 for the structure of D).

$$V = \{r_1, r_2, y\} \cup \{v_{i,z} : S_i \in S, i \in [m], z \in [n^2]\} \cup \{w_i : i \in [n], u_i \in U\}.$$

A is defined as

$$A = A_1 \cup A_2 \cup A_3$$

where:

$$A_1 = \{(r_1, r_2, 2)\} \cup \{(r_1, y, 1)\} \cup \{(r_1, v_{i,1}, 4) : i \in [m]\} \cup \\ \{(v_{i,j}, v_{i,j+1}, 4) : i \in [m], j \in [n^2 - 1]\} \cup \{(y, w_i, 2) : i \in [n]\}$$

$$A_2 = \{(r_2, r_1, 2)\} \cup \{(r_2, y, 1)\} \cup \{(r_1, v_{i,1}, 4) : i \in [m]\} \cup \\ \{(v_{i,j}, v_{i,j+1}, 4) : i \in [m], j \in [n^2 - 1]\} \cup \{(y, w_i, 2) : i \in [n]\}$$

$$A_3 = \{(r_1, y, 3)\} \cup \{(r_1, v_{i,1}, 3) : i \in [m]\} \cup \{(v_{i,j}, v_{i,j+1}, 3) : i \in [m], j \in [n^2 - 1]\} \cup \\ \{(v_{i,n^2}, w_j, 3) : u_j \in S_i, i \in [m], j \in [n]\}$$

Now, T_1 is the temporal arborescence induced by A_1 , that is $T_1 = (V, A_1)$, and T_2 is the temporal arborescence induced by A_2 , that is $T_2 = (V, A_2)$ (see Fig. 3). Note that $|A_1 \setminus A_2| = |A_2 \setminus A_1| = 2$, since $A_1 \setminus A_2 = \{(r_1, r_2, 2), (r_1, y, 1)\}$, while $A_2 \setminus A_1 = \{(r_2, r_1, 2), (r_2, y, 1)\}$.

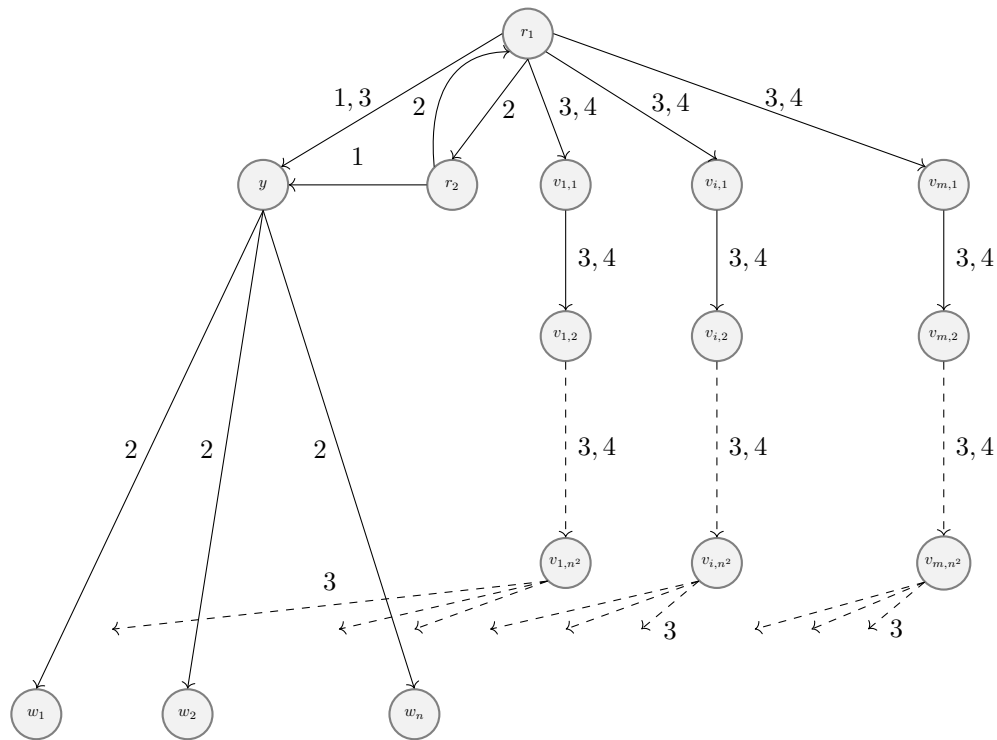
We define a reconfiguration from T_1 to T_2 as *canonical* if it has the following properties. First, in some order, each w_i , $i \in [n]$, is disconnected from y as follows (we call this the *disconnection step* of the reconfiguration):

1. For some $j \in [m]$, each arc on the path from r_1 to v_{j,n^2} , associated with timestamp 4, is flipped with the arc having the same endpoints and timestamp 3 (starting from $(r_1, v_{j,1}, 4)$ and ending with $(v_{j,n^2-1}, v_{j,n^2}, 4)$).
2. Each arc $(y, w_i, 2)$, $i \in [n]$, is flipped with an arc $(v_{j,n^2}, w_i, 3)$, $j \in [m]$, so that there is a path from r_1 to v_{j,n^2} with all the arcs having timestamps 3.

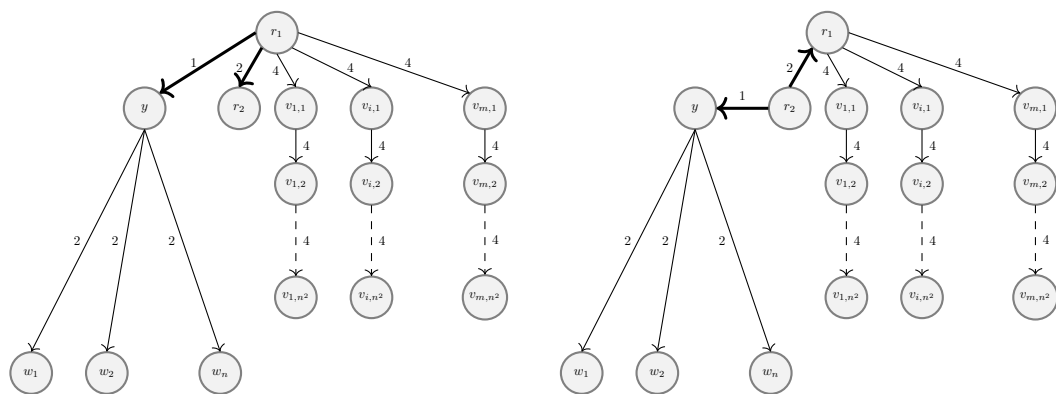
Once the disconnection step is applied and each w_i , $i \in [n]$, is disconnected from y , a canonical reconfiguration flips arc $(r_1, y, 1)$ and $(r_1, y, 3)$. Then the root of the temporal arborescence is changed by flipping arcs $(r_1, r_2, 2)$ and $(r_2, r_1, 2)$. After these arc flips, $(r_1, y, 3)$ is flipped with arc $(r_2, y, 1)$. In order to compute T_2 , each arc $(v_{j,n^2}, w_i, 3)$, $j \in [m]$ and $i \in [n]$, flipped in the disconnection step, is flipped with $(y, w_i, 2)$. Finally, for each path from r_1 to v_{j,n^2} , $j \in [m]$, having arcs with timestamp 3, each arc on the path is flipped with the arc having the same endpoints and timestamp 4 (starting from $(v_{j,n^2-1}, v_{j,n^2}, 3)$ and ending with $(r_1, v_{j,1}, 3)$).

We start by proving that a canonical reconfiguration is correct, that is it computes only temporal arborescences.

¹ Since in this section we consider optimization versions of problems, we do not include in the problem instances the value of a solution



■ **Figure 2** The input digraph D associated with an instance of SET COVER. Each dashed arrow outgoing from $v_{i,2}$, $i \in [m]$, represent a path containing vertices $v_{i,j}$, $j \in \{3, \dots, n^2 - 1\}$, and not shown in the figure. The dashed arrows outgoing from v_{1,n^2} , v_{i,n^2} , v_{m,n^2} represent arcs connecting these vertices with some vertices w_z , $z \in [n]$ (the precise arcs depends on the instance of SET COVER).



■ **Figure 3** Arborescence T_1 (left) and T_2 (right). The four arcs in bold belong to exactly one the two temporal arborescence, the other arcs belong to both T_1 and T_2 .

► **Lemma 2.** *Each arborescence computed by a canonical reconfiguration from T_1 to T_2 is a temporal arborescence of D .*

We prove now the first direction of the reduction.

► **Lemma 3.** *Let (S, U) be an instance of SET COVER and let (D, T_1, T_2) be the corresponding instance of TEMPORAL ARBORESCENCE RECONFIGURATION. Given a set cover of size k we can compute in polynomial time a reconfiguration from T_1 to T_2 consisting of $2kn^2 + 2n + 3$ flips.*

Now, we consider the second part of the reduction, where we prove that a reconfiguration from T_1 to T_2 must apply the disconnection step of a canonical reconfiguration.

► **Lemma 4.** *Let (S, U) be an instance of SET COVER and let (D, T_1, T_2) be the corresponding instance of TEMPORAL ARBORESCENCE RECONFIGURATION. Given a reconfiguration from T_1 to T_2 consisting of $2kn^2 + 2n + 3$ arc flips we can compute in polynomial time a solution of SET COVER on instance (S, U) of size k .*

Proof. We start by proving that a reconfiguration from T_1 to T_2 must apply the disconnection step of a canonical reconfiguration.

First, consider arc $(r_1, r_2, 2)$ of T_1 and arc $(r_2, r_1, 2)$ of T_2 . Note that $(r_1, r_2, 2)$ ($(r_2, r_1, 2)$, respectively) is the only arc of D incoming into r_2 (into r_1 , respectively). Hence whenever $(r_1, r_2, 2)$ is flipped, and hence removed, by a reconfiguration, it must be flipped with $(r_2, r_1, 2)$, and r_2 must become the root of the computed temporal arborescence, otherwise either both r_1 and r_2 have not incoming arcs or r_2 is not connected with other vertices of the temporal arborescence. Note that this arc flip defines r_2 as the root of the computed arborescence and creates a temporal path $(r_2, r_1, 2), (r_1, y, 1)$, if this latter arc (of T_1) belongs to the arborescence, which is not time-respecting. It follows that before $(r_1, r_2, 2)$ and $(r_2, r_1, 2)$ are flipped, $(r_1, y, 1)$ must be flipped with another arc that must be incoming to y (since r_1 remains the root of the arborescence), that is with $(r_2, y, 1)$ or $(r_1, y, 3)$.

Consider $(r_2, y, 1)$ and notice that arcs $(r_1, y, 1)$ and $(r_2, y, 1)$ cannot be flipped, since this flip creates a temporal path $(r_1, r_2, 2), (r_2, y, 1)$, which is not time-respecting, and we have observed that $(r_1, r_2, 2)$ is not flipped before $(r_1, y, 1)$. Arcs $(r_1, y, 1)$ and $(r_1, y, 3)$ cannot be flipped unless y is a leaf, that is all the arcs $(y, w_i, 2)$, with $i \in [n]$, have been flipped. Indeed, if an arc $(y, w_i, 2)$, $i \in [n]$, belongs to a temporal arborescence, then by flipping $(r_1, y, 1)$ and $(r_1, y, 3)$ we have a temporal path $(r_1, y, 3), (y, w_i, 2)$, which is not time-respecting. It follows that, before $(r_1, y, 1)$ is flipped each vertex w_i , $i \in [n]$, must first be disconnected from y . By construction the only incoming arcs to a vertex w_i , $i \in [n]$, other than $(y, w_i, 2)$, are $(v_{j, n^2}, w_i, 3)$, for some $j \in [m]$, hence each vertex w_i must first be disconnected from y by flipping an arc $(y, w_i, 2)$ with an arc $(v_{j, n^2}, w_i, 3)$, for some $j \in [m]$. This implies that the disconnection step of the canonical reconfiguration is applied. This requires that each arc on the path from r_1 to v_{j, n^2} , which have timestamp 4 in T_1 , is flipped with the arc having the same endpoints and timestamp 3.

Consider the temporal arborescence T' constructed by the disconnection step. For each w_i , $i \in [n]$, the disconnection step flips all the arcs of one path from r_1 to some v_{j, n^2} , $j \in [m]$, such that $w_i \in S_j$; then we can define a set cover as follows:

$$S^* = \{S_j : \text{the path from } r_1 \text{ to } w_{j, n^2} \text{ is modified in the disconnection step}\}.$$

We claim that S^* contains at most k sets. Note that a reconfiguration from T' to T_2 requires, as in a canonical reconfiguration, to delete arcs in $A(T') \setminus (A(T_2) \cap A(T_1))$ and insert arcs in $(A(T_2) \cap A(T_1)) \setminus A(T')$.

10:10 On the Complexity of Temporal Arborescence Reconfiguration

Recall that the reconfiguration of T_1 in T_2 consists of $2kn^2 + 2n + 3$ flips. If S^* consists of at least $k + 1$ sets, then by the definition of S^* the disconnection step includes at least $k + 1$ paths, thus requiring at least $2(k + 1)n^2$ arc flips for these paths, plus $2n$ arc flips for the arcs incident in w_i , $i \in [n]$. We have that $2(k + 1)n^2 + 2n > 2kn^2 + 2n + 3$, since $n \geq 2$. Hence S^* contains at most k sets, thus completing the proof. \blacktriangleleft

Based on Lemma 3 on Lemma 4, on the fact that the digraph D contains $O(n^2m)$ vertices and on the hardness of approximation of SET COVER [2, 5, 20], we can prove the following result.

► **Theorem 5.** *TEMPORAL ARBORESCENCE RECONFIGURATION is not approximable within factor $b \ln |V(D)|$, for any constant $0 < b < 1$, unless $P = NP$, even when the two input temporal arborescences differ for two pairs of arcs.*

Distance One

We have shown that TEMPORAL ARBORESCENCE RECONFIGURATION is hard (also to approximate) when $T_1 = (V, A(T_1))$ and $T_2 = (V, A(T_2))$ have distance two. On the other hand when T_1 and T_2 have distance one, thus $A(T_1) \setminus A(T_2)$ contains a single arc a_1 and $A(T_2) \setminus A(T_1)$ contains a single arc a_2 , the problem is easy to solve in polynomial time. Indeed, since by flipping a_1 with a_2 in T_1 , hence by removing a_1 and inserting a_2 , we obtain T_2 , it follows that the arc flip produces a spanning time-respecting arborescence and thus can always be applied.

5 W[1]-Hardness

In all the above reductions (Section 3 and Section 4) and also the reduction in [13], the number of required arc flips is always a function of n . Therefore, an algorithm with complexity of the form $f(p)n^c$, with constant c and f only depending on p (number of arc flips of a reconfiguration from T_1 to T_2), is not excluded. We show that this is unlikely by proving that the TEMPORAL ARBORESCENCE RECONFIGURATION problem is W[1]-hard under this parameter p , and that in fact it is W[1]-hard in parameter $p + \max_{(u,v,t) \in A(D)} t$.

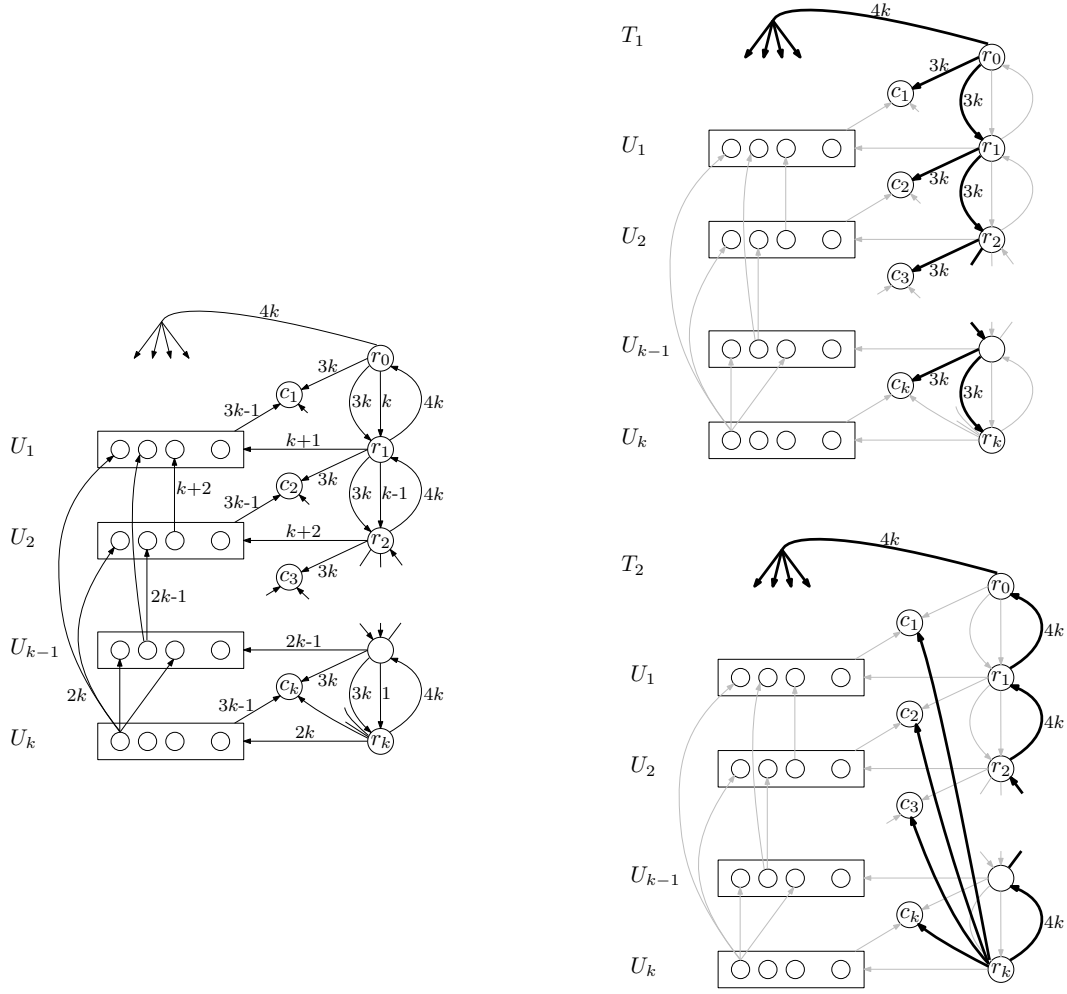
We reduce MULTICOLORED CLIQUE to TEMPORAL ARBORESCENCE RECONFIGURATION. MULTICOLORED CLIQUE, given an undirected graph $G = (V, E)$, whose vertices are colored with k colors, asks whether there exists a clique, called multicolored clique, containing one vertex from each color. The problem is W[1]-hard when the parameter is the number of colors [6].

Let $G = (V, E)$ be an instance a MULTICOLORED CLIQUE, with vertices partitioned into color classes V_1, \dots, V_k . For $i, j \in [k]$, we will denote $E_{i,j} = \{uv \in E : u \in V_i, v \in V_j\}$. Construct an instance (D, T_1, T_2, p) of TEMPORAL ARBORESCENCE RECONFIGURATION as follows.

Let us first construct D , which is shown in Figure 4 (we provide the main intuitions after the description of the construction). We define the vertex set of D as $V(D) = R \cup C \cup U$, where

$$\begin{aligned} R &= \{r_0, r_1, \dots, r_k\}, \\ C &= \{c_1, c_2, \dots, c_k\}, \\ U &= \{u' : u \in V(G)\}. \end{aligned}$$

For $i \in [k]$, we will denote $U_i = \{u' : u \in V_i\}$.



■ **Figure 4** Main construction for the $W[1]$ -hardness proof. Left: the temporal graph D . Top-right: the initial temporal arborescence T_1 . Bottom-right: the target temporal arborescence T_2 . Note that the timestamps 0 of arcs (r_k, c_i) are not shown.

As for the arc set $A(D)$, add the following arc sets:

- *R-R arcs*: for each $i \in \{0, 1, \dots, k-1\}$, add the arc $e_i = (r_i, r_{i+1}, 3k)$; the arc $e'_i = (r_i, r_{i+1}, k-i)$; and the arc $f_i = (r_{i+1}, r_i, 4k)$.
- *r_0 - U arcs*: for each $u \in V(G)$, add the arc $(r_0, u', 4k)$.
- *r_i - U_i arcs*: for each color class $i \in [k]$ and each $u \in V_i$, add the arc $(r_i, u', k+i)$. Note that $i > 0$, hence r_0 is not concerned here.
- *U_i - U_j arcs*: for each $i, j \in [k]$ with $j < i$ and each $uv \in E_{i,j}$ with $u \in V_i$ and $v \in V_j$, add an arc $(u', v', k+i)$. That is, each vertex u' has an outgoing arc to v' whenever v is a neighbor of u in a “lower” color class. In terms of Figure 4, this means that all arcs between the U_i sets go upwards. The tail of the arc determines its timestamp.
- *R-C arcs*: for each color class $i \in [k]$, add the arcs $(r_{i-1}, c_i, 3k)$ and $(r_k, c_i, 0)$.
- *U_i - c_i arcs*: for each color class $i \in [k]$, and each $u \in V_i$, add the arc $(u', c_i, 3k-1)$.

The arcs of the initial temporal arborescence T_1 consist of: the *R-R* arcs e_i for $i \in \{0, 1, \dots, k-1\}$, so that there is a path of arcs at time $3k$ from r_0 to r_k ; the *r_0 - U* arcs $(r_0, u', 4k)$ for $u \in V$; the *R-C* arcs $(r_{i-1}, c_i, 3k)$ for $i \in [k]$.

10:12 On the Complexity of Temporal Arborescence Reconfiguration

The arcs of the target temporal arborescence T_2 consist of the same arc set as T_1 , except that: no e_i arc is in T_2 , and instead each R - R arc f_i is in T_2 ; no $(r_{i-1}, c_i, 3k)$ arc is present, and instead each R - C arc $(r_k, c_i, 0)$ is in T_2 . It is not difficult to verify that T_1 and T_2 are temporal arborescences (hence time-respecting).

The intuition behind this construction is as follows. To transform T_1 into T_2 , one must first re-root from r_0 to r_1 , then to r_2 , and so on until r_k is the root. If we re-root from r_0 to r_1 , we need to insert the arc $(r_1, r_0, 4k)$. This cannot be done in the very first arc flip though, because the arc $(r_0, c_1, 3k)$ in the R - C group would violate temporality. So any solution must first create an alternate path from r_0 to c_1 before the first re-rooting. One can show that the only way to achieve this is to choose some $u'_1 \in U_1$ and create the path $r_0 \rightarrow r_1 \rightarrow u'_1 \rightarrow c_1$, using arcs at times $k, k+1, 3k-1$. Once this is done, we can safely re-root to r_1 .

Next, we must re-root to r_2 . As before, we cannot insert $(r_2, r_1, 4k)$ because of $(r_1, c_2, 3k)$. So we must create an alternate path $r_1 \rightarrow r_2 \rightarrow u'_2 \rightarrow c_2$ for some $u'_2 \in U_2$. However this time, the arc $(r_1, u'_1, k+1)$ from the previous step is also an issue and we must also have an alternate path from r_1 to u'_1 . The key idea is that the most efficient way to do this is, after choosing u'_2 , to apply a flip that removes $(r_1, u'_1, k+1)$ and inserts $(u'_2, u'_1, k+2)$. This arc exists only if $u_2 u_1 \in E(G)$, forcing us to choose u'_2, u'_1 that form a clique of size 2.

The same idea applies for every $i \in [k]$. Before re-rooting from r_{i-1} to r_i , we must find an alternate path $r_{i-1} \rightarrow r_i \rightarrow u'_i \rightarrow c_i$ by choosing some $u'_i \in U_i$. At this point, there are u'_1, \dots, u'_{i-1} that are used as in-neighbors of c_1, \dots, c_{i-1} . The most efficient setup is to choose u'_i that allows inserting the $(u'_i, u'_j, k+i)$ arcs for all those $j < i$, requiring all corresponding u_j 's to be neighbors of u_i in G . In other words, there are k phases to apply, one for each re-rooting to each r_i , and at each phase i we must choose a u_i (and corresponding u'_i) that is a neighbor of all the previously chosen u_j 's, thereby forming a clique. The specific arc timestamps in the construction are chosen to enforce this behavior.

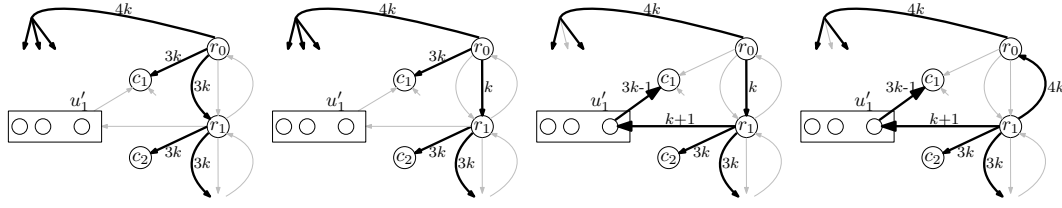
We will show that G contains a multicolored clique if and only if T_1 can be transformed into T_2 using at most $p = 2k + \sum_{i=1}^k (i+3)$ arc flips. In essence, each term in the summation represents the arc flips needed to re-root from r_{i-1} to r_i , and the $2k$ term is there for a cleanup phase after having re-rooted to r_k . Note that since p is a function of k only, this shows $W[1]$ -hardness in parameter p being the number of required arc flips. Also note that in fact, all timestamps assigned to arcs are a function of k , so the problem is $W[1]$ -hard in parameter $p+t$, where $t = \max_{(u,v,t') \in A(D)} t'$.

► **Theorem 6.** *The TEMPORAL ARBORESCENCE problem is $W[1]$ -hard when parameterized by the number of arc flips plus the maximum timestamp.*

Proof. First note that the construction of D from G can be carried out in polynomial time. As mentioned above, we show that G contains a multicolored clique if and only if T_1 can be transformed into T_2 using at most $p = 2k + \sum_{i=1}^k (i+3)$ arc flips.

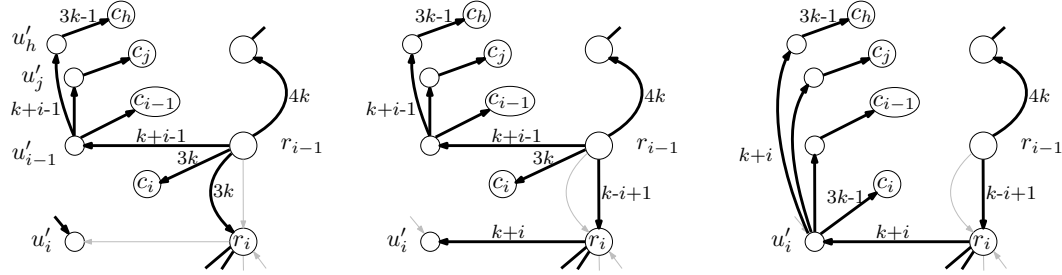
(\Rightarrow) Suppose that G has a multicolored clique $K = \{u_1, \dots, u_k\}$, where for each $i \in [k]$ the vertex u_i belongs to color class V_i . As shown in Figure 5, starting from T_1 , one can re-root from r_0 to r_1 (each step can easily be checked to maintain a temporal arborescence, hence time-respecting):

- Remove $e_0 = (r_0, r_1, 3k)$ and insert $e'_0 = (r_0, r_1, k)$, so that r_0 reaches r_1 with the arc at time k instead of the arc at time $3k$;
- Remove $(r_0, u'_1, 4k)$ and insert $(r_1, u'_1, k+1)$, which is now possible. Then remove $(r_0, c_1, 3k)$ and insert $(u'_1, c_1, 3k-1)$;
- Remove e'_0 and insert $(r_1, r_0, 4k)$, thereby re-rooting to r_1 .



■ **Figure 5** A sequence of arc flips to re-root from r_0 to r_1 .

Note that this requires $4 = 1 + 3$ flips. Now let $i \geq 2$ and let us see how to re-root from r_{i-1} to r_i (illustrated in Figure 6). Assume that we have reached a temporal arborescence such that: r_{i-1} is the root; $(r_{i-1}, u'_{i-1}, k + i - 1)$ is active; $(u'_{i-1}, u'_j, k + i - 1)$ is active for each $j < i - 1$; $(u'_j, c_j, 3k - 1)$ is active for each $j \leq i - 1$. Also assume that r_{i-1} reaches r_0 using the f_j upwards arcs at time $4k$, and that r_0 uses $4k$ arcs to reach all the v'_j other than u'_1, \dots, u'_{i-1} . Note that all these conditions hold for $i = 2$ after applying the re-rooting from r_0 to r_1 . We show how to re-root from r_{i-1} to r_i , such that the same properties hold but with r_i as the root. To achieve this:



■ **Figure 6** A sequence of arc flips to re-root from r_{i-1} to r_i . Here, we assume $h < j < i - 1$. The middle state is obtained by two arc flips that insert e'_{i-1} and (r_i, u'_i) . The rightmost state is obtained by making u'_i the in-neighbor of every $u'_j, j < i$. The last step is not shown and consists in flipping e'_{i-1} to f_{i-1} to re-root to r_i .

- Remove $e_{i-1} = (r_{i-1}, r_i, 3k)$ and add $e'_{i-1} = (r_{i-1}, r_i, k - (i - 1))$, so that r_{i-1} now reaches r_i with an arc at timestamp $k - i + 1$. This preserves temporality since this is akin to lowering the timestamp for the arc from r_{i-1} to r_i , which is an outgoing arc from the root r_{i-1} .
- Remove $(r_0, u'_i, 4k)$ and add $(r_i, u'_i, k + i)$. This preserves temporality since the new path from r_{i-1} to u'_i uses arcs at respective times $k - i + 1$ and $k + i$.
- Remove $(r_{i-1}, c_i, 3k)$ and add $(u'_i, c_i, 3k - 1)$, which is correct since the latter has time $3k - 1 > k + i$.
- For each $j < i - 1$, remove the incoming arc $(u'_{i-1}, u'_j, k + i - 1)$ of u'_j and add $(u'_i, u'_j, k + i)$ (which exists because $u_i u_j \in E(G)$). This is temporarily correct since u'_i is currently reachable with arcs of timestamp at most $k + i$, each arc from u'_i to u'_j has timestamp $k + i$, and each arc from u'_j to c_j has timestamp $3k - 1 > k + i$.
- Remove $(r_{i-1}, u'_{i-1}, k + i - 1)$ and add $(u'_i, u'_{i-1}, k + i)$, which preserves temporality as in the previous step.
- Finally, re-root to r_i by removing e'_{i-1} and adding $f_{i-1} = (r_i, r_{i-1}, 4k)$. This preserves temporality because, at this point we have the situation from Figure 6 on the right. The only vertices that r_{i-1} was reaching without going through r_i were r_{i-2}, \dots, r_0 and u' vertices using $(r_0, u', 4k)$ arcs, and all the underlying paths consisted of arcs at time $4k$.

10:14 On the Complexity of Temporal Arborescence Reconfiguration

Observe that all the assumptions made before handling step i are true for the next step. Also note that to re-root from r_{i-1} to r_i , the above requires $3 + (i - 2) + 2 = i + 3$ flips.

Once we reach a point where r_k is the root, we can: replace every $(u'_j, c_j, 3k - 1)$ with $(r_k, c_j, 0)$ for $j \in [k]$ (k arc flips); remove all the $(u'_k, u'_j, 2k)$ arcs and insert $(r_0, u'_j, 4k)$ for $j < k$ ($k - 1$ arc flips); replace $(r_k, u'_k, 2k)$ with $(r_0, u'_k, 4k)$ (1 arc flip). This last step adds $2k$ arc flips.

Overall, we have reached T_2 using $\sum_{i=1}^k (i + 3) + 2k = p$ arc flips.

(\Leftarrow) Suppose that T_1 can be transformed into T_2 using at most p flips. It can be shown that this implies that u_1, \dots, u_k form a multicolored clique of G . The proof is omitted for space reasons and can be found in the full version – the main idea is that the steps described in the forward direction are essentially forced to achieve p flips. Since MULTICOLORED CLIQUE is W[1]-hard (for parameter k), the parameterized reduction we have described implies that TEMPORAL ARBORESCENCE RECONFIGURATION is W[1]-hard for parameters number of arc flips plus maximum timestamp. \blacktriangleleft

6 Conclusion

We have analyzed the complexity TEMPORAL ARBORESCENCE RECONFIGURATION, proving that it is NP-hard for two timestamps, it is inapproximable within factor $b \ln |V(D)|$, for any $0 < b < 1$, if the two temporal arborescences differ only for two arc pairs, and it is W[1]-hard when parameterized by the number of arc flips needed to transform one arborescence into the other plus maximum timestamp.

A natural future direction is to further study the approximation complexity of the problem, in particular if it is possible to achieve a $c \ln |V(D)|$ approximation factor, for some constant $c \geq 1$. A second future direction is to further investigate the problem when the input temporal digraph has specific properties (for example bounded treewidth or bounded degree), both in the approximation and parameterized complexity framework.

References

- 1 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Christoforos L. Raptopoulos. The temporal explorer who returns to the base. *J. Comput. Syst. Sci.*, 120:179–193, 2021. doi:10.1016/j.jcss.2021.04.001.
- 2 Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k -restrictions. *ACM Trans. Algorithms*, 2(2):153–177, 2006. doi:10.1145/1150334.1150336.
- 3 Marco Amoroso, Daniele Anello, Vincenzo Auletta, Raffaele Cerulli, Diodato Ferraioli, and Andrea Raiconi. Contrasting the spread of misinformation in online social networks. *J. Artif. Intell. Res.*, 69:847–879, 2020. doi:10.1613/JAIR.1.11509.
- 4 Daniela Bubboloni, Costanza Catalano, Andrea Marino, and Ana Silva. On computing optimal temporal branchings. In Henning Fernau and Klaus Jansen, editors, *Fundamentals of Computation Theory - 24th International Symposium, FCT 2023, Trier, Germany, September 18-21, 2023, Proceedings*, volume 14292 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2023. doi:10.1007/978-3-031-43587-4_8.
- 5 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.
- 6 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/J.TCS.2008.09.065.

- 7 Guillaume Fertin, Julien Fradin, and Géraldine Jean. The maximum colorful arborescence problem: How (computationally) hard can it be? *Theoretical Computer Science*, 852:104–120, 2021.
- 8 Ziyun Guang, Matthew Smith-Erb, and Layla Oesper. A weighted distance-based approach for deriving consensus tumor evolutionary trees. *Journal Title Here*, pages 1–9, 2023.
- 9 Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88(9):234, 2015.
- 10 Petter Holme and Jari Saramäki. A map of approaches to temporal networks. In *Temporal Network Theory*, pages 1–24. Springer, 2019.
- 11 Silu Huang, Ada Wai-Chee Fu, and Ruifeng Liu. Minimum spanning trees in temporal graphs. In Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 419–430. ACM, 2015. doi:10.1145/2723372.2723717.
- 12 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. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011. doi:10.1016/J.TCS.2010.12.005.
- 13 Takehiro Ito, Yuni Iwamasa, Naoyuki Kamiyama, Yasuaki Kobayashi, Yusuke Kobayashi, Shun-ichi Maezawa, and Akira Suzuki. Reconfiguration of time-respecting arborescences. In Pat Morin and Subhash Suri, editors, *Algorithms and Data Structures - 18th International Symposium, WADS 2023, Montreal, QC, Canada, July 31 - August 2, 2023, Proceedings*, volume 14079 of *Lecture Notes in Computer Science*, pages 521–532. Springer, 2023. doi:10.1007/978-3-031-38906-1_34.
- 14 Takehiro Ito, Yuni Iwamasa, Yasuaki Kobayashi, Yu Nakahata, Yota Otachi, and Kunihiro Wasa. Reconfiguring (non-spanning) arborescences. *Theor. Comput. Sci.*, 943:131–141, 2023. doi:10.1016/J.TCS.2022.12.007.
- 15 Naoyuki Kamiyama and Yasushi Kawase. On packing arborescences in temporal networks. *Inf. Process. Lett.*, 115(2):321–325, 2015. doi:10.1016/J.IPL.2014.10.005.
- 16 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.
- 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 Ning Li and Jennifer C. Hou. Topology control in heterogeneous wireless networks: Problems and solutions. In *Proceedings IEEE INFOCOM 2004, The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, March 7-11, 2004*. IEEE, 2004. doi:10.1109/INFOCOM.2004.1354497.
- 19 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.
- 20 Dana Moshkovitz. The projection games conjecture and the np-hardness of ln n-approximating set-cover. *Theory Comput.*, 11:221–235, 2015. doi:10.4086/TOC.2015.V011A007.
- 21 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/A11040052.

Partial Temporal Vertex Cover with Bounded Activity Intervals

Riccardo Dondi¹  

Università degli Studi di Bergamo, Italy

Fabrizio Montecchiani  


Università degli Studi di Perugia, Italy

Giacomo Ortali  

Università degli Studi di Perugia, Italy

Tommaso Piselli  

Università degli Studi di Perugia, Italy

Alessandra Tappini  

Università degli Studi di Perugia, Italy

Abstract

Different variants of **Vertex Cover** have recently garnered attention in the context of temporal graphs. One of these variants is motivated by the need to summarize timeline activities in social networks. Here, the activities of individual vertices, representing users, are characterized by time intervals. In this paper, we explore a scenario where the temporal span of each vertex’s activity interval is bounded by an integer ℓ , and the objective is to maximize the number of (temporal) edges that are covered. We establish the APX-hardness of this problem and the NP-hardness of the corresponding decision problem, even under the restricted condition where the temporal domain comprises only two timestamps and each edge appears at most once. Subsequently, we delve into the parameterized complexity of the problem, offering two fixed-parameter algorithms parameterized by: (i) the number k of temporal edges covered by the solution, and (ii) the number h of temporal edges *not* covered by the solution. Finally, we present a polynomial-time approximation algorithm achieving a factor of $\frac{3}{4}$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Fixed parameter tractability; Theory of computation → Approximation algorithms analysis

Keywords and phrases Temporal Graphs, Temporal Vertex Cover, Parameterized Complexity, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.11

Funding This work was partially supported by: (i) MUR PRIN Proj. 2022TS4Y3N – “EXPAND: scalable algorithms for EXPLoratory Analyses of heterogeneous and dynamic Networked Data”; (ii) MUR PRIN Proj. 2022ME9Z78 – “NextGRAAL: Next-generation algorithms for constrained GRaph visuALization”; (iii) University of Perugia, Fondi di Ricerca di Ateneo, edizione 2021, project “AIDMIX – Artificial Intelligence for Decision making: Methods for Interpretability and eXplainability”.

1 Introduction

The temporal graph model is designed to capture the dynamic evolution of interactions over time [14, 12, 17, 13]. A temporal graph can be viewed as a labeled graph, where every edge is endowed with time labels signifying the timestamps where the edge is defined, and thus where the interaction represented by the edge is observed; see Figure 1 for an illustration.

¹ Corresponding author

Numerous foundational problems originally formulated for static graphs have recently been extended to temporal graphs. On static graphs, the **Vertex Cover** problem asks for a subset of vertices with minimum cardinality, such that it covers all the edges of the input graph, that is, such that for each edge at least one of its endpoints belongs to the subset. Following this line of research, different adaptations of **Vertex Cover** on temporal graphs have been explored in the literature [1, 11, 19]. Here we focus on the approach introduced in [19], motivated by the need to summarize interaction timelines of users in social networks.

Informally, a *temporal vertex cover* of a temporal graph G is a subset² \mathcal{C} of its vertices and an assignment of time intervals to every vertex of \mathcal{C} , such that for every edge e of G and for every time label t of e , at least one end-vertex of e is part of \mathcal{C} and the endowed time interval includes t (see Section 2 for a formal definition). In other words, a temporal vertex cover assigns an activity interval to a subset of users, such that for every observed interaction at least one involved user is part of the solution and active. Based on this idea, the objective function of the **MinTimelineCover** problem is to find a temporal vertex cover of minimum size (i.e., minimizing the sum of the interval lengths).

Recently, a sequence of works investigated the computational complexity of the **MinTimelineCover** problem, proving that it is NP-hard [19], even in the restricted scenarios when each label is associated with a single edge [4], and when the temporal graph is defined over two timestamps only [7]. In terms of parameterized complexity, **MinTimelineCover** parameterized by the solution size has first been shown to admit a fixed-parameter algorithm for temporal graphs defined over two timestamps [7] and, subsequently, this restriction has been removed [5].

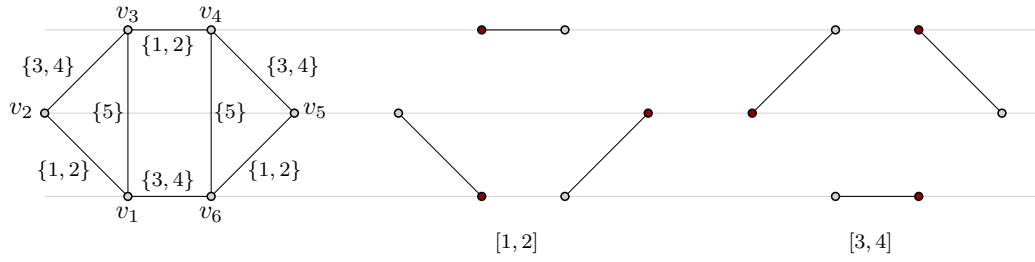
The complexity of approximating **MinTimelineCover** has been also studied. A result given in [7] implies that, assuming the Unique Games conjecture, **MinTimelineCover** cannot be approximated within a constant factor, even for graphs defined on two timestamps only. On the positive side, the problem can be approximated within factor $O(T \log n)$, on a temporal graph with n vertices and T timestamps [6].

In this paper, we introduce and explore a new problem, ℓ -**TimelineCover(k)** and its optimization version ℓ -**MaxTimelineCover**, in which we relax the constraint that all the edges have to be covered, bounding instead the length of the vertex activity intervals by an integer $\ell \geq 1$. This last constraint is motivated by the observation that a solution of **MinTimelineCover** may define long activity intervals for some vertices, while in several applications we observe short time activities of users [19]. Hence, the ℓ -**TimelineCover(k)** problem asks for the definition of one interval of length at most ℓ for each vertex, so that at least k edges of the temporal graph are covered (or the maximum number of edges are covered for ℓ -**MaxTimelineCover**); see Figure 1 for an example. From a graph theory point of view, ℓ -**TimelineCover(k)** can be seen as a temporal variant of **Partial Vertex Cover** [8, 16]: Given a graph and two positive integers h and p , **Partial Vertex Cover** asks whether there exists a set of at most h vertices that cover at least p edges of the graph.

Our main contribution can be summarized as follows.

- We prove, in Section 3, that ℓ -**TimelineCover(k)** is NP-hard and ℓ -**MaxTimelineCover** is APX-hard, even in the restricted case where the time domain consists of two timestamps (and $\ell = 1$) and each edge appears at most once. Note that if ℓ is equal to the number of timestamps, then the problem admits a trivial solution where each vertex has an interval equal to the number of timestamps and all the edges are covered. Denote by T the

² We note that an equivalent definition can be made by replacing \mathcal{C} with the entire vertex set of G , and allowing for vertices with an empty assigned time interval.



■ **Figure 1** (Left) An example of a temporal graph $\langle G, \lambda \rangle$, where G is a graph and λ is a time-labeling function that maps every edge of G onto a set of timestamps; for example, the edge $\{v_1, v_2\}$ is associated to timestamps $\{1, 2\}$, while the edge $\{v_2, v_3\}$ to timestamps $\{3, 4\}$. (Center and Right) A solution of ℓ -TimelineCover(12) (hence at least 12 temporal edges have to be covered), for $\ell = 2$, defines: interval $[1, 2]$ for v_1, v_3, v_5 , thus covering edges at timestamps 1 and 2; interval $[3, 4]$ for v_2, v_4, v_6 , thus covering edges at timestamps 3 and 4. Note that the edges defined at timestamp 5 ($\{v_1, v_3\}, \{v_4, v_6\}$) are not covered.

number of timestamps over which the temporal graph is defined. These results imply that ℓ -TimelineCover(k) parameterized by $\ell + T$ admits no XP (and hence no FPT) algorithm, unless $P=NP$.

- Next, in Section 4, we focus on the parameterized complexity of the ℓ -TimelineCover(k) problem and consider two parameters: the number h of temporal edges left uncovered by the solution, and the number k of temporal edges that are covered by the solution. For both parameterizations, we prove that the problem is fixed-parameter tractable.
- Finally, in Section 5, we focus again on the approximability of the ℓ -MaxTimelineCover problem, and we present a polynomial-time approximation algorithm of factor $\frac{3}{4}$.

In Section 2 we give some definitions and we introduce the ℓ -TimelineCover(k) and ℓ -MaxTimelineCover problems. We conclude the paper in Section 6 with open problems that naturally stem from our research. Some of the proofs are deferred to the journal version.

2 Preliminaries

A *temporal graph* is a pair $\langle G, \lambda \rangle$ such that $G = (V, E)$ is a simple (undirected) graph and $\lambda : E \rightarrow 2^{\mathbb{N}}$ is a time-labeling function that maps every edge of G onto a set of integers, called *timestamps* in the following (see the example in Figure 1). Up to a relabeling, we can assume that the minimum timestamp over all edges of G is equal to 1, while T denotes the maximum timestamp (and hence it upperbounds the number of timestamps).

We say that an edge $e \in E$ of a temporal graph $\langle G, \lambda \rangle$ is *active* in $t \in \lambda(e)$ and the pair (e, t) is called a *temporal edge*, while E_t is the set of temporal edges active in t .

A *temporal vertex cover* of $\langle G, \lambda \rangle$ is a pair (\mathcal{C}, σ) , such that: (i) $\mathcal{C} \subseteq V$; (ii) σ maps each vertex v of \mathcal{C} to an interval $[l_v, r_v]$ such that $1 \leq l_v \leq r_v \leq T$; and (iii) for every edge e and for every value $t \in \lambda(e)$, there is a vertex $v \in \mathcal{C}$ such that $t \in [l_v, r_v]$ and $e = \{u, v\}$. An ℓ -*partial temporal vertex cover* of $\langle G, \lambda \rangle$ is a function σ , called *assignment*, such that: (i) σ maps each vertex v of V to an interval $[l_v, r_v]$ such that $1 \leq l_v \leq r_v \leq T$; and (ii) $r_v - l_v + 1 \leq \ell$. A temporal edge (e, t) , where $e = \{u, v\}$, is *covered* by σ if either $t \in [l_u, r_u]$ or $t \in [l_v, r_v]$. Note that, in a ℓ -partial temporal vertex cover, we can assume w.l.o.g. each vertex is assigned to an interval of length exactly ℓ .

We are now ready to formalize the definition of ℓ -TimelineCover(k) and of the corresponding optimization version ℓ -MaxTimelineCover.

► **Problem 1.** ℓ -TimelineCover(k)

Input: a temporal graph $\langle G, \lambda \rangle$ and two positive integers ℓ and k .

Output: an ℓ -partial temporal vertex cover of $\langle G, \lambda \rangle$ that covers at least k temporal edges.

► **Problem 2.** ℓ -MaxTimelineCover

Input: a temporal graph $\langle G, \lambda \rangle$ and a positive integer ℓ .

Output: an ℓ -partial temporal vertex cover of $\langle G, \lambda \rangle$ that covers the maximum number of temporal edges over all ℓ -partial temporal vertex covers of $\langle G, \lambda \rangle$.

3 Hardness for Single Labeling

In this section, we prove that ℓ -TimelineCover(k) is NP-hard, even if the input temporal graph $\langle G, \lambda \rangle$ has the following properties: (1) each edge has a single label and (2) $T = 2$. As a corollary of this result, we prove that ℓ -MaxTimelineCover is APX-hard for the same restriction. The result is proven via a reduction from Max 2-3-SAT(h), a variant of Max 2-SAT(h) where each literal appears in at most three clauses. Given a set X of variables and a set of clauses C on X , where each clause consists of exactly two literals and each literal appears in at most three clauses, Max 2-3-SAT(h) asks for a truth assignment to the variables in X that satisfies at least h clauses in C . Note that we assume that each clause in C consists of exactly two literals. Indeed the APX-hardness proof of Max 2-3-SAT(h) in [3, 2] constructs only clauses consisting of exactly two literals.

Construction. Consider an instance $\langle X, C, h \rangle$ of Max 2-3-SAT(h), where $X = \{x_1, \dots, x_q\}$ is a set of variables and $C = \{C_1, \dots, C_z\}$ is a set of clauses, each one defined over two literals. A clause of C is written as $x_{i,A} \vee x_{j,B}$, with $A, B \in \{T, F\}$, where $x_{i,T}$ ($x_{i,F}$, respectively) represents a positive literal (a negative literal, respectively).

In the following, given $\langle X, C, h \rangle$, we define a corresponding instance $\langle G, \lambda, k, \ell \rangle$ of ℓ -TimelineCover(k), with $k = 24q + h$, $\ell = 1$ and $T = 2$; see Figure 2 for an illustration. Note that, since $T = 2$, the labels belong to interval $[1, 2]$. The set V is defined as follows:

$$V = \{v_{i,T}, v_{i,F}, a_{i,1}, a_{i,2}, a_{i,3}, a_{i,4}, b_{i,1}, b_{i,2}, b_{i,3}, b_{i,4} : x_i \in X\}.$$

Next, we define the set E_t of temporal edges:

$$E_t = \{(\{v_{i,T}, a_{i,p}\}, 1), (\{v_{i,F}, b_{i,p}\}, 1) : 1 \leq p \leq 4, 1 \leq i \leq q\} \cup$$

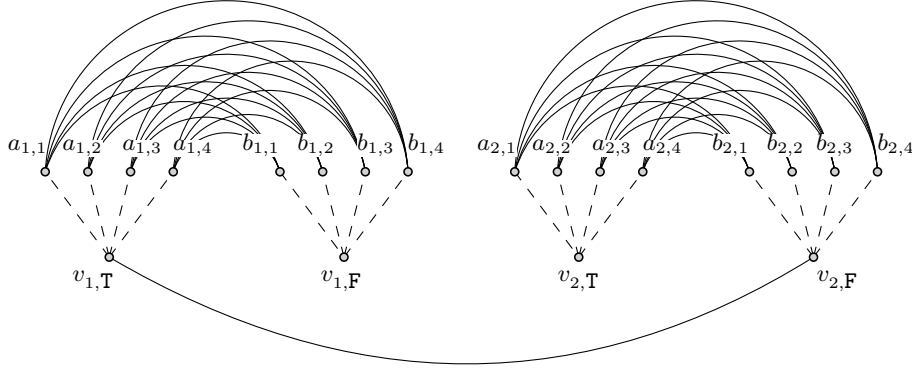
$$\{(\{a_{i,s}, b_{i,t}\}, 2) : 1 \leq i \leq q \wedge 1 \leq s, t \leq 4\} \cup$$

$$\{(\{v_{i,A}, v_{j,B}\}, 2) : 1 \leq i, j \leq q \wedge A, B \in \{T, F\} \wedge (x_{i,A} \vee x_{j,B}) \in C\}.$$

Clearly $\langle G, \lambda \rangle$ is defined over two timestamps. We prove below that each pair of vertices of $\langle G, \lambda \rangle$ is connected by at most one temporal edge.

► **Fact 1.** Let $\langle X, C, h \rangle$ be an instance of Max 2-3-SAT(h), and let $\langle G, \lambda, k, \ell \rangle$ be the corresponding instance of ℓ -TimelineCover(k). For every edge $e \in E$, it holds $|\lambda(e)| = 1$.

Proof. The edges connecting $v_{i,T}$ and $a_{i,p}$, $1 \leq i \leq q$ and $1 \leq p \leq 4$, are active only at timestamp 1 and the same property holds for the edges connecting $v_{i,F}$ and $b_{i,p}$, $1 \leq i \leq q$ and $1 \leq p \leq 4$. The edges between $a_{i,p}$ and $b_{i,s}$, with $1 \leq p \leq 4$ and $1 \leq s \leq 4$, for each $1 \leq i \leq q$, are active only at timestamp 2. Finally, the edges between $v_{i,A}$ and $v_{j,B}$, $1 \leq i \leq q$ and $1 \leq j \leq q$, are active only at timestamp 2. ◀



■ **Figure 2** An example of a temporal graph G built by the reduction for clause $(x_{1,T} \vee x_{2,F})$. The temporal edges defined at timestamp 1 are dashed, while those defined at timestamp 2 are solid .

Correctness. A solution of ℓ -TimelineCover(k) on $\langle G, \lambda, k, \ell \rangle$ is called *canonical* if every temporal edge $(\{v_{i,T}, a_{i,p}\}, 1)$, for $1 \leq i \leq q$ and $1 \leq p \leq 4$, and every temporal edge $(\{v_{i,F}, b_{i,p}\}, 1)$, for $1 \leq i \leq q$ and $1 \leq p \leq 4$, are covered by such a solution. We start by proving the following property.

► **Lemma 1.** *Given an instance $\langle X, C, h \rangle$ of Max 2-3-SAT(h), consider a corresponding instance $\langle G, \lambda, k, \ell \rangle$ of ℓ -TimelineCover(k). Then, starting from a feasible solution of ℓ -TimelineCover(k) on $\langle G, \lambda, k, \ell \rangle$, we can compute a feasible canonical solution of ℓ -TimelineCover(k) on $\langle G, \lambda, k, \ell \rangle$ that covers at least the same number of temporal edges.*

Proof. Consider an assignment σ to V and assume that there exist two vertices $a_{i,p}, b_{i,s} \in V$, for some i with $1 \leq i \leq q$, $1 \leq p, s \leq 4$ that are both assigned to $t = 1$. Notice that the temporal edge $(\{a_{i,p}, b_{i,s}\}, 2)$ is not covered and that each of $a_{i,p}, b_{i,s}$ covers at most one temporal edge (active in $t = 1$). Then we can modify the solution σ of ℓ -TimelineCover(k) by assigning one of the two vertices, w.l.o.g. $a_{i,p}$, to $t = 2$ so that the temporal edge $(\{a_{i,p}, b_{i,s}\}, 2)$, is now covered, while $(\{v_{i,T}, a_{i,p}\}, 1)$ is now possibly not covered. Notice that by iteratively applying this modification we can compute a solution of ℓ -TimelineCover(k) that covers the same number of temporal edges as σ , such that every $a_{i,p}$ or every $b_{i,s}$ is assigned to $t = 2$. Indeed assume this is not the case, then there exist two vertices $a_{i,p}, b_{i,s}$ both assigned to $t = 1$, thus by applying the modification described before we can compute a solution with the desired property.

Note that we assume that in σ either every $a_{i,p}$, $1 \leq p \leq 4$, or every $b_{i,s}$, $1 \leq s \leq 4$, is assigned to $t = 2$ and that either every $a_{i,p}$, $1 \leq p \leq 4$, or every $b_{i,s}$, $1 \leq s \leq 4$, is assigned to $t = 1$. Indeed, assume w.l.o.g. that every $a_{i,p}$ is assigned to $t = 2$, then all the temporal edges defined in timestamp 2 and incident in some $b_{i,s}$ are covered by vertices $a_{i,p}$. Hence we can assume that every $b_{i,s}$ is assigned to 1.

Now, we claim that at most one of $v_{i,T}, v_{i,F}$, $1 \leq i \leq q$, is assigned to $t = 2$. Indeed, assume that both $v_{i,T}, v_{i,F}$ are assigned to $t = 2$ (thus not to $t = 1$). Since either every vertex $a_{i,p}$ or every vertex $b_{i,s}$ is assigned to timestamp 2, it follows that either all temporal edges $(\{v_{i,T}, a_{i,p}\}, 1)$, with $1 \leq i \leq q$, and $1 \leq p \leq 4$, or all the temporal edges $(\{v_{i,F}, b_{i,p}\}, 1)$, $1 \leq i \leq q$, and $1 \leq p \leq 4$ are not covered. Assume w.l.o.g. that every $a_{i,p}$ is assigned to time $t = 2$. Since both $v_{i,T}$ and $v_{i,F}$ are assigned to time 2 and each literal in X belongs to at most three clauses, each of $v_{i,T}, v_{i,F}$ is assigned to $t = 2$ and it covers at most three temporal edges. Then we can compute a solution of ℓ -TimelineCover(k) on $\langle G, \lambda, k \rangle$ by assigning $v_{i,T}$ to $t = 1$, while each $a_{i,p}$ is assigned to $t = 2$ (or $v_{i,F}$ assigned to $t = 1$ and each $b_{i,p}$ is assigned to

11:6 Partial Temporal Vertex Cover with Bounded Activity Intervals

$t = 2$). The number of covered temporal edges with respect to solution σ is increased at least by one and we have that either $v_{i,T}$ is assigned to time 1 (if every $a_{i,p}$ is assigned to time 2) or $v_{i,F}$ is assigned to time 1 (if every $b_{i,p}$ is assigned to time 2). Then every temporal edge $(\{v_{i,T}, a_{i,p}\}, 1)$, $1 \leq i \leq q$, and $1 \leq p \leq 4$, and every temporal edge $(\{v_{i,F}, b_{i,p}\}, 1)$, $1 \leq i \leq q$, and $1 \leq p \leq 4$, is covered by the solution. Thus we have computed a canonical solution of ℓ -TimelineCover(k) on $\langle G, \lambda, k, \ell \rangle$ that covers at least the same number of temporal edges as σ , hence concluding the proof. \blacktriangleleft

Now, we can prove the main result of this section.

► **Theorem 2.** ℓ -TimelineCover(k) is NP-hard even on temporal graphs defined on two timestamps and where each edge is assigned a single time label.

Proof. Note that, by Fact 1, each edge is assigned a single time label and by construction, the temporal graph is defined on two timestamps.

We start by proving the following fact.

► **Fact 2.** Given a solution of Max 2-3-SAT(h) on instance $\langle X, C, h \rangle$ we can compute in polynomial time a canonical solution of ℓ -TimelineCover(k) on the corresponding instance $\langle G, \lambda, k, \ell \rangle$ with $k = 24q + h$ (hence that covers at least k temporal edges).

Proof. Consider a solution of Max 2-3-SAT(h) on instance $\langle X, C, h \rangle$, we define a solution σ of ℓ -TimelineCover(k) on $\langle G, \lambda, k, \ell \rangle$ as follows. For each variable x_i , $1 \leq i \leq q$, that is set to true, then $v_{i,F}$ is assigned to $t = 1$, each $a_{i,p}$, with $1 \leq p \leq 4$, is assigned to $t = 1$, $v_{i,T}$ is assigned to $t = 2$ and each $b_{i,p}$, with $1 \leq p \leq 4$, is assigned to $t = 2$. For each variable x_i , $1 \leq i \leq q$, that is set to false, then $v_{i,T}$ is assigned to $t = 1$, each $b_{i,p}$, with $1 \leq p \leq 4$, is assigned to $t = 1$, $v_{i,F}$ is assigned to $t = 2$ and each $a_{i,p}$, with $1 \leq p \leq 4$, is assigned to $t = 2$. By construction the solution σ is canonical, hence the $8q$ temporal edges defined at time $t = 1$ are covered. Each temporal edge $(\{a_{i,p}, b_{i,s}\}, 2)$, with $1 \leq p, s \leq 4$, is covered (we have $16q$ such temporal edges). Finally, by construction, for each satisfied clause, the corresponding temporal edge defined in $t = 2$ is covered (we have h such temporal edges). \blacktriangleleft

For the second direction, we prove the following fact.

► **Fact 3.** Given a solution of ℓ -TimelineCover(k) on the instance $\langle G, \lambda, k, \ell \rangle$ with $k = 24q + h$ (hence that covers at least k temporal edges), we can compute in polynomial time a solution Max 2-3-SAT(h) on instance $\langle X, C, h \rangle$ (hence that satisfies h clauses).

Proof. By Lemma 1 we can consider a canonical solution σ of ℓ -TimelineCover(k) on instance $\langle G, \lambda, k, \ell \rangle$. By construction σ covers the $8q$ temporal edges defined at time $t = 1$. Notice that we can assume that exactly one of $v_{i,T}, v_{i,F}$ is assigned to $t = 1$. If both $v_{i,T}, v_{i,F}$ are assigned to $t = 1$, we can define all the vertices $a_{i,p}$ (all the vertices $b_{j,p}$, respectively) assigned to $t = 1$ and assign $v_{i,T}$ ($v_{i,F}$, respectively) to $t = 2$. Hence exactly one of $v_{i,T}, v_{i,F}$ is assigned to $t = 1$, and exactly one of $v_{i,T}, v_{i,F}$ is assigned to $t = 2$. Moreover, we can assume that for each i with $1 \leq i \leq q$, the temporal edges incident to $a_{i,p}$ and $b_{i,s}$, with $1 \leq p, s \leq 4$, are covered.

Now, construct a truth assignment as follows. For each $1 \leq i \leq q$, if $v_{i,T}$ is assigned to $t = 2$, then set the corresponding variable x_i to true, if $v_{i,F}$ is assigned to $t = 2$, then set the corresponding variable x_i to false. By construction if a temporal edge $(\{x_{i,A}, x_{j,B}\}, 2)$ is covered, then the corresponding clause is satisfied, thus we have defined a truth assignment that satisfies at least h clauses, hence a solution of Max 2-3-SAT(h), concluding the proof. \blacktriangleleft

By Fact 2 and by Fact 3 it follows that we have designed a polynomial-time reduction from Max 2-3-SAT(h) to ℓ -TimelineCover(k). By the NP-hardness of Max 2-3-SAT(h) (the decision version) [3], it follows that also ℓ -TimelineCover(k) on temporal graphs defined on two timestamps and where each edge is assigned a single time label is NP-hard. ◀

We note that the reduction described above can be used to prove the APX-hardness of ℓ -MaxTimelineCover, thus implying that ℓ -MaxTimelineCover does not admit a PTAS. Later, in Section 5, we will prove that ℓ -MaxTimelineCover admits an approximation algorithm of factor $\frac{3}{4}$.

► **Theorem 3.** *ℓ -MaxTimelineCover is APX-hard.*

Proof. The result follows from the fact that essentially the same reduction described in this section is an L -reduction from the optimization version of Max 2-3-SAT(h) to ℓ -MaxTimelineCover (for details on L -reduction we refer to [20]).

Denote by I an instance of the optimization version of Max 2-3-SAT(h) and by I' the corresponding instance of ℓ -MaxTimelineCover. Let $OPT_S(I)$ be the value of an optimum solution of the optimization version of Max 2-3-SAT(h) on instance I . Let $OPT_M(I')$ be the value of an optimum solution of ℓ -MaxTimelineCover on instance I' .

By Fact 2, we have that

$$OPT_M(I') \leq 24q + OPT_S(I)$$

and, observing that, since there is a truth assignment that satisfies at least $\frac{1}{2}q$ clauses, we have that $OPT_S(I) \geq \frac{1}{2}q$. It follows that

$$OPT_M(I') \leq 24q + OPT_S(I) \leq 48 OPT_S(I) + OPT_S(I) = 49 OPT_S(I)$$

Consider the value A' (number of covered temporal edges) of a feasible solution of ℓ -MaxTimelineCover on instance I' and the value A (number of satisfied clauses) of a feasible solution of the optimization version of Max 2-3-SAT(h) on instance I .

By Fact 3, we have that, given a feasible solution of ℓ -MaxTimelineCover of value A' on I' we can compute in polynomial time a feasible solution of the optimization version of Max 2-3-SAT(h) on instance I of value A such that

$$|OPT_S(I) - A| \leq |OPT_M(I') - A'|$$

Thus we have designed an L -reduction from the optimization version of Max 2-3-SAT(h) to ℓ -MaxTimelineCover. Since the optimization version of Max 2-3-SAT(h) is known to be APX-hard [2, 3], the APX-hardness holds also for ℓ -MaxTimelineCover. ◀

4 Fixed Parameter Tractability

In this section, we show that ℓ -TimelineCover(k) is FPT when parameterized by: the number $h = |E| - k$ of temporal edges that may not be covered by the solution (Section 4.1); parameter k and parameter $n + \ell$ (Section 4.2), where n denotes the number of vertices of the input graph.

4.1 Parameter h

The result is obtained by a parameterized reduction to Almost 2-SAT(p), which is known to be FPT when parameterized by p [18, 15], with a similar approach applied for MinTimelineCover in [7]. We recall that, given a 2-CNF formula and a positive integer p , Almost 2-SAT(p) asks whether it is possible to remove at most p clauses so that the resulting formula is satisfiable.

► **Theorem 4.** ℓ -TimelineCover(k) is FPT when parameterized by h .

Proof. Given an instance $I = \langle G, \lambda \rangle$ of ℓ -TimelineCover(k) and denoted by $h = |E| - k$ the number of temporal edges that are not covered, we define a corresponding instance I' of Almost 2-SAT(p), with $p = h$, as follows. To ease the notation, we shall assume that all timestamps $t \in [1, T]$ are such that $E_t \neq \emptyset$; if this is not the case, the algorithm can be easily modified to avoid any computation in those timestamps t for which $E_t = \emptyset$.

- For each vertex $v \in V$ and for each timestamp $i \in [1, T]$, we create a variable v_i .
- For each vertex $v \in V$, for each timestamp $i \in [1, T]$, and for each $j > i + \ell$, we create $h + 1$ copies of clause $(\bar{v}_i \vee \bar{v}_j)$, which is called a *vertex clause*.
- For each edge $\{u, v\} \in E$ such that $\lambda(\{u, v\}) = i$ (with $i \in [1, T]$), we create a clause $(u_i \vee v_i)$, which is called a *temporal edge clause*.

Intuitively, each copy of the vertex clauses models the fact that a vertex is assigned to an interval of length exactly ℓ . More formally, denoted by i the first timestamp of the interval that is assigned to a vertex v , the clause $(\bar{v}_i \vee \bar{v}_j)$ ensures that, for each $j > i + \ell$, time j does not belong to the interval assigned to v . Also, a temporal edge clause $(u_i \vee v_i)$ models the fact that a temporal edge $(e = \{u, v\}, i)$ is covered, because u or v is assigned some time interval that includes i .

Note that $p = h$, and that I' can be computed in polynomial time. We now prove that I' is a yes-instance of Almost 2-SAT(p) if and only if I is a yes-instance of ℓ -TimelineCover(k).

Let I be a yes-instance of ℓ -TimelineCover(k). Since I is a yes-instance, we know that $k' \geq k$ temporal edges are covered and hence $h' = |E| - k' \leq h$ temporal edges are not covered. The corresponding Almost 2-SAT(p) instance I' contains h' clauses that are not satisfied, and these clauses are temporal edge clauses. Indeed, since we have $h + 1$ copies for each vertex clause, having one vertex clause that is not satisfied would imply that $h + 1$ clauses of the formula are not satisfied, but we observed $h' \leq h$. By removing the $h' \leq p$ temporal edge clauses that are not satisfied, the formula is satisfied. This implies that I' is a yes-instance of Almost 2-SAT(p).

Now, let I' be a yes-instance of Almost 2-SAT(p). Since I' is a yes-instance, the formula is satisfied by removing at most $p = h$ clauses. Since the formula contains $h + 1 = p + 1$ copies of each vertex clause, the clauses that are not satisfied are temporal edge clauses, while all vertex clauses are satisfied. Each unsatisfied temporal edge clause in I' implies that there is a temporal edge in I that is not covered. It follows that I is a yes-instance of ℓ -TimelineCover(k). ◀

4.2 Parameter k

We shall assume that every vertex is incident to at least one temporal edge, otherwise, we can just remove such a vertex and solve the problem for the obtained subgraph of G . We distinguish two cases.

• **Case 1:** $k \leq \frac{n}{2}$. We first compute a spanning forest \mathcal{F} of the graph G . Assuming w.l.o.g. that G contains no isolated vertex, we note that \mathcal{F} has at least $\frac{n}{2}$ edges. We then root each tree of \mathcal{F} , and randomly select k edges from \mathcal{F} . Next, we associate each temporal edge $(\{u, v\}, i)$ of the k selected edges to vertex v , where u is the parent of v in the forest. This implies that no two temporal edges are associated to the same vertex. Let (e, i) be one of such temporal edges and let v be the vertex associated to it. We map v to an interval $[i, i + \ell - 1]$. We apply the process mentioned above for each of the k edges. This approach allows us to cover k temporal edges.

• **Case 2:** $k > \frac{n}{2}$. In this case, we use dynamic programming as follows. First, to ease the notation, we shall assume that all timestamps $t \in [1, T]$ are such that $E_t \neq \emptyset$; if this is not the case, the algorithm can be easily modified to avoid any computation in those timestamps t for which $E_t = \emptyset$.

Suppose that we already performed our computations on each timestamp smaller than i , with $i \in [2, T]$, and that we are now analyzing timestamp i . Suppose we have the records $R_1^{i-1}, \dots, R_g^{i-1}$ associated with timestamp $i - 1$; the cardinality g of this set of records depends on the parameter k and will be analyzed later. Each record R_j^{i-1} , with $j \in [1, g]$, is composed of the following information:

- A table A_j^{i-1} where we store all the vertices v that we previously assigned to an interval starting in timestamp x , with $i - \ell \leq x \leq i - 1$. In this table, we also associate to v the value x of the first timestamp where v is assigned. These vertices are said to be *on-vertices* or simply *on*.
- A set of vertices \bar{A}_j^{i-1} that were previously assigned to an interval starting in timestamp x such that $x + \ell < i - 1$ by the algorithm. Notice that these vertices are not going to cover any temporal edge (e, q) where $q \geq i$. These vertices are said to be *off-vertices* or simply *off*.
- A number s_j^{i-1} of temporal edges that are covered by the vertices in A_j^{i-1} and \bar{A}_j^{i-1} . This is the *score* of the record.

Observe that the number of possible different sets of off-vertices is a function of n , which is upperbounded by $2k$ because $k \geq \frac{n}{2}$ by assumption. Also, the score of possible different costs is upperbounded by k by definition, since we only care about covering k edges. Concerning the vertices that are on-vertices in some interval $[x, x + \ell - 1]$, observe that we can assume that there are no more than k temporal edges active in this interval, since in this case we can simply assign this interval to every vertex of the graph and the solution is trivial. Also, we assume that we assign a vertex to an interval $[x, x + \ell - 1]$ if and only if there exists a temporal edge (e, x) incident to v . If this is not the case, we could simply associate v to the interval that starts with the first timestamp where a temporal edge incident to v is covered, thus covering not fewer temporal edges with v . Since there are less than k temporal edges (e, q) such that $q \in [x, x + \ell - 1]$ and by the above assumption, we can assume w.l.o.g. that there are less than k vertices that are on-vertices in each record.

Now, we prove the next lemma:

► **Lemma 5.** *For each $i - 1 \in [2, T]$, where $R_1^{i-1}, \dots, R_g^{i-1}$ are the records at timestamp $i - 1$, we have $g \in O(2^{k \log k})$ and $\max_{i=1}^g |R_j^{i-1}| \in O(k \log T)$.*

Proof. Overall, each table in one record contains $O(k)$ items. Concerning the size of a single item, each item of A_j^{i-1} , for each $j \in [1, g]$, represents a value in $O(T)$. Thus, $\max_{i=1}^g |R_j^{i-1}| \in O(k \log T)$. Concerning the value of g , we note that the number of possible values x associated with a vertex in A_j^{i-1} is at most k , because there are k possible temporal edges incident to v in an interval of length ℓ starting in x . Consequently, we have $O(2^{k \log k})$ distinct records, i.e., $g \in O(2^{k \log k})$. ◀

Algorithm description. In the base case, consider timestamp 1. Let S_1, \dots, S_{2^n} be all the possible subsets of vertices of V and assume that S_1, \dots, S_g , with $g \leq 2^n$, are the possible subsets of the vertices that have a temporal edge active in timestamp 1. For each $j \in [1, g]$, we create a record R_j^1 by setting: $A_j^1 = S_j$ and each of such vertices is associated to the value (timestamp) 1; $\bar{A}_j^1 = \emptyset$; the score s_j^1 can be computed in $O(k)$ time.

We now consider the inductive case. Let $R_1^{i-1}, \dots, R_g^{i-1}$ be the set of records computed at timestamp $i - 1$. For each R_j^{i-1} , where $j \in [1, g]$, we proceed as follows. We consider the vertex set $V \setminus (A_j^{i-1} \cup \bar{A}_j^{i-1})$ and every possible subset of this set. For each subset S , we

11:10 Partial Temporal Vertex Cover with Bounded Activity Intervals

construct a new record R_p^i that we will associate with time i . We assume that these and only these are the vertices that are associated with the interval $[i, i + \ell - 1]$ for the partial solution described by R_p^i . Let S' be the set of vertices in A_j^{i-1} that are assigned to interval $[i - \ell, i - 1]$; note that the information about which vertices of A_j^{i-1} are assigned to interval $[i - \ell, i - 1]$ is contained in A_j^{i-1} (these are the vertices of A_j^{i-1} that are associated with timestamp $i - \ell - 1$).

- The vertices of R_p^i that are on-vertices (hence those in A_p^i), are going to be the vertices in $(A_j^{i-1} \setminus S') \cup S$; each vertex in S is associated with value i , each vertex in $A_p^i \setminus S$ has the same timestamp it is associated with in A_j^{i-1} . Note that we assume that each vertex u in S has a temporal edge defined in i and covered by u .
- The off-vertices of R_p^i (hence those in $\overline{A_p^i}$) are going to be the vertices in $A_j^{i-1} \cup S'$.
- The score of R_p^i is $s_j^{i-1} + s$, where s is the number of temporal edges covered by vertices of S (that we just associated with an interval $[i, i + \ell - 1]$) that are not covered by vertices in A_j^{i-1} . Computing the score s of R_p^i requires $O(k^2)$ time. Indeed, there exist at most k temporal edges in the interval $[i, i + \ell - 1]$. We identify the temporal edges defined in $[i, i + \ell - 1]$ and not covered by on-vertices of A_j^{i-1} in $O(k^2)$ time (for each temporal edge defined in $[i, i + \ell - 1]$ we can check if it is covered by some on-vertex of A_j^{i-1} in $O(k)$ time, since A_j^{i-1} contains less than k on-vertices). Then, for each uncovered temporal edge, we check that it is covered by some vertex in S in $O(k^2)$ time (for each temporal edge we can check if it is covered by some vertex of S in $O(k)$ time, since S contains at most k vertices).

Following from the discussion above, the time complexity needed to perform the above operations does not depend on ℓ , but only on k . Since the number of vertices of the graph is $n \leq 2k$, updating a record requires $O(f(k))$ time, where $f(\cdot)$ is a computable function.

Hence, by Lemma 5 we have the following theorem.

► **Theorem 6.** ℓ -TimelineCover(k) is FPT when parameterized by k .

It is worth noting that, since in each timestamp a vertex can cover at most $n - 1$ edges, it follows that $k \leq \ell \cdot n^2$. Thus, we observe the following.

► **Theorem 7.** ℓ -TimelineCover(k) is FPT when parameterized by $n + \ell$.

5 A $\frac{3}{4}$ -Approximation Algorithm

In this section, we present an approximation algorithm achieving factor $\frac{3}{4}$ based on randomized rounding and inspired by the approximation algorithm given in [9] for Max Sat.

► **Theorem 8.** There is a polynomial-time approximation algorithm for ℓ -MaxTimelineCover with factor $\frac{3}{4}$.

To prove Theorem 8, we first define an ILP formulation for the ℓ -MaxTimelineCover problem (Section 5.1). Next, we describe an algorithm based on randomized rounding an LP relaxation of this formulation (Section 5.2).

5.1 ILP formulation

We present an Integer Linear Programming (ILP) formulation of ℓ -MaxTimelineCover. We make use of the following variables:

- For each temporal edge $(\{v_i, v_j\}, t)$, the variable $e_{i,j,t}$ is 1 if $(\{v_i, v_j\}, t)$ is covered, and it is 0 otherwise.
- For each vertex v_i and for each $t \in [1, T - \ell + 1]$, the variable $A_i(t)$ is 1 if v_i is assigned to interval $[t, t + \ell - 1]$, and it is 0 otherwise.

$$\max \sum_{i,j,t} e_{i,j,t} \quad (1)$$

s. t.

$$e_{i,j,t} \leq \sum_{t_1 \in [t-\ell+1, t]} A_i(t_1) + \sum_{t_2 \in [t-\ell+1, t]} A_j(t_2) \quad \forall (\{v_i, v_j\}, t) \quad (2)$$

$$\sum_t A_i(t) \leq 1 \quad \forall v_i \in V \quad (3)$$

$$e_{i,j,t} \in \{0, 1\}, \quad \forall (\{v_i, v_j\}, t) \quad (4)$$

$$A_i(t) \in \{0, 1\} \quad \forall v_i \in V, \quad (5)$$

$$\forall t \in [1, 2, \dots, T - \ell]$$

Inequality (2) guarantees that a variable $e_{i,j,t}$ can be set to 1 only if at least one end-vertex is mapped to an interval containing t , while inequality (3) guarantees that each vertex is mapped to at most one interval.

5.2 The Approximation Algorithm

The $\frac{3}{4}$ -factor approximation algorithm for the ℓ -MaxTimelineCover problem is presented in Algorithm 1. The algorithm solves in polynomial time an LP relaxation of the ILP formulation described in Section 5.1, where variables $e_{i,j,t} \in [0, 1]$ and $A_i(t) \in [0, 1]$. We denote by $A_i^*(t)$ and $e_{i,j,t}^*$ the values of variables $A_i(t)$ and $e_{i,j,t}$, respectively, of the optimal solution of the LP relaxation.

Starting from a solution of the relaxation, the approximation algorithm defines a solution for ℓ -MaxTimelineCover by assigning each vertex $v_i \in V$ to interval $[t, t + \ell - 1]$ with probability $A_i^*(t)$.

■ **Algorithm 1** $\frac{3}{4}$ -approximation algorithm for the ℓ -MaxTimelineCover problem.

Solve the LP relaxation of the ILP formulation from Section 5.1, with constraints $e_{i,j,t} \in [0, 1]$ and $A_i(t) \in [0, 1]$

Let $e_{i,j,t}^*$ and $A_i^*(t)$ be the values of a solution to the relaxation of the ILP from Section 5.1
For every vertex v_i , define it active in interval $[t, t + \ell - 1]$ with probability $A_i^*(t)$

Let $\mathbb{E}(\sigma)$ be the expected value of a solution σ returned by Algorithm 1. Denote by $P[e_{i,j,t}]$ the probability that the temporal edge $(\{v_i, v_j\}, t)$ is covered. It holds that

$$\mathbb{E}(\sigma) = \sum_{i,j,t} P[e_{i,j,t}].$$

Consider now $P[e_{i,j,t}]$, it holds that

$$P[e_{i,j,t}] = 1 - P[\overline{e_{i,j,t}}],$$

where $\overline{e_{i,j,t}}$ is the event that the temporal edge $(\{v_i, v_j\}, t)$ is not covered by solution σ . We have that

$$1 - P[\overline{e_{i,j,t}}] = 1 - P[(\{v_i, v_j\}, t) \text{ not cov. by } v_i \wedge (\{v_i, v_j\}, t) \text{ not cov. by } v_j]. \quad (6)$$

11:12 Partial Temporal Vertex Cover with Bounded Activity Intervals

Now, we have that

$$P[(\{v_i, v_j\}, t) \text{ not cov. by } v_i \wedge (\{v_i, v_j\}, t) \text{ not cov. by } v_j] = \left(1 - \sum_{t_1 \in [t-\ell+1, t]} A_i^*(t_1)\right) \left(1 - \sum_{t_2 \in [t-\ell+1, t]} A_j^*(t_2)\right) \quad (7)$$

By combining the previous equations we have that

$$1 - P[\overline{e_{i,j,t}}] = 1 - \left(1 - \sum_{t_1 \in [t-\ell+1, t]} A_i^*(t_1)\right) \left(1 - \sum_{t_2 \in [t-\ell+1, t]} A_j^*(t_2)\right). \quad (8)$$

From the arithmetic mean inequality, we have that

$$\begin{aligned} 1 - \left(1 - \sum_{t_1 \in [t-\ell+1, t]} A_i^*(t_1)\right) \left(1 - \sum_{t_2 \in [t-\ell+1, t]} A_j^*(t_2)\right) &\geq \\ 1 - \left(\frac{1 - \sum_{t_1 \in [t-\ell+1, t]} A_i^*(t_1) + 1 - \sum_{t_2 \in [t-\ell+1, t]} A_j^*(t_2)}{2}\right)^2 &= \\ 1 - \left(1 - \frac{(\sum_{t_1 \in [t-\ell+1, t]} A_i^*(t_1) + \sum_{t_2 \in [t-\ell+1, t]} A_j^*(t_2))}{2}\right)^2. \end{aligned}$$

Recall that $e_{i,j,t}^*$ is the value of variable $e_{i,j,t}$ returned by the relaxation of the ILP formulation of Section 5.1. By Inequality (2) of this formulation, we have that:

$$\sum_{t_1 \in [t-\ell+1, t]} A_i^*(t_1) + \sum_{t_2 \in [t-\ell+1, t]} A_j^*(t_2) \geq e_{i,j,t}^*. \quad (9)$$

Thus

$$\begin{aligned} 1 - P[\overline{e_{i,j,t}}] &\geq \\ 1 - \left(1 - \frac{(\sum_{t_1 \in [t-\ell+1, t]} A_i^*(t_1) + \sum_{t_2 \in [t-\ell+1, t]} A_j^*(t_2))}{2}\right)^2 &\geq 1 - \left(1 - \frac{e_{i,j,t}^*}{2}\right)^2. \end{aligned}$$

Hence, $P[e_{i,j,t}]$ can be bounded as follows:

$$P[e_{i,j,t}] = 1 - P[\overline{e_{i,j,t}}] \geq 1 - \left(1 - \frac{e_{i,j,t}^*}{2}\right)^2.$$

The function

$$1 - \left(1 - \frac{e_{i,j,t}^*}{2}\right)^2$$

is a concave function and it has value 0 for $e_{i,j,t}^* = 0$ and value $\frac{3}{4}$ for $e_{i,j,t}^* = 1$. It follows that

$$P[e_{i,j,t}] \geq 1 - \left(1 - \frac{e_{i,j,t}^*}{2}\right)^2 \geq \frac{3}{4} e_{i,j,t}^*,$$

thus concluding the proof.

6 Conclusion

In this paper, we introduced and studied the ℓ -TimelineCover(k) problem (and its optimization version ℓ -MaxTimelineCover), a variant of the classical Vertex Cover problem inspired by a recent stream of work on temporal graphs. We have established the NP-hardness of ℓ -TimelineCover(k) and the APX-hardness of ℓ -MaxTimelineCover, under the restricted condition where the temporal domain consists of only two timestamps and each edge appears at most once. We have presented two fixed-parameter algorithms for the following parameters: (i) the number k of temporal edges covered by the solution, and (ii) the number h of temporal edges *not* covered by the solution. Furthermore, we have contributed a $\frac{3}{4}$ -approximation algorithm for ℓ -MaxTimelineCover based on randomized rounding.

There are some interesting research directions to explore. First, the parameterized complexity of the problem can be further investigated, similarly to what has been done for MinTimelineCover in [7]. Second, it would be interesting to improve the approximation factor for ℓ -MaxTimelineCover, possibly considering the semidefinite programming technique applied for Max Sat [10]. A third possible direction involves expanding the definition of vertex activity by permitting a finite number of intervals during which a vertex can be active, as done for MinTimelineCover in [7, 19].

References

- 1 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *J. Comput. Syst. Sci.*, 107:108–123, 2020. doi:10.1016/j.jcss.2019.08.002.
- 2 Piotr Berman and Marek Karpinski. On some tighter inapproximability results. *Electron. Colloquium Comput. Complex.*, TR98-029, 1998. arXiv:TR98-029.
- 3 Piotr Berman and Marek Karpinski. On some tighter inapproximability results (extended abstract). In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 1999. doi:10.1007/3-540-48523-6_17.
- 4 Riccardo Dondi. Untangling temporal graphs of bounded degree. *Theor. Comput. Sci.*, 969:114040, 2023. doi:10.1016/J.TCS.2023.114040.
- 5 Riccardo Dondi and Manuel Lafond. An FPT algorithm for temporal graph untangling. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation, IPEC 2023, September 6-8, 2023, Amsterdam, The Netherlands*, volume 285 of *LIPICs*, pages 12:1–12:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.IPEC.2023.12.
- 6 Riccardo Dondi and Alexandru Popa. Timeline cover in temporal graphs: Exact and approximation algorithms. In Sun-Yuan Hsieh, Ling-Ju Hung, and Chia-Wei Lee, editors, *Combinatorial Algorithms - 34th International Workshop, IWOCA 2023, Tainan, Taiwan, June 7-10, 2023, Proceedings*, volume 13889 of *Lecture Notes in Computer Science*, pages 173–184. Springer, 2023. doi:10.1007/978-3-031-34347-6_15.
- 7 Vincent Froese, Pascal Kunz, and Philipp Zschoche. Disentangling the computational complexity of network untangling. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 2037–2043. ijcai.org, 2022. doi:10.24963/ijcai.2022/283.
- 8 Rajiv Gandhi, Samir Khuller, and Aravind Srinivasan. Approximation algorithms for partial covering problems. *Journal of Algorithms*, 53(1):55–84, 2004.

- 9 Michel X. Goemans and David P. Williamson. New $3/4$ -approximation algorithms for the maximum satisfiability problem. *SIAM J. Discret. Math.*, 7(4):656–666, 1994. doi:10.1137/S0895480192243516.
- 10 Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995. doi:10.1145/227683.227684.
- 11 Thekla Hamm, Nina Klobas, George B. Mertzios, and Paul G. Spirakis. The complexity of temporal vertex cover in small-degree graphs. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelfth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 10193–10201. AAAI Press, 2022.
- 12 Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88(9):234, 2015.
- 13 Petter Holme and Jari Saramäki. A map of approaches to temporal networks. In *Temporal Network Theory*, pages 1–24. Springer, 2019.
- 14 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.
- 15 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.
- 16 Julián Mestre. A primal-dual approximation algorithm for partial vertex cover: Making educated guesses. *Algorithmica*, 55:227–239, 2009.
- 17 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.
- 18 Igor Razgon and Barry O’Sullivan. Almost 2-sat is fixed-parameter tractable. *J. Comput. Syst. Sci.*, 75(8):435–450, 2009. doi:10.1016/J.JCSS.2009.04.002.
- 19 Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. The network-untangling problem: from interactions to activity timelines. *Data Min. Knowl. Discov.*, 35(1):213–247, 2021. doi:10.1007/s10618-020-00717-5.
- 20 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

Parameterized Algorithms for Multi-Label Periodic Temporal Graph Realization

Thomas Erlebach  

Department of Computer Science, Durham University, Durham, UK

Nils Morawietz  

Friedrich Schiller University Jena, Institute of Computer Science, Jena, Germany

Petra Wolf   

LaBRI, CNRS, Université de Bordeaux, Bordeaux INP, France

Abstract

In the periodic temporal graph realization problem introduced by Klobas et al. [SAND '24] one is given a period Δ and an $n \times n$ matrix D of desired fastest travel times, and the task is to decide if there is a simple periodic temporal graph with period Δ such that the fastest travel time between any pair of vertices matches the one specified by D . We generalize the problem from simple temporal graphs to temporal graphs where each edge can appear up to ℓ times in each period, for some given integer ℓ . For the resulting problem MULTI-LABEL PERIODIC TGR, we show that it is fixed-parameter tractable for parameter n and for parameter $vc + \Delta$, where vc is the vertex cover number of the underlying graph. We also show the existence of a polynomial kernel for parameter $nu + d_{\max}$, where nu is the number of non-universal vertices of the underlying graph and d_{\max} is the largest entry of D . Furthermore, we show that the problem is *NP*-hard for each $\ell \geq 5$, even if the underlying graph is a tree, a case that was known to be solvable in polynomial time if the task is to construct a simple periodic temporal graph, that is, if $\ell = 1$.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Fixed-Parameter Tractability, Almost-Clique, Kernelization, Dynamic Network, Temporal Graph

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.12

Funding *Thomas Erlebach*: Supported by EPSRC grants EP/S033483/2 and EP/T01461X/1.

Petra Wolf: Supported by the French ANR, project ANR-22-CE48-0001 (TEMPOGRAL).

1 Introduction

Graph realization problems are problems where one is given information about a certain property of a graph, such as the matrix of shortest-path distances or the degree sequence, and wants to decide whether there exists a graph for which that property matches the given information (and to find such a graph, if it exists). A wide range of graph realization problems have been studied for static graphs for many years, with the work by Erdős and Gallai [9] on realizing a given degree sequence and the work by Hakimi and Yau [10] on realizing a given distance matrix by an edge-weighted graph being two particularly early examples. In recent years, temporal graphs, i.e., graphs whose edge set may change in each time step, have received substantial attention. In temporal graphs, one usually considers paths that traverse at most one edge in each time step (and we also do so in this paper), although non-strict paths where several edges can be traversed in the same time step have also been studied. Many classical graph problem have been adapted and studied in the temporal graph setting (see [14] for an introduction to temporal graphs and [4] for a broader overview of different classes of time-varying graphs). Therefore, considering graph realization problems in the temporal graph setting is a natural and timely direction. Very recently, Klobas et al. [12] have started this line of research and introduced the following periodic temporal



© Thomas Erlebach, Nils Morawietz, and Petra Wolf;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 12; pp. 12:1–12:16

Leibniz International Proceedings in Informatics



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

graph realization problem (PERIODIC TGR): Given a period Δ and an $n \times n$ integer matrix D that specifies the desired fastest travel times for each pair of vertices, find a simple periodic temporal graph \mathcal{G} with period Δ such that the fastest travel times in \mathcal{G} match those given by D (or decide that no such temporal graph exists). Here, a periodic temporal graph is *simple* if each edge of the underlying graph appears exactly once in each period.

Klobas et al. [12] noted that the consideration of periodic temporal graphs can be motivated by, for example, railway networks, satellite networks, or social networks. They showed that PERIODIC TGR is *NP*-hard for any $\Delta \geq 3$ and also *W*[1]-hard when parameterized by the feedback vertex number of the underlying graph. (The latter result also applies to the non-periodic version of the problem if the distance matrix can have entries equal to ∞ .) Here, the underlying graph is the graph containing edges between all vertex pairs that have distance 1 according to D . They showed that the problem can be solved in polynomial time if the underlying graph is a tree or a cycle, and that it is fixed-parameter tractable (FPT) when parameterized by the feedback edge number of the underlying graph. Finally, they raised a number of interesting questions for future research, including the investigation of PERIODIC TGR with the vertex cover number of the underlying graph as parameter and with parameter combinations that include a structural parameter and the period Δ .

Our contribution. In this paper, we follow up on the work by Klobas et al. [12]. Furthermore, we generalize PERIODIC TGR from simple periodic temporal graphs to ℓ -label periodic temporal graphs, i.e., to periodic temporal graphs where each edge of the underlying graph is allowed to appear up to ℓ times in each period. This problem (MULTI-LABEL PERIODIC TGR) is defined as follows: Given a period Δ and an $n \times n$ integer matrix D that specifies the desired fastest travel times for each pair of vertices and a positive integer ℓ , find an ℓ -label periodic temporal graph \mathcal{G} with period Δ such that the fastest travel times in \mathcal{G} match those given by D (or decide that no such temporal graph exists). Clearly, PERIODIC TGR is the special case of MULTI-LABEL PERIODIC TGR where $\ell = 1$. We also consider the non-periodic version (MULTI-LABEL TGR). While Klobas et al. mainly considered parameters that relate to how close the underlying graph is to being a tree, we explore also the opposite end of the spectrum and consider a parameter that measures how close the underlying graph is to being a clique, namely the number of non-universal vertices. As the problem is trivial if the underlying graph is a clique, parameters that measure closeness to a clique are interesting candidates for obtaining FPT algorithms.

We obtain the following main results:

- The known *NP*-hardness proof for PERIODIC TGR [12] only applies to MULTI-LABEL PERIODIC TGR with $\ell = 1$ and leaves open the complexity for $\ell \geq 2$. We show that MULTI-LABEL PERIODIC TGR is *NP*-hard for every $\ell \geq 1$ even if the largest entry in D is 3. For $\ell \geq 3$, we show *NP*-hardness even if the underlying graph has a size-1 feedback vertex set.
- In contrast to the known result that PERIODIC TGR can be solved in polynomial time for trees [12], we show that MULTI-LABEL PERIODIC TGR is *NP*-hard for any $\ell \geq 5$ even if the underlying graph is a star.
- Both MULTI-LABEL PERIODIC TGR and MULTI-LABEL TGR are FPT for parameter n . Here, n is the number of vertices. (For PERIODIC TGR, this result is implied by the FPT algorithm for parameter feedback edge number by Klobas et al. [12], but our algorithm is conceptually simpler and can handle the multi-label problem variants.)
- MULTI-LABEL PERIODIC TGR is FPT for parameter $vc + \Delta$, where vc is the vertex cover number of the underlying graph.

- MULTI-LABEL TGR can be solved in $O(pn^4)$ time if D has no entries equal to ∞ , $\ell \geq n^2$, and the underlying graph is such that for each pair (u, v) of vertices the number of u - v paths is at most p . For trees and cycles, we have $p \leq 2$, and hence the algorithm runs in polynomial time.
- MULTI-LABEL PERIODIC TGR admits a polynomial kernel for parameter $\text{nu} + d_{\max}$ and is hence FPT for that parameter, where nu is the number of non-universal vertices of the underlying graph and d_{\max} is the largest entry of D .

The remainder of the paper is structured as follows. After discussing further related work below, we give formal definitions and present preliminary results in Section 2. Our hardness results are presented in Section 3, and our algorithmic results in Section 4. Section 5 gives conclusions and open problems.

Proofs of statements marked with (\star) are deferred to the full version.

Related work. For a general introduction to temporal graphs, we refer to the article by Michail and Spirakis [14]. The only previous work dealing with the problems we consider (but only for the case of simple temporal graphs) is the recent work by Klobas et al. [12], which has already been discussed above. Other settings where the task is to assign time labels to the edges of a graph in order to create a temporal graph have also been studied, but mainly with the goal of ensuring certain temporal connectivity properties rather than realizing pre-specified journey durations. For example, Akrida et al. [1] studied the problem of assigning (multiple) labels to the edges of a given graph in such a way that the resulting temporal graph is temporally connected (i.e., there exist u - v journeys for all pairs (u, v) of vertices), with the objective of minimizing the total number of labels used. They showed that $O(n)$ labels suffice. Klobas et al. [11] showed that the problem can be solved optimally in polynomial time but becomes *NP*-hard if restrictions are placed on the lifetime of the temporal graph or if connectivity needs to be established only for a subset of the vertices. Mertziou et al. [13] studied variations of the problem where the goal is to minimize the maximum number of labels assigned to any single edge, termed the *temporality* of the temporal graph. Note that the parameter ℓ that we consider in this paper corresponds to the temporality. Enright et al. [8] considered the problem of reordering the snapshots of a given temporal graph in order to minimize reachability.

2 Preliminaries

For details about parameterized complexity we refer to the standard monographs [5, 7].

For any integers i, j with $i \leq j$ we write $[i, j]$ for the set $\{i, i + 1, i + 2, \dots, j\}$. We use standard notation for (static) graphs (see, e.g., [6]). For a graph $G = (V, E)$ and a vertex v of G , we denote by $N_G(v)$ the *neighbors* of v in G and define $N_G[v] := N_G(v) \cup \{v\}$. If the graph is clear from the context, we may omit the subscript. We write uv to denote an edge $\{u, v\}$ in an undirected graph.

A *temporal graph* is a graph that evolves over discrete time steps and whose vertex set remains the same while the edge set may be different in each time step. Two standard ways to represent a temporal graph \mathcal{G} with lifetime L are as follows: The first representation uses a pair $(G = (V, E), \lambda)$, where $G = (V, E)$ is an undirected graph and $\lambda : E \rightarrow (2^{[1, L]} \setminus \{\emptyset\})$ is a function that assigns to each $e \in E$ the non-empty set of time steps during which e is present. The graph G is called the *underlying graph* of \mathcal{G} . We also call λ a *multi-labeling* to emphasize that an edge can receive more than one label. The second representation uses a sequence (G_1, G_2, \dots, G_L) of *snapshots* or *layers*, where $G_i = (V, E_i)$, for $1 \leq i \leq L$, is the graph on vertex set V that contains all edges that are present in time step i . The underlying

graph is then the graph $G = (V, E)$ with $E = \bigcup_{i \in [1, L]} E_i$. The two representations are mathematically equivalent, and each can be transformed into the other in a straightforward way. For implementation purposes, we assume in this paper that temporal graphs are represented in the form (G, λ) , but for ease of exposition we will also often use terminology that refers to the representation using explicit snapshots. We use the convention that $n = |V|$ and $m = |E|$ throughout.

If a temporal graph \mathcal{G} with lifetime L is considered as a non-periodic graph, one assumes that the graph ceases to exist once time step L has passed. If \mathcal{G} is considered as a periodic graph (with period L), then it is assumed that the snapshots repeat after L time steps, i.e., $G_{i+zL} = G_i$ for all $i \in [1, L]$ and all positive integers z . A temporal graph with lifetime or period L is called *simple* if every edge of the underlying graph appears in only one snapshot among the first L time steps, i.e., if $|\lambda(e)| = 1$ for all $e \in E$. For simple temporal graphs, we also write $\lambda(e) = t$ instead of $\lambda(e) = \{t\}$ if $t \in [1, L]$ is the time step in which edge e appears. For periodic temporal graphs, we usually denote the period by Δ instead of L .

If an edge e is present in time step t of a temporal graph \mathcal{G} , we say that (e, t) is a *time-edge* of \mathcal{G} . A *u-v journey* (or *u-v temporal path*) in \mathcal{G} is a sequence $((e_1, t_1), (e_2, t_2), \dots, (e_r, t_r))$ of time-edges such that $t_i < t_{i+1}$ for $1 \leq i < r$ and (e_1, e_2, \dots, e_r) is a *u-v path* in the underlying graph of \mathcal{G} . The journey *starts* or *begins* at u in time step t_1 , *reaches* or *arrives* at v in time step t_r , and has *duration* or *travel time* $t_r - t_1 + 1$. For vertices u, v and any time step t , an *earliest-arrival u-v journey at time t* is a *u-v journey* that begins at u at some time $\geq t$ and minimizes the time when it arrives at v . A *fastest u-v journey* is a *u-v journey* of minimum duration, and the duration of that journey is referred to as *the fastest travel time from u to v*.

A *distance matrix* D is an $n \times n$ matrix whose values are non-negative integers or ∞ . If all values are non-negative integers, we say that D is *finite-valued*. The rows (and columns) of D correspond to n vertices, and we use V to denote the set of these n vertices. For two vertices $u, v \in V$, we use D_{uv} to denote the entry in row u and column v of D , and that entry specifies the desired fastest travel time from u to v . We say that a temporal graph \mathcal{G} with vertex set V *realizes* D if, for any two vertices $u, v \in V$, the duration of a fastest *u-v journey* in \mathcal{G} is equal to D_{uv} . (If $D_{uv} = \infty$, this means that \mathcal{G} does not contain any *u-v journey*.) We can assume that $D_{uv} = 0$ if and only if $u = v$, as otherwise there cannot exist a temporal graph that realizes D . Furthermore, we can also assume for any pair (u, v) with $u \neq v$ that $D_{uv} = 1$ if and only if $D_{vu} = 1$, as a journey with duration one uses a single time-edge and thus is also a journey in the opposite direction. We only consider distance matrices that satisfy these assumptions throughout this paper. The graph $G = (V, E)$ that contains precisely those edges uv for which $D_{uv} = D_{vu} = 1$ is called the *underlying graph induced by D* as any temporal graph that realizes D must have underlying graph G .

As mentioned in the introduction, Klobas et al. [12] introduced the problem of constructing, for a given $n \times n$ distance matrix D and period Δ , a periodic simple temporal graph with n vertices and period Δ that realizes D . We generalize this problem by allowing multiple labels per edge, with an input parameter ℓ specifying how many labels an edge can receive at most:

MULTI-LABEL PERIODIC TGR

Input: An integer ℓ , an $n \times n$ distance matrix D , and a period Δ .

Question: Is there a periodic temporal graph \mathcal{G} with period Δ that realizes D and in which no edge receives more than ℓ labels?

For this problem we assume that D is finite-valued, as otherwise the problem could be split into independent subproblems on temporally connected components. Note that, contrary to the case of non-periodic temporal graphs, the temporal reachability relation in periodic temporal graphs is symmetric and transitive.

Furthermore, we also consider the non-periodic variant of the problem:

MULTI-LABEL TGR

Input: An integer ℓ and an $n \times n$ distance matrix D .

Question: Is there a non-periodic temporal graph \mathcal{G} (with arbitrary lifetime) that realizes D and in which no edge receives more than ℓ labels?

For the non-periodic variant, we may allow the distance matrix to contain entries equal to ∞ . We use $d_{\max} = \max\{D_{uv} \mid u, v \in V, D_{uv} \neq \infty\}$ to refer to the largest finite entry of D .

Basic observations. In the following, we present some basic observations about the problems under consideration. First, we observe that each yes-instance of MULTI-LABEL PERIODIC TGR can be realized with at most n^2 labels per edge.

► **Lemma 1** (\star). *Let I be an instance of MULTI-LABEL PERIODIC TGR or MULTI-LABEL TGR with $\ell \geq n^2$. Then reducing ℓ to n^2 yields an equivalent instance.*

The argument to show Lemma 1 is that a solution only needs to realize one fastest u - v journey for each of the $n(n-1) < n^2$ vertex pairs (u, v) , and for each such u - v journey it suffices to assign at most one additional label to every edge. Thus, it can never be necessary to assign more than n^2 labels to an edge.

Hence, in the following we assume that for each instance of MULTI-LABEL PERIODIC TGR under consideration, $\ell \leq n^2$. Moreover, we can further assume that $\ell \leq \Delta$, since no edge can receive more than Δ labels.

Next, we observe that a yes-instance can be realized by using time labels of value at most $\ell \cdot d_{\max} \cdot n^2$.

► **Lemma 2** (\star). *Let $I := (\ell, D)$ ($I := (\ell, D, \Delta)$) be a yes-instance of MULTI-LABEL TGR (MULTI-LABEL PERIODIC TGR). There is a solution for I with largest time label at most $\ell \cdot d_{\max} \cdot m \leq \ell \cdot d_{\max} \cdot n^2$.*

The proof of Lemma 2 considers gaps (sequences of edgeless snapshots) between non-empty snapshots. For MULTI-LABEL TGR it is clear that gaps of length greater than $d_{\max} - 1$ are never necessary and can be reduced by removing empty snapshots in the gap. As there are at most $m\ell$ snapshots with at least one edge, the result follows. For MULTI-LABEL PERIODIC TGR, if there is a gap that is longer than $d_{\max} - 1$, we can perform a cyclic shift of the time labels so that the longest gap appears in the final steps of the period. All gaps before that final gap can then be reduced to size at most $d_{\max} - 1$ in the same way as in the non-periodic case, showing the lemma.

Note that for MULTI-LABEL PERIODIC TGR, we can thus reduce Δ to at most $\ell \cdot n^2 \cdot d_{\max} \geq \ell \cdot m \cdot d_{\max} + d_{\max}$ if the period Δ is larger than $\ell \cdot m \cdot d_{\max} + d_{\max}$.

► **Corollary 3.** *For an instance (ℓ, D, Δ) of MULTI-LABEL PERIODIC TGR with $\Delta > \ell \cdot n^2 \cdot d_{\max}$, the instance $(\ell, D, \ell \cdot n^2 \cdot d_{\max})$ is an equivalent instance of MULTI-LABEL PERIODIC TGR.*

Note that this also implies the existence of polynomial kernels for MULTI-LABEL PERIODIC TGR of size $\mathcal{O}(\ell \cdot n^2 \cdot d_{\max}) \subseteq \mathcal{O}(n^4 \cdot d_{\max})$, since ℓ can be reduced to n^2 and Δ can be reduced to $\ell \cdot n^2 \cdot d_{\max}$.

3 NP-Hardness for Multi-Label Periodic TGR on restricted instances

In this section, we present three hardness results for MULTI-LABEL PERIODIC TGR on very restricted instances. Recall that MULTI-LABEL PERIODIC TGR for $\ell = 1$ is known to be *NP*-hard for each $\Delta \geq 3$ and that for $\ell = 1$, MULTI-LABEL PERIODIC TGR and MULTI-LABEL TGR are known to be *NP*-hard and $W[1]$ -hard when parameterized by the feedback vertex set number [12]. All three of our hardness results are obtained by reductions from a restricted version of VERTEX COVER. First, we show that for each $\ell \geq 1$, MULTI-LABEL PERIODIC TGR is *NP*-hard even if $d_{\max} = 3$. Afterwards, we show that for each $\ell \geq 5$, MULTI-LABEL PERIODIC TGR is *NP*-hard even on stars, which stands in stark contrast to the fact that for $\ell = 1$ the problem can be solved in polynomial time on trees [12]. Finally, we show hardness for $\ell \in \{3, 4\}$ on graphs that are very close to trees, that is, on graphs with a feedback vertex set of size 1.

We start by showing that for each $\ell \geq 1$, MULTI-LABEL PERIODIC TGR is *NP*-hard even if $d_{\max} = 3$ and the underlying graph is a dense split graph, i.e., a graph where the non-universal vertices form an independent set.

► **Theorem 4.** *For each $\ell \geq 1$, MULTI-LABEL PERIODIC TGR is *NP*-hard even if the underlying graph is a dense split graph, $\Delta = 6$, and $d_{\max} = 3$.*

Proof. We reduce from VERTEX COVER which is known to be *NP*-hard even if the input graph has maximum degree 3, contains no cycle of length three or four, and no two vertices of degree 3 are adjacent [15].

VERTEX COVER

Input: A graph $G = (V, E)$ and an integer k .

Question: Is there a *vertex cover* of size at most k for G , that is, a set of vertices S of size at most k , such that each edge of E is incident with at least one vertex of S ?

Let $\ell \geq 1$. (Our reduction does not actually depend on ℓ ; it thus shows *NP*-hardness for all values of ℓ simultaneously.)

Let $I := (G = (V, E), k)$ be an instance of VERTEX COVER with the above restrictions and let $n := |V|$ with $n \geq 13$. Clearly, we can assume $k < n$ as I is trivially a yes-instance otherwise. Without loss of generality, we assume that G contains four isolated edges x_1x_2 , y_1y_2 , z_1z_2 , and w_1w_2 . (We can ensure this property for any graph by adding four isolated edges and increasing k by four.) These four edges will be helpful to prove that D can be realized with only one label per edge, if I is a yes-instance of VERTEX COVER.

We construct an instance $I' := (\ell, D, \Delta)$ of MULTI-LABEL PERIODIC TGR as follows: We set $\Delta = 6$. The underlying graph $G' := (V', E')$ of I' is set to be a dense split graph with $V' = V \cup S$, where V is an independent set and $S = \{s_1, s_2, \dots, s_k\}$ is the vertex set of a clique of size k . The edge set of G' is therefore $E' = E'_1 \cup E'_2$, where $E'_1 = \{vs \mid v \in V, s \in S\}$ and $E'_2 = \{s_i s_j \mid 1 \leq i < j \leq k\}$. Next, we describe the distance matrix D . As always, we have $D_{uu} = 0$ for all $u \in V'$ and $D_{uv} = D_{vu} = 1$ for all $uv \in E'$. Orient G by picking for each edge $uv \in E$ a direction (u, v) arbitrarily, and denote the resulting set of arcs by A . We assume that the arcs corresponding to the four isolated edges mentioned above are (x_2, x_1) , (y_2, y_1) , (z_2, z_1) , and (w_2, w_1) . For every $(u, v) \in A$, set $D_{uv} = 2$ and $D_{vu} = 3$. For every pair (u, v) of vertices that do not form an edge in E , set $D_{uv} = D_{vu} = 3$. This completes the construction of I' . We show that I admits a vertex cover of size at most k if and only if I' is a yes-instance of MULTI-LABEL PERIODIC TGR.

(\Rightarrow) Let $C = \{c_1, c_2, \dots, c_k\}$ be a vertex cover of size k for I . (If I has a vertex cover smaller than k , we can add arbitrary vertices to it until $|C| = k$.) In the following, we describe a labeling of the edges of G' that realizes D . This labeling is visualized in Figure 1.

Without loss of generality, assume $c_{k-3} = x_1$, $c_{k-2} = y_1$, $c_{k-1} = z_1$, and $c_k = w_1$. For $i = 1, 2, \dots, k-2$, assign one time label to each edge in E'_1 that is incident with s_i in G' as follows:

- For each incoming arc (u, c_i) of c_i in A , set $\lambda(us_i) := 1$.
- Set $\lambda(c_i s_i) := 2$.
- For each outgoing arc (c_i, v) of c_i in A , set $\lambda(vs_i) := 3$.
- For each other vertex $r \in V \setminus N_G[c_i]$, set $\lambda(rs_i) := 5$.

Note that these labels generate journeys of duration 2 exactly for the incoming and outgoing arcs of c_i in A .

For $i = k-1$ and $i = k$, we do essentially the same, but all assigned time labels are shifted by two time steps. Recall that $c_{k-1} = z_1$ ($c_k = w_1$) and that z_1 (w_1) has no outgoing neighbor and only one incoming neighbor, namely z_2 (w_2). The time labels are assigned as follows:

- Set $\lambda(z_2 s_{k-1}) := \lambda(w_2 s_k) := 5$ and $\lambda(s_{k-1} z_1) := \lambda(s_k w_1) := 6$.
- For each vertex $r \in V \setminus \{z_1, z_2\}$, set $\lambda(rs_{k-1}) = 3$, and
- For each vertex $r \in V \setminus \{w_1, w_2\}$, set $\lambda(rs_k) = 3$.

Note that these labels again generate journeys of duration 2 exactly for the arcs incident with z_1 and w_1 , that is, arcs (z_2, z_1) and (w_2, w_1) . As C is a vertex cover of G , every edge in E is an incoming or outgoing arc in A of at least one vertex in C . Hence, it is clear that a u - v journey of duration 2 is created for all pairs (u, v) with $D_{uv} = 2$.

Finally, set $\lambda(e) := 4$ for all $e \in E'_2$.

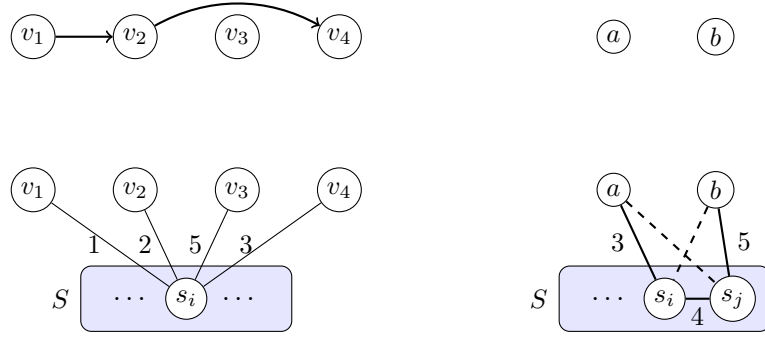
We claim that λ is a solution to I' . First, note that each edge $e \in E'$ receives only a single label. We have already shown above that the journeys of duration 2 that are created are exactly those for all vertex pairs (u, v) with $D_{uv} = 2$. Furthermore, we can show that a journey of duration 3 is generated for each vertex pair $(u, v) \in V \times V$ as follows: Choose $i = k-1$ if $u \notin \{z_1, z_2\}$ and $i = k$ otherwise. Since $\{z_1, z_2\}$ and $\{w_1, w_2\}$ are disjoint, this implies that at time step 3 the edge us_i exists. Similarly, choose $j = k-3$ if $v \notin \{x_1, x_2\}$ and $j = k-2$ otherwise. Since $\{x_1, x_2\}$ and $\{y_1, y_2\}$ are disjoint, this implies that at time step 5 the edge $s_j v$ exists. The u - v journey of duration 3 is then as follows: Take edge us_i at time 3, edge $s_i s_j$ at time 4, and edge $s_j v$ at time 5. Thus, λ solves I' , and hence I' is a yes-instance of MULTI-LABEL PERIODIC TGR.

(\Leftarrow) Assume that λ is a solution that realizes D and maps each $e \in E'$ to a subset of $[1, 6]$. For each $s_i \in S$, let N_i denote the set of edges in E'_1 that are incident with s_i . Note that for every distance $D_{uv} = 2$, there is an $i \in [1, k]$, such that a u - v journey with duration 2 is realized by edges of N_i , since such a journey must pass from u to s_i and then from that s_i to v .

▷ **Claim 5.** Let P_i be the set of vertex pairs $(u, v) \in V \times V$ for which λ realizes a u - v journey of duration 2 using edges of N_i . Then there exists a vertex u_i that all vertex pairs in P_i have in common.

Proof. We show that it is impossible that λ realizes journeys of duration two for two disjoint vertex pairs (u, v) and (a, b) in $V \times V$ using the edges of N_i . This implies that any two vertex pairs in P_i share a common vertex. As E does not contain cycles of length 3, this implies further that there exists one vertex that is common to all vertex pairs in P_i .

Assume for a contradiction that λ realizes journeys of duration 2 for two disjoint vertex pairs (u, v) and (a, b) in $V \times V$ using edges of N_i . Assume without loss of generality that us_i is present at time 1 and vs_i at time 2. Let j be such that as_i is present at time j and bs_i at time $j+1$, where we use the convention that $6+1=1$. Also, let z be a vertex in V



■ **Figure 1** An example how the distances between different vertices are realized in the constructed instance of MULTI-LABEL PERIODIC TGR in the proof of Theorem 4. Top left shows an input instance of VERTEX COVER excluding the additional isolated arcs (w_2, w_1) , (x_2, x_1) , (y_2, y_1) , and (z_2, z_1) . Bottom left shows how the distances representing the arcs incident with vertex v_2 can be realized if vertex v_2 is selected to be the i th vertex of the vertex cover. Top right represents any two distinct vertices a and b of G and bottom right shows how an a - b journey of duration 3 can be realized by using two of the four vertices of $\{s_{k-3}, s_{k-2}, s_{k-1}, s_k\} \subseteq S$. Here, $i = k - 1$ if $a \notin \{z_1, z_2\}$ and $i = k$ otherwise, and $j = k - 3$ if $b \notin \{x_1, x_2\}$ and $j = k - 2$ otherwise. The labels of the dashed edges are not depicted, since they depend on a , b , i , and j .

that is adjacent to no vertex in $\{u, v, a, b\}$ in G (such a vertex must exist as we assume that $n \geq 13$, G has maximum degree 3, and G contains the edges uv and ab). Let k be a time step in which the edge $s_i z$ is active. Note that $k \notin \{6, 1, 2, 3\}$ as otherwise z has a journey of duration 2 from or to u or v , a contradiction to λ being a solution while z is adjacent to neither u nor v . Hence, $k \in \{4, 5\}$. By symmetry, we can assume $k = 4$.

Now we show that we obtain a contradiction no matter what the value of j is:

- $j = 1$: We have journeys of duration 2 from u to v , u to b , a to v and a to b , implying that G must contain the 4-cycle $uvab$, a contradiction.
 - $j = 2$: We have journeys of duration 2 from u to v , u to a , a to b , and v to b , implying a 4-cycle $uvba$, a contradiction.
 - $j = 6$: We have journeys of duration 2 from a to b , a to u , b to v , and u to v , implying a 4-cycle $abvu$, a contradiction.
 - $j = 3$: We have an a - z journey of duration 2, a contradiction to z not being adjacent to a .
 - $j = 4$: We have a z - b journey of duration 2, a contradiction to z not being adjacent to b .
 - $j = 5$: We have a z - a journey of duration 2, a contradiction to z not being adjacent to a .
- As all cases lead to a contradiction, the assumption that λ realizes journeys of duration 2 for two disjoint vertex pairs (u, v) and (a, b) cannot hold. \triangleleft

By Claim 5, all vertex pairs for which λ realizes a journey of duration 2 using the edges of N_i have a common vertex u_i . As a journey of duration 2 must be realized for every edge uv of G (either from u to v or from v to u , depending on how the edge has been oriented), the set $U = \{u_1, u_2, \dots, u_k\}$ is a vertex cover of G . Furthermore, $|U| \leq k$, and hence I is a yes-instance of VERTEX COVER. \blacktriangleleft

Note that in contrast to hardness for $d_{\max} = 3$ that we have just shown, for each $\ell \geq 2$, it is not difficult to see that MULTI-LABEL PERIODIC TGR can be solved in polynomial time if $d_{\max} \leq 2$: If $\Delta = 1$, the problem is polynomial time solvable, since there is only a single temporal multi-labeling. If $d_{\max} = 1$, the underlying graph is a clique and the distance matrix is realizable by any periodic multi-labeling. If $d_{\max} = 2$, then the instance is a trivial

no-instance, if the underlying graph has diameter larger than 2. Otherwise, if the underlying graph has diameter at most two, we can label each edge with label 1 and 2 (since $\ell \geq 2$) and so ensure a path between any two vertices of duration 2 that starts at time step 1.

► **Observation 6.** *For each $\ell \geq 2$ MULTI-LABEL PERIODIC TGR can be solved in polynomial time if $d_{\max} \leq 2$.*

We now shift to considering the structure of the realized graph. Next, we show that for $\ell \geq 5$, MULTI-LABEL PERIODIC TGR is NP-hard even on stars. This implies that MULTI-LABEL PERIODIC TGR is NP-hard even if $\ell + \text{vc} \in \mathcal{O}(1)$, and so FPT algorithms for parameter $\ell + \text{vc}$ are impossible, unless $P = NP$.

► **Theorem 7 (\star).** *For each $\ell \geq 5$, MULTI-LABEL PERIODIC TGR is NP-hard even if the underlying graph is a star.*

Proof sketch. Let $\ell \geq 5$. We again reduce from VERTEX COVER where the input graph contains no cycle of length three or four, the input graph has a maximum degree of 3, and no two vertices of degree 3 are adjacent.

Let $I := (G = (V, E), k)$ be an instance of VERTEX COVER with the above restrictions and let $n := |V|$ be larger than 10. We construct an instance $I' := (\ell, D, \Delta)$ of MULTI-LABEL PERIODIC TGR as follows: The underlying graph $G' := (V', E')$ of I' is a star with center c and leaf set $V \cup \{v^*, w^*\}$. We set $D_{w^*v^*} := D_{v^*w^*} := n^2$ and for each vertex $v \in V$, we set $D_{vv^*} := D_{v^*v} := D_{vw^*} := D_{w^*v} := n^2$. For each two distinct vertices u and v of V , we set $D_{uv} := D_{vu} := 2$ if uv is an edge of E , and $D_{uv} := D_{vu} := n^2$, otherwise. Finally, we set $\Delta := (k + 2) \cdot (n^2 + 1) = k \cdot n^2 + 2n^2 + k + 2$. This completes the construction of I' .

Note that each temporal path between any two vertices u and v of G' distinct from the center vertex c is of the form ucv . Since each vertex of $V' \setminus \{c\}$ has only one incident edge in G' , we may say in the following that for a temporal multi-labeling, a vertex $v \in V' \setminus \{c\}$ is *active* in time step i , if the edge cv exists in time step i .

Observe that for each vertex $u \in V$, journeys of travel time 2 from u to all its neighbors in G and vice versa can be realized in three consecutive time steps: u is active in the first and the third of these time steps and all vertices of $N_G(u)$ are active in the second time step. Hence, the journeys of duration 2 from u to all its neighbors in G start in the first time step and end in the second time step, and the journeys of duration 2 from all neighbors of u in G to u start in the second time step and end in the third time step. If G admits a vertex cover of size k , all required journeys of travel time 2 can thus be realized in k such groups of three consecutive time steps, with a separation of $n^2 - 2$ edgeless time steps between them (to avoid creating journeys of travel time shorter than n^2 between pairs of vertices that are independent in G). In addition to these $k(n^2 + 1)$ time steps, a further $2(n^2 + 1)$ time steps can be used to realize all the required journeys of travel time n^2 . For the other direction, we can show that any feasible realization must have a similar structure, implying the existence of a vertex cover of size at most k . The detailed proof of correctness is deferred to the full version. ◀

Since the hardness result above only holds for $\ell \geq 5$, we note that we can also show that for $\ell \geq 3$, the problem is still NP-hard even on graphs with a size-1 feedback vertex set.

► **Theorem 8 (\star).** *For each $\ell \geq 3$, MULTI-LABEL PERIODIC TGR is NP-hard even if the underlying graph has diameter two and a feedback vertex set of size one.*

4 Parameterized algorithms

In this section, we present FPT algorithms for MULTI-LABEL PERIODIC TGR and MULTI-LABEL TGR for several parameter combinations. First, we give FPT algorithms for both problems parameterized by n in Section 4.1. Section 4.2 presents our FPT algorithm for MULTI-LABEL PERIODIC TGR parameterized by $vc + \Delta$. Section 4.3 discusses our $O(pn^4)$ -time algorithm for instances of MULTI-LABEL TGR with finite-valued D , $\ell \geq n^2$, and underlying graphs that have at most p different u - v paths for each vertex pair (u, v) . Finally, the polynomial kernel for parameter $d_{\max} + \text{nu}$ is shown in Section 4.4, where nu denotes the number of non-universal vertices in the underlying graph.

4.1 Parameterization by the number of vertices

In this section, we present an FPT algorithm for parameter n for MULTI-LABEL PERIODIC TGR. The key idea is to enumerate all possibilities for the sequence of snapshots that contain at least one edge (and some extra information that specifies for each pair (u, v) a snapshot in which a u - v journey of shortest duration begins). For each possibility, we use an integer linear program (ILP) to decide whether it is possible to assign these snapshots to time steps in such a way that the resulting periodic temporal graph realizes D . The approach extends to MULTI-LABEL TGR as well.

► **Theorem 9.** *MULTI-LABEL PERIODIC TGR can be solved in $n^{\mathcal{O}(\ell \cdot n^2)} \cdot |I|^{\mathcal{O}(1)}$ time and $n^{\mathcal{O}(n^4)} \cdot |I|^{\mathcal{O}(1)}$ time, where $|I|$ denotes the encoding length of the instance.*

Proof. Let an instance $I = (\ell, D, \Delta)$ of MULTI-LABEL PERIODIC TGR be given. Recall that we can assume that $\ell \leq n^2$ due to Lemma 1. Let $K = \ell m$ and $T = \min\{K, \Delta\}$. Note that $T \leq \ell m \in \mathcal{O}(n^2 \cdot n^2) \subseteq n^{\mathcal{O}(1)}$. We observe that there are at most K non-empty snapshots in any realization, as each of the m edges can occur in at most ℓ snapshots. Furthermore, it is clear that there are at most Δ non-empty snapshots, so the number of non-empty snapshots is at most T . The number of sequences of at most T non-empty snapshots in which each of the m edges occurs in at most ℓ snapshots (and in at least one snapshot) can be bounded by $T^{\ell m} = T^K$, as each such sequence can be encoded by assigning to each of ℓm edge copies (with ℓ copies of each edge) a number in $[1, T]$ that identifies the snapshot in which it occurs. (If an edge occurs fewer than ℓ times, this can be captured by assigning some of its copies the same number.) Thus, we can enumerate all such sequences in $T^{\mathcal{O}(K)} \subseteq n^{\mathcal{O}(\ell \cdot n^2)}$ time.

For each such sequence \mathcal{S} of non-empty snapshots, we enumerate all possibilities of assigning to each vertex pair (u, v) with $D_{uv} > 1$ a number s_{uv} in $[1, |\mathcal{S}|]$ that identifies the snapshot in which the journey from u to v that realizes the duration D_{uv} starts. (Note that if $s_{uv} = i$, this means that the journey starts in the i -th non-empty snapshot. That snapshot will be present in some time step t_i in $[1, \Delta]$ that has not yet been determined.) The number of possibilities to be enumerated is bounded by $T^{n^2} \subseteq n^{\mathcal{O}(n^2)}$.

Intuitively, we want to proceed along the following lines: For each combination of a sequence \mathcal{S} of snapshots and an assignment of values s_{uv} to vertex pairs (u, v) with $D_{uv} > 1$, we want to use an ILP to check whether we can assign the i -th snapshot of \mathcal{S} to a time step t_i , for all i , in such a way that the resulting periodic temporal graph realizes D . To be able to formulate the constraints of the ILP, we use an auxiliary temporal graph, without gaps between the snapshots of \mathcal{S} , to determine for each pair (u, v) of vertices and each starting snapshot i the snapshot at which v can first be reached if starting at u in snapshot i . The constraints of the ILP can then express that the gaps inserted between the snapshots must be such that (1) the duration of the u - v journey starting in snapshot s_{uv} is equal to D_{uv} , and (2) the duration of the u - v journey starting in any other snapshot is at least D_{uv} . The variables of the ILP represent the time steps to which the snapshots get assigned.

Formally, we process each combination of a sequence \mathcal{S} of snapshots and assignment of values s_{uv} to vertex pairs (u, v) with $D_{uv} > 1$ as follows. Let $L = |\mathcal{S}|$ and $\mathcal{S} = (S_1, S_2, \dots, S_L)$. Build a periodic temporal graph $\mathcal{G}_{\mathcal{S}}$ with period L such that the edges present in time steps $i + zL$ for all integers $z \geq 0$ are those of S_i , for $1 \leq i \leq L$. For every $(u, v) \in V \times V$ with $D_{uv} > 1$ and every $i \in [1, L]$, we denote by $q(u, v, i)$ the tuple (j, z) , such that each fastest u - v journey in $\mathcal{G}_{\mathcal{S}}$ starting at time i ends in time step $j + zL$, where $1 \leq j \leq L$. Note that computing $q(u, v, i)$ can be done in polynomial time, since finding a path of shortest duration from u to v starting at time step i can be done in polynomial time on non-periodic temporal graphs [3, 17], and we can simply unroll $\mathcal{G}_{\mathcal{S}}$ n times to obtain a non-periodic temporal graph on which the shortest duration of any journey between u and v remains the same. This is discussed in detail for a more general class of periodic temporal graphs in [2, Remark 1]. Note that in a periodic temporal graph, for any pair (u, v) of vertices for which a u - v journey exists, there exists a u - v journey of shortest duration that starts in a snapshot of the first period. Therefore, it suffices to consider only start times $i \in [1, L]$ for u - v journeys in $\mathcal{G}_{\mathcal{S}}$.

We want to determine whether there exist values t_i for $1 \leq i \leq L$ with $1 \leq t_1 < t_2 < \dots < t_L \leq \Delta$ such that the periodic temporal graph \mathcal{G} defined as follows realizes D :

- for each $i \in [1, L]$, S_i is the set of edges of \mathcal{G} that are present in time step t_i , and
- no edge is present in any of the time steps in $[1, \Delta] \setminus \{t_i \mid 1 \leq i \leq L\}$.

Observe that $q(u, v, i) = (j, z)$ if and only if the earliest-arrival journey from u to v in \mathcal{G} starting at time t_i reaches v in time step $t_j + z\Delta$ (and thus has duration $\delta(u, v, t_i) = t_j + z\Delta - t_i + 1$). This is because $\mathcal{G}_{\mathcal{S}}$ can be obtained from \mathcal{G} by removing all empty snapshots.

The temporal graph \mathcal{G} realizes D if, for all (u, v) with $D(u, v) > 1$, $\delta(u, v, t_i) = D_{uv}$ for at least one t_i and $\delta(u, v, t_i) \geq D_{uv}$ for all t_i . The purpose of the value s_{uv} that we have enumerated is to give a value of i with the property that $\delta(u, v, t_i) = D_{uv}$. We can then formulate the following ILP with variables t_1, t_2, \dots, t_L to check whether there exist values of these variables such that \mathcal{G} realizes D :

$$\begin{aligned}
t_j + z\Delta - t_i + 1 &= D_{uv} && \forall (u, v) \text{ with } D_{uv} > 1, i = s_{uv}, q(u, v, i) = (j, z) \\
t_j + z\Delta - t_i + 1 &\geq D_{uv} && \forall (u, v) \text{ with } D_{uv} > 1, \forall i \neq s_{uv}, q(u, v, i) = (j, z) \\
t_1 &\geq 1 && \\
t_i - t_{i-1} &\geq 1 && \forall i \text{ with } 2 \leq i \leq L \\
t_L &\leq \Delta &&
\end{aligned} \tag{ILP}$$

Note that there is no objective function as we only want to check feasibility, i.e., check whether there exist values t_1, \dots, t_L that satisfy the constraints. The first two constraints express that the earliest-arrival path from u to v starting in time step $t_{s_{uv}}$ has duration D_{uv} and the earliest-arrival paths from u to v starting in any other time step have duration at least D_{uv} . The last three constraints ensure that $1 \leq t_1 < t_2 < \dots < t_L \leq \Delta$. Thus, a feasible solution of the ILP gives a periodic temporal graph that is a solution to the given instance I of MULTI-LABEL PERIODIC TGR.

As the ILP has $L \leq T$ variables, we can solve each such ILP instance in $(\log L)^{\mathcal{O}(L)} \cdot |I|^{\mathcal{O}(1)} \subseteq (\log n)^{\mathcal{O}(\ell \cdot n^2)} \cdot |I|^{\mathcal{O}(1)}$ time [16]. We solve the ILP once for each combination of a sequence \mathcal{S} of non-empty snapshots and an assignment of values s_{uv} to all pairs (u, v) with $D_{uv} > 1$. Thus, we solve $n^{\mathcal{O}(\ell \cdot n^2)}$ different ILPs. The resulting overall running-time is then $n^{\mathcal{O}(\ell \cdot n^2)} \cdot |I|^{\mathcal{O}(1)}$. The claimed running-time follows because we can assume $\ell \leq n^2$. The algorithm is correct because, if I is a yes-instance, one of the enumerated combinations of a sequence of snapshots and an assignment of values s_{uv} corresponds to a realization of D , and for that combination a realization of D will be obtained from the solution of the ILP. ◀

12:12 Parameterized Algorithms for Multi-Label Periodic Temporal Graph Realization

For the special case $\ell = 1$ of MULTI-LABEL PERIODIC TGR, our FPT algorithm of Theorem 9 has a running time of $n^{\mathcal{O}(n^2)} \cdot |I|^{\mathcal{O}(1)}$ and is a conceptually simpler FPT algorithm for PERIODIC TGR than the FPT algorithm for n that is implied by the algorithm for PERIODIC TGR parameterized by the feedback edge number of the underlying graph from [12]. The main advantage of our approach is that it extends to arbitrary ℓ . Furthermore, our approach also works for the non-periodic version.

► **Corollary 10** (\star). *MULTI-LABEL TGR can be solved in $n^{\mathcal{O}(\ell \cdot n^2)} \cdot |I|^{\mathcal{O}(1)}$ time and $n^{\mathcal{O}(n^4)} \cdot |I|^{\mathcal{O}(1)}$ time, where $|I|$ denotes the encoding length of the instance.*

4.2 Parameterization by the vertex cover number plus the period

In this section we give an FPT algorithm for MULTI-LABEL PERIODIC TGR parameterized by $\text{vc} + \Delta$. The key idea of our approach is to show that any given instance can be reduced to one where the number of vertices in the independent set that have the same neighbors can be bounded by a function of the parameter. It then suffices to apply the FPT algorithm for parameter n to this reduced instance.

► **Theorem 11.** *There is an FPT algorithm for MULTI-LABEL PERIODIC TGR parameterized by $\text{vc} + \Delta$.*

Recall that ℓ is upper bounded by Δ . Hence, to show Theorem 11, it is sufficient to present an FPT algorithm for parameter $\text{vc} + \Delta + \ell$. Let (ℓ, D, Δ) be the given instance of MULTI-LABEL PERIODIC TGR. Recall that vc denotes the size of a minimum vertex cover of the underlying graph $G = (V, E)$. Observe that the number of possible label sets assigned to any particular edge $e \in E$ can be bounded by Δ^ℓ : Each of the up to ℓ labels assigned to the edge is a value in $[1, \Delta]$, and combinations where fewer than ℓ labels are assigned to the edge can be modeled as assigning the same label several times.

We call two vertices $u, v \in V$ *distance twins* if they have the same distance to every vertex in $V \setminus \{u, v\}$ and to each other. This means that their rows in the distance matrix D are identical, and their columns in D are identical, up to the obvious difference in the intersection of their rows with their columns: $D_{uu} = 0$, $D_{uv} = D_{vu}$, and $D_{vv} = 0$. Note that the distance twin relation is an equivalence relation on V .

Let C be a vertex cover of G of size vc , and let $I = V \setminus C$ be the independent set that is the complement of C . Partition I into *neighborhood classes* $\mathcal{I} = \{I_1, I_2, \dots, I_t\}$ based on adjacency to C , i.e., two vertices $u, v \in I$ are in the same class I_j if and only if $N(u) = N(v)$. Note that, $t \leq 2^{\text{vc}}$.

Consider one part I_j of the partition \mathcal{I} . For each vertex $u \in I_j$, there are at most $\Delta^{\text{vc} \cdot \ell}$ different ways of assigning label sets to the (at most vc) edges incident with u . For any fixed labeling λ of E , call two vertices $u, v \in I_j$ *label twins* if for every vertex $w \in N(u)$, $\lambda(uw) = \lambda(vw)$. The label twin relation partitions the set I_j into equivalence classes that we call *label classes*. Note that there can be at most $\Delta^{\text{vc} \cdot \ell}$ label classes for I_j .

Observe that all vertices u, v in a label class have the same distance to every vertex in $V \setminus \{u, v\}$ and to each other. This means that, if λ realizes D , then u and v must be distance twins. The distance twin relation partitions I_j into *distance classes*. If I_j contains more than $\Delta^{\text{vc} \cdot \ell}$ distance classes, the given instance is a no-instance, because at most $\Delta^{\text{vc} \cdot \ell}$ different distance classes can be realized by the at most $\Delta^{\text{vc} \cdot \ell}$ different label classes.

► **Observation 12.** *If a neighborhood class contains more than $\Delta^{\text{vc} \cdot \ell}$ distance classes, then the considered instance is a trivial no-instance of MULTI-LABEL PERIODIC TGR.*

If I_j contains at most $\Delta^{\text{vc}\cdot\ell}$ different distance classes, it suffices to keep $2\Delta^{\text{vc}\cdot\ell}$ of the vertices in each such class, while any additional vertices of the class can be deleted (i.e., the corresponding rows and columns of these vertices can be removed from D).

► **Lemma 13** (\star). *Let I_j be a neighborhood class and let F denote a distance class of I_j of size more than $2\Delta^{\text{vc}\cdot\ell}$. Then removing one vertex of F from G yields an equivalent instance.*

With this statement at hand, we are now able to present the algorithm for MULTI-LABEL PERIODIC TGR.

Proof of Theorem 11. For each neighborhood class I_j of \mathcal{I} and each distance class F of I_j with $|F| > 2\Delta^{\text{vc}\cdot\ell}$, remove $|F| - 2\Delta^{\text{vc}\cdot\ell}$ arbitrary vertices from F . Due to Lemma 13, this yields an equivalent instance, where each distance class contains at most $2\Delta^{\text{vc}\cdot\ell}$ vertices. If the resulting instance contains at most $\text{vc} + 2^{\text{vc}} \cdot 2\Delta^{2\text{vc}\cdot\ell}$ vertices, the FPT algorithm is obtained by applying the algorithm behind Theorem 9. Otherwise, if the resulting instance contains more than $\text{vc} + 2^{\text{vc}} \cdot 2\Delta^{2\text{vc}\cdot\ell}$ vertices, there is a neighborhood class I_j such that I_j has more than $\Delta^{\text{vc}\cdot\ell}$ distance classes. This is correct, since the new instance contains exactly vc vertex cover vertices, at most 2^{vc} neighborhood classes, and at most $2\Delta^{\text{vc}\cdot\ell}$ vertices in each distance class. Due to Observation 12, we can thus correctly output that the instance under consideration is a trivial no-instance of MULTI-LABEL PERIODIC TGR. ◀

Recall that MULTI-LABEL PERIODIC TGR is *NP*-hard even if $\ell = 1$ and $\Delta = 3$ [12] and that Theorem 7 shows that MULTI-LABEL PERIODIC TGR is *NP*-hard even if $\ell = 5$ and $\text{vc} = 1$. Hence, neither of the considered parameters can be omitted to still obtain an FPT algorithm for MULTI-LABEL PERIODIC TGR. Still, the question remains whether there is an FPT algorithm parameterized by vc alone for the case $\ell = 1$ (PERIODIC TGR), or if one can replace Δ in the combined parameter by some potentially smaller parameter. In particular, the parameterized complexity of MULTI-LABEL PERIODIC TGR when parameterized by $\text{vc} + d_{\max} + \ell$ is open.

4.3 Efficient algorithm for Multi-Label TGR on graphs with few paths

While we have shown MULTI-LABEL PERIODIC TGR to be *NP*-hard for any $\ell \geq 5$ even if the underlying graph is a star, we show in this section that MULTI-LABEL TGR can be solved in polynomial time if the underlying graph is a tree and $\ell \geq n(n-1)$ and D is finite-valued. In fact, our result is more general and solves the problem in polynomial time whenever the number of different u - v paths in the underlying graph can be bounded by a polynomial, for each vertex pair (u, v) .

As a subproblem we consider the problem PATH REALIZATION that is defined as follows: Given a distance matrix D , a pair (u, v) with $u \neq v$, and a u - v path $P = (u_0 = u, u_1, \dots, u_r = v)$ in the underlying graph, decide if one can assign one time label to each edge on P in such a way that, in the temporal graph on P with those time labels, the u - v journey has duration D_{uv} , while any u_i - u_j journey with $0 \leq i < j \leq r$ has duration at least $D_{u_i u_j}$. Here, we use the convention that the duration of a u_i - u_j journey is ∞ if there is no u_i - u_j journey in the temporal graph on P with the time labels assigned.

Intuitively, the purpose of solving PATH REALIZATION is to decide whether it is possible to assign time labels to the edges of P in such a way that a u - v journey of duration D_{uv} is realized while no u' - v' journey that is too short (i.e., has duration strictly less than $D_{u'v'}$) is created. The key ingredient of the proof is the following lemma that shows that PATH REALIZATION can be solved in polynomial time. For the main result of this section we only

need to be able to solve PATH REALIZATION for finite-valued distance matrices, but since our approach can also handle entries equal to ∞ , we present the lemma for this more general case.

► **Lemma 14** (\star). *PATH REALIZATION can be solved in $O(r^2)$ time.*

We say that the underlying graph $G = (V, E)$ is p -path-diverse if the number of u - v paths in G is bounded by p for each pair (u, v) of vertices in V .

► **Theorem 15** (\star). *MULTI-LABEL TGR can be solved in $O(pn^4)$ time if the underlying graph is p -path-diverse and $\ell \geq n(n-1)$ and D is finite-valued.*

The maximum number of u - v paths for any vertex pair (u, v) can be bounded by $n!$, so the running-time of the algorithm of Theorem 15 is bounded by $O(n! \cdot n^4)$. Thus, the algorithm is an FPT algorithm for MULTI-LABEL TGR parameterized by n that is simpler and more efficient than that of Theorem 9 in Section 4.1, but only works for instances with a finite-valued distance matrix and $\ell \geq n(n-1)$.

As trees are 1-path-diverse and cycles are 2-path-diverse, we obtain the following corollary.

► **Corollary 16**. *MULTI-LABEL TGR can be solved in $O(n^4)$ time if the underlying graph is a tree or a cycle, $\ell \geq n(n-1)$, and D is finite-valued.*

4.4 A polynomial kernel

In this section, we present a kernel for MULTI-LABEL PERIODIC TGR for the combined parameter $d_{\max} + \text{nu}$, where $\text{nu} := |\{v \in V \mid N[v] \neq V\}|$ denotes the number of non-universal vertices of the underlying graph. Note that nu is never larger than the number of entries of D of value larger than 1, since for each non-edge $\{u, v\}$ of G , $D_{uv} > 1$ and $D_{vu} > 1$. Hence, the kernel we present also implies a kernel for MULTI-LABEL PERIODIC TGR for the parameter combination $d_{\max} + |\{D_{uv} \mid u, v \in V, D_{uv} > 1\}|$. We also show that this kernel transfers to MULTI-LABEL TGR for finite-valued distance matrices.

► **Theorem 17** (\star). *MULTI-LABEL PERIODIC TGR admits a kernel of size $\mathcal{O}(\min\{\ell \cdot \text{nu}^4 \cdot d_{\max}, \text{nu}^8 \cdot d_{\max}\})$. More precisely, this kernel has $\mathcal{O}(\text{nu}^2 \cdot d_{\max})$ vertices and a period of $\mathcal{O}(\min\{\ell \cdot \text{nu}^4 \cdot d_{\max}, \text{nu}^8 \cdot d_{\max}\})$ and does not increase the value of ℓ .*

Proof. Let $I := (\ell, D, \Delta)$ be an instance of MULTI-LABEL PERIODIC TGR where d_{\max} denotes the largest non-infinite entry of D and where $G = (V, E)$ is the underlying graph implied by the distance matrix D . Moreover, let X denote the set of all vertices of G that are not universal and let $\text{nu} := |X|$.

If $n \in \mathcal{O}(\text{nu}^2)$, then D contains $\mathcal{O}(\text{nu}^4)$ entries of size at most d_{\max} each. Moreover, due to Lemma 2, we can reduce Δ to $\mathcal{O}(\ell \cdot \text{nu}^4 \cdot d_{\max}) \subseteq \mathcal{O}(\text{nu}^8 \cdot d_{\max})$. This then implies a polynomial kernel of the desired size. Hence, in the following, we assume that $n > \text{nu}^2$. Note that this implies that there is at least one universal vertex v^* in G . Hence, for any two vertices u and w of G distinct from v^* , there is the path uv^*w of length two in G . Consequently if I is a yes-instance of PERIODIC TGR, the largest possible time that can be realized between any two vertices of G is $\Delta + 1$, since traversing the path uv^*w takes time at most $\Delta + 1$. In other words, I is a trivial no-instance of PERIODIC TGR if D contains an entry larger than $\Delta + 1$. In the following, we thus assume that $d_{\max} \leq \Delta + 1$. We distinguish two cases.

Case 1: $\Delta \geq 3d_{\max}$. We show in this case that I is a trivial yes-instance of MULTI-LABEL PERIODIC TGR even when assigning only a single label to each edge. This proof is deferred to the full version.

Case 2: $\Delta < 3d_{\max}$. Let R_X denote the set of directed pairs of vertices of X for which the distance is not trivially realized, that is, $R_X := \{(u, v) \in X \times X \mid u \neq v, D_{uv} > 1\}$. If G contains no more than $\text{nu}^2 \cdot d_{\max}$ universal vertices, then $n \in \mathcal{O}(\text{nu}^2 \cdot d_{\max})$ and the polynomial kernel follows directly, since $\ell \leq \Delta \in \mathcal{O}(d_{\max})$ and D contains $\mathcal{O}(\text{nu}^2)$ entries of value larger than 1. Otherwise, if G contains at least $\text{nu}^2 \cdot d_{\max}$ universal vertices, we remove an arbitrary set Z of universal vertices from G such that $\text{nu}^2 \cdot d_{\max}$ universal vertices remain. This then gives the kernel of desired size due to the above argumentation. In the full version, we show that the so obtained instance $I' := (\ell, D', \Delta)$ of MULTI-LABEL PERIODIC TGR is a yes-instance if and only if I is a yes-instance of MULTI-LABEL PERIODIC TGR. ◀

Note that this implies the following polynomial kernel for PERIODIC TGR.

► **Corollary 18.** *PERIODIC TGR admits a polynomial kernel of size $\mathcal{O}(\text{nu}^4 \cdot d_{\max}^2)$. More precisely, this kernel has $\mathcal{O}(\text{nu}^2 \cdot d_{\max})$ vertices and a period of $\mathcal{O}(\text{nu}^4 \cdot d_{\max})$.*

Moreover, we can derive the following result for MULTI-LABEL TGR on finite-valued distance matrices.

► **Corollary 19** (★). *On finite-valued distance matrices, MULTI-LABEL TGR is FPT when parameterized by nu and admits a kernel of size $\mathcal{O}(\text{nu}^4 + \text{nu}^2 \cdot d_{\max})$. More precisely, this kernel has $\mathcal{O}(\text{nu}^2)$ vertices and does not increase the value of ℓ .*

5 Conclusion and open questions

In this paper, we have studied multi-label versions of the temporal realization problem introduced by Klobas et al. [12] and presented various hardness results and FPT algorithms for different parameters or parameter combinations. There are a number of interesting directions for future work. While our hardness results exclude FPT algorithms for MULTI-LABEL PERIODIC TGR parameterized by the vertex cover number alone (unless $P = NP$), the question whether such an FPT algorithm exists for PERIODIC TGR remains open. With respect to the polynomial kernel of size $\mathcal{O}(\text{nu}^4 \cdot d_{\max}^2)$ that we have obtained, an interesting question is whether a kernel whose size is a polynomial of nu alone exists. To answer this question, one first has to analyze whether the problem admits a polynomial kernel for parameter n alone. A question in relation to our FPT algorithms for parameter n is whether the subproblem that we solve using ILP can be solved more efficiently using a combinatorial algorithm.

Furthermore, it would be interesting to analyze the computational complexity of MULTI-LABEL PERIODIC TGR on stars and trees for $\ell \in \{2, 3, 4\}$. Klobas et al. [12] have shown that the problem on trees is polynomial for $\ell = 1$, while we have shown that it is NP -hard on stars for $\ell \geq 5$, so the status for $\ell \in \{2, 3, 4\}$ is open for stars and trees.

For MULTI-LABEL TGR, NP -hardness has so far only been shown in the case that the distance matrix can have entries equal to ∞ and $\ell = 1$ [12]. It would be interesting to analyze the complexity for finite-valued distance matrices and for $\ell > 1$. For the problem variant where a maximum allowed label L is specified as part of the input (i.e., a bound on the lifetime of the temporal graph that can be built to realize D), our NP -hardness proofs of Section 3 should translate. We expect that our FPT algorithms for parameter n and for parameter $\text{vc} + \Delta$ (which then becomes $\text{vc} + L$) can also be adapted to that case even if the distance matrix contains entries of value ∞ .

Given that we have shown that MULTI-LABEL PERIODIC TGR is NP -hard even if $d_{\max} = 3$ for every $\ell \geq 1$, and that it can be solved in polynomial time if $d_{\max} \leq 2$ for all $\ell \geq 2$, it would be interesting to settle the complexity of the problem for $\ell = 1$ and $d_{\max} = 2$.

References

- 1 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. The complexity of optimal design of temporally connected graphs. *Theory Comput. Syst.*, 61(3):907–944, 2017. doi:10.1007/S00224-017-9757-X.
- 2 Emmanuel Arrighi, Niels Grüttemeier, Nils Morawietz, Frank Sommer, and Petra Wolf. Multi-parameter analysis of finding minors and subgraphs in edge periodic temporal graphs. *CoRR*, abs/2203.07401, 2022. doi:10.48550/ARXIV.2203.07401.
- 3 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.
- 4 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *Int. J. Parallel Emergent Distributed Syst.*, 27(5):387–408, 2012. doi:10.1080/17445760.2012.668546.
- 5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 6 Reinhard Diestel. *Graph Theory*. Springer-Verlag, 2000.
- 7 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 8 Jessica A. Enright, Kitty Meeks, and Fiona Skerman. Assigning times to minimise reachability in temporal graphs. *J. Comput. Syst. Sci.*, 115:169–186, 2021. doi:10.1016/J.JCSS.2020.08.001.
- 9 Paul Erdős and Tibor Gallai. Graphs with points of prescribed degree (in Hungarian). *Mat. Lapok*, 11:264–274, 1961.
- 10 S. L. Hakimi and S. S. Yau. Distance matrix of a graph and its realizability. *Quarterly of Applied Mathematics*, 22(4):305–317, 1965.
- 11 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. The complexity of computing optimum labelings for temporal connectivity. In *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *LIPICs*, pages 62:1–62:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.MFCS.2022.62.
- 12 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Temporal graph realization from fastest paths. In *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, volume 292 of *LIPICs*, pages 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.SAND.2024.16.
- 13 George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019. doi:10.1007/S00453-018-0478-6.
- 14 Othon Michail and Paul G. Spirakis. Elements of the theory of dynamic networks. *Commun. ACM*, 61(2):72, 2018. doi:10.1145/3156693.
- 15 Owen J. Murphy. Computing independent sets in graphs with large girth. *Discret. Appl. Math.*, 35(2):167–170, 1992. doi:10.1016/0166-218X(92)90041-8.
- 16 Victor Reis and Thomas Rothvoss. The subspace flatness conjecture and faster integer programming. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 974–988. IEEE, 2023. doi:10.1109/FOCS57990.2023.00060.
- 17 Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proc. VLDB Endow.*, 7(9):721–732, 2014. doi:10.14778/2732939.2732945.

Computational Power of Opaque Robots

Caterina Feletti ✉ 

Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy

Lucia Mambretti ✉ 

Milan, Italy

Carlo Mereghetti ✉ 

Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy

Beatrice Palano ✉ 

Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy

Abstract

In the field of distributed computing by robot swarms, the research comprehends manifold models where robots operate in the Euclidean plane through a sequence of *look-compute-move* cycles. Models under study differ for (i) the possibility of storing constant-size information, (ii) the possibility of communicating constant-size information, and (iii) the synchronization mode. By varying features (i,ii), we obtain the noted four base models: *OBLLOT* (silent and oblivious robots), *FSTA* (silent and finite-state robots), *FCOM* (oblivious and finite-communication robots), and *LUMI* (finite-state and finite-communication robots). Combining each base model with the three main synchronization modes (*fully synchronous*, *semi-synchronous*, and *asynchronous*), we obtain the well-known 12 models. Extensive research has studied their *computational power*, proving the hierarchical relations between different models. However, only *transparent* robots have been considered.

In this work, we study the taxonomy of the 12 models considering *collision-intolerant opaque* robots. We present six witness problems that prove the majority of the computational relations between the 12 models. In particular, the last witness problem depicts a peculiar issue occurring in the case of *obstructed visibility* and asynchrony.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Mobile robots, Look-Compute-Move, Computational complexity, Opaque robots, Distributed computing, Obstructed visibility, Collision intolerance

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.13

Related Version *Preprint*: <https://arxiv.org/abs/2401.16893v1>

Acknowledgements C. Feletti, C. Mereghetti, and B. Palano are members of the *Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica* (GNCS-INdAM). The authors thank the anonymous referees for their helpful comments which contributed to improving the paper.

1 Introduction

In the far-ranging field of distributed computing, a significant area concerns *computing by mobile entities* [16, 17], where tasks are required to be solved by multiple simple and limited entities (also called *robots*) that can move in the environment. In this realm, manifold theoretical models have been introduced to formalize realistic scenarios (e.g. sensor or drone swarms, dynamic networks, software agents). One of the most studied is the *look-compute-move* (LCM) model [16, 17], where robots, once activated, execute a *cycle* of three steps: they *look* at the environment, they *compute* the next position executing a distributed algorithm, and they *move* to the computed position.

Under the umbrella of the LCM macro-model, a vast combination of models has been proposed to formalize different robot capabilities and to study how model settings affect its computational power. In this respect, robots are assumed to possess very limited and



© Caterina Feletti, Lucia Mambretti, Carlo Mereghetti, and Beatrice Palano; licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 13; pp. 13:1–13:19

Leibniz International Proceedings in Informatics



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

restricted features, in order to find the minimal sets of capabilities which are required to achieve a given task. Accordingly, robots are assumed to be *autonomous*, *indistinguishable*, *anonymous*, and *homogeneous*: namely, they act without any central control, they cannot distinguish themselves by external appearance or by ids, they possess the same features, they execute the same algorithm in a decentralized way. Moreover, most of the literature considers *punctiform* robots that cannot communicate with other robots (*silent*), without any persistent memory (*oblivious*), without any agreement on a global coordinate system, or chirality, or a unit measure (*disoriented*). Besides robot capabilities, different model *environments* have been proposed to study diverse scenarios. The existing models can be mainly divided into two groups: the models where robots act on the Euclidean plane [1, 14, 18, 23], and the models where robots act on discrete spaces (generally graphs, rings, or lattices) [7, 9, 12, 24]. According to the *synchronization*, robots may adhere to different modes (fair/unfair, synchronous/asynchronous, sequential...) [8]. In general, robots may be synchronized (time is globally divided into rounds) or not. Specifically, three modes are mainly studied in literature: the *fully synchronous* mode (**FULLY**), where all robots execute each step of the LCM cycle synchronously in one round, the *semi-synchronous* mode (**SEMI**), where at each round an arbitrary but nonempty subset of robots act synchronously, and the *asynchronous* mode (**ASYNCH**), where robots act without any synchronization assumption.

The traditional problems studied for swarms of mobile entities include **Pattern Formation** [1, 13, 14, 18, 26, 29, 30, 31], **Gathering** [5, 7, 12, 15, 22], **Scattering** [21, 25], **Flocking** [4]. A common goal of the algorithmic investigation is to reduce the model capabilities required to solve a given problem or to prove the impossibility of solving it under a certain set of capabilities. This approach has led to describing the *computational power* of a given model (i.e. the set of problems it can solve) and outlining the hierarchical relations (dominance, equivalence, or orthogonality) among different models. In the last decade, multiple works [2, 6, 9, 10, 11, 19] have inspected and compared the computational power of different models which differ in robot features and synchronization mode. According to the robot features, they have investigated how the communication and storage capabilities affect the computational power of the robots. Starting from the classical model where robots are both *oblivious* and *silent* (i.e. without any means of storage or communication), researchers have investigated how the possession of a persistent memory or communication means changes the power of such models. To characterize these extra properties, they proposed to add a *constant-size light* to each robot which can assume a color chosen among a constant and fixed set of colors. Such light is *persistent* (so the color is maintained until the next update), it can be updated at the beginning of a move step, and it can be internally or externally visible. Specifically, the literature focuses on four classes of robots: the *OBLLOT* class, where robots are assumed to be *oblivious* and *silent*, the *FSTA* class, where each robot is embedded with an *internal light* (visible just to the robot, thus providing a persistent memory), the *FCOM* class, where each robot is embedded with an *external light* (visible just to the other robots, thus providing communication means), and the *LUMI* class, where each robot is embedded with an external and internal light. According to the synchronization mode, each class has been studied under the three settings: **FULLY**, **SEMI**, and **ASYNCH**.

Besides some trivial relations between a pair of models that only differ because the first one enjoys a capability that the second one lacks, other model relations may not be obvious to identify. This is especially true for models characterized by completely different capabilities, so it may be difficult to understand which combination of capabilities is more powerful. In these cases, the literature has attempted to illustrate some *simulators* to prove the equivalence between models, or some *witness problems* to prove their strict dominance

or orthogonality. Specifically, in [2, 6, 19, 20], the authors study the computational power of transparent robots that can move on the Euclidean plane, assuming multiple robots can occupy the same positions (*multiplicity*). In [9, 10, 11], the authors make the same effort but for robots acting on graphs. In [3], the authors consider *energy-constraint* robots, i.e. robots that necessitate an idle round to restore the needed energy to perform a new cycle.

Related works and our contributions. Our work is inspired by the papers [2, 6, 19, 20] where the authors exhibit the complete taxonomy of the 12 models of robots that can freely move on the Euclidean plane. Such models vary for the synchronization mode and for the possibility to memorize and communicate. However they are assumed to be transparent, thus always guaranteeing complete visibility for the swarm, and collision-tolerant, thus allowing robots to occupy the same position at the same time.

In this paper, we investigate the computational power of *opaque robots*, i.e. robots that cannot see beyond a collinear robot. Opaqueness introduces a remarkable difficulty in the design of correct algorithms to solve some classical problems [1, 13, 14]. In fact, the *obstructed visibility* leads to critical issues to be addressed in the algorithmic strategies: robots may not be aware of the cardinality of the swarm, robots may not be aware if there are some moving robots in the ASYNCH mode, robots may not know the complete topology of the current configuration, robots may compute the next action based on partial information. As a matter of fact, *ad hoc* techniques are needed to cope with this visibility limitation [27, 28].

Besides the *opaqueness* feature, our model differs from [2, 6, 19, 20] since robots do not tolerate collisions (so we drop the *multiplicity* assumption). The reason behind this choice is twofold, and it is coherent with the related literature [1, 13, 14, 27, 28]. Firstly, assuming collision intolerance leads to the formalization and analysis of more realistic models, as does assuming robot opaqueness. Secondly, dropping the multiplicity assumption is coherent with the hypothesis of obstructed visibility in the case of collinearity. As a matter of fact, a multiplicity of two robots forms a “degenerate” collinearity with any other robot of the swarm, for which it would be unnatural to state the visibility relation in this special case. In this respect, some witness problems introduced in [2, 20] cannot be applied under our model, which needs a new study with specific witness problems.

In the first part of this work, we expose a preliminary study of the relations between transparent and opaque models. Intuitively, a transparent model seems to computationally dominate the same model but with opaque robots. In Section 3 we formally prove this strict dominance: endowing a model with transparency increases its computational power, allowing it to solve more problems. As a consequence, this result highlights that constant-size (internal or external) lights are not always sufficient to compensate for robot obstructed visibility.

In the second part of this work (Section 4), we present six witness problems showing the majority of the hierarchical relations among models of *collision-intolerant opaque robots*, thus providing a first overview of their computational taxonomy. For the sake of space, all relations proved in this work will be compactly shown in the theorems in Section 4 (i.e. without splitting them in multiple corollaries). See Appendix A for the proofs of such theorems.

2 Preliminaries

2.1 Models

This work compares 12 robot models that differ in some features. We here introduce in detail all the *core features* that such models share, and the *variable features* under study.

Core features. We investigate swarms of autonomous computational mobile robots, which act in the Euclidean plane \mathbb{R}^2 . Robots are *indistinguishable* (they cannot be distinguished by external appearance), *anonymous* (they are not provided with any id), *homogeneous* (they execute the same algorithm), and *punctiform* entities. We consider *opaque* robots so that in the case of three collinear robots p, q, r , the endpoint robots p, r cannot see each other. We assume robots are in the worst condition about orientation: they are *completely disoriented* so that they do not share a global common coordinate system (i.e. no agreement on origin, axis direction, chirality, or unit distance). Moreover, we assume that the local coordinate system of any robot may change from one activation to another (*variable disorientation*).

All the robots in the swarm are provided with the same deterministic algorithm, which is executed every time the robot is activated. At each time, a robot can be either *idle* or *active*, according to the scheduler. When activated, a robot r executes a *Look-Compute-Move* cycle: it takes the *snapshot* of its visible area (*look*), it executes the algorithm using the sole snapshot as input (*compute*), and it travels along a straight trajectory towards the computed destination (*move*). The snapshot of r contains all the positions (according to the coordinate system of r) and, possibly, the external colors of the robots visible to r : no other information about the swarm can be perceived (e.g. whether robots are idle/active, still/moving, ...). If the computed destination position is equal to the current one, r is said to perform a *null movement*. After the *move* step, r becomes idle again. We consider *rigid* models, i.e. no adversary can stop the motion of a robot¹.

We deal with a *collision-intolerant model* meaning that it does not tolerate either multiplicity (i.e. no robot can occupy the same location as another robot at the same time) or overlapping trajectories (robots r and s have overlapping trajectories if (i) r is moving from a to a' , (ii) s is moving from b to b' , and (iii) the segments $\bar{aa'}$ and $\bar{bb'}$ have points in common). We refer to both multiplicity and overlapping trajectories as *collisions*.

Variable features. Regarding the *memory and communication* features of robots, we consider the four models mainly proposed in the literature. In the *OBLOT* model, robots are assumed to be *oblivious* (i.e. they do not have any persistent memory to store data about past cycles) and *silent* (i.e. they do not have any means to communicate with other robots). In the *FSTA* model, robots are provided with a persistent *internal light* which can assume a color chosen from a constant-size set. Such internal light plays the role of a constant-size persistent memory. In the *FCOM* model, robots are equipped with a persistent *external light* visible only to other robots, which can assume a color chosen in a constant-size set of colors. Indeed, external lights can be exploited by the swarm to communicate some messages to the visible robots. Lastly, the *LUMI* model gather the features of both *FSTA* and *FCOM*. This model assumes *luminous* robots, which are equipped with a light that can be colored using a constant-size set of colors. Such light is both visible to the robot itself (working as an internal state) and visible to the other robots (working as an external communication means).

Regarding the *activation and synchronization* of robots, we consider the three modes mainly studied in the literature. In the *fully synchronous* mode (**FULLY**), time is split into atomic rounds, within which all robots are activated together and execute their LCM steps completely synchronously. The *semi-synchronous* mode (**SEMI**) differs from **FULLY** just for the fact that at each round an arbitrary but nonempty subset of robots is activated. In the *asynchronous* mode (**ASYNCH**), every robot is activated independently from the others, and

¹ In [2, 6, 19, 20], the authors consider both rigid and non-rigid models. In the next model comparisons (transparent vs opaque), we consider only rigid models.

every cycle step lasts a finite but unpredictable amount of time. For the SEMI and ASYNCH modes, robots do not know which are the activated robots at each instant. Moreover, we always assume the *fairness condition*: for each time t and for each robot r , there exists a time $t' > t$ such that r is activated. This condition allows us to compute time complexity considering the number of *epochs*, where an epoch is a minimal time frame within which each robot is activated at least once. The selection of the subset of robots activated at every time is made by an adversarial *scheduler*. Formally, let $\mathcal{R} = \{r_1, \dots, r_n\}$ be a swarm of n robots, and let \mathcal{T} be a time domain which can be discrete $\mathbb{N}_{\geq 0}$ (in FULLY and SEMI) or continuous $\mathbb{R}_{\geq 0}$ (in ASYNCH). An *activation scheduling* is a function $S : \mathcal{T} \rightarrow 2^{\mathcal{R}}$ defining the subset of the swarm that is activated at a specific time.

Notation. We use the notation \overline{X}^Y to indicate a model for opaque robots that possess all the above core features and that has X as communication-storage setting and Y as synchronization mode, where $X \in \{\text{OBLOT}, \text{FSTA}, \text{FCOM}, \text{LUMI}\}$ and $Y \in \{\text{F}, \text{S}, \text{A}\}$ (FULLY, SEMI, ASYNCH, resp.). Consistently with the notation used in [2, 6, 19, 20], we indicate with X^Y the same model as \overline{X}^Y but considering transparent robots which tolerate collisions. We refer to these two classes of models as the *opaque* and *transparent framework*.

2.2 Problems

Robot swarms are distributed systems aimed at solving problems. Since in these models robots can just move in the plane, the literature studies problems requiring a swarm to form (a sequence of) geometric patterns, and/or to travel along specific trajectories. Formally, let us assume a swarm of n robots $\mathcal{R} = \{r_1, \dots, r_n\}$ on the Euclidean plane. When no ambiguity arises, we indicate with r_i both the robot and the point on the plane where r_i is located. Given an absolute coordinate system Z on \mathbb{R}^2 , we define the *configuration* of the swarm at time t as the set $C = \{(x_1, l_1), \dots, (x_n, l_n)\}$ where $x_i \in \mathbb{R}^2$ is the position of r_i according to Z , and l_i is the light color of r_i , at time t . In the OBLOT model, we always assume $l_i = \text{off}$ for every $r_i \in \mathcal{R}$. A configuration is *valid* if $x_i \neq x_j$, for each $i \neq j$. We define \mathcal{C} as the set of all the valid configurations for \mathcal{R} . We say that a configuration C guarantees *complete visibility* if there are no collinearities among robots.

A *problem* P for a swarm of robots is defined² by a sequence $(\phi_0, \tau_0, \phi_1, \tau_1, \dots, \phi_m, \tau_m, \dots)$ where each ϕ_i is a condition on the configuration of the swarm, and where τ_i is a condition on the intermediate configurations that the swarm is requested to fulfill while reaching a new configuration where ϕ_{i+1} holds true. We call such sequence the *request of the problem* P . The initial condition ϕ_0 must include the clause stating that $l_i = \text{off}$ for every $r_i \in \mathcal{R}$. Except for this clause, since P might be solved without lights and under any synchronization mode, ϕ_i, τ_i must not impose any conditions on light colors or the number of cycles, for each i .

Starting from an initial configuration C_0 for which ϕ_0 is true, P is said to be *solved* under a scheduling mode if, for each scheduling Σ under the given mode, there exists an algorithm \mathbb{A} through which the swarm forms a sequence of configurations (C_1, \dots, C_m, \dots) such that ϕ_i holds in C_i for each $i \geq 1$, and such that τ_{i-1} holds during the formation of C_i starting from C_{i-1} . If the request of the problem is finite, the last condition τ_m requires the swarm to stay still after having satisfied the last condition ϕ_m of the request. If Σ works on a time domain $\mathcal{T} \in \{\mathbb{N}_{\geq 0}, \mathbb{R}_{\geq 0}\}$, we define the *evolution* of \mathbb{A} w.r.t. Σ and C_0 as the function $\mathcal{E} : \mathcal{T} \rightarrow \mathcal{C}$ such that $\mathcal{E}(0) = C_0$ and $\mathcal{E}(t)$ is the configuration reached at time $t > 0$

² For our purposes.

executing \mathbb{A} according to the scheduling. Indeed, there must exist a sequence of time instants $0 < t_1 < t_2 < \dots < t_m \dots$ upon which the evolution of \mathbb{A} both satisfies $\mathcal{E}(t_i) = C_i$ and guarantees the validity of the conditions τ_i in the corresponding time intervals.

2.3 Computational relations

Given a model M , we indicate with $\mathcal{P}(M)$ the set of problems solved under M , i.e. the *computational power* of M . Given two models M_1, M_2 , we define the following relations:

- M_1 is *computationally not less powerful* than M_2 , formally $M_1 \geq M_2$, if $\mathcal{P}(M_1) \supseteq \mathcal{P}(M_2)$, i.e any problem solvable in M_2 is solvable in M_1 ;
- M_1 is *computationally more powerful* than M_2 , formally $M_1 > M_2$, if $\mathcal{P}(M_1) \supset \mathcal{P}(M_2)$, i.e any problem solvable in M_2 is solvable in M_1 and there exists a problem solvable in M_1 that is not solvable in M_2 ;
- M_1 is *computationally orthogonal* to M_2 , formally $M_1 \perp M_2$, if $\mathcal{P}(M_1) \setminus \mathcal{P}(M_2) \neq \emptyset$ and $\mathcal{P}(M_2) \setminus \mathcal{P}(M_1) \neq \emptyset$, i.e there exists a problem solvable in M_1 (M_2 , resp.) that is not solvable in M_2 (M_1 , resp.);
- M_1 is *computationally equivalent* to M_2 , formally $M_1 \equiv M_2$, if $\mathcal{P}(M_1) = \mathcal{P}(M_2)$, i.e M_1 and M_2 solve the same set of problems.

The following relations trivially follow from the definitions of the models:

$$\begin{aligned} LUMI^Y &\geq FSTA^Y \geq OBLOT^Y \quad \text{and} \quad LUMI^Y \geq FCOM^Y \geq OBLOT^Y \\ X^F &\geq X^S \geq X^A \end{aligned}$$

where $Y \in \{F, S, A\}$ and $X \in \{OBLOT, FSTA, FCOM, LUMI\}$. Indeed, the same relations hold in the opaque framework.

3 Transparent vs opaque robots

► **Theorem 1.** *Let P be a problem solved in \overline{X}^Y . Then P is solved under X^Y .*

Proof. Let $\overline{\mathbb{A}}$ be an algorithm solving P under \overline{X}^Y . We can easily construct an algorithm \mathbb{A} solving P under X^Y . Given a robot r and given its snapshot as input σ of all the robots, \mathbb{A} computes $\mathbb{A}(\sigma) := \overline{\mathbb{A}}(\overline{\sigma})$ where $\overline{\sigma}$ is the snapshot obtained by σ removing all the robots which would be hidden from r in case of opaqueness. \mathbb{A} perfectly simulates $\overline{\mathbb{A}}$, thus correctly solving P for transparent robots. ◀

► **Corollary 2.** *For each $Y \in \{F, S, A\}$ and $X \in \{OBLOT, FSTA, FCOM, LUMI\}$,*

$$\overline{X}^Y \leq X^Y.$$

► **Problem 1 (Line-Stretch).** Let us consider an initial configuration where $n > 3$ robots are equally spaced along the same line, say γ . Let d be the distance between two adjacent robots. The problem asks the endpoint robots to move away from their adjacent robot and stop in order to form a new distance $d + \frac{d}{n}$ with them. They are allowed to travel only along γ . The other robots must stay still. See Figure 1.



■ **Figure 1** Line-Stretch.

► **Lemma 3.** *Line-Stretch is solved under \overline{OBLOT}^A .*

Proof. The problem is solved under the weakest model of the transparent framework. In fact, the endpoint robots can compute and head to their destination since they can count all the robots and at least two internal robots fix d . The final configuration is stable. ◀

► **Lemma 4.** *Line-Stretch cannot be solved under \overline{LUMI}^F .*

Proof. The problem cannot be solved under the strongest model of the opaque framework. Since the n robots are always collinear by request and are provided with constant-size lights, they cannot compute n either by sight or by using their lights to communicate/store the cardinality of the swarm, and so the endpoint robots will never accomplish the task. In fact, lights are inefficient for keeping a counter of the robots, due to their constant size. ◀

► **Theorem 5.** *For each $Y \in \{F, S, A\}$ and $X \in \{OBLOT, FSTA, FCOM, LUMI\}$,*

$$\overline{X}^Y < X^Y.$$

Proof. The result derives by combining Corollary 2 with Lemma 3 and Lemma 4. In fact, it holds that $\text{Line-Stretch} \in \mathcal{P}(X^Y)$ whereas $\text{Line-Stretch} \notin \mathcal{P}(\overline{X}^Y)$ for any X, Y . ◀

► **Theorem 6.** *Let P be a problem solved by an algorithm \mathbb{A} under X^Y always avoiding collisions, such that P is defined for a swarm with fixed cardinality, say k . If, given any evolution of \mathbb{A} , every robot can see k robots, then the problem can be solved even in \overline{X}^Y .*

Proof. Since at any activation, each robot is aware it sees the whole swarm, it can compute its next action by executing \mathbb{A} . This computation results in the solution of the problem considering opaque robots. ◀

4 Taxonomy of opaque models

We present our witness problems to prove some strict dominance ($>$) and orthogonality (\perp) relations among opaque models. Thanks to Theorem 1 and Theorem 6, one of the witness problems presented in [2] can be used to prove some hierarchical relations to hold in our opaque framework too. However, other witness problems in [2, 20] are not compliant with our collision-intolerant models; thus, we present specific problems that fit our assumptions.

4.1 Weakness of \overline{OBLOT}

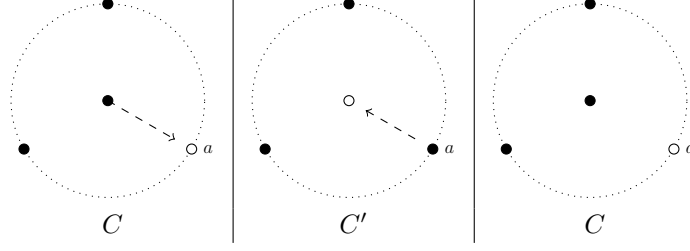
► **Problem 2 (Triangle Round-Trip).** Let C be a configuration where 3 robots are placed so that two of them lay on the vertices of an equilateral triangle (let a be the empty vertex), while the third robot lays on the triangle center. From C , the robot in the center has to move to a , forming the new configuration C' . Then, robots have to form C again, where a is again the empty vertex. See Table 1.

Triangle Round-Trip is a sub-case of the problem N -gon Round-Trip defined in [2] (see Definition 1).

► **Lemma 7.** *Triangle Round-Trip $\notin \mathcal{P}(\overline{OBLOT}^F)$.*

Proof. The problem has been shown to not belong to \overline{OBLOT}^F (see Lemma 3 in [2]). In fact, using oblivious and silent robots, there is no way to identify the former empty vertex a due to the full symmetry of C' . By the contrapositive of Theorem 1, the result holds. ◀

■ **Table 1** Configurations in Triangle Round-Trip.



► **Lemma 8.** $\text{Triangle Round-Trip} \in \left(\mathcal{P} \left(\overline{\mathcal{FSTA}}^A \right) \cap \mathcal{P} \left(\overline{\mathcal{FCOM}}^A \right) \right)$.

Proof. The problem has been shown to be solved in \mathcal{FSTA}^A and \mathcal{FCOM}^A (see Lemma 4-5 in [2]). Since in this version of the problem the cardinality of the swarm is fixed and the robots never create collinearities or collisions, we can apply Theorem 6 to state that Triangle Round-Trip can be solved both in $\overline{\mathcal{FSTA}}^A$ and $\overline{\mathcal{FCOM}}^A$. ◀

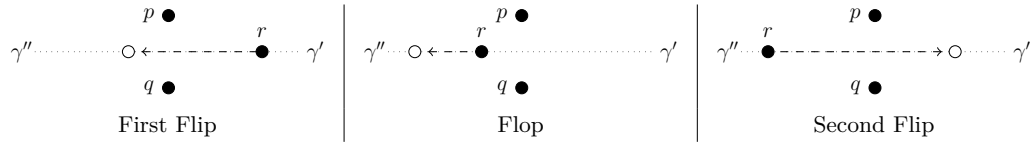
► **Theorem 9.** Given the schedulers $Y_1 = F$, $Y_2 = S$, $Y_3 = A$, it holds

$$\begin{aligned} \overline{\mathcal{FSTA}}^{Y_i} &> \overline{\text{OBLOT}}^{\{Y_j\}_{j \geq i}} \\ \overline{\mathcal{FCOM}}^{Y_i} &> \overline{\text{OBLOT}}^{\{Y_j\}_{j \geq i}} \\ \overline{\mathcal{LUMI}}^{Y_i} &> \overline{\text{OBLOT}}^{\{Y_j\}_{j \geq i}}. \end{aligned}$$

4.2 Orthogonality between $\overline{\mathcal{FSTA}}$ and $\overline{\mathcal{FCOM}}$

► **Problem 3 (Flip-Flop-Flip).** Let p , q and r be three robots forming a strictly isosceles triangle so that $\text{dist}(p, r) = \text{dist}(q, r)$. Let γ be the perpendicular bisector to the line segment $\bar{p}q$ passing through the point $b \in \bar{p}q$. Let γ' (γ'' , resp.) be the semi-line of γ starting from b and which contains (does not contain, resp.) r . The problem requires r to perpetually perform three subsequent actions (see Table 2), in an infinite loop: (i) r must reach a point on $\gamma'' \setminus \{b\}$; (ii) r must reach a different point on γ'' in order to move away from p, q ; (iii) r must reach a point on $\gamma' \setminus \{b\}$. The problem requires r to never leave γ and to never stop so that p, q, r form an equilateral triangle. Robots p, q must stay still.

■ **Table 2** Configurations in Flip-Flop-Flip.



► **Lemma 10.** $\text{Flip-Flop-Flip} \in \left(\mathcal{P} \left(\overline{\mathcal{FSTA}}^A \right) \cap \mathcal{P} \left(\overline{\mathcal{FCOM}}^F \right) \right)$.

Proof. We solve the problem in these two models using three colors (flip1, flop and flip2), assuming w.l.o.g. all robots start with the color flip1. The problem request guarantees that each robot can recognize its role by geometric conditions. In $\overline{\mathcal{FSTA}}^A$, r moves along γ changing its internal color following the perpetual scheme $(\text{flip1} - \text{flop} - \text{flip2})^\infty$, so that at each activation, r knows which is the current action to be performed. The robots p, q do

not need to change their colors. In the $\overline{\mathcal{FCOM}}^F$ model, all the robots synchronously update their external colors following the above scheme, so that at each round each robot knows what actions (color setting and move step) have to be accomplished. ◀

► **Lemma 11.** $\text{Flip-Flop-Flip} \notin \left(\mathcal{P} \left(\overline{\mathcal{OBLOT}}^F \right) \cup \mathcal{P} \left(\overline{\mathcal{FCOM}}^S \right) \right)$.

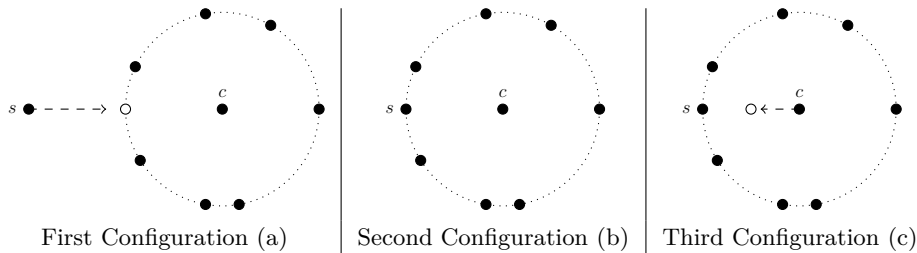
Proof. Flip-Flop-Flip cannot be solved under an $\overline{\mathcal{OBLOT}}$ model since r would not have any means to understand which movement it has to perform. Suppose by contradiction that there exists an $\overline{\mathcal{OBLOT}}$ algorithm \mathbb{A} solving Flip-Flop-Flip. Let σ be the snapshot taken by r which makes \mathbb{A} compute its first Flop action. Being in $\overline{\mathcal{OBLOT}}$, σ contains only the positions of the three robots in the current local coordinate system of r . Let us now assume that Flip-Flop-Flip starts from an initial configuration where the snapshot taken by r is identical to σ . Since \mathbb{A} has no further information as input, its output is still a Flop, which causes r to perform an erroneous action. Contradiction.

Flip-Flop-Flip cannot be solved under the $\overline{\mathcal{FCOM}}^S$ model too. By contradiction, suppose that the problem is solved by an algorithm \mathbb{A} . Let S be a SEMI activation scheduling under which \mathbb{A} solves the problem. We show that there exists a SEMI activation scheduling S' such that Flip-Flop-Flip is not solved by \mathbb{A} . Let t be the first round in S where r executes the first Flip. Let S' be a scheduling such that $S'(t') = S(t')$, $\forall t' \leq t$. Clearly, r executes its first Flip at the t -th round under S' . Suppose that, in the $(t+1)$ -th activation round under S' , r is the only one that gets activated, namely $S'(t+1) = \{r\}$. Yet, r has no memory of the previous activation rounds. As a consequence, r makes again a Flip. Contradiction. ◀

► **Theorem 12.** $\frac{\overline{\mathcal{LUMI}}^A}{\overline{\mathcal{LUMI}}^F} > \frac{\overline{\mathcal{FCOM}}^A}{\overline{\mathcal{FCOM}}^{S,A}}$, $\frac{\overline{\mathcal{LUMI}}^S}{\overline{\mathcal{LUMI}}^F} > \frac{\overline{\mathcal{FCOM}}^{S,A}}{\overline{\mathcal{FCOM}}^{S,A}}$.

► **Problem 4 (Newcomer Introducing).** Consider $n+2$ robots, with $n \geq 7$. Let n robots be placed on the same circle whose ray length is ρ . Let c be a robot lying in the center of the circle. Let s be a robot external to the circle so that s can see c . The problem requires sequentially forming two configurations. First, s must travel along the line \overline{sc} and stop on the boundary of the circle. Second, c must travel along the radius defined by s and stop in a position c' so that $\text{dist}(s, c') = \frac{1}{2}\rho$. All the other robots must stay still. See Table 3.

■ **Table 3** Configurations in Newcomer Introducing.



► **Lemma 13.** $\text{Newcomer Introducing} \notin \mathcal{P} \left(\overline{\mathcal{FSTA}}^F \right)$.

Proof. The impossibility of solving the problem with just internal lights derives from the fact that starting from the second configuration (see Table 3.b) c has no way to recognize which robot is s . Since s can be anywhere in the disposition of the $n+1$ robots on the circle, a constant set of colors would not be sufficient to store robot indices. ◀

13:10 Computational Power of Opaque Robots

► **Lemma 14.** *Newcomer Introducing* $\in \mathcal{P}(\overline{\mathcal{FCOM}^A})$.

Proof. We show a possible $\overline{\mathcal{FCOM}^A}$ algorithm solving *Newcomer Introducing* with two colors: off and s. All the robots are initially set to color off. Each robot can determine its role by the geometry of the configurations (c sees $n \geq 7$ robots equidistant from itself and an external robot, s sees at least four robots forming a circle with a robot on its center, while the other robots can see they lay on a circle with at least other $n - 2 \geq 5$ robots). When s is activated, it sets its light to s and starts to move. This color is maintained also in its next activations. When c is activated, if it sees a robot s on the circle, it can compute its destination correctly. The last configuration is stable: no other robot will move. ◀

► **Theorem 15.** *Given the schedulers* $Y_1 = F$, $Y_2 = S$, $Y_3 = A$,

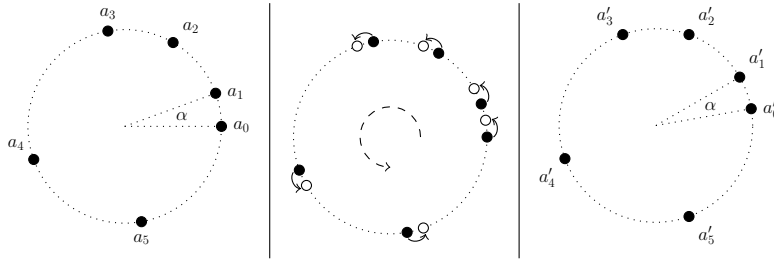
$$\overline{\mathcal{LUMI}^{Y_i}} > \overline{\mathcal{FSTA}^{\{Y_j\}_{j \geq i}}}.$$

► **Theorem 16.** $\overline{\mathcal{FSTA}^{F,S,A}} \perp \overline{\mathcal{FCOM}^{S,A}}$.

4.3 Power of FULLY

► **Problem 5 (Spinning).** The problem is defined recursively, without any stop conditions. Consider a configuration C where $n \geq 5$ robots $\{r_0, \dots, r_{n-1}\}$ are located on a circle centered in O . Let a_0, \dots, a_{n-1} be the related positions of the robots such that it is possible to establish a global clockwise direction (e.g. the one going from a_0 to a_2 , passing through a_1). Let α be the angle $a_0 \hat{O} a_1$, which is the minimum angle in $\{a_i \hat{O} a_{i+1}\}_{0 \leq i \leq n-1}$. The problem requires the given configuration to form a new configuration C' by rotating each r_i from a_i to a'_i of an angle $\frac{\alpha}{2}$, following the agreed clockwise direction. Robots are required only stop on the target points lying on the circumference. Recursively, the problem demands the same request starting from C' . See Table 4.

■ **Table 4** Configurations in Spinning.



► **Lemma 17.** *Spinning* $\in (\mathcal{P}(\overline{\mathcal{OBL\O T}^F}) \cap \mathcal{P}(\overline{\mathcal{LUMI}^A}))$.

Proof. The problem is solvable in $\overline{\mathcal{OBL\O T}^F}$: each robot always has complete visibility of the swarm, so it is able to determine the rotation center and the rotation angle. The FULLY mode guarantees that all the robots agree on the same rotation-angle, at each round.

The problem is solvable under $\overline{\mathcal{LUMI}^A}$, by using these colors: off, a0, a1, moving0, moving1, m0, m1, moving, moved, end. The algorithm solving the problem executes the same sub-routine perpetually. This sub-routine implements a complete circle rotation of the swarm. At the beginning of each circle rotation, all robots are off. In the first epoch, the robots r_0 and r_1 set their lights as a0 and a1, respectively. After this setting, robot a0 (a1,

resp.) computes its destination position, sets its light to `moving0` (`moving1`, resp.) and starts moving. If a robot r , which is not `moving0` or `moving1` colored, sees a `moving0` or `moving1` robot, r does nothing. When a `moving0` (`moving1`, resp.) robot is activated, it just updates its light to `m0` (`m1`, resp.). Once the rotation angle through `m0` and `m1` has been fixed, the other robots can start their rotation. If an `off` robot r sees both `m0` and `m1` on the circle, it sets its light as `moving` and starts its rotation. When a `moving` robot is activated, it sets its light to `moved`. When a robot sees only `m0`, `m1`, `moved`, or `end` robots, then it updates its color to `end`. In the last phase of the sub-routine, if an `end` robot can see only `end` or `off` robots, it resets its color to `off`. Once all robots are `off`, the circle rotation is ready to restart. ◀

► **Lemma 18.** $Spinning \notin \left(\mathcal{P} \left(\overline{FSTA}^S \right) \cup \mathcal{P} \left(\overline{FCOM}^S \right) \right)$.

Proof. `Spinning` is not solvable under \overline{FSTA}^S since an activated robot r cannot know what movements other robots have already made, thus it cannot determine the rotation-angle.

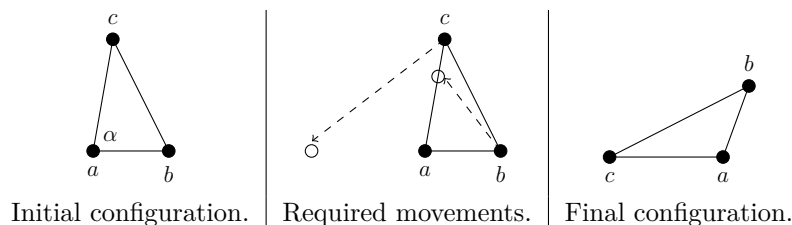
`Spinning` is not solvable under \overline{FCOM}^S . Suppose that, by contradiction, there exists an algorithm \mathbb{A} solving `Spinning`. In particular, the problem is solved under an activation scheduler S . Let r_0 be the robot in position a_0 . Let t_1 be the activation time, under S , of the first round during which r_0 performs a non-null movement. Let S' be another scheduling such that $S'(t) := S(t) \ \forall t < t_1$ and $S'(t_1) = S'(t_1 + 1) := \{r_0\}$. If \mathbb{A} is executed under S' , then the execution is the same as S until time $t_1 - 1$. At time t_1 , r_0 behaves in the same way as it did under scheduling S but, as no other robot has been activated, then there is no way to keep track of the fact that r_0 has already moved. At time $t_1 + 1$, r_0 is activated again but it cannot understand from geometric conditions that it must stay still. Contradiction. ◀

► **Theorem 19.**

$$\begin{aligned} \overline{OBLOT}^F &> \overline{OBLOT}^{S,A}, & \overline{FSTA}^F &> \overline{FSTA}^{S,A}, & \overline{FCOM}^F &> \overline{FCOM}^{S,A}, \\ \overline{OBLOT}^F &\perp \overline{FCOM}^{S,A}, & \overline{OBLOT}^F &\perp \overline{FSTA}^{S,A}. \end{aligned}$$

► **Problem 6 (Angle-Shift).** Consider an initial configuration with three robots forming an acute and scalene triangle. Let a, b, c be the three robots, where a is placed on the greatest angle, say α , whereas c is placed on the smallest angle. Fixing a as the rotation center and following the direction given by a, b, c , the problem requires b to rotate of α and c to rotate of $\pi - \alpha$. The robots are not allowed to stop anywhere else on the plane. Afterwards, the robots must stay still. See Table 5.

■ **Table 5** Angle-Shift.



► **Lemma 20.** $Angle-Shift \in \left(\mathcal{P} \left(\overline{OBLOT}^F \right) \setminus \mathcal{P} \left(\overline{LUMI}^S \right) \right)$.

Proof. `Angle-Shift` is solvable under any FULLY model: if b and c perform their cycles at the same time, then they correctly compute their target position. The final configuration is stable since it always forms an obtuse triangle (terminal condition).

13:12 Computational Power of Opaque Robots

Instead, the swarm can suffer from information loss in SEMI, making Angle-Shift unsolvable even under $\overline{\mathcal{LUMI}}^S$. In fact, suppose that in the initial configuration only b is activated. After b 's movement, the three robots turn out to be collinear in the reached configuration. As a result, c has no means to recompute α , whether c uses the geometry of the configuration or uses constant-size lights. The same happens even if only c is activated. ◀

► **Theorem 21.** $\overline{\mathcal{LUMI}}^F > \overline{\mathcal{LUMI}}^{S,A}$, $\overline{\mathcal{OBL\O{T}}^F} \perp \overline{\mathcal{LUMI}}^{S,A}$, $\overline{\mathcal{FST\A}}^F \perp \overline{\mathcal{LUMI}}^{S,A}$.

4.4 Opaqueness and asynchrony

We now introduce the Pseudo-Polygon problem which shows a peculiar issue occurring in case of obstructed visibility and asynchrony.

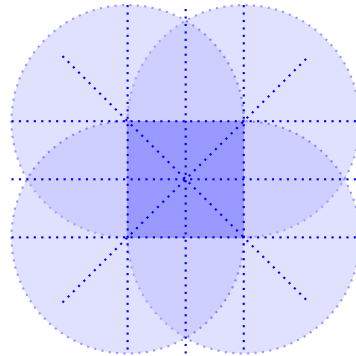
► **Definition 22.** Given a regular n -gon \mathcal{N} , for any $n \geq 4$, a pseudo-polygon \mathcal{Q} is a subset of vertices of \mathcal{N} , such that $|\mathcal{Q}| \geq \frac{n}{2} + 1$. We call \mathcal{N} the associated polygon with respect to \mathcal{Q} .

Given a pseudo-polygon \mathcal{Q} , it is always possible to determine the associated polygon, which is unique. In fact, as \mathcal{Q} contains at least three vertices, the circumscribed circle is univocally defined. Moreover, since \mathcal{Q} contains more than half of the vertices of the associated n -gon, there always exist at least two vertices that are adjacent in \mathcal{N} . So, it is always possible to univocally establish the associated polygon from a pseudo-polygon.

► **Definition 23.** A safe zone of a regular polygon is the locus of all points x in the plane such that:

- x is external to the regular polygon;
- x is not aligned with any of the two vertices of the associated polygon;
- x does not lie on the bisector of any edge of the associated polygon (equivalently, x is not equally distanced from any two adjacent vertices);
- if ℓ is the length of the edge of the polygon, then the distance between x and any vertex of the polygon is at least ℓ .

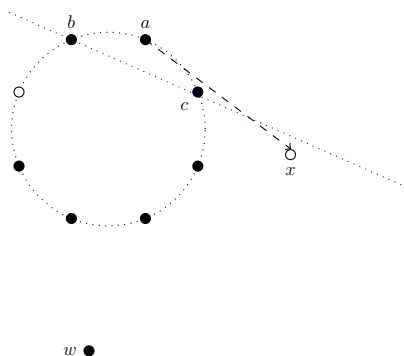
Figure 2 depicts the (complement of the) safe zone of a square.



■ **Figure 2** The safe zone of the square comprehends all the points not belonging to the blue-colored (infinite) lines and zones.

► **Problem 7 (Pseudo-Polygon).** Let \mathcal{N} be a regular n -gon with $n \geq 6$ vertices. Let \mathcal{Q} be a pseudo-polygon of $m \geq \frac{n}{2} + 2$ vertices, associated with \mathcal{N} . Consider a swarm of $m + 1$ robots, where m robots lay on \mathcal{Q} and let the last robot, w , lay in the safe zone of \mathcal{N} . Let a be the farthest robot from w . Let b, c be the first two found robots, starting from a and following

both directions on the perimeter of the associated polygon, one per each direction taken. Assume $\text{dist}(b, w) > \text{dist}(c, w)$. The problem requires a to move away from b towards a point x such that (i) x belongs to the safe zone of \mathcal{N} , (ii) x belongs to the halfplane delimited by the line bc that does not contain a , and (iii) x must not be on any line passing by the position of w and any other robot on \mathcal{Q} . Note that requests (i,iii) are imposed in order to have x visible by every robot. See Figure 3.



■ **Figure 3** The Pseudo-Polygon problem associated with an octagon.

► **Lemma 24.** $\text{Pseudo-Polygon} \notin \mathcal{P}(\overline{\mathcal{FSTA}}^A)$.

Proof. Pseudo-Polygon cannot be solved in the ASYNCH mode, only using internal lights. Let us consider the problem instance given by Figure 3 where the pseudo-polygon of the initial configuration is composed of $\frac{n}{2} + 3$ vertices, with $n = 8$. Let us assume b is activated for the first time during the movement of a , when a is hidden by c (i.e. b, c, a are collinear). When b looks at its snapshot, it recognizes a feasible initial configuration (it sees a pseudo-polygon with $\frac{n}{2} + 2$ robots, and the robot w). According to this configuration, b erroneously elects itself as the robot that has to move away from the pseudo-polygon. It has no means to understand if a exists or not. On the other hand, a has no means to know if b has updated its internal light to memorize it is not the elected robot to move. ◀

False election. The impossibility of solving Pseudo-Polygon in the asynchronous modes with just internal lights derives from a critical issue that is typical of swarms with obstructed visibility. This critical issue can be described as the *false election* phenomenon. Such phenomenon can be informally described as follows: from a stable configuration, the given problem requires the use of a leader election routine to elect the unique robot (the *true leader*) which has to execute a non-null movement to reach the next configuration. All the other robots have to stay still. In ASYNCH, a robot r executes its look step while the true leader is moving and is hidden from r . However, r cannot deduce the presence of the true leader from its snapshot. So, applying the same leader election routine, r elects itself as the (*false*) leader, thus starting an unrequested movement.

The false election phenomenon must be examined when trying to transpose a SEMI algorithm in ASYNCH. In particular, the use of lights must be considered as a possible method to avoid false elections. As we have shown in Lemma 24 for Pseudo-Polygon, internal lights are not sufficient to cope with them. Instead, the next lemma proves that external lights are required (and sufficient) to correctly solve the Pseudo-Polygon problem in ASYNCH.

► **Lemma 25.** $\text{Pseudo-Polygon} \in \left(\mathcal{P}(\overline{\mathcal{OBL\mathcal{O}T}}^S) \cap \mathcal{P}(\overline{\mathcal{FCOM}}^A) \right)$.

Proof. Pseudo-Polygon is solvable in \overline{OBLOT}^S (i.e. in any synchronous model), since complete visibility is guaranteed at any activation time and all the movements (null and non-null) are univocally determined by geometric conditions. In fact, each robot can determine \mathcal{Q} , the watcher w , and the robot a (the farthest from w). The robot a can compute its final destination and move there. If a robot is not the farthest from the watcher, or if it sees two robots that are not part of the pseudo-polygon, then it stands still.

Pseudo-Polygon needs at least external lights to be solvable in the ASYNCH mode. We show here an algorithm that needs 4 colors: off (default), on, a, b. In the first epoch, every robot updates its color according to its role: robot a turns into a, robot b turns into b, whereas the remainder turns into on. Afterward, let r be an activated robot that sees no off robots and that notes there is only one robot (the watcher) out of the pseudo-polygon. Let V_r be the set of colors r can see.

- if $V_r = \{a, b, on\}$, r turns into on and stays still;
- if $V_r = \{a, on\}$, r turns into b and stays still;
- if $V_r = \{b, on\}$, and if r is the farthest robot from w , it turns into a and starts moving;
- if $V_r = \{on\}$, it means r is b and stays still (robot a is hidden).

If a robot r sees two robots not belonging to the pseudo-polygon, then r does not move (the final configuration is already formed or is about to be formed). ◀

► **Theorem 26.** $\overline{OBLOT}^S > \overline{OBLOT}^A$, $\overline{FSTA}^S > \overline{FSTA}^A$, $\overline{FSTA}^A \perp \overline{OBLOT}^S$.

5 Relation map

Table 6 summarizes the results proved in this work, showing the relations ($>$, $<$, \perp , and \equiv) that hold between the pairs of models in our opaque framework. The map shows also which of the six witness problems (TRT for Triangle Round-Trip, FFF for Flip-Flip-Flip, NWC for Newcomer Introducing, SPIN for Spinning, ASH for Angle-Shift, PSE for Pseudo-Polygon) have been used to prove such relations. For some pairs of models (gray cells), the knowledge about what kind of relation holds is still now incomplete. E.g. between \overline{FSTA}^F and \overline{FCOM}^F two possible relations ($<$ or \perp) can exist: so far we have built Newcomer Introducing as witness problem proving that $\text{Newcomer Introducing} \in \left(\mathcal{P}(\overline{FCOM}^F) \setminus \mathcal{P}(\overline{FSTA}^F) \right)$. To prove the orthogonality relation, we should find a witness problem B such that $B \in \left(\mathcal{P}(\overline{FSTA}^F) \setminus \mathcal{P}(\overline{FCOM}^F) \right)$. Instead, to prove the strict dominance relation, we should find that any problem in \overline{FSTA}^F can be solved also under \overline{FCOM}^F . For the pairs of models where the relation is unknown in the opaque framework, we have reported the relation holding in the transparent framework in red.

6 Conclusions

We have investigated the computational power of the 12 models of collision-intolerant opaque robots, thus presenting the taxonomy of the problems solved in such framework. We have taken inspiration from [2, 6, 19, 20] where the authors provide the complete map of the relations held by the same 12 models but considering collision-tolerant transparent robots.

Thus far, the relations proven here in our opaque framework are the same as in the corresponding transparent framework. The natural question that arises from this observation is whether the relation map of the opaque models is completely identical to the relation map of the transparent models. To answer this question, future works should find the missing relations among the twelve opaque models in order to obtain the complete hierarchy in the

■ **Table 6** Relation map.

\uparrow	\overline{LUMI}^f	\overline{FCOM}^f	\overline{FSTA}^f	\overline{OBLOT}^f	\overline{LUMI}^s	\overline{FCOM}^s	\overline{FSTA}^s	\overline{OBLOT}^s	\overline{LUMI}^A	\overline{FCOM}^A	\overline{FSTA}^A
\overline{OBLOT}^A	< TRT	< TRT	< TRT	< SPIN	< TRT	< TRT	< TRT	< PSE	< TRT	< TRT	< TRT
\overline{FSTA}^A	< NWC	< or \perp , < NWC,	< SPIN	\perp TRT, SPIN	< NWC	\perp NWC, FFF	< PSE	\perp PSE, TRT	< NWC	\perp NWC, FFF	
\overline{FCOM}^A	< FFF	< FFF	\perp FFF, NWC	\perp NWC, SPIN	< FFF	< or \equiv , <	\perp FFF, NWC	> or \perp , \perp NWC,	< FFF		
\overline{LUMI}^A	< ASH	< or \perp , < ASH,	\perp ASH, NWC	\perp ASH, TRT	< or \equiv , \equiv	> or \perp , > FFF,	> or \perp , > NWC,	> or \perp , > TRT,			
\overline{OBLOT}^s	< TRT	< TRT	< TRT	< SPIN	< TRT	< TRT	< TRT				
\overline{FSTA}^s	< NWC	< or \perp , < NWC,	< SPIN	\perp TRT, SPIN	< NWC	\perp NWC, FFF					
\overline{FCOM}^s	< FFF	< FFF	\perp FFF, NWC	\perp SPIN, NWC	< FFF						
\overline{LUMI}^s	< ASH	< or \perp , < ASH,	\perp ASH, NWC	\perp ASH, TRT							
\overline{OBLOT}^f	< TRT	< TRT	< TRT								
\overline{FSTA}^f	< NWC	< or \perp , < NWC,									
\overline{FCOM}^f	< or \equiv , \equiv										

opaque framework. Among the others, it is worth mentioning the yet unknown relation between \overline{LUMI}^S and \overline{LUMI}^A . In the transparent framework, the two models were proven to be computationally equivalent [6] through the design of a simulator which, with the help of extra light colors, simulates any SEMI algorithm in the ASYNCH mode. This simulator is not adequate to prove the same relation considering opaque robots, precisely because of their obstructed visibility. With the Pseudo-Polygon problem, we have presented the *false election* phenomenon whose formalization and investigation will be preparatory to answer this interesting open question: is it possible to simulate a \overline{LUMI}^S algorithm in the ASYNCH mode, thus proving that \overline{LUMI}^S and \overline{LUMI}^A are two equivalent models also in the opaque framework? Are constant-size lights sufficient to always avoid the phenomenon of false elections? In addition, it would be necessary to formalize and study all the critical issues caused by obstructed visibility: such formalizations may be essential for the correct investigation of the missing relations.

In conclusion, further research directions could broaden the range of robot models to be compared by considering non-rigid models and/or less popular synchronization modes (e.g. sequential, round-robin, etc.).

References

- 1 Kaustav Bose, Manash Kumar Kundu, Ranendu Adhikary, and Buddhadeb Sau. Arbitrary pattern formation by asynchronous opaque robots with lights. *Theor. Comput. Sci.*, 849:138–158, 2021. doi:10.1016/J.TCS.2020.10.015.
- 2 Kevin Buchin, Paola Flocchini, Irina Kostitsyna, Tom Peters, Nicola Santoro, and Koichi Wada. Autonomous mobile robots: Refining the computational landscape. In *35th International Parallel and Distributed Processing Symposium Workshops, IPDPS*, pages 576–585. IEEE, 2021. doi:10.1109/IPDPSW52791.2021.00091.
- 3 Kevin Buchin, Paola Flocchini, Irina Kostitsyna, Tom Peters, Nicola Santoro, and Koichi Wada. On the computational power of energy-constrained mobile robots: Algorithms and cross-model analysis. In *29th International Colloquium on Structural Information and Communication Complexity, SIROCCO*, volume 13298 of *Lecture Notes in Computer Science*, pages 42–61. Springer, 2022. doi:10.1007/978-3-031-09993-9_3.

- 4 Davide Canepa and Maria Gradinariu Potop-Butucaru. Stabilizing flocking via leader election in robot networks. In *9th International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS*, volume 4838 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2007. doi:10.1007/978-3-540-76627-8_7.
- 5 Gianlorenzo D’Angelo, Gabriele Di Stefano, Ralf Klasing, and Alfredo Navarra. Gathering of robots on anonymous grids and trees without multiplicity detection. *Theor. Comput. Sci.*, 610:158–168, 2016. doi:10.1016/J.TCS.2014.06.045.
- 6 Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. Autonomous mobile robots with lights. *Theor. Comput. Sci.*, 609:171–184, 2016. doi:10.1016/J.TCS.2015.09.018.
- 7 Shantanu Das, Riccardo Focardi, Flaminia L. Luccio, Euripides Markou, and Marco Squarcina. Gathering of robots in a ring with mobile faults. *Theor. Comput. Sci.*, 764:42–60, 2019. doi:10.1016/J.TCS.2018.05.002.
- 8 Xavier Défago, Maria Potop-Butucaru, and Sébastien Tixeuil. Fault-tolerant mobile robots. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2019. doi:10.1007/978-3-030-11072-7_10.
- 9 Mattia D’Emidio, Daniele Frigioni, and Alfredo Navarra. Synchronous robots vs asynchronous lights-enhanced robots on graphs. In *16th Italian Conference on Theoretical Computer Science, ICTCS*, pages 169–180. Elsevier, 2015. doi:10.1016/J.ENTCS.2016.03.012.
- 10 Mattia D’Emidio, Daniele Frigioni, and Alfredo Navarra. Characterizing the computational power of anonymous mobile robots. In *36th International Conference on Distributed Computing Systems, ICDCS*, pages 293–302. IEEE Computer Society, 2016. doi:10.1109/ICDCS.2016.58.
- 11 Mattia D’Emidio, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Characterizing the computational power of mobile robots on graphs and implications for the euclidean plane. *Inf. Comput.*, 263:57–74, 2018. doi:10.1016/J.IC.2018.09.010.
- 12 Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Multiple agents rendezvous in a ring in spite of a black hole. In *7th International Conference on Principles of Distributed Systems, OPODIS*, volume 3144 of *Lecture Notes in Computer Science*, pages 34–46. Springer, 2003. doi:10.1007/978-3-540-27860-3_6.
- 13 Caterina Feletti, Carlo Mereghetti, and Beatrice Palano. Uniform circle formation for swarms of opaque robots with lights. In *20th International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS*, volume 11201 of *Lecture Notes in Computer Science*, pages 317–332. Springer, 2018. doi:10.1007/978-3-030-03232-6_21.
- 14 Caterina Feletti, Carlo Mereghetti, and Beatrice Palano. $O(\log n)$ -time uniform circle formation for asynchronous opaque luminous robots. In *27th International Conference on Principles of Distributed Systems, OPODIS*, volume 286 of *LIPICs*, pages 5:1–5:21, 2023. doi:10.4230/LIPICs.OPODIS.2023.5.
- 15 Paola Flocchini. Gathering. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 63–82. Springer, 2019. doi:10.1007/978-3-030-11072-7_4.
- 16 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012. doi:10.2200/S00440ED1V01Y201208DCT010.
- 17 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors. *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*. Springer, 2019. doi:10.1007/978-3-030-11072-7.
- 18 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Distributed computing by mobile robots: uniform circle formation. *Distributed Comput.*, 30(6):413–457, 2017. doi:10.1007/S00446-016-0291-X.
- 19 Paola Flocchini, Nicola Santoro, Yuichi Sudo, and Koichi Wada. On asynchrony, memory, and communication: Separations and landscapes. In *27th International Conference on Principles*

- of *Distributed Systems, OPODIS*, volume 286 of *LIPICs*, pages 28:1–28:23, 2023. doi:10.4230/LIPICs.OPODIS.2023.28.
- 20 Paola Flocchini, Nicola Santoro, and Koichi Wada. On memory, communication, and synchronous schedulers when moving and computing. In *23rd International Conference on Principles of Distributed Systems, OPODIS*, volume 153 of *LIPICs*, pages 25:1–25:17, 2019. doi:10.4230/LIPICs.OPODIS.2019.25.
 - 21 Taisuke Izumi, Daichi Kaino, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. On time complexity for connectivity-preserving scattering of mobile robots. *Theor. Comput. Sci.*, 738:42–52, 2018. doi:10.1016/J.TCS.2018.04.047.
 - 22 Sayaka Kamei, Anissa Lamani, Fukuhito Ooshita, Sébastien Tixeuil, and Koichi Wada. Gathering on rings for myopic asynchronous robots with lights. In *23rd International Conference on Principles of Distributed Systems, OPODIS*, volume 153 of *LIPICs*, pages 27:1–27:17, 2019. doi:10.4230/LIPICs.OPODIS.2019.27.
 - 23 Peter Kling and Friedhelm Meyer auf der Heide. Continuous protocols for swarm robotics. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 317–334. Springer, 2019. doi:10.1007/978-3-030-11072-7_13.
 - 24 Giuseppe Antonio Di Luna. Mobile agents on dynamic graphs. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 549–584. Springer, 2019. doi:10.1007/978-3-030-11072-7_20.
 - 25 Moumita Mondal and Sruti Gan Chaudhuri. Uniform scattering of robots on alternate nodes of a grid. In *23rd International Conference on Distributed Computing and Networking*, pages 254–259. ACM, 2022. doi:10.1145/3491003.3493231.
 - 26 Giuseppe Prencipe. Pattern formation. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 37–62. Springer, 2019. doi:10.1007/978-3-030-11072-7_3.
 - 27 Gokarna Sharma, Ramachandran Vaidyanathan, and Jerry L. Trahan. Constant-time complete visibility for robots with lights: The asynchronous case. *Algorithms*, 14(2):56, 2021. doi:10.3390/A14020056.
 - 28 Gokarna Sharma, Ramachandran Vaidyanathan, Jerry L. Trahan, Costas Busch, and Suresh Rai. Complete visibility for robots with lights in $O(1)$ time. In *18th International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS*, pages 327–345, 2016. doi:10.1007/978-3-319-49259-9_26.
 - 29 Kazuo Sugihara and Ichiro Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *J. Field Robotics*, 13(3):127–139, 1996.
 - 30 Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999. doi:10.1137/S009753979628292X.
 - 31 Masafumi Yamashita and Ichiro Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theor. Comput. Sci.*, 411(26-28):2433–2453, 2010. doi:10.1016/J.TCS.2010.01.037.

A Proofs of theorems

The proofs of the following theorems hold combining the previously stated lemmas and by transitivity. We use the compacted notation $\{\overline{X}_1, \dots, \overline{X}_m\}^{Y_1, \dots, Y_h}$ to indicate all the models in $\{\overline{X}_i^{Y_j}\}_{\substack{1 \leq i \leq m \\ 1 \leq j \leq h}}$ where $X_i \in \{OBLLOT, FSTA, FCOM, LUMI\}$ and $Y_j \in \{F, S, A\}$.

► **Theorem 9.** *Given the schedulers $Y_1 = F$, $Y_2 = S$, $Y_3 = A$, it holds*

$$\overline{FSTA}^{Y_i} > \overline{OBLLOT}^{\{Y_j\}_{j \geq i}}$$

13:18 Computational Power of Opaque Robots

$$\overline{FCOM}^{Y_i} > \overline{OBL\mathcal{O}T}^{\{Y_j\}_{j \geq i}}$$

$$\overline{LUMI}^{Y_i} > \overline{OBL\mathcal{O}T}^{\{Y_j\}_{j \geq i}}.$$

Proof. Triangle Round-Trip cannot be solved under $\overline{OBL\mathcal{O}T}^{F,S,A}$ (by Lemma 7) but it can be solved under $\{\overline{FSTA}, \overline{FCOM}, \overline{LUMI}\}^{A,S,F}$ (by Lemma 8). Combining the results, we obtain that $\overline{OBL\mathcal{O}T}$ is strictly dominated by \overline{FSTA} and \overline{FCOM} for a given synchronization mode $Y_i \in \{F, S, A\}$. The other strict dominances are derived by transitivity. ◀

► **Theorem 12.**

$$\overline{LUMI}^A > \overline{FCOM}^A$$

$$\overline{LUMI}^S > \overline{FCOM}^{S,A}$$

$$\overline{LUMI}^F > \overline{FCOM}^{S,A}$$

$$\overline{FCOM}^F > \overline{FCOM}^{S,A}.$$

Proof. Flip-Flop-Flip is solved under \overline{FCOM}^F and $\overline{LUMI}^{A,S,F}$ (by Lemma 10) but it cannot be solved under $\overline{FCOM}^{S,A}$ (by Lemma 11). Combining the results, the strict dominance relations follow. ◀

► **Theorem 15.** *Given the schedulers $Y_1 = F$, $Y_2 = S$, $Y_3 = A$, it holds*

$$\overline{LUMI}^{Y_i} > \overline{FSTA}^{\{Y_j\}_{j \geq i}}.$$

Proof. By Lemma 14, Newcomer Introducing is solved under $\overline{LUMI}^{A,S,F}$. By Lemma 13, Newcomer Introducing cannot be solved under $\overline{FSTA}^{F,S,A}$. Combining the results, the strict dominance relations follow. ◀

► **Theorem 16.**

$$\overline{FSTA}^{F,S,A} \perp \overline{FCOM}^{S,A}.$$

Proof. By Lemma 10 and Lemma 11, Flip-Flop-Flip is solved in $\overline{FSTA}^{F,S,A}$ but not in $\overline{FCOM}^{S,A}$. By Lemma 14 and Lemma 13, Newcomer Introducing is solved in $\overline{FCOM}^{S,A}$ but not in $\overline{FSTA}^{F,S,A}$. Combining the results, the orthogonality relations follow. ◀

► **Theorem 19.**

$$\overline{OBL\mathcal{O}T}^F > \overline{OBL\mathcal{O}T}^{S,A}$$

$$\overline{FSTA}^F > \overline{FSTA}^{S,A}$$

$$\overline{FCOM}^F > \overline{FCOM}^{S,A}$$

$$\overline{OBL\mathcal{O}T}^F \perp \overline{FCOM}^{S,A}$$

$$\overline{OBL\mathcal{O}T}^F \perp \overline{FSTA}^{S,A}.$$

Proof. The above relations hold combining the previous lemmas and by transitivity:

- the strict dominance of \overline{X}^F over $\overline{X}^{S,A}$ derives from Lemma 17 and Lemma 18, for each $X \in \{\overline{OBL\mathcal{O}T}, \overline{FSTA}, \overline{FCOM}\}$. In fact, Spinning is solved in $\{\overline{OBL\mathcal{O}T}, \overline{FSTA}, \overline{FCOM}\}^F$ but it is not solved in $\{\overline{OBL\mathcal{O}T}, \overline{FSTA}, \overline{FCOM}\}^{S,A}$;

- the orthogonality between \overline{OBLOT}^F over $\overline{FCOM}^{S,A}$ holds since **Spinning** is solved in \overline{OBLOT}^F but not in $\overline{FCOM}^{S,A}$, and since **Newcomer Introducing** is solved in $\overline{FCOM}^{S,A}$ but not in \overline{OBLOT}^F (by Lemma 14, Lemma 13);
- the orthogonality between \overline{OBLOT}^F over $\overline{FSTA}^{S,A}$ holds since **Spinning** is solved in \overline{OBLOT}^F but not in $\overline{FSTA}^{S,A}$, and since **Triangle Round-Trip** is solved in $\overline{FSTA}^{S,A}$ but not in \overline{OBLOT}^F (by Lemma 8, Lemma 7). ◀

► **Theorem 21.**

$$\begin{aligned}\overline{LUMI}^F &> \overline{LUMI}^{S,A} \\ \overline{OBLOT}^F &\perp \overline{LUMI}^{S,A} \\ \overline{FSTA}^F &\perp \overline{LUMI}^{S,A}.\end{aligned}$$

Proof. The above relations hold combining the previous lemmas and by transitivity:

- the strict dominance of \overline{LUMI}^F over $\overline{LUMI}^{S,A}$ straightforwardly derives from Lemma 20. In fact, **Angle-Shift** is solved in \overline{LUMI}^F but it is not solved in $\overline{LUMI}^{S,A}$;
- the orthogonality between \overline{OBLOT}^F over $\overline{LUMI}^{S,A}$ holds since **Angle-Shift** is solved in \overline{OBLOT}^F but not in $\overline{LUMI}^{S,A}$, and since **Triangle Round-Trip** is solved in $\overline{LUMI}^{S,A}$ but not in \overline{OBLOT}^F (by Lemma 8, Lemma 7);
- the orthogonality between \overline{FSTA}^F over $\overline{LUMI}^{S,A}$ holds since **Angle-Shift** is solved in \overline{FSTA}^F but not in $\overline{LUMI}^{S,A}$, and since **Newcomer Introducing** is solved in $\overline{LUMI}^{S,A}$ but not in \overline{FSTA}^F (by Lemma 14, Lemma 13). ◀

► **Theorem 26.**

$$\begin{aligned}\overline{OBLOT}^S &> \overline{OBLOT}^A \\ \overline{FSTA}^S &> \overline{FSTA}^A \\ \overline{FSTA}^A &\perp \overline{OBLOT}^S.\end{aligned}$$

Proof. The above relations hold combining the previous lemmas and by transitivity:

- for each $X \in \{\overline{OBLOT}, \overline{FSTA}\}$, \overline{X}^S strictly dominates \overline{X}^A since **Pseudo-Polygon** can be solved in \overline{X}^S but not in \overline{X}^A (by Lemma 25 and Lemma 24);
- the orthogonality between \overline{FSTA}^A and \overline{OBLOT}^S holds since **Pseudo-Polygon** is solved in \overline{OBLOT}^S but not in \overline{FSTA}^A , and since **Triangle Round-Trip** is solved in \overline{FSTA}^A but not in \overline{OBLOT}^S (by Lemma 8 and Lemma 7). ◀

Forming Large Patterns with Local Robots in the OBLOT Model

Christopher Hahn  

Universität Hamburg, Germany

Jonas Harbig  

Paderborn University, Germany

Peter Kling  

Universität Hamburg, Germany

Abstract

In the *arbitrary pattern formation* problem, n autonomous, mobile robots must form an arbitrary pattern $P \subseteq \mathbb{R}^2$. The (deterministic) robots are typically assumed to be indistinguishable, disoriented, and unable to communicate. An important distinction is whether robots have memory and/or a limited viewing range. Previous work managed to form P under a natural symmetry condition if robots have *no memory but an unlimited viewing range* [23] or if robots have a *limited viewing range but memory* [26]. In the latter case, P is only formed in a shrunk version that has constant diameter.

Without memory and with limited viewing range, forming arbitrary patterns remains an open problem. We provide a partial solution by showing that P can be formed under the same symmetry condition if the robots' initial diameter is ≤ 1 . Our protocol partitions P into rotation-symmetric components and exploits the initial mutual visibility to form one cluster per component. Using a careful placement of the clusters and their robots, we show that a cluster can move in a coordinated way through its component while “drawing” P by dropping one robot per pattern coordinate.

2012 ACM Subject Classification Theory of computation \rightarrow Self-organization

Keywords and phrases Swarm Algorithm, Swarm Robots, Distributed Algorithm, Pattern Formation, Limited Visibility, Oblivious

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.14

Related Version *Extended Version*: <https://arxiv.org/abs/2404.02771> [14]

Funding *Jonas Harbig*: This work was partially supported by the German Research Foundation (DFG) under the project number ME 872/14-1.

1 Introduction

Swarm robotics considers many, simple autonomous robots that must coordinate to reach a common goal. Applications include exploration and rescue missions in hazardous environments (like the deep sea or space [15]), medicine (for precise surgery or drug injection [19]), or biology (to model and understand the behavior of animal populations [21]). While the degree of necessary cooperation varies between applications, a central aspect is almost always the deployment of robots to a given set of coordinates.

Model & Problem. The mentioned deployment aspect motivates the *arbitrary pattern formation* problem, where a swarm of $n \in \mathbb{N}$ autonomous, mobile robots must form (in an arbitrary rotation and translation) a *pattern* $P \subseteq \mathbb{R}^2$ of $|P| = n$ *coordinates*. We assume the well-known *OBLOT* (*OBL*ivious *robOT*) model [10] for (deterministic) point robots in \mathbb{R}^2 with the following characteristics: Robots are *oblivious* (have no memory), *anonymous* (have no IDs), *homogeneous* (execute the same protocol), and *identical* (look the same). They are also *disoriented*, such that each robot perceives its surroundings in its own, local coordinate



© Christopher Hahn, Jonas Harbig, and Peter Kling;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 14; pp. 14:1–14:20

Leibniz International Proceedings in Informatics



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

system that might be arbitrarily rotated and translated compared to other robots (and even vary over time). A central feature of our work is that robots have a *limited viewing range*: they perceive their surroundings up to a constant distance. Without loss of generality, we normalize this viewing range to 1. We consider the *fully-synchronous* ($\mathcal{F}\text{SYNC}$) time model, where robots synchronously go through an LCM-cycle consisting of three phases: a LOOK- (observe surroundings), a COMPUTE- (calculate target), and a MOVE- (move to target) phase.

A key aspect that determines whether a pattern can be formed is its symmetry. For example, a swarm that starts as a perfectly regular n -gon cannot form an arrow (which is, intuitively, less symmetric): The robots may have identical local views and, thus, perform exactly the same computations and movements; the swarm would be forever trapped in a, possibly scaled, n -gon formation. One can measure the symmetry of a pattern P by its *symmetry* $\text{sym}(P)$. It counts how often P covers itself when rotated full circle around its center (see Definition 2.1). A swarm that starts with symmetry s can only form patterns whose symmetry is a multiple of s [20, 13]. This holds even for an *unlimited* viewing range and for robots *with memory*. In fact, for oblivious robots (still with unlimited viewing range), these are *exactly* the patterns that can be formed, even in an asynchronous setting [23, 13].

Under limited viewing range, the situation is more elusive. Robots *with memory* can form a *scaled* version of P under the above symmetry condition [26]. Basically, the robots first form a *near-gathering* (a formation in which robots have mutual visibility), use their memory to “maintain” the symmetry, and then apply the protocol from [13] to form a shrunk P that fits into the viewing range. In the $\mathcal{F}\text{SYNC}$ setting, this holds even for *non-rigid* moves (an adversary can stop robots during their move). On the negative side, oblivious robots with *non-rigid* moves cannot always form P , even if the symmetry condition holds [26].

It remains open whether the patterns that can be formed by oblivious robots with limited viewing range (and rigid movements) are also characterized by the symmetry condition.

Our Contribution. We make a decisive step towards characterizing patterns that *oblivious* robots with *limited viewing range* can form without *down-scaling*. Our main result is the following theorem (see Section 2 for formal definitions):

► **Theorem 1.1.** *A connected pattern P can be formed by $|P|$ oblivious OBLLOT robots with limited viewing range in the $\mathcal{F}\text{SYNC}$ model from a near-gathering I if and only if $\text{sym}(I) \mid \text{sym}(P)$. The formation takes $O(n)$ rounds, which is worst-case optimal.*

Starting from a near-gathering avoids another challenging open problem: Can we reach a near gathering from any connected formation *without increasing the symmetry*. If that were the case, together with Theorem 1.1 it would show that (under rigid movements) obliviousness and limited viewing range do not weaken the robots’ pattern formation abilities. Note that a recent near-gathering protocol for our model [4] avoids *collisions* (two robots moving to the same spot), a major cause of symmetry increase for most gathering protocols.

Requiring that P is connected (see Section 2) is natural for robots of limited viewing range. However, our technique can be adapted to form disconnected patterns, as long as they contain a connected component of size ≥ 3 (see Section 5 for a brief discussion).

In a nutshell, our protocol partitions the input pattern P into $\text{sym}(P)$ rotation-symmetric *components*. Using the initial mutual visibility, we let the robots form one cluster, called *drawing formation*, per component. Such a drawing formation relies on a careful placement of its contained robots to store information about the component it is responsible for and to coordinate its movement. We show how the drawing formation’s robots can compute and coordinately move along a deliberately constructed path through the component in order to “draw” the pattern by dropping one robot at each contained coordinate.

Further Related Work. The arbitrary pattern formation problem has been considered in numerous settings and variants. To name just a few, there are results for pattern formation

- on a grid [1],
- with obstructed view [2],
- with axis agreement [12],
- for robots without a common chirality [5],
- for pattern sequences [7], and
- in three dimensional space [25].

See [24] for a survey on pattern formation. A more recent and general overview of results and open problems in swarm robotics and related areas can be found in [9].

There is also work dedicated to forming a specific pattern like

- a point [8, 4, 3] (gathering),
- an arbitrarily tight near-gathering [6, 16] (convergence), and
- a uniform circle [22, 18, 11].

Again, a rather up-to-date and good overview can be found in [9].

Somewhat different in spirit but in our context relevant is [17]. The authors show how three or more robots with limited viewing range but *arbitrarily precise* sensors can form a *TuringMobile* to simulate a Turing machine that can, e.g., store and process real numbers. To showcase the model's power, the authors provide, amongst others, a pattern formation protocol for any dimension and an initially disconnected swarm, but under the strong requirement that (some) robots form initially a TuringMobile. Note that while we use robot placement to encode information, we do this in an inherently discrete way, requiring a sensor precision of order only $\min \{ 1/\sqrt{|P|}, \text{mindist}(P), 1/\text{sym}(P) \}$. A precision of order $1/\sqrt{|P|}$ is already required to measure distances in any near-gathering of $|P|$ robots. Similarly, to form P , robots must naturally be able to measure the minimal distance $\text{mindist}(P)$ that occurs in P . The final term stems from the fact that our drawing starts from a near-gathering of symmetricity $\text{sym}(P)$, in which robot distances are of order $O(1/\text{sym}(P))$. In [14], we add a discussion of the required measuring precision.

Outline. Section 2 introduces preliminaries like further notions and notation. Section 3 contains the major part of our protocol description and its analysis. That section formalizes notions like drawing formations or drawing paths and details how we coordinate the robots that form a drawing formation. At the section's end, we prove Theorem 1.1 under the assumptions that there is a drawing path that adheres to certain conditions (namely Definition 3.15) and that $\text{sym}(P) < |P|/2$. The construction of such a drawing path is subject of Section 4. We conclude with a small discussion and open problems in Section 5.

2 Preliminaries & Notation

This section extends the model and problem description from Section 1.

Geometric Notation. For two points $p, q \in \mathbb{R}^2$ we define $\text{dist}(p, q) = \|p - q\|_2$ as their Euclidean distance. We extend this notation in the natural way to sets $S \subseteq \mathbb{R}^2$, such that, e.g., $\text{dist}(p, S) = \min \{ \text{dist}(p, q) \mid q \in S \}$. We use a set-like notation for sequences $S = (s_i)_{i=1}^n$, like $p_1 \in S$, $S \subseteq \mathbb{R}^2$, or $\text{dist}(p, S)$. The minimal distance between two points in a set (or sequence) $S \subseteq \mathbb{R}^2$ is $\text{mindist}(S) := \min \{ \text{dist}(p, q) \mid p, q \in S, p \neq q \}$. For $p \in \mathbb{R}^2$ and $r \in \mathbb{R}$ the set $\mathcal{B}(p, r) = \{ q \in \mathbb{R}^2 \mid \text{dist}(p, q) < r \}$ denotes the open ball around p with radius r .

14:4 Forming Large Patterns with Local Robots in the OBLLOT Model

For a set $S \subseteq \mathbb{R}^2$ we write

- its power set as $\mathcal{P}(S)$,
- its closure as \bar{S} , and
- its boundary as $\partial S = \bar{S} \cap \overline{\mathbb{R}^2 \setminus S}$.

To highlight the usage of directions (in contrast to points), we use vector notation like $\vec{u}, \vec{v} \in \mathbb{R}^2$. Let $G_S = (S, E)$ with $E = \{ \{p, q\} \subseteq S \mid \text{dist}(p, q) \in (0, 1] \}$ be the *unit disc graph* of S . Then S is *connected* if G_S is connected and $p, q \in \mathbb{R}^2$ are *connected* if $\{p, q\}$ is connected. We use $\angle(\vec{u}, \vec{v}) \in (-\pi, \pi]$ for the signed angle between \vec{u} and \vec{v} . If not stated otherwise, explicit coordinates for points $p = (r, \phi) \in \mathbb{R}^2$ are given in *polar coordinates*.

Patterns & Configurations. Remember that we consider the *arbitrary pattern formation problem*: a swarm \mathcal{R} of $n := |\mathcal{R}| \in \mathbb{N}$ *OBLLOT* robots with a viewing range of 1 must form a target pattern $P \subseteq \mathbb{R}^2$ of $|P| = n$ coordinates. Since robots are oblivious, we use the standard assumption that, each round, they receive P as their sole input in an arbitrary but fixed coordinate system (i.e., robots receive the exact same numerical values).

Since robots are deterministic and indistinguishable, the *configuration* at any time is uniquely described by the robots' positions $\text{pos}(\mathcal{R}) = \{ \text{pos}(r) \mid r \in \mathcal{R} \} \subseteq \mathbb{R}^2$. If the robot identity is irrelevant for the matter at hand, we identify $r \in \mathcal{R}$ with the position $\text{pos}(r)$ and \mathcal{R} with the configuration $\text{pos}(\mathcal{R})$. A *near-gathering* is a configuration of diameter ≤ 1 .

W.l.o.g., we assume that P 's smallest enclosing circle is centered at the origin (otherwise, robots translate P accordingly). We measure P 's symmetry via its *symmetricity*:

► **Definition 2.1** (Symmetricity [13]). *Consider a set $P \subseteq \mathbb{R}^2$ whose smallest enclosing circle is centered at $c \in \mathbb{R}^2$. A m -regular partition of P is a partition of P into $k = |P|/m$ regular m -gons with common center c . The symmetricity of P is defined as $\text{sym}(P) := \max \{ m \in \mathbb{N} \mid \text{there is a } m\text{-regular partition of } P \}$.*

In Definition 2.1, a single point is considered a 1-gon with an arbitrary center. Thus, any P has a 1-regular partition. Note that, if the origin is an element of P , then $\text{sym}(P) = 1$.¹ At some places, we use the shorthand s_P for the symmetricity $\text{sym}(P)$ of a set P .

Symmetricity allows us to characterize patterns that can be formed by synchronous, oblivious robots with an unlimited viewing range:

► **Theorem 2.2** (Symmetry Condition, [13, Theorem 1]). *A pattern P can be formed by $|P|$ oblivious OBLLOT robots with unlimited viewing range in the FSYNC model from configuration I if and only if $\text{sym}(I) \mid \text{sym}(P)$.*

Further Time Models. Remember that we assume the *fully-synchronous* time model (FSYNC), where each *round* robots synchronously execute the LOOK-, COMPUTE-, and MOVE-phases of their LCM-cycle. Two other natural models are the *semi-synchronous* time model (SSYNC; a subset of robots is active each round and executes its phases synchronously) and the *asynchronous* time model (ASYNC; robots execute phases completely asynchronously).

3 Forming Patterns via Drawing

Given a pattern P of symmetricity $s_P \in \mathbb{N}$, we “draw” P using s_P *drawing formations*. Each drawing formation consists of a carefully arranged subset of *state robots* and is responsible to form one of s_P symmetric subpatterns $P' \subseteq P$. The state robots' careful placement enables

¹ One might assume a n -gon together with its center forms a rather symmetric set of size $n + 1$. But robots can easily break the perceived symmetry, since the center robot basically functions as a leader.

them to coordinately move through P' along a specific *drawing path*. While doing so, some state robots are *dropped* at nearby pattern coordinates to form P' .

We start in Section 3.1 by formalizing the idea of drawing formations and related concepts. Section 3.2 details the legal arrangements of state robots in a drawing formation and shows how we can use this to coordinate state robots. Equipped with this coordination, Section 3.3 formalizes the drawing path \mathbf{v} and what properties a drawing formation F must have in order to traverse \mathbf{v} while suitably dropping robots. Afterward, Section 3.4 shows how we can create s_P different drawing formations and use them to draw suitable, symmetric subpatterns of P . Finally, Section 3.5 puts everything together to prove Theorem 1.1. We often define algorithms implicitly during the proofs. To better illustrate the algorithms, we give high-level pseudocode in [14].

3.1 Drawing Formations & Movement

We first define the notion *drawing hull*, representing the general shape of a drawing formation.

► **Definition 3.1** (Drawing Hull). *A drawing hull $H = (a, \vec{d}, \phi, \Delta)$ consists of an anchor $a \in \mathbb{R}^2$, a direction $\vec{d} \in \mathbb{R}^2$ with $\|\vec{d}\|_2 = 1$, a span $\phi \in (0, \pi/3]$, and a diameter $\Delta \in (0, 1]$.*

As illustrated in Figure 1, one should think of a drawing hull $H = (a, \vec{d}, \phi, \Delta)$ as the point set $\{x \in \mathbb{R}^2 \mid \text{dist}(x, a) \leq \Delta \wedge \angle(\vec{d}, x - a) \in [0, \phi)\}$.² With this in mind, we sometimes abuse notation and identify H with this set to write, e.g., $\text{pos}(r) \in H$ for a robot r .

A *drawing formation* is defined by a drawing hull and all robots contained in it. These robots form a tight cluster whose exact placement inside the hull (the drawing formation’s *state*) allows us to coordinate their movement (see Section 3.2).

► **Definition 3.2** (Drawing Formation). *A drawing formation $F = (H_F, \mathcal{R}_F)$ consists of a drawing hull H_F and the robot set $\mathcal{R}_F := \{r \in \mathcal{R} \mid \text{pos}(r) \in H_F\}$. We call $r \in \mathcal{R}_F$ a state robot of F and $\mathcal{S}_F := \text{pos}(\mathcal{R}_F)$ the state of F . The size of F is $|\mathcal{R}_F|$.*

We sometimes identify a drawing formation with its hull, allowing us to, e.g., speak of a drawing formation’s anchor or diameter.

A drawing formation F forms a given pattern by “moving” F along a specific *drawing path* (see Section 3.3) that visits all pattern coordinates, dropping one state robot per pattern coordinate along the way. The following definition formalizes such *moves* (see Figure 3 for an illustration).

► **Definition 3.3** (Move). *Consider a drawing formation $F = (H_F, \mathcal{R}_F)$ with drawing hull $H_F = (p, \vec{d}, \phi)$ in configuration \mathcal{R} . Let \mathcal{R}' denote the configuration after the next LCM cycle. We say F moves from p (in configuration \mathcal{R}) to p' (in configuration \mathcal{R}') if a state robot subset $\mathcal{R}_{F'} \subseteq \mathcal{R}_F$ of F forms a drawing formation $F' = ((p', \vec{d}, \phi), \mathcal{R}_{F'})$ in configuration \mathcal{R}' . We call the robots $\mathcal{R}_F \setminus \mathcal{R}_{F'}$ dropped robots.*

When moving from one drawing path vertex to the next, the remaining state robots change state (their placement in the drawing formation) to encode the progress on the drawing path. To ensure that a drawing formation can adopt any (reasonable) state after a movement, we restrict its movement distance to $1 - \Delta$ (s.t. each state robot can reach any other location in the resulting drawing formation of diameter Δ).

² Note that $\phi \leq \pi/3$ ensures that Δ is indeed the diameter of the point set H .

► **Observation 3.4.** *Consider a drawing formation F of diameter Δ that moves from position p to p' . If $\text{dist}(p, p') \leq 1 - \Delta$, the robots that are not dropped can form any state in the resulting drawing formation.*

3.2 States of a Drawing Formation

Given a target pattern P , our protocol considers only drawing formations F with fixed span $\phi = 2\pi/\text{sym}(P)$ (depending only on P) and fixed diameter Δ (constant). Moving F between vertices of the drawing path (see Section 3.3) requires a coordinated movement of F 's state robots. To achieve this, any robot must

1. decide whether it is one of F 's state robots and, if so,
2. know the current progress on the drawing path.

To achieve (1), we use a careful placement of three *defining robots* that allows any robot r that sees them to deduce the remaining hull parameters (anchor and direction); once all four hull parameters are known, r can compute the hull H_F and decide whether it lies inside H_F or not. To achieve (2), we require that any additional state robots are placed on an ϵ -grid ($\epsilon > 0$ fixed, depending only on P) that is aligned with the defining robots; using an arbitrary but fixed enumeration scheme for ℓ robots on such a grid, all state robots can agree on the same ordering of states and use it (in combination with F 's size ℓ) to encode the progress on the drawing path.

Legal States. We continue to formalize this idea for a given parameter $\epsilon > 0$. The placement of the defining robots r_1 , r_2 , and r_3 of a drawing formation F with anchor a and direction \vec{d} is as follows:

1. r_1 is at the anchor a ,
2. r_2 is at distance ϵ in direction \vec{d} from anchor a , and
3. r_3 is at distance $\in \{2\epsilon, 4\epsilon, \dots\}$ in direction \vec{d} from r_2 .

Further state robots (if any) must be placed on the non-negative 2ϵ -grid with origin r_2 and whose x -axis is aligned with \vec{d} (see Figure 2).

The robot pair $\{r_1, r_2\}$ can be identified since they are the only state robots with distance ϵ . And since r_3 at distance $\geq 2\epsilon$ is closer to r_2 than to r_1 , robots can distinguish r_1 from r_2 , from which they can infer both the hull's anchor and direction.

We get the following set of potential state robot locations:

► **Definition 3.5** (ϵ -Granular Locations). *Consider a drawing formation $F = (H_F, \mathcal{R}_F)$ with anchor a and direction \vec{d} . Let \vec{d}_\perp be the unit vector with $\angle(\vec{d}, \vec{d}_\perp) = \pi/2$. The set of ϵ -granular locations of F is*

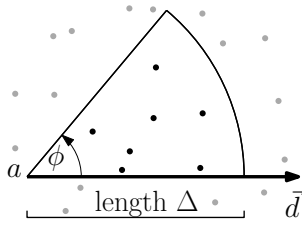
$$L_F(\epsilon) := \{a, a + (1 + 2i)\epsilon \cdot \vec{d} + 2j\epsilon \cdot \vec{d}_\perp \mid i, j \in \mathbb{N}_0\} \cap H_F. \quad (1)$$

States (i.e., state robot placements) considered legal by our protocol consist of all possible placements on ϵ -granular locations with the mentioned restrictions on the three defining robots' positions.

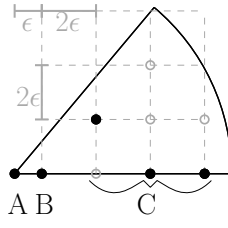
► **Definition 3.6** (ϵ -Granular States). *Consider a drawing formation F with anchor a and direction \vec{d} . The set of ϵ -granular states of F is*

$$A_F(\epsilon) := \{\mathcal{S} \cup \mathcal{T} \mid \mathcal{S} \in \mathcal{P}(L_F), \mathcal{T} \in T(\epsilon)\}.$$

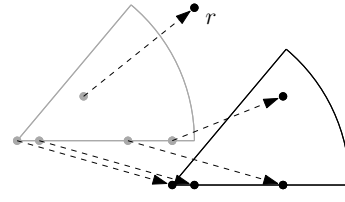
with $T(\epsilon) := \bigcup_{i=1}^{\lfloor (\Delta/\epsilon - 1)/2 \rfloor} \{a, a + \epsilon \cdot \vec{d}, a + (1 + 2i)\epsilon \cdot \vec{d}\}$ being the sets of defining robots. For $\ell \in \mathbb{N}$ we define $A_F^\ell(\epsilon) := \{\mathcal{S} \in A_F(\epsilon) \mid |\mathcal{S}| = \ell\}$ as the set of all ϵ -granular states of F that can be adopted with ℓ state robots.



■ **Figure 1** Drawing formation F with drawing hull H_F and state robots $r \in \mathcal{R}_F$ (black) and other robots $r \notin \mathcal{R}_F$ (gray).



■ **Figure 2** ϵ -granular drawing formation; locations A and B must each contain a robot, the locations at C must in sum contain ≥ 1 robots.



■ **Figure 3** Movement of a drawing formation where robot r is dropped.

Our protocol considers only drawing formations that adhere to the above restrictions, leading us to the following definition:

► **Definition 3.7** (ϵ -Granular Drawing Formation). *A drawing formation F is ϵ -granular if F is in an ϵ -granular state and if F 's state robots know³ the fixed parameters ϵ , Δ , and ϕ .*

Note that all subsets of robots in the current configuration that fulfill the definition above are ϵ -granular drawing formations. We require that $r \in F$ knows the parameter ϵ , Δ and ϕ of F . Therefore $r \in F$ can check all subsets in its viewing range whether they are drawing formations with these parameters. With a viewing range of $1 \geq \Delta$, r observes all robots in F and can compute the drawing hull of F .

► **Observation 3.8.** *Let F be a ϵ -granular drawing formation. All state robots in F can compute the anchor of F and \vec{d} .*

As a final property, we only want non-overlapping drawing formations. If two drawing hulls were overlapping, a robot might be state robot in two drawing formations that move in different directions.

► **Definition 3.9** (Validity). *An ϵ -granular drawing formation $F = (H_F, \mathcal{R}_F)$ is valid in configuration $\mathcal{R} \supseteq \mathcal{R}_F$ if for any other ϵ -granular drawing formation $F' = (H_{F'}, \mathcal{R}_{F'})$ we have $H_F \cap H_{F'} = \emptyset$.*

Counting via States. Given an ϵ -granular drawing formation F of size ℓ (i.e., consisting of ℓ state robots), we can easily enumerate $A_F^\ell(\epsilon)$ in a way that depends solely on the relative positions of its locations. In particular, all state robots can use this enumeration and thus agree on the order of states, which basically equips the state robots with a shared counter. A concrete implementation is depicted in [14].

► **Definition 3.10** (i -th State). *Using an arbitrary, unique state enumeration on $A_F^\ell(\epsilon)$ that depends solely on the relative position of locations, for $i \in \{1, 2, \dots, |A_F^\ell(\epsilon)|\}$ we define the i -th state of F as the corresponding state in this enumeration.*

We conclude with a lower bound on the number of states that an ϵ -granular drawing formation may have.

³ The parameters are either hard-coded into the protocol or can be computed from the target pattern P .

► **Lemma 3.11.** *Consider a drawing formation F with hull diameter Δ and span $\phi \in (0, \pi/3]$. We have $|A_F^3(\epsilon)| = \lfloor (\Delta/\epsilon - 1)/2 \rfloor$ and $|A_F^\ell(\epsilon)| = \Omega(\Delta^3 \cdot \phi/\epsilon^3)$ for $4 \leq \ell \leq |L_F(\epsilon)| - 1$.*

Proof. For $\ell = 3$ robots, only the third defining robot r_3 can choose between multiple locations; the first and second defining robots r_1 and r_2 have a fixed location inside H_F . Since r_3 can be placed on any of the locations of the form $a + (1 + 2i)\epsilon \cdot \vec{d}$ for $i \in \mathbb{N}$ that lies in H_F , we get $|A_F^3| = \lfloor (\Delta/\epsilon - 1)/2 \rfloor$.

For $\ell = 4$, observe that there are $k := |L_F(\epsilon)| = \Omega(\Delta^2 \cdot \phi/\epsilon^2)$ ϵ -granular locations in F , since the 2ϵ -grid allows for $\Omega(1/\epsilon^2)$ many locations per unit area and the total area covered by F 's hull is $\pi \cdot \Delta^2 \cdot \phi/(2\pi) = \Delta^2 \cdot \phi/2$. Again, the first two defining robots have a fixed location, while the third defining robot may occupy one of $\Omega(\Delta/\epsilon)$ many locations. The remaining $\ell - 3 \geq 1$ robots can be arranged on the remaining $k - 3$ locations in $\binom{k-3}{\ell-3}$ ways. By the lemma's restriction on ℓ we have $\ell - 3 \geq 4$ and $\ell - 3 \leq k - 4$, such that $\binom{k-3}{\ell-3} \geq \binom{k-3}{k-4} = \binom{k-3}{1} = \Omega(\Delta^2 \cdot \phi/\epsilon^2)$. Together, we get the desired bound. ◀

3.3 Drawing a Pattern via a Drawing Path

This section introduces the *drawing path* of a (sub-) pattern P , which is a path in \mathbb{R}^2 that visits all pattern coordinates. This path should allow a drawing formation to move along its vertices while dropping one state robot per pattern coordinate along the way to form P . An instructive illustration of the idea can be found in [14].

A drawing path \mathbf{v} has a parameter δ that controls the maximal distance between consecutive vertices as well as between each pattern coordinate and the path. Moreover, \mathbf{v} must depend only on P , such that oblivious robots can all recalculate \mathbf{v} each LCM-cycle.

► **Definition 3.12 (Drawing Path).** *Consider any pattern P . A path $\mathbf{v} = (v_j)_{j=1}^k$ of k vertices $v_j \in \mathbb{R}^2$ is a δ -drawing path of P if*

1. \mathbf{v} can be calculated from P ,
2. $\forall p \in P: \text{dist}(p, \mathbf{v}) \leq 1 - \delta$, and
3. $\forall j \in \{1, 2, \dots, k-1\}: \text{dist}(v_j, v_{j+1}) \leq 1 - \delta$.

Choosing δ equal to the diameter of a drawing formation F enables F to traverse \mathbf{v} while forming any state (Observation 3.4). We omit δ if it is irrelevant for the matter at hand.

When traversing \mathbf{v} , we want a drawing formation to drop a robot at pattern coordinate $p \in P$ when leaving the latest vertex v_j that is close enough to p . This ensures that, at any time after being dropped, the dropped robot has a distance of at least $1 - \delta$ to the drawing formation that dropped it. We say p is *covered* by vertex v_j .

► **Definition 3.13 (Covered Coordinates).** *Consider a δ -drawing path $\mathbf{v} = (v_j)_{j=1}^k$ of a pattern P . Coordinate $p \in P$ is covered by vertex v_j if $j \in \{1, 2, \dots, k\}$ is the maximal index for which $\text{dist}(p, v_j) \leq 1 - \delta$. Let $\text{cov}(v_j)$ denote the set of all coordinates covered by v_j .*

We extend $\text{cov}(\bullet)$ in the natural way to subsequence \mathbf{v}' of \mathbf{v} , such that $\text{cov}(\mathbf{v}') = \bigcup_{u \in \mathbf{v}'} \text{cov}(u)$.

Care must be taken once a drawing formation dropped so many robots that it reached size $\ell = 3$: It must not drop further robots before the path's end, since the remaining two robots would no longer form a drawing formation and could not coordinate (see Section 3.2). We capture this (possibly non-existent) path region in the following Definition 3.14.

► **Definition 3.14 (Tail of a Drawing Path).** *The tail $\text{tail}(\mathbf{v})$ of a drawing path $\mathbf{v} = (v_j)_{j=1}^k$ is the longest suffix $(v_j)_{j=s}^k$ s.t. $\sum_{j=s}^k |\text{cov}(v_j)| < 4$.*

As a final notion, we declare when a drawing formation and a drawing path are *compatible* (i.e., can be used to form the path's pattern). Here, we use $\text{hops}(v_s, v_t)$ to denote the number of edges between two path vertices $v_s, v_t \in \mathbf{v}$.

► **Definition 3.15 (Compatibility).** *An ϵ -granular drawing formation F with diameter Δ and span Φ is compatible with a δ -drawing path $\mathbf{v} = (v_j)_{j=1}^k$ of a pattern P if*

1. $\epsilon < \text{mindist}(P)$ and $\Delta \leq \delta$,
2. $\forall s < t$ s.t. $\bigcup_{j=s}^{t-1} \text{cov}(v_j) = \emptyset$: $\text{hops}(v_s, v_t) \leq |A_F^4(\epsilon)|$,
3. $|\text{tail}(\mathbf{v})| \leq |A_F^3(\epsilon)|$, and
4. $|\text{cov}(\text{tail}(\mathbf{v}))| = 3$ and $\text{cov}(\text{tail}(\mathbf{v})) \subseteq \mathcal{B}(v_k, 1)$.

Property 1 ensures that the distance ϵ (identifying F 's defining robots) does not occur in P and that F can traverse \mathbf{v} (by Observation 3.4). Property 2 requires that, after dropping a robot, the state space $A_F^\ell \supseteq A_F^4$ of the remaining ℓ robots is large enough to encode the progress towards the next vertex where a robot is dropped. These two properties are used in Lemma 3.16 to prove that F can traverse the non-tail of \mathbf{v} while appropriately dropping robots.

Lemma 3.17 uses Property 3 to traverse the tail and Property 4 to drop the final three robots at the tail's end. This final drop is slightly more involved: if the last three coordinates form, e.g., a straight path of edge length 1, our drawing formation cannot drop all robots at once. With the help of Property 4, we handle this via an intermediate step.

► **Lemma 3.16.** *Consider a compatible drawing path $\mathbf{v} = (v_j)_{j=1}^k$ of a pattern P . Let \mathcal{R} be the configuration formed by a drawing formation F of size $|P|$ in state 1 anchored in v_1 that is compatible with \mathbf{v} . Then F can traverse \mathbf{v} by taking one edge per LCM-cycle while dropping one robot at each coordinate in $\text{cov}(v_j)$ when leaving $v_j \notin \text{tail}(\mathbf{v})$.*

Proof.

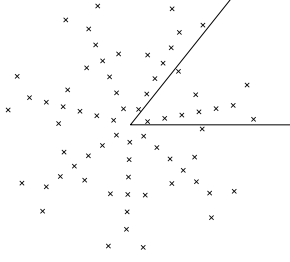
Enumeration of States. We defined in Definition 3.10 an enumeration of A_F^ℓ , let the i -th state of A_F^ℓ be $\text{state}(i, \ell)$. We define the following unique states for the path, using an enumeration that includes different sizes ℓ of drawing formations fitting to the number of not dropped robots at a node.

$$\begin{aligned} f(v_i) &:= |\text{cov}((v_j)_{j=i}^k)| \\ g(v_i) &:= \max(|(v_j)_{j < i}^{i-1}|) \text{ with } \text{cov}((v_j)_{j < i}^{i-1}) = \emptyset \\ \text{state}(v_i) &:= \text{state}(g(v_i) + 1, f(v_i)) \end{aligned}$$

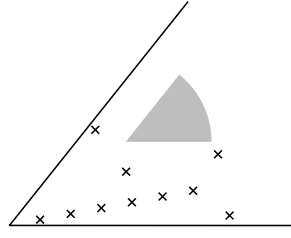
From (3) and (4) of Definition 3.15 follows directly, that all such states exist.

Induction Proof. Assumption: F with anchor on v_i with $|F| = |\text{cov}((v_j)_{j=i}^k)|$ in state i and dropped robots on $\text{cov}((v_j)_{j=1}^{i-1})$. Let $H_F = (a, \vec{d}, \phi, \Delta)$ be the drawing hull of F (Definition 3.1). We define the coordinate system \mathcal{S} as the coordinate system with x direction \vec{d} and origin at v_1 . Start: At node v_1 this is initially given.

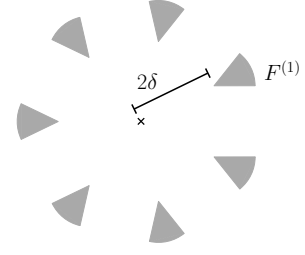
Step: No coordinates $p \in \mathcal{B}(v_i, 1 - \delta)$ contain robots $r \notin F$, otherwise $p \in \text{cov}(v_j) \cap \mathcal{B}(v_i, 1 - \delta)$, $j < i$ with is a contradiction to Definition 3.13. Therefore F is valid (Definition 3.9). $r \in F$ knows anchor a and direction vector \vec{d} of F (see Observation 3.8) and F is unambiguous because it is valid (i.e. $r \in F$ is not in another ϵ -granular drawing formation). Because \mathbf{v} is a drawing path (Definition 3.12), r can compute \mathbf{v} from P . With the assumption that a (the anchor of F) is at v_i and F in state $\text{state}(v_i)$, it can determine the coordinate system \mathcal{S} . $\text{dist}(v_i, v_{i+1}) \leq 1 - \delta$ (by Definition 3.12) and $1 - \delta \leq 1 - \Delta$ (by Definition 3.15). From Observation 3.4 follows that F can move from v_i to v_{i+1} . F moves such that its new anchor is v_{i+1} , $|\text{cov}(v_i)|$ robots are dropped onto the coordinates $\text{cov}(v_i)$ and its new state $\text{state}(v_{i+1})$. ◀



■ **Figure 4** Pattern P with symmetricity $s_P = 7$ and cone $C^{(1)}$.



■ **Figure 5** Symmetric component $P^{(1)}$ with an aligned drawing formation $F^{(1)}$.



■ **Figure 6** Initial drawing pattern \mathcal{I} for $s_P = 7$.

► **Lemma 3.17.** Consider a compatible drawing path $\mathbf{v} = (v_j)_{j=1}^k$ of a pattern P . Let \mathcal{R} be the configuration that has

1. one robot at each coordinate in $P \setminus \text{cov}(\text{tail}(\mathbf{v}))$ and
 2. a drawing formation F of size 3 in state $|\text{tail}(\mathbf{v})|$ anchored in v_k that is compatible with \mathbf{v} .
- Then F can dissolve within two LCM-cycles while dropping one robot at each coordinate in $\text{cov}(\text{tail}(\mathbf{v}))$.

The proof of Lemma 3.17 is given in Appendix B.

3.4 Full Pattern via Many Drawing Formations

As shown in Section 3.3, we can draw any pattern P if we start in a suitable drawing formation F (and have a compatible drawing path). But we must first form such a drawing formation from the initial near-gathering, which might have a symmetricity $s > 1$. In that case, since any drawing formation has symmetricity 1, we cannot form F (by Theorem 2.2). Instead, we show how to form $\text{sym}(P)$ symmetric copies of F that are placed such that they

1. have symmetricity $\text{sym}(P)$ (we have $s \mid \text{sym}(P)$ or we cannot form P , even globally) and
2. do not interfere with each other (if using suitable drawing paths, see Section 4).

We start by partitioning the pattern P of symmetricity $s_p := \text{sym}(P)$ into s_P symmetric components, each of which will be drawn by its own drawing formation.

► **Definition 3.18 (Cone & Symmetric Component).** Let $\vec{e}_x := (1, 0)$. For a pattern P of symmetricity s_P , define the i -th cone

$$C^{(i)} := \{p \in \mathbb{R}^2 \mid \angle(\vec{e}_x, p) \in [(i-1) \cdot 2\pi/s_P, i \cdot 2\pi/s_P)\} \quad (2)$$

and the i -th symmetric component $P^{(i)} := P \cap C^{(i)}$.

Note that the symmetric components are pairwise disjoint and that $P = \bigcup_{i=1}^{s_P} P^{(i)}$. See Figure 4 for an illustration.

To form pattern P , we first form a suitable *initial drawing pattern* of diameter ≤ 1 . This initial drawing pattern places each robot $r \in \mathcal{R}$ in one of s_P ϵ -granular drawing formations $F^{(i)}$ of size $|P|/s_P$ in state 1. If there exists a drawing path $\mathbf{v}^{(1)}$ for $P^{(1)}$ that is compatible with $F^{(1)}$ and starts at the anchor of $F^{(1)}$, we immediately get a corresponding (rotation-symmetric) drawing path for each $F^{(i)}$. Assuming that, additionally, those drawing paths lie “sufficiently inside” their respective cone $C^{(i)}$, we will prove a generalization of Lemmas 3.16 and 3.17, basically showing that the different $F^{(i)}$ can draw their $P^{(i)}$ without interfering with each other. The existence of suitable drawing paths is shown in Section 4.

To enable the robots to deduce the symmetric component $P^{(i)}$ they are drawing⁴, we require $F^{(i)}$ to be *aligned* with $P^{(i)}$:

► **Definition 3.19.** Fix $i \in \{1, 2, \dots, s_P\}$ and the i -th symmetric component $P^{(i)}$ of a pattern P with symmetry s_P . Let F be a drawing formation with direction \vec{d} and span ϕ . Then F is aligned with $P^{(i)}$ if $\angle(\vec{d}, \vec{e}_x) = (i-1) \cdot 2\pi/s_P$ and if $\phi = \min\{2\pi/s_P, \pi/3\}$.

Figure 5 gives an illustration. With this, we define the *initial drawing pattern* of a pattern P and a suitable drawing formation F as follows (illustrated in Figure 6):

► **Definition 3.20 (Initial Drawing Pattern).** Fix a pattern P of symmetry s_P . An initial drawing pattern \mathcal{I} for P is a configuration of $|P|$ robots that consists of s_P drawing formations $\{F^{(i)}\}_{i=1}^{s_P}$ of diameter $\Delta \leq 1/6$ in state 1 such that:

1. $F^{(1)}$ is aligned with $P^{(1)}$ and anchored in $(2\Delta, \pi/s_P)$.
2. $F^{(i)}$ is a rotation of $F^{(1)}$ by $(i-1) \cdot 2\pi/s_P$.

We say \mathcal{I} is ϵ -granular if the $F^{(i)}$ are ϵ -granular.

Note that, by construction, each $F^{(i)}$ is aligned with $P^{(i)}$. Moreover, \mathcal{I} is a near-gathering configuration (i.e., has diameter ≤ 1) and has symmetry s_P , such that we can form \mathcal{I} from any near-gathering for which the symmetry condition (Theorem 2.2) holds.

It remains to prove that once the initial drawing pattern is formed, each drawing formation $F^{(i)}$ forms its symmetric component $P^{(i)}$ and does not interfere with the operation of any other drawing formation $F^{(j)}$ with $j \neq i$.

► **Lemma 3.21.** Assume the current configuration is an ϵ -granular initial drawing pattern \mathcal{I} for a pattern P of symmetry s_P . Consider a drawing path $\mathbf{v}^{(1)} = (v_j^{(1)})_{j=1}^k$ of symmetric component $P^{(1)}$ that is compatible with $F^{(1)}$ such that

1. the path $\mathbf{v}^{(1)}$ starts in the anchor of $F^{(1)}$,
2. $\mathbf{v}^{(1)}$ lies in the first cone (i.e., $\mathbf{v}^{(1)} \subseteq C^{(1)}$) and

$$\text{dist}(\mathbf{v}^{(1)}, \partial C^{(1)}) > \max\{\epsilon, \Delta \cdot \sin(2\pi/s_P)\}. \quad (3)$$

Then P can be formed in $O(k)$ many LCM-cycles.

Proof. From Lemmas 3.16 and 3.17 we know, $P^{(1)}$ can be formed by $F^{(1)}$ assuming $P^{(1)}$ is the whole pattern. The traversal of this path takes at most k rounds to reach $v_k^{(1)}$ plus 2 rounds for dropping the last robots. With the other symmetric components $P^{(i)}$ next to $P^{(1)}$, $F^{(1)}$ can still form $P^{(1)}$ if $F^{(1)}$ is always valid. We will prove in the following, that $F^{(1)}$ is always valid.

Validity. $F^{(1)} = (\mathcal{R}_F, H_F)$ is valid, if there exists no subset $\mathcal{R}_G \subseteq \mathcal{R}$ which fulfills the criteria of Definition 3.7 such that $G = (\mathcal{R}_G, H_G)$ a ϵ -granular drawing formation $\mathcal{R}_G \neq \mathcal{R}_F$ and $H_F \cap H_G \neq \emptyset$. $F^{(1)}$ is aligned with $C^{(1)}$ (Definition 3.19). It follows directly, that $F^{(1)} \subseteq C^{(1)}$ and naturally for all symmetric drawing formation $F^{(i)}$ that $F^{(i)} \subseteq C^{(i)}$. Therefore, those drawing formations have disjoint hulls. From the proof of Lemma 3.16 we know that all dropped robots on $p \in P^{(1)}$ have a distance $\geq \Delta$ to H_F , therefore those robots cannot be part of \mathcal{R}_G . It is left to show, that $r \in C^{(i)}$ with $i \neq 1$ cannot build a drawing formation G that intersects H_F .

⁴ Since robots are disoriented, they cannot deduce which $P^{(i)}$ they are drawing. But they can deduce $P^{(i)}$'s coordinates in their own, local coordinate systems.

14:12 Forming Large Patterns with Local Robots in the OBLLOT Model

Two dropped robots r_1, r_2 on $p_1, p_2 \in P$ have $\text{dist}(r_1, r_2) > \epsilon$, this follows from the fact that $\epsilon \leq \text{mindist}(P)$ (Property 1 of Definition 3.15). Let F be anchored on $v_j^{(1)}$. By prerequisite (1) $\text{dist}(v_j^{(1)}, \partial C^{(1)}) > \epsilon$. Because $F^{(1)}$ is aligned to $C^{(1)}$, $\text{dist}(\partial C^{(1)}, \mathcal{R}_F) > \epsilon$. Therefore $r_1 \in F^{(1)}$ and $r_2 \notin F^{(1)}$ have $\text{dist}(r_1, r_2) > \epsilon$. Therefore, drawing formation G must have a pair of $\text{dist}(r_1, r_2) > \epsilon$ which are part of the same drawing formation $F^{(i)}$. There must exist a third robot $r_3 \notin F^{(i)}$ collinear to r_1, r_2 with $\text{dist}(r_3, \{r_1, r_2\}) \leq \Delta$. W.l.o.g $i = 1$. Let $H_F = (a, \vec{d}, \Phi, \Delta)$ be the drawing hull of $F^{(1)}$. Because $F^{(1)}$ is aligned to $C^{(1)}$ (Definition 3.19) \vec{d} is parallel to one side of its boundary and the line segment $\text{line}(a, -\vec{d}, \Delta)$ can never cut this side. $\text{line}(a, -\vec{d}, \Delta - \epsilon)$ cuts the other side of $C^{(1)}$ boundary with angle Φ . The length of $\text{line}(a, -\vec{d}, \Delta - \epsilon)$ must be $\geq \frac{\text{dist}(\partial C^{(1)})}{\sin(\Phi)}$. The line segment has a length of Δ . $\Phi = \frac{2\pi}{s_P}$ (Definition 3.19) $\text{dist}(a, \partial C^{(1)}) \geq \Delta \cdot \sin\left(\frac{2\pi}{s_P}\right)$ (assumption (2) of this lemma). This resolves to $\Delta - \epsilon \geq \Delta \cdot \sin\left(\frac{2\pi}{s_P}\right) / \sin\left(\frac{2\pi}{s_P}\right) = \Delta$, which is obviously a contradiction. Therefore, $\text{line}(a, -\vec{d}, \Delta - \epsilon)$ is completely in $C^{(1)}$ and cannot contain $r_3 \notin F^{(1)}$. ◀

3.5 Putting Everything Together

Section 3.4 showed how to form a pattern P assuming we start in a suitable initial drawing pattern \mathcal{I} and if a compatible drawing path $\mathbf{v}^{(1)}$ for $P^{(1)}$ exists. We continue by showing the existence of such a path for patterns with symmetricity $s_P := \text{sym}(P) < |P|/2$ (Lemma 3.22, proven in Section 4 and Appendix A); patterns of larger symmetricity can be handled without drawing formations (Lemma 3.23). Afterward, in Lemma 3.24, we prove that each robot can distinguish in which phase of our protocol it is:

- (i) forming \mathcal{I} ,
- (ii) being part of a valid ϵ -granular drawing formation F ,
- (iii) dropping the last three robots at the tail's end, or
- (iv) having been dropped at a pattern coordinate.

Putting everything together, we conclude this section with the proof of Theorem 1.1.

► **Lemma 3.22.** *Consider the ϵ -granular drawing formation $F^{(1)}$ of the initial drawing pattern for a connected pattern P of symmetricity $s_P < |P|/2$. The parameter ϵ can be chosen such that $F^{(1)}$ has $|L_{F^{(1)}}(\epsilon)| \geq 2 + |P^{(1)}|$ ϵ -granular locations. Moreover, there exists a drawing path $\mathbf{v}^{(1)}$ of symmetric component $P^{(1)}$ that is compatible with $F^{(1)}$ such that*

1. the path $\mathbf{v}^{(1)}$ starts in the anchor $a = (2\Delta, \pi/s_P)$ of $F^{(1)}$,
2. $\mathbf{v}^{(1)}$ lies in the first cone (i.e., $\mathbf{v}^{(1)} \subseteq C^{(1)}$) and

$$\text{dist}\left(\mathbf{v}^{(1)}, \partial C^{(1)}\right) > \max\{\epsilon, \Delta \cdot \sin(2\pi/s_P)\}. \quad (4)$$

► **Lemma 3.23.** *Consider a pattern P with symmetricity $s_P \geq |P|/2$. P can be formed.*

With a separate algorithm, we can form patterns with symmetricity n or $n/2$. The robots form P scaled down to diameter 1. Then, the robots scale the small pattern back to its original (large) size. Because of the high symmetricity, the scaling can be performed locally. The proof can be found in Appendix C.

► **Lemma 3.24.** *Let $r \in \mathcal{R}$ be a robot in a configuration described in Lemma 3.22 executing the protocol from Lemma 3.21. Then r can locally distinguish between the following situations:*

- (i) r is in an initial configuration before the initial drawing pattern is formed
- (ii) $r \in H_F$ of a valid ϵ -granular drawing formation F
- (iii) $r \in F_{inter}$ (see Definition B.1)⁵
- (iv) r has been dropped from a drawing formation

⁵ F_{inter} is a slight alteration of the ϵ -granular drawing formation used in the very last step of the execution.

Proof.

- (i) If a robot $r \in \mathcal{R}$ sees $|P|$ robots, it is in a near-gathering and observes all robots \mathcal{R} . In that case, r checks whether \mathcal{R} equals the initial drawing pattern \mathcal{I} (Definition 3.20) or any of the execution steps resulting from the protocol described in Lemma 3.21 that are still a near-gathering ($\Theta(1)$ many). If not, r can conclude that it is in the initial configuration.
- (ii) Is clear by Observation 3.8.
- (iii) Is shown in Lemma B.3.
- (iv) When a robot is dropped from a drawing formation it is on $p \in P$. We know, that $\text{mindist}(P) > \epsilon$, therefore dropped robots can never form an ϵ -granular drawing formation. Moreover, in Lemma 3.21 we have shown that all ϵ -granular drawing formations in the configuration are valid. Therefore, no robot $r' \in H_F$ can be part of another ϵ -granular drawing formation F' . Similar arguments are true for F_{inter} . Hence, a robot observing that it is not in one of the first three situations knows that it has been dropped. \blacktriangleleft

► **Theorem 1.1 (restated).** *A connected pattern P can be formed by $|P|$ oblivious OBLLOT robots with limited viewing range in the FSYNC model from a near-gathering I if and only if $\text{sym}(I) \mid \text{sym}(P)$. The formation takes $O(n)$ rounds, which is worst-case optimal.*

Proof. The first direction follows from Theorem 2.2, since a pattern where $s_I := \text{sym}(I)$ does not divide $s_P := \text{sym}(P)$ cannot be formed.

For the second direction, assume $s_I \mid s_P$. By Lemma 3.23 the pattern can be formed if $s_P \geq |P|/2$, so assume $s_P < |P|/2$. Then we must execute the protocol described in Lemma 3.21, whose prerequisites (esp. the existence of a suitable drawing path) can be fulfilled: By Lemma 3.24 we know, that a robot can locally decide between the phases necessary to start and execute the protocol. If \mathcal{R} is in an initial near-gathering before forming the initial drawing pattern \mathcal{I} , the robots collectively form \mathcal{I} (which has symmetricity s_P and, thus, can be formed by Theorem 2.2). From Lemma 3.22's guarantee on $|L_{F^{(1)}}(\epsilon)|$ we get that the drawing formation has enough locations for the number of robots as well as the existence of a suitable drawing path. Thus, we can apply Lemma 3.21 to get that one robot is dropped at each $p \in P$ after at most $O(|P|)$ rounds. By Lemma 3.24, robots can realize that they have been dropped and remain idle on their respective pattern coordinate. \blacktriangleleft

4 Existence of Suitable Drawing Paths

In the previous section, we showed that a drawing formation $F^{(i)}$ which is placed in $C^{(i)}$ can traverse a drawing path v of $P^{(i)}$ if $F^{(i)}$ is compatible to $v^{(i)}$. We omitted all details on how we can create such a drawing path. In this section, we will construct a path and prove that it fulfills all required properties of Lemma 3.22. The final proof of Lemma 3.22 can be found in Appendix A.

Outline. For a symmetric component $P^{(1)}$, we define a tree $T^{(1)}$ with $O(n)$ nodes such that its nodes cover all points of $P^{(1)}$. The tree's root node will be $(2\Delta, \pi/s_P)$, the initial position of the drawing formation $F^{(1)}$ aligned to $P^{(1)}$. It is clear that a simple traversal of $T^{(1)}$ will fulfill the requirement (1) and (4) of compatibility. To additionally fulfill (2) and (3), we construct a tail that fits the requirements and append it to the traversal. To prove that such

14:14 Forming Large Patterns with Local Robots in the OBLLOT Model

a tail always exists we show, that it is always possible to rotate P s.t. $P^{(1)}$ contains ≥ 3 connected positions.⁶ We use these three positions to append a tail to $T^{(1)}$ that fulfills the requirement of compatibility.

4.1 Tree Construction

► **Definition 4.1** (Drawing-Tree). *Let $P^{(1)}$ be a symmetric component of P . We call $T^{(1)}$ constructed with algorithm Algorithm 1 a drawing-tree of $P^{(1)}$.*

■ **Algorithm 1** CONSTRUCTDRAWINGTREE($P^{(1)}$).

```

root ← (2Δ, π/sP)
Baseleft ← linear line (2Δ, π/sP) + i · (4δ, 2π/sP) for i ∈ {1, ⋯, ∞} starting at root
Baseright ← linear line (2Δ, π/sP) + i · (4δ, 0) for i ∈ {1, ⋯, ∞} starting at root
T(1) ← root + Baseleft + Baseright                                     ▷ Base Tree
while cov(T(1)) ≠ P(1) do                                       ▷ grow the tree inside the cone
  let (p, t), p ∈ P(1) \ cov(T(1)), t ∈ T(1) be the pair with minimal distance
  if dist(p, t) < 1 − δ then
    add p to T(1) and connect it to t
  else
    t' ← (p + t)/2                                                   ▷ Intermediate node between p and t
    add t' to T(1) and connect it to t; add p to T(1) and connect it to t'
  remove all subtrees of T(1) that do not cover any p ∈ P(1) and return T(1)

```

It is clear that $T^{(1)}$ is computable from P . The nodes cover all positions in $P^{(1)}$. Distances between neighboring nodes are 4δ on the linear lines and at most $1 - \delta$ everywhere else. So $T^{(1)}$ fulfills all requirements for a drawing path for $\delta \leq 0.2$.⁷

► **Observation 4.2.** *A reasonable short and deterministically computable traversal of $T^{(1)}$ as defined in Definition 4.1 is a drawing path for $\delta \leq 0.2$. We define $trav(T^{(1)})$ to represent the path of this traversal.*

Summary of Lemma 3.22 proof. $trav(T^{(1)})$ is a drawing path, but it is not necessarily compatible to a drawing formation because the tail does always not fulfill the requirements of Definition 3.15. The tail is the last part of the path, which covers in total ≤ 3 pattern positions (Definition 3.14). For the compatibility, it must have a length that is traversable by a drawing formation of 3 robots, i.e. length $\leq |A_P^3(\epsilon)|$. Additionally, all positions covered by the tail must be in the distance ≤ 1 to the last node of the tail. This allows the remaining 3 robots of the drawing formation to reach the last three pattern positions from the last node of the tail. In Lemma A.1 we first show that there exists suitable start point z_{start} and end point z_{end} for such a tail. z_{end} is a suitable end point, when it has at least 3 positions $p_1, p_2, p_3 \in P^{(1)}$ in its 1-surrounding. z_{start} must have a constant distance to z_{end} and cover at least one additional position $p_4 \in P^{(1)} \setminus \{p_1, p_2, p_3\}$. To prove Lemma 3.22 we construct a compatible drawing path out of z_{start} , z_{end} and $T^{(1)}$. We connect z_{start} and z_{end} with intermediate nodes in a straight line and add possibly up to 3 additional intermediate nodes

⁶ While the pattern is connected by definition, the cuts in symmetric components can disconnect parts of the component. E.g. if the pattern is a multi-helix spiral

⁷ It holds for $\delta \leq 0.2$ that $4\delta \leq 1 - \delta$

around z_{end} to cover the positions p_1, p_2 and p_3 . z_{start} is connected with a straight line of intermediate nodes to the end of $T^{(1)}$ as well. Because z_{start} and z_{end} have a constant distance and cover, together with the intermediate nodes, at least the positions p_1, \dots, p_4 we know that the tail fulfills the requirements mentioned above. The full proof of Lemma 3.22 can be found in Appendix A.

5 Discussion and Future Work

This section discusses some additional aspects of our arbitrary pattern formation protocol for oblivious robots with a limited viewing range and highlights some open questions.

Near-Gathering with Symmetry Preservation. We presented a protocol that starts in a near-gathering (all robots within a constant diameter). The authors of [4] gave a class of near-gathering protocols in the same model we consider. They proved that their protocols, starting from any connected initial configuration, reach a near-gathering in $\mathcal{O}(|P|^2)$ rounds. However, their class contains protocols that increase the symmetricity during the execution. For the application of pattern formation, it is essential that the swarms symmetricity does not exceed $\text{sym}(P)$. It remains an interesting open question whether there is a suitable near-gathering protocol that preserves the initial symmetricity.

Synchronicity. We assume the fully-synchronous $\mathcal{F}\text{SYNC}$ scheduler for our protocol. In the related work, many papers assume $\mathcal{A}\text{SYNC}$. The authors of [13] proved that, for an unlimited viewing range, $\mathcal{F}\text{SYNC}$ has the same pattern formation capabilities as $\mathcal{A}\text{SYNC}$. An unlimited viewing range makes it much easier to maintain common knowledge in the swarm (like a common coordinate system). For a limited viewing range, our protocol must maintain this information during execution (using the ϵ -granular drawing formations). In the $\mathcal{A}\text{SYNC}$ model, where only a part of the drawing formation might be activated, we would have to ensure that “partially” moving a drawing formation does not destroy the encoded information (e.g., by encoding information redundantly). It is a crucial part of ϵ -granular drawing formations that their robots can identify them, and we would have to maintain this property under partial movements. While it seems challenging, a careful design might be able to solve this.

Connectedness. In our main theorem, we assume that the unit disc graph of P is connected. This is a natural assumption for robots with limited visibility because they cannot interact beyond their viewing range. Whenever such pattern formation is used in a real-world application, basically only connected patterns are meaningful (e.g., for creating an ad-hoc-network). However, our protocol is capable of forming patterns with less connectivity. It applies to any pattern P where a compatible drawing path can be created. When we translate Definition 3.15 to a pattern, we get the following condition:

Let $\text{perm}(P)$ be a permutation of P . There exist $\text{perm}(P) = (p_i)_{i=1}^k$ such that

1. $\text{dist}(p_i, p_{i+1}) \leq |A_F^{k-i}(\epsilon)|$ for $1 \leq i \leq k-2$
2. p_{k-2}, p_{k-1} and p_k must have a smallest enclosing circle of radius ≤ 1

Condition (1) follows from (1) and (2) of Definition 3.15, and condition (2) is necessary such that a drawing path can fulfill (4) of Definition 3.15.

Besides the last three robots, the maximal distance between two pattern positions is dependent on $|A_F^{k-i}(\epsilon)|$. In the proof of Lemma 3.11 we have shown that $|A_F^{k-i}(\epsilon)| = \mathcal{O}\left(\binom{\epsilon^{-1}}{k-i-3}\right)$. We can choose ϵ freely to reach any distance necessary.

References

- 1 Kaustav Bose, Ranendu Adhikary, Manash Kumar Kundu, and Buddhadeb Sau. Arbitrary pattern formation on infinite grid by asynchronous oblivious robots. *Theor. Comput. Sci.*, 815:213–227, 2020. doi:10.1016/J.TCS.2020.02.016.
- 2 Kaustav Bose, Manash Kumar Kundu, Ranendu Adhikary, and Buddhadeb Sau. Arbitrary pattern formation by asynchronous opaque robots with lights. *Theor. Comput. Sci.*, 849:138–158, 2021. doi:10.1016/J.TCS.2020.10.015.
- 3 Jannik Castenow, Matthias Fischer, Jonas Harbig, Daniel Jung, and Friedhelm Meyer auf der Heide. Gathering anonymous, oblivious robots on a grid. *Theor. Comput. Sci.*, 815:289–309, 2020. doi:10.1016/J.TCS.2020.02.018.
- 4 Jannik Castenow, Jonas Harbig, Daniel Jung, Peter Kling, Till Knollmann, and Friedhelm Meyer auf der Heide. A unifying approach to efficient (near)-gathering of disoriented robots with limited visibility. In *OPODIS 2022, December 13-15, 2022, Brussels, Belgium*, volume 253 of *LIPICs*, pages 15:1–15:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.OPODIS.2022.15.
- 5 Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Asynchronous arbitrary pattern formation: the effects of a rigorous approach. *Distributed Comput.*, 32(2):91–132, 2019. doi:10.1007/S00446-018-0325-7.
- 6 Reuven Cohen and David Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM J. Comput.*, 34(6):1516–1528, 2005. doi:10.1137/S0097539704446475.
- 7 Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. *Distributed Comput.*, 28(2):131–145, 2015. doi:10.1007/S00446-014-0220-9.
- 8 Bastian Degener, Barbara Kempkes, Tobias Langner, Friedhelm Meyer auf der Heide, Peter Pietrzyk, and Roger Wattenhofer. A tight runtime bound for synchronous gathering of autonomous robots with limited visibility. In *SPAA 2011, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*, pages 139–148. ACM, 2011. doi:10.1145/1989493.1989515.
- 9 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors. *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340. Springer, 2019. doi:10.1007/978-3-030-11072-7.
- 10 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Moving and computing models: Robots. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340, pages 3–14. Springer, 2019. doi:10.1007/978-3-030-11072-7_1.
- 11 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Distributed computing by mobile robots: uniform circle formation. *Distributed Comput.*, 30(6):413–457, 2017. doi:10.1007/S00446-016-0291-X.
- 12 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theor. Comput. Sci.*, 407(1-3):412–447, 2008. doi:10.1016/J.TCS.2008.07.026.
- 13 Nao Fujinaga, Yukiko Yamauchi, Hiroataka Ono, Shuji Kijima, and Masafumi Yamashita. Pattern formation by oblivious asynchronous mobile robots. *SIAM J. Comput.*, 44(3):740–785, 2015. doi:10.1137/140958682.
- 14 Christopher Hahn, Jonas Harbig, and Peter Kling. Forming large patterns with local robots in the OBLLOT model (arXiv-version with extended appendix). arXiv. URL: <https://arxiv.org/abs/2404.02771>.
- 15 Chang-kwon Kang, Farbod Fahimi, Rob Griffin, D Brian Landrum, Bryan Mesmer, Guangsheng Zhang, Taeyoung Lee, Hikaru Aono, Jeremy Pohly, Jesse McCain, et al. Marsbee-swarm of flapping wing flyers for enhanced mars exploration. Technical report, NASA, 2019.
- 16 David G. Kirkpatrick, Irina Kostitsyna, Alfredo Navarra, Giuseppe Prencipe, and Nicola Santoro. Separating bounded and unbounded asynchrony for autonomous robots: Point

- convergence with limited visibility. In *PODC 2021, Italy, July 26-30, 2021*, pages 9–19. ACM, 2021. doi:10.1145/3465084.3467910.
- 17 Giuseppe Antonio Di Luna, Paola Flocchini, Nicola Santoro, and Giovanni Viglietta. Turing-mobile: a turing machine of oblivious mobile robots with limited visibility and its applications. *Distributed Comput.*, 35(2):105–122, 2022. doi:10.1007/S00446-021-00406-6.
 - 18 Moumita Mondal and Sruti Gan Chaudhuri. Uniform circle formation by swarm robots under limited visibility. In *ICDCIT 2020, Bhubaneswar, India, January 9-12, 2020*, volume 11969, pages 420–428. Springer, 2020. doi:10.1007/978-3-030-36987-3_28.
 - 19 Fernando Soto, Jie Wang, Rajib Ahmed, and Utkan Demirci. Medical Micro/Nanorobots in Precision Medicine. *Advanced Science*, 7(21), November 2020. doi:10.1002/adv.202002203.
 - 20 Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999. doi:10.1137/S009753979628292X.
 - 21 Yuko Ulrich, Mari Kawakatsu, Christopher K. Tokita, Jonathan Saragosti, Vikram Chandra, Corina E. Tarnita, and Daniel J. C. Kronauer. Response thresholds alone cannot explain empirical patterns of division of labor in social insects. *PLOS Biology*, 19(6):e3001269, June 2021. doi:10.1371/journal.pbio.3001269.
 - 22 Giovanni Viglietta. Uniform circle formation. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340, pages 83–108. Springer, 2019. doi:10.1007/978-3-030-11072-7_5.
 - 23 Masafumi Yamashita and Ichiro Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theor. Comput. Sci.*, 411(26-28):2433–2453, 2010. doi:10.1016/J.TCS.2010.01.037.
 - 24 Yukiko Yamauchi. A survey on pattern formation of autonomous mobile robots: asynchrony, obliviousness and visibility. *Journal of Physics: Conference Series*, 473:012016, December 2013. doi:10.1088/1742-6596/473/1/012016.
 - 25 Yukiko Yamauchi, Taichi Uehara, and Masafumi Yamashita. Brief announcement: Pattern formation problem for synchronous mobile robots in the three dimensional euclidean space. In *PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 447–449. ACM, 2016. doi:10.1145/2933057.2933063.
 - 26 Yukiko Yamauchi and Masafumi Yamashita. Pattern formation by mobile robots with limited visibility. In *SIROCCO 2013, Ischia, Italy, July 1-3, 2013*, volume 8179 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2013. doi:10.1007/978-3-319-03578-9_17.

A Proof of Lemma 3.22

► **Lemma A.1.** *Let P be a pattern with symmetricity $s_P < |P|/2$. There exist for each symmetric component $P^{(1)}$ with cone $C^{(1)}$ points z_{start} and z_{end} such that*

1. $|\mathcal{B}(z_{end}, 1)| \geq 3$
2. if $|P^{(i)}| > 3$
 - a. $|\mathcal{B}(z_{start}, 1 - \delta) \cup \mathcal{B}(z_{end}, 1)| \geq 4$
 - b. $dist(z_{start}, z_{end}) = \mathcal{O}(1)$
 - c. $dist(\{z_{start}, z_{end}\}, \partial C^{(1)}) = \Omega(1/s_P + mindist(P))$

Proof.

(1). Consider a finite subset $S \subset \mathcal{R}^2$ of symmetricity s and size $|S| \geq 3s$. Assume the unit disc graph $\mathcal{U}(S)$ is connected. It is a simple geometric fact that there exists a subset $C \subseteq S$ of size $|C| = 3$ and with $\angle(C) < 2\pi/s$ such that $\mathcal{U}(C)$ is connected. We stated and proved this in the appendix (Lemma D.1). Our pattern P is such a set and a symmetric component $P^{(1)}$ is a subset with $\angle(C) < 2\pi/s$, therefore there exists a rotation of P such that $p_1, p_2, p_3 \in P^{(1)}$ with $\mathcal{U}(\{p_1, p_2, p_3\})$ connected. W.l.o.g. we assume this is the rotation of P . Then, there exists a point z_{end} with $\mathcal{B}(z_{end}, 1) \supseteq \{p_1, p_2, p_3\}$.

14:18 Forming Large Patterns with Local Robots in the OBLLOT Model

Trivial cases for (a), (b) and (c). If $P^{(1)}$ contains 4 positions with $\mathcal{U}(\{p_1, p_2, p_3, p_4\})$ connected, it is clear that z_{start} and z_{end} can be placed with $\mathcal{B}(z_{start}, 1 - \delta) \cup \mathcal{B}(z_{end}, 1) \supseteq \{p_1, p_2, p_3, p_4\}$ (a). If $|\mathcal{B}(z_{end}, 1 - \delta)| \leq 3$ (w.l.o.g. $p_4 \notin \mathcal{B}(z_{end}, 1 - \delta)$), we place z_{start} in distance $\leq 1 - \delta$ to p_4 (b). z_{start} and z_{end} have a constant distance (c). If $\mathcal{B}(z_{end}, 1)$ contains more than three positions that are not all connected, the placement for z_{start} is analog.

(a) and (b). We assume that $\mathcal{B}(z_{end}, 1) = \{p_1, p_2, p_3\}$. Because $\mathcal{U}(P)$ is connected, there exists a path from p_1 to $p' \in P^{(1)} \setminus \{p_1, p_2, p_3\}$ in $\mathcal{U}(P)$. Let $p_1, w_1 \dots, w_k, p'$ be a shortest path from p_1 to p' . The path can only contain 3 consecutive nodes in one symmetric component (otherwise the component would contain 4 connected positions), therefore there exist $w_j \in P^{(2)}$ with $j \leq 3$ and $w_l \notin P^{(2)}$ with $l \leq j + 3$. If $w_l \in P^{(1)}$, we find $p_4 = w_l$ with $dist(p_1, p_3) \leq 6$. If $w_l \in P^{(3)}$, there exist a rotational symmetric point in $P^{(1)}$, let this be p_4 . It is clear that $dist(p_1, p_4) \leq dist(p_1, w_l) \leq 6$. We can place z_{start} in the distance $1 - \delta$ of p_4 to fulfill (a) and (b).

(c). z_{start} can be placed relatively freely in a radius of $1 - \delta$ around p_4 , this easily fulfill (c). There exist cases where z_{end} must be placed directly on one of the three connected positions, let this be p_1 . From Lemma D.1 we can follow, that w.l.o.g. p_{end} lies on the bisector of $C^{(1)}$. Let $p' \in P^{(2)}$ be the rotational symmetric point to p_1 . Naturally, $dist(z_{end}, \partial C^{(1)}) = \frac{1}{2} dist(p_1, p') \geq mindist(P)$. \blacktriangleleft

► **Lemma 3.22 (restated).** Consider the ϵ -granular drawing formation $F^{(1)}$ of the initial drawing pattern for a connected pattern P of symmetry $s_P < |P|/2$. The parameter ϵ can be chosen such that $F^{(1)}$ has $|L_{F^{(1)}}(\epsilon)| \geq 2 + |P^{(1)}|$ ϵ -granular locations. Moreover, there exists a drawing path $\mathbf{v}^{(1)}$ of symmetric component $P^{(1)}$ that is compatible with $F^{(1)}$ such that

1. the path $\mathbf{v}^{(1)}$ starts in the anchor $a = (2\Delta, \pi/s_P)$ of $F^{(1)}$,
2. $\mathbf{v}^{(1)}$ lies in the first cone (i.e., $\mathbf{v}^{(1)} \subseteq C^{(1)}$) and

$$dist(\mathbf{v}^{(1)}, \partial C^{(1)}) > \max\{\epsilon, \Delta \cdot \sin(2\pi/s_P)\}. \quad (4)$$

Proof. Let $T^{(1)}$ be the drawing tree of $P^{(1)}$ Definition 4.1. $trav(T^{(1)})$ has all properties for a drawing path (Observation 4.2). Let $F^{(1)}$ be a drawing formation with $\epsilon = \Theta(\min(1/s_P, mindist(P), 1/\sqrt{|P|}))$ and $\Delta = 0.1$ and $\Phi = 2\pi/s_P$. To make $trav(T^{(1)})$ compatible with $F^{(1)}$ we append the points z_{start} and z_{end} from Lemma A.1. We add $b = \mathcal{O}(P)$ intermediate nodes w_1, \dots, w_b between the end of $trav(T^{(1)})$ and z_{start} . We add $b' = \mathcal{O}(1)$ intermediate nodes $w_{b+1}, \dots, w_{b+b'}$ between z_{start} and z_{end} . The resulting path is

$$\mathbf{v}^{(1)} := trav(T^{(1)}) + (w_i)_{i=0}^b + (z_{start}) + (w_i)_{i=b+1}^{b+b'} + (z_{end})$$

We make sure, that

$$cov((w_i)_{i=b+j}^{b+b'} + (z_{end})) \subseteq \mathcal{B}(z_{end}, 1) \text{ with } |cov((w_i)_{i=b+j}^{b+b'} + (z_{end}))| \geq 3$$

for $1 \leq j \leq b'$. This is possible by placing up to 3 intermediate nodes in distance $\leq \delta$ to z_{end} . This number of nodes is sufficient to reach z_{start} , respectively z_{end} , with distances $\leq 1 - \delta$ between w_i and w_{i+1} , because z_{start} has a distance $\mathcal{O}(|P|)$ from any node of $trav(T^{(1)})$ and z_{end} has a constant distance from z_{start} (see Lemma A.1). There obviously exist deterministic methods to define the intermediate paths, chose z_{end} , z_{start} , and determine $trav(T^{(1)})$ with $hops(trav(T^{(1)})) = \mathcal{O}(|P|)$.

Compatibility. We show that conditions (1) - (4) from Definition 3.15 are fulfilled. **(1)** with $\delta = 0.1$ this is fulfilled **(2)** From Lemma A.1 we know that $|\mathcal{B}(z_{end})| = 3$. From the equation in the beginning of this proof follows $\text{cov}(\text{tail}(v^{(1)})) \subseteq \mathcal{B}(z_{end})$, **(3)** If $|P^{(1)}| \geq 4$ than $|\mathcal{B}(z_{start}, 1 - \delta) \cup \mathcal{B}(z_{end}, 1)| \geq 4$ (by Lemma A.1 (2)) Hence, the tail of $v^{(1)}$ must start after z_{start} . $|\text{tail}(v^{(1)})| = \mathcal{O}(1)$ in this case **(3)** is fulfilled (see Lemma 3.11). If $|P^{(1)}| = 3$ we know that $\epsilon = \mathcal{O}(1/|P|)$. By Lemma 3.11) follows that $\mathcal{A}_F^3(\epsilon) = \Omega(|P|)$. $\text{tail}(v^{(1)}) = v^{(1)}$ with length $\mathcal{O}(|P|)$ in this case. This fulfills **(3)**. **(4):** With $\epsilon = \mathcal{O}(1/\sqrt{|P|})$ we have $|\mathcal{A}_F^4(\epsilon)| = \Omega(|P|)$ (see Lemma 3.11). We have $|v^{(1)}| = \mathcal{O}(|P|)$. This fulfills **(4)**

Requirements (1) and (2) of Lemma 3.22. (1) the root node of $T^{(1)}$ has coordinate $(2\Delta, \pi/s_P)$ and is the start of $v^{(1)}$. (2) By construction of Definition 4.1 is clear that $\text{dist}(t, \partial c(1)) \geq \text{dist}((2\Delta, \pi/s_P), \partial c(1)) = \Delta \cdot \sin(\frac{2\pi}{s_P})$ for $t \in T^{(1)}$. By Lemma 3.17 we know that $\text{dist}(z_i, \partial C^{(1)}) = \Omega(\text{mindist}(P)), i \in \{1, 2\}$. $\epsilon < \min(1/s_P, \text{mindist}(P), 1/\sqrt{|P|}) \cdot \Delta$. With this choice of ϵ we can create a ϵ -granular drawing formation that has more locations than $|P^{(1)}| + 2$.

Such that F is an ϵ -granular drawing formation, the parameter Δ, ϕ and ϵ must be known. Δ and Φ are given above and computable from P . $\epsilon = \min(1/s_P, \text{mindist}(P), 1/\sqrt{|P|}) \cdot c$ for a constant $c < 1$. The constant can be deterministically determined (but we never write it down). ◀

B Proof of Lemma 3.17

► **Lemma 3.17 (restated).** Consider a compatible drawing path $\mathbf{v} = (v_j)_{j=1}^k$ of a pattern P . Let \mathcal{R} be the configuration that has

1. one robot at each coordinate in $P \setminus \text{cov}(\text{tail}(\mathbf{v}))$ and
 2. a drawing formation F of size 3 in state $|\text{tail}(\mathbf{v})|$ anchored in v_k that is compatible with \mathbf{v} .
- Then F can dissolve within two LCM-cycles while dropping one robot at each coordinate in $\text{cov}(\text{tail}(\mathbf{v}))$.

We have shown in Lemma 3.16 that on all coordinates outside the $\text{tail}(v)$, F can drop robots while traversing the drawing path v . On the tail it cannot further drop robots before reaching the end; otherwise $|\mathcal{R}_F| \leq 2$, which is not an ϵ -granular drawing formation anymore. In fact, it can never be a valid drawing formation because two robots can not encode the direction \vec{d} in a model without a compass. Therefore, F does not drop robots onto $\text{cov}(\text{tail}(v))$ during the traversal. Instead, the drawing formation moves onto v_k , and the robots will move from there onto $\text{cov}(\text{tail}(v))$. Because $\text{dist}(\text{cov}(\text{tail}(v)), v_k) \leq 1$ for a compatible path, the drawing formation is close enough to all remaining positions. However, not all robots in \mathcal{R}_F are on v_k ; they can have a distance up to Δ . If $\text{dist}(v_k, p) > 1 - \Delta, p \in \text{cov}(\text{tail}(v))$ than the drawing formation F can be placed inconveniently such that $\text{dist}(r, p) > 1, r \in \mathcal{R}_F$. We will add an intermediate step that reshapes the drawing formation such that all robots have a distance of ≤ 1 to the coordinate they must obtain in the end. Robots may leave the drawing hull H_F , but most properties of a drawing formation must still be fulfilled. This intermediate shape F_{inter} must be valid in the sense that robots in F_{inter} must be able to determine the position of v_k and the direction vector \vec{d} to compute the global coordinate system. In the following proof, we will define the intermediate shape and prove that robots can obtain this information.

► **Definition B.1** (ϵ -intermediate-shape). Let P be a pattern and $v = (v_1, \dots, v_k)$ be its drawing path with

1. $|\text{cov}(\text{tail}(v))| = 3$ and
2. $\mathcal{B}(v_k, 1) \supseteq \text{cov}(\text{tail}(v))$

14:20 Forming Large Patterns with Local Robots in the OBLLOT Model

Let $\text{cov}(\text{tail}(v)) = \{p_1, p_2, p_3\}$. The intermediate shape F_{inter} defines the following positions for a set of three robots r_1, r_2, r_3 .

- r_1 is on v_k
- r_2 is distance $\epsilon/2$ in direction p_2 and
- r_3 is distance $\epsilon/3$ in direction p_3

► **Observation B.2.** Let F_{inter} be an intermediate shape as in Definition B.1. It is clear, that $\text{dist}(r_1, p_1) \leq 1$, $\text{dist}(r_2, p_2) \leq 1$ and $\text{dist}(r_3, p_3) \leq 1$.

► **Lemma B.3.** Let P be a pattern and $v = (v_1, \dots, v_k)$ be a drawing path which is compatible with an ϵ -ganular drawing formation. Let F_{inter} be an intermediate shape as defined in Definition B.1 which is on v_k . Let $\text{dist}(v_k, \mathcal{R} \setminus F_{\text{inter}}) > \epsilon$. All robots $r \in F_{\text{inter}}$ can decide, that they are in F_{inter} and can compute the positions of $\text{tail}(v_k)$ in their local coordinate system.

Proof. To decide, whether a robot $r \in F_{\text{inter}}$ is in F_{inter} it observes other robots in distance ϵ . Because the $\text{dist}(v_k, \mathcal{R} \setminus F_{\text{inter}}) > \epsilon$ it only finds one triple of robots with distances $\epsilon/2$ and $\epsilon/3$. Let $r_1, r_2, r_3, p_1, p_2, p_3$ be as in Definition B.1. The triangle r_1, r_2, r_3 is never equilateral (one side has length $\epsilon/2$ and another $\epsilon/3$). With chirality, all three robots know, who is r_1, r_2 and r_3 . They know the coordinates p_1, p_2 and p_3 as well as the position of v_k in the global coordinate system. The positions of r_1, r_2 and r_3 in the global coordinate system are also defined. Therefore, they can translate/rotate their local coordinate system and compute $\text{tail}(v_k)$. ◀

Lemma 3.17 follows immediately from Observation B.2 and Lemma B.3.

C Proof of Lemma 3.23


Proof. For $s_p = |P|/2$, the pattern P has coordinates $\cup_{i=0}^{s_p-1} \{(D_1, \alpha_1 + i \cdot 2\pi/s_p), (D_2, \alpha_2 + i \cdot 2\pi/s_p)\}$ with $\alpha_1, \alpha_2 \leq 2\pi/s_p$. $s_p = |P|$ is just a special case with $D_1 = D_2$ and $\alpha_1 = \alpha_2$. Let $\text{pos}_t(r_i)$ be the position of the robot r_i in round t . In a configuration with symmetricity $|P|/2$, the positions are as follows $\text{pos}_t(r_i) = (d_1(t), \beta_1(t) + ((i-1)/2) \cdot 2\pi/s_p)$ if i uneven, $\text{pos}_t(r_i) = (d_2(t), \beta_2(t) + (i/2) \cdot 2\pi/s_p)$ if i even. The robots form in the first round of the execution the pattern P scaled to a diameter ≤ 1 . From there on, they scale the pattern up. Therefore $\frac{d_1(t)}{d_2(t)} = \frac{D_1}{D_2}$ and $\beta_1(t) = \alpha_1, \beta_2(t) = \alpha_2$ for $t > 1$. It is clear, that the "uneven" (i is uneven) robots on $\text{pos}_t(r_i) = (d_1(t), \alpha_1 + ((i-1)/2) \cdot 2\pi/s_p)$ can distinguish themselves from the "even" robots on $\text{pos}_t(r_i) = (d_2(t), \alpha_2 + (i/2) \cdot 2\pi/s_p)$ when $\frac{d_1(t)}{d_2(t)} = \frac{D_1}{D_2}$. This is enough to compute the global coordinate system. The robots move onto positions with the same angle and $d_1(i+1) = \min\left(d_1(i) + 1, d_1(i) \cdot \frac{d_2(i)+1}{d_2(i)}, D_1\right)$ $d_2(i+1) = \min\left(d_2(i) + 1, d_2(i) \cdot \frac{d_1(i)+1}{d_1(i)}, D_2\right)$ This reached $d_1(t) = D_1$ and $d_2(t) = D_2$ after $\leq \max(D_1, D_2) \leq |P|$ rounds. ◀

D Auxiliary Results

► **Lemma D.1.** Consider a finite connected subset $S \subset \mathcal{R}^2$ of symmetricity $s := \text{sym}(S) \in \mathbb{N}$ and size $|S| \geq 3s$. There exists a subset $C \subseteq S$ of size $|C| = 3$ and with $\angle(C) < 2\pi/s$ such that C is connected.

The proof can be found in [14].

Efficient Shape Formation by 3D Hybrid Programmable Matter: An Algorithm for Low Diameter Intermediate Structures

Kristian Hinnenthal ✉ 🏠 

Paderborn University, Germany

David Liedtke ✉ 🏠 

Paderborn University, Germany

Christian Scheideler ✉ 🏠 

Paderborn University, Germany

Abstract

This paper considers the shape formation problem within the 3D hybrid model, where a single agent with a strictly limited viewing range and the computational capacity of a deterministic finite automaton manipulates passive tiles through pick-up, movement, and placement actions. The goal is to reconfigure a set of tiles into a specific shape termed an *icicle*. The icicle, identified as a dense, hole-free structure, is strategically chosen to function as an intermediate shape for more intricate shape formation tasks. It is designed for easy exploration by a finite state agent, enabling the identification of tiles that can be lifted without breaking connectivity. Compared to the line shape, the icicle presents distinct advantages, including a reduced diameter and the presence of multiple removable tiles. We propose an algorithm that transforms an arbitrary initially connected tile structure into an icicle in $\mathcal{O}(n^3)$ steps, matching the runtime of the line formation algorithm from prior work. Our theoretical contribution is accompanied by an extensive experimental analysis, indicating that our algorithm decreases the diameter of tile structures on average.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Programmable Matter, Shape Formation, 3D Model, Finite Automaton

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.15

Related Version *Full Version*: 10.48550/arXiv.2401.17734 [15]

Funding This work was supported by the DFG Project SCHE 1592/10-1.

1 Introduction

Advancements in molecular engineering have led to the development of a series of computing DNA robots designed for nano-scale operations. These robots are intended to perform simple tasks such as transporting cargo, facilitating communication, navigating surfaces of membranes, and pathfinding [29, 1, 22, 4]. Envisioning the future of nanotechnology, we anticipate a scenario where a collective of computing particles collaboratively acts as programmable matter – a homogeneous material capable of altering its shape and physical properties programmably. There are numerous potential applications: For environmental remediation, particles may construct nanoscale filtration systems to remove pollutants from air or water. They may also be deployed within the human body to construct intricate structures for targeted drug delivery, perform nanoscale surgeries, or repair damaged tissues at a cellular level. Additionally, they could assemble nanoscale circuits and components, enabling the development of more efficient and compact electronic devices. Each of those scenarios is an application of the *shape formation problem*, which is the subject of this paper.



© Kristian Hinnenthal, David Liedtke, and Christian Scheideler;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 15; pp. 15:1–15:20

Leibniz International Proceedings in Informatics



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

Over the past few decades, various models of programmable matter have emerged, primarily distinguished by the activity of entities within them. Passive systems consist of entities (called tiles) that undergo movement and bonding exclusively in response to external stimuli, such as current or light, or based on their inherent structural properties, such as specific glues on the surfaces of tiles. Examples of these passive systems include the DNA tile assembly models aTAM, kTAM, and 2HAM, which are extensively discussed in the survey [25], as well as population protocols [2], and slime molds [5]. In contrast, active systems consist of entities (called particles, agents or robots) that independently perform computation and movement to accomplish tasks. Notable examples encompass the Amoebot model [7], modular self-reconfigurable robots [27, 30], the nubot model [34], metamorphic robots [6, 31], and swarm robotics [33].

While fabricating computing DNA robots remains challenging, producing simple passive tiles from folded DNA strands is efficient and scalable [14]. The hybrid model of programmable matter [12, 13, 16, 23, 19] offers a compromise between feasibility and utility. This model involves a small number of active agents with the computational capabilities of deterministic finite automata together with a large set of passive building blocks, called tiles. Agents can manipulate the structure of tiles by picking up a tile, moving it, and placing it at some spot. A key advantage of the hybrid approach lies in the reusability of agents upon completing a task, where in purely active systems, particles become part of the formed structure.

In this paper, we address the shape formation problem within the 3D hybrid model, with the ultimate goal of transforming an arbitrary initial arrangement of tiles into a predefined shape. We consider tiles in the shape of rhombic dodecahedra, i.e., polyhedra featuring 12 congruent rhombic faces, positioned at nodes within the adjacency graph of face-centered cubic (FCC) stacked spheres (see Figure 1a). Unlike rectangular tiles, the rhombic dodecahedron presents a distinct advantage: it allows an agent to orbit around a tile without risking connectivity. This property is particularly valuable in liquid or low gravity environments, where it prevents unintended separation between the agents and the tiles.

Achieving universal 3D shape formation faces a key challenge: identifying tiles that can be lifted without disconnecting the tile structure (referred to as *removable* tiles). Even if such tiles exist, locating them requires exploring the tile structure, demanding $\Omega(D \log(\Delta))$ memory bits for graphs with a diameter D and degree Δ [11]. When limited to constant memory, navigating plane labyrinths requires two placeable markers (pebbles) [17, 3]. In the 2D context, finding removable tiles is impossible without prior modification of the tile structure, as discussed in [13]. In 3D, complexity increases significantly, with instances where any tile movement can locally disconnect the structure. As discussed above, the agent is unable to verify whether this disconnection also occurs globally. To address these challenges, we make the assumption that the agent carries a tile initially, using it to uncover removable tiles through successive tile movements. It is still entirely unclear whether otherwise a removable tile can be found in all 3D instances. For that reason, our primary goal is to construct an intermediate structure that is easily navigable by constant-memory agents and allows the identification of removable tiles without relying on an initially carried tile.

1.1 Our Contribution

The intermediate structure we propose is termed an *icicle*, characterized by a platform representing a parallelogram and downward-extending lines of tiles from the platform (see Figure 1a). We present a single-agent algorithm that transforms any initially connected tile structure into an icicle in $\mathcal{O}(n^3)$ steps, matching the efficiency of the line formation algorithm from prior work [16]. While both the icicle and the line enable agents without an

initial tile to find removable tiles, the icicle presents distinct advantages. In the best-case scenario, the diameter D of an icicle can be as low as $\mathcal{O}(n^{\frac{1}{3}})$, whereas a line consistently maintains a diameter of n . Furthermore, an icicle encompasses multiple removable tiles, which removes the necessity to traverse the intermediate shape completely to locate a removable tile. Our paper includes comprehensive simulation results, indicating that, on average, our algorithm reduces the diameter of the tile structure. In addition, the runtime observed in the simulations consistently falls below the bound established in our runtime analysis. Across all simulations, the runtime remains well within the vicinity of n^2 . It is noteworthy that we identified an edge case where the diameter could increase by a factor of $\mathcal{O}(n^{\frac{1}{3}})$, although we believe this to be the worst-case.

1.2 Related Work

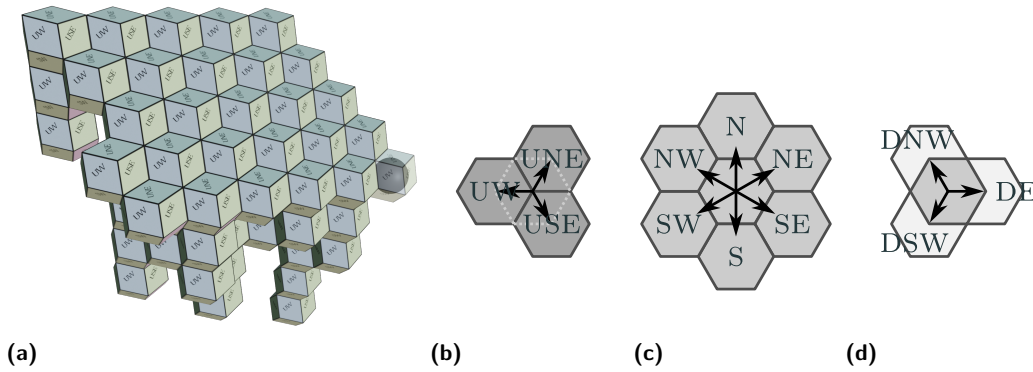
The 3D variant of the hybrid model was introduced in [16], where the authors presented an algorithm capable of transforming any connected input configuration into a line in $\mathcal{O}(n^3)$ steps. In [19], the authors address the coating problem, providing a solution that solves the problem in worst-case optimal $\mathcal{O}(n^2)$ steps. They assume a single active agent that has access to a constant number of distinguishable tile types.

Significant progress has been made in recent years regarding the 2D version of the hybrid model. For instance, in [13], the authors address the 2D shape formation problem, presenting algorithms for a single active agent that efficiently constructs line, block, and tree structures - each being hole-free structures with specific advantages and disadvantages - in worst-case optimal $\mathcal{O}(n^2)$ steps. Another publication, [12], explores the recognition of parallelograms with a specific height-to-length ratio. The most recent publication [23] solves the problem of maintaining a line of tiles in presence of multiple agents and dynamic failures of the tiles.

Closely tied to the hybrid model is the well-established Amoebot model, where computing particles traverse an infinite triangular lattice through expansions and contractions. In [8], the authors showcase the construction of simple shapes like hexagons or triangles within the Amoebot model. Expanding on this work, [9] introduces a universal shape formation algorithm capable of constructing an arbitrary input shape using a constant number of equilateral triangles, with the scale depending on the number of amoebots. Notably, this work assumes common chirality, a sequential activation schedule, and randomization. Subsequent improvements are presented in [10], where a deterministic algorithm is introduced, enabling amoebots to form any Turing-computable shape without the need for common chirality or randomization. In [20], the authors consider shape formation in the presence of a finite number of faults, where a fault resets an amoebot's memory. They solve the hexagon formation problem, assuming the existence of a fault-free leader. A recent extension of the Amoebot model, discussed in [24], considers joint movements of Amoebots. The authors simulate various shape formation algorithms as a proof of concept.

In both [13] and this paper, shape formation algorithms are introduced that construct an intermediate shape, intended to serve as the foundation for more advanced shape formation algorithms. A similar strategy is explored in [18], where 2D lattice-based modular robots initially transform into a canonical shape before achieving the final desired shape. An approach that does not rely on canonical intermediate structures is considered in [26]. The authors present primitives for the Amoebot model that establish shortest path trees within the amoebot structure and subsequently directly route amoebots to their target position.

The concept of shape formation is extensively studied in the field of modular robotics and metamorphic robots, often referred to as self-reconfiguration. A comprehensive survey on this topic can be found in [28]. In the field of swarm robotics, shape formation is often closely related to the problem of computing collision-free paths [32, 21].



■ **Figure 1** (a) An example configuration that has the shape of an icicle; the agent (depicted as a sphere) is positioned at a tiled node within the platform representing a parallelogram. (b–d) The twelve compass directions divided into upwards (b), plane (c) and downwards directions (d).

1.3 Model Definition

We consider a single active agent r with limited sensing and computational power that operates on a finite set of passive *tiles* positioned at nodes of some specific underlying graph G , which we define in the following. Consider the close packing of equally sized spheres at each point of the infinite face-centered cubic lattice. Let $G = (V, E)$ be the adjacency graph of spheres in that packing, and consider an embedding of G in \mathbb{R}^3 in which all edges have equal length, e.g., the trivial embedding where the edge length equals the radius of the spheres. Cells in the dual graph of G w.r.t. that embedding have the shape of rhombic dodecahedra, i.e., polyhedra with 12 congruent rhombic faces (see Figure 1a). This is also the shape of every cell in the Voronoi tessellation of G , i.e., that shape completely tessellates 3D space. Consider a finite set of tiles that have the shape of rhombic dodecahedra. Tiles are passive, in the sense that they cannot perform any computation or movement on their own. A node $v \in V$ is *tiled*, if there is a passive tile positioned at v ; otherwise node v is *empty*. Each node can hold at most one tile and each tile is placed at at most one node at a time. Each node in V has precisely twelve neighbors whose relative positions are described by the twelve compass directions UNE, UW, USE, N, NW, SW, S, SE, NE, DNW, DSW and DE (see Figures 1b–1d). Take note that G contains infinitely many copies of the infinite triangular lattice, which serves as the underlying graph in the 2D variant. This allows us to visually depict 3D examples as a stack of 2D hexagonal tiles, as shown in Figure 1.

A *configuration* $C = (\mathcal{T}, p)$ is the set \mathcal{T} that contains all tiled nodes together with the agent's position p . We call C *connected*, if $G|_{\mathcal{T}}$ is connected or if $G|_{\mathcal{T} \cup \{p\}}$ is connected and the agent carries a tile, where $G|_W$ denotes the subgraph of G induced by some nodeset W . That is, we allow the subgraph induced by all tiled nodes to disconnect, as long as a tile carried by the agent maintains connectivity. This constraint prevents the agent and tiles to drift apart, e.g., in liquid or low gravity environments.

The agent r is the only active entity in this model. It has strictly limited sensing and computing power and can act on passive tiles by picking up a tile, moving and placing it at some spot. We assume that tiles can be moved through other tiles, e.g., by Particularly, we assume an agent with the computational capabilities of a deterministic finite automaton that performs discrete steps of *Look-Compute-Move* cycles. In the *look*-phase, the agent observes whether its current position p and the twelve neighbors of p are tiled or empty. The agent is equipped with a compass that allows it to distinguish the relative positioning of

its neighbors using the twelve above mentioned compass directions. Its initial rotation and chirality can be arbitrary, but we assume that it remains consistent throughout the execution. For ease of presentation, our algorithms and their analysis are described according to the robot’s local view, i.e., we do not distinguish between local and global compass directions. Based on the information gathered in the look phase, the agent determines its next state transition according to the finite automaton in the *compute*-phase. In the *move* phase, the agent performs an *action* that corresponds to the prior state transition. It either (i) moves to an empty or tiled node adjacent to p , (ii) places a tile at p , if $p \notin \mathcal{T}$ and r carries a tile, (iii) picks up a tile from p , if $p \in \mathcal{T}$ and r carries no tile, or (iv) terminates. The agent can carry at most one tile at a time and during actions (ii) and (iii) the agent loses and gains a tile, respectively. It’s worth noting that we allow the agent to move through tiles while carrying one simultaneously. From a practical standpoint, this capability can be facilitated by conceptualizing tiles as hollow and foldable. It is assumed that the agent is initially positioned at a tiled node, as otherwise, there might be no valid action available. Additionally, we assume that the agent initially carries a tile, a justification for which was provided in Section 1. While the agent is technically a finite automaton, we describe algorithms from a higher level of abstraction textually and through pseudocode. It is easy to see that a constant number of variables of constant-size domain each can be incorporated into the agent’s constantly many states.

1.4 Problem Statement

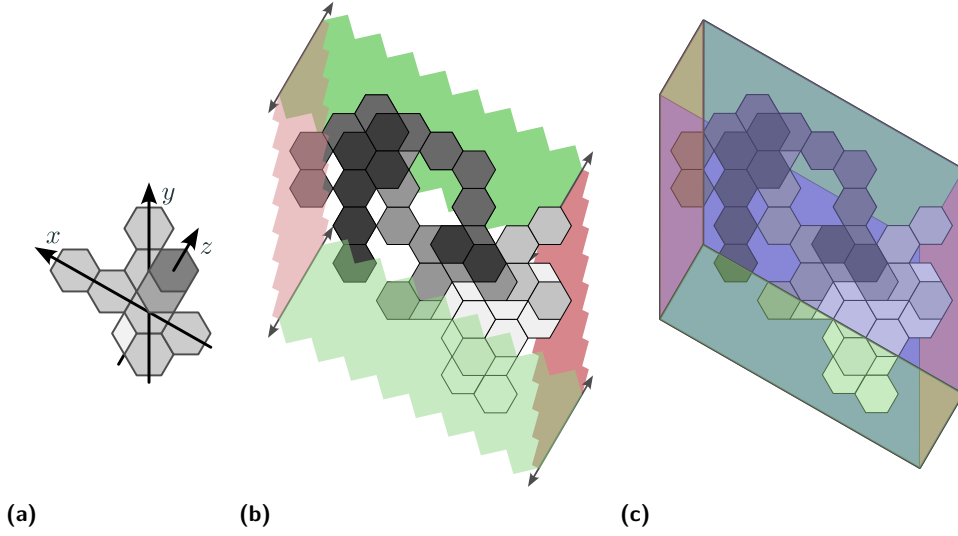
Consider an arbitrary initially connected configuration $C^0 = (\mathcal{T}^0, p^0)$ with $p^0 \in \mathcal{T}$. Superscripts in our notation generally refer to step numbers and may be omitted if they are clear from the context. An algorithm solves the *icicle formation problem*, if its execution results in a sequence of connected configurations $C^0 = (\mathcal{T}^0, p^0), \dots, C^T = (\mathcal{T}^T, p^T)$ such that nodes in \mathcal{T}^T are in the shape of an icicle (which we define below), C^t results from C^{t-1} for $1 \leq t \leq T$ by applying some action (i)–(iii) to p^{t-1} , and the agent terminates (iv) in step T .

For some node $v \in V$, we denote $v + x$ the node that is neighboring v in some compass direction x and $-x$ the opposite compass direction of x , e.g., $-UNE = DSW$. We call a maximal consecutive array of tiles in direction N and S a *column*, in direction NW and SE a *row*, and in direction UNE and DSW a *tower*. A *parallelogram* is a maximal consecutive array of equally sized columns c_0, \dots, c_m (ordered from west to east) whose southernmost tiles at nodes v_0, \dots, v_m are contained in the same row, i.e., $v_i + SE = v_{i+1}$ for all $0 \leq i < m$. In a *partially filled* parallelogram, column c_0 can have smaller size than columns c_1, \dots, c_m .

An *icicle* is defined as a connected set of towers whose uppermost tiles are contained within the same (partially filled) parallelogram, as illustrated in Figure 1a. In other words, tiles “grow” from a single uppermost parallelogram in the DSW direction, hence the chosen name “icicle”. Notably, in an icicle, any tile with a neighboring tile at UNE but not at DSW (some locally DSW -most tile below the parallelogram) can be picked up without violating connectivity (it is *removable*). If there is no such tile, i.e., all towers have size one, the northernmost tile of the westernmost column is removable.

1.5 Structure of the Paper

In Section 2, we introduce all essential terminology. Following that, we present a non-halting icicle-formation algorithm in Section 3, prove that it converges any initially connected configuration into an icicle in Section 4, and provide its termination criteria and runtime analysis in Section 5. Finally, we discuss the results obtained from simulation in Section 6.



■ **Figure 2** Illustrating the x -, y -, and z -coordinate axes (a), the bounding cylinder (b), which infinitely extends in directions UNE and DSW as indicated by the arrows, and the bounding box (c) of an example configuration. For ease of distinction, tiles are shaded according to their z -coordinate, with brighter shades representing lower z -coordinates. In the example, there is one layer that contains two fragments (darkest shade of gray), and four layers that each contain a single fragment.

2 Preliminaries

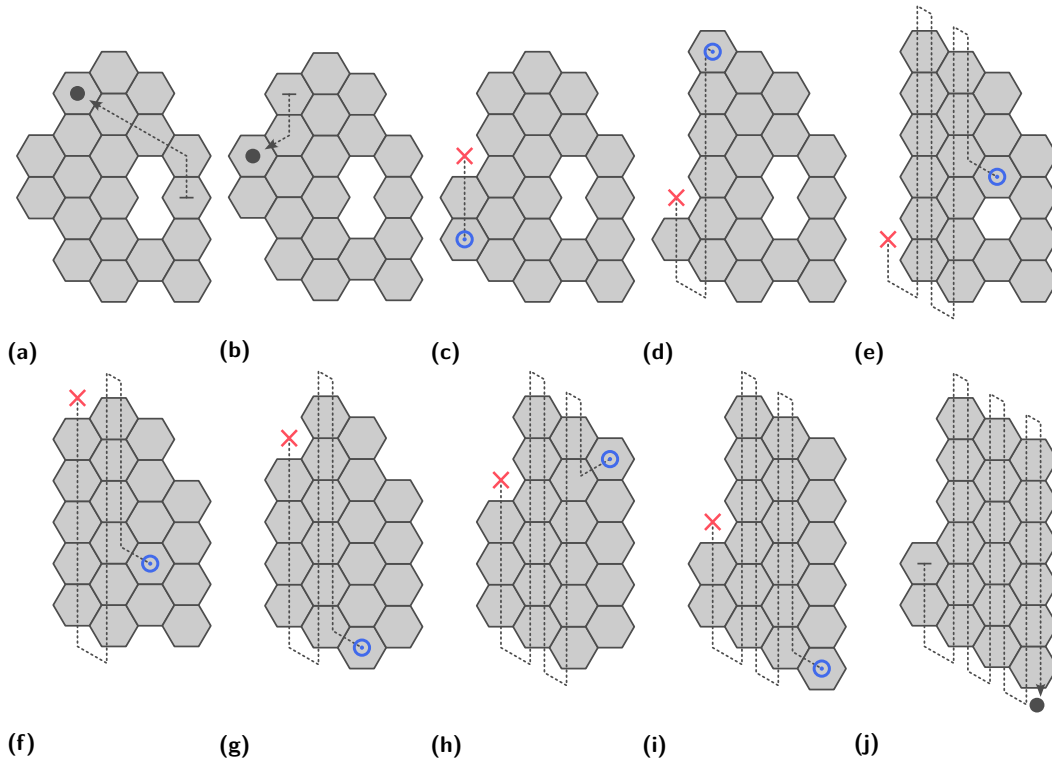
We assign x, y and z coordinates to each node $v \in V$, denoted by $c(v) = (x(v), y(v), z(v))$, where the x -coordinates grow from SE to NW, y -coordinates from S to N, and z -coordinates from DSW to UNE (see Figure 2a). The coordinates transition between neighbors as follows:

► **Observation 1.** Let w be some reference node with $c(w) = (0, 0, 0)$. The following holds:

$$\begin{array}{lll}
 c(w + \text{UNE}) = (0, 0, 1) & c(w + \text{UW}) = (1, -1, 1) & c(w + \text{USE}) = (0, -1, 1) \\
 c(w + \text{N}) = (0, 1, 0) & c(w + \text{NW}) = (1, 0, 0) & c(w + \text{SW}) = (1, -1, 0) \\
 c(w + \text{S}) = (0, -1, 0) & c(w + \text{SE}) = (-1, 0, 0) & c(w + \text{NE}) = (-1, 1, 0) \\
 c(w + \text{DSW}) = (0, 0, -1) & c(w + \text{DE}) = (-1, 1, -1) & c(w + \text{DNW}) = (0, 1, -1)
 \end{array}$$

Given some nodeset S , let x_{min}^S, x_{max}^S be the minimum and maximum x -coordinate of any node in S , and define $y_{min}^S, y_{max}^S, z_{min}^S$ and z_{max}^S accordingly. We normalize coordinates according to the minimum coordinates in the initial set of tiled nodes \mathcal{T}^0 , i.e., we set $x_{min}^{\mathcal{T}^0} = y_{min}^{\mathcal{T}^0} = z_{min}^{\mathcal{T}^0} = 0$. The *bounding cylinder* $\mathfrak{C}(S)$ is the set of all nodes (both empty and tiled) whose coordinates are bounded by the minimum and maximum x - and y -coordinates in S , i.e., $\mathfrak{C}(S) = \{v \in V \mid x_{min}^S \leq x(v) \leq x_{max}^S, y_{min}^S \leq y(v) \leq y_{max}^S\}$ (see Figure 2b). Similarly, in the *bounding box* $\mathfrak{B}(S)$ we further bound by the z -coordinate, i.e., $\mathfrak{B}(S) = \{v \in \mathfrak{C}(S) \mid z_{min}^S \leq z(v) \leq z_{max}^S\}$. We refer to the extent of a bounding box along the x -, y - and z -axes as its *width*, *height*, and *depth*. Note that by the choice of our coordinate axes, the bounding box is always a filled (potentially degenerated) parallelepiped (a 3D rhomboid; see Figure 2c). A node v is *inside* the bounding cylinder (box) of S , if $v \in \mathfrak{C}(S)$ ($v \in \mathfrak{B}(S)$); otherwise, v is *outside* of the bounding cylinder (box) of S .

A *layer* L_i is the set of all nodes with z -coordinate i that are contained in the bounding cylinder of all tiled nodes, i.e., $L_i = \{v \in \mathfrak{C}(\mathcal{T}) \mid z(v) = i\}$. We refer to nodes with z -coordinate greater than and less than i as the nodes *above* and *below* layer L_i , respectively. The nodeset of a connected component of $G|_{L_i \cap \mathcal{T}}$ is called a *fragment (of L_i)* (see Figure 2).



■ **Figure 3** The parallelogram formation algorithm on a 2D configuration. The agent performs multiple steps between each depicted configuration. In (a) and (b) the agent finds a westernmost column, and in (j) the agent terminates. In all other cases, a tile is shifted from the cross to the circle, where the dashed lines indicate the path traversed before placing the tile. The path back to where the tile is picked up as well as the movement to the next column (e.g., (e)–(f)) is not shown.

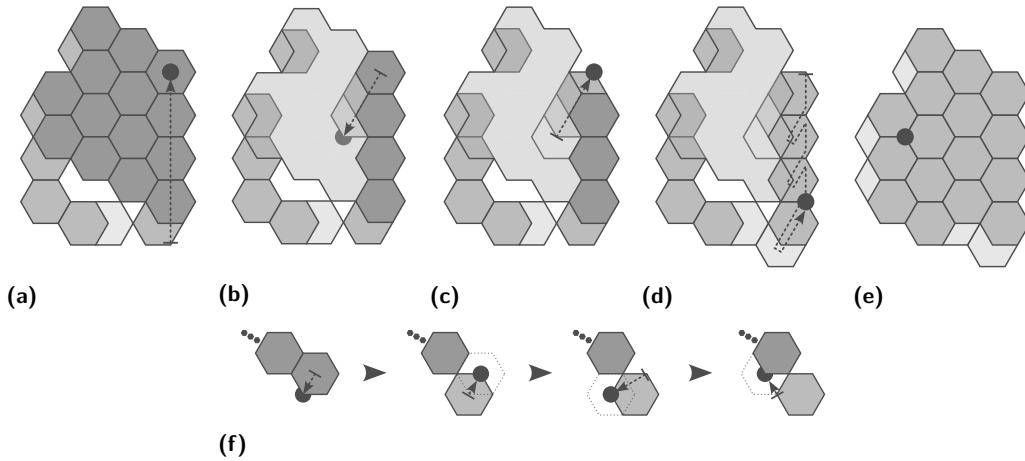
3 The Algorithm

From a high-level perspective, the agent iteratively transforms locally uppermost fragments into partially filled parallelograms. This involves rearranging tiles within the same layer and, at times, positioning tiles below the current layer to ensure connectivity. Whenever the agent encounters tiles of some layer above, it moves further upwards. Once a parallelogram is successfully formed, the subsequent step entails its *projection*. Essentially, during this projection, each tile in the fragment is shifted to the first empty node in the DSW direction.

In the following, we provide detailed textual descriptions of the parallelogram formation and projection procedures `BUILDPAR` and `PROJECT`, as well as the full icicle formation algorithm `BUILDICICLE`. For completeness, their pseudocodes can be found in Appendix A.

3.1 A 2D Parallelogram Formation Algorithm

Refer to Figure 3 for an illustrative example of the algorithm in action. The algorithm initiates with the agent searching for a locally westernmost column. In configurations where multiple columns share the same x -coordinate and are locally westernmost, the agent prioritizes finding the northernmost among them. This is achieved by moving in the NW, SW, and N directions, prioritized in that order, until no more tile is encountered in any of these directions. Eventually the agent stops upon reaching the northernmost tiled node v of some column c . We refer to the steps involved in finding column c as the *search phase*.

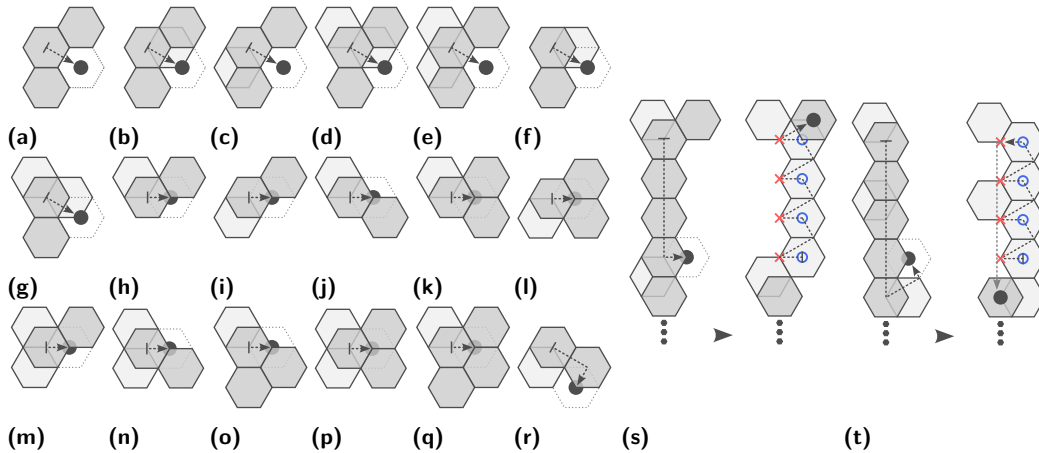


■ **Figure 4** During a projection, the agent (black disk) shifts each tile of a fragment in direction DSW. Detailed in (a-d) is the projection of a single column; (e) is a snapshot of the configuration after the projection. The special case of a parallelogram with a height of one is shown in (f). To maintain connectivity in that case, the agent moves SW + DNW to transition below the next column.

Subsequently, it executes the BUILDPAR procedure, which we describe in the following: Starting from node v , the agent traverses each column in the configuration from N to S. If, during the traversal of the first column c , the agent encounters either a more western column (as depicted in Figure 3b) or a column with the same x -coordinate as c to the north while moving N in the next column c' , it discontinues the current traversal and transitions to the search phase. Notably, in the latter case, it first fully traverses column c' in direction N and afterwards moves to the first column west of c' . This technical detail will play an important role in the runtime analysis. While traversing a column in the S direction, the agent actively looks for an empty node that *violates* the shape of a (partially filled) parallelogram with westernmost column c . Specifically, it checks the two empty nodes immediately above (excluding column c) and below each column, as well as each empty neighbor to the east of the column. Upon finding such a violating empty node w , the agent first places its carried tile at w and then returns to column c to retrieve the tile from v . Subsequently, this exchange of tiles is termed as a *tile shift* from v to w or as *shifting* (the tile) from v to w (recall that the agent initially carries a tile that was never placed at any node). After picking up the tile at v , the agent moves to an adjacent tile and transitions to the search phase again. The agent terminates at the empty node S of the easternmost column once the configuration is fully traversed without encountering any violating nodes. Any of the following conditions are sufficient for an empty node w to be considered violating: (1) w has a tile at N, NE and SE (e.g., Figure 3c), (2) w has a tile at S and SE (e.g., Figure 3d) and is not N of the westernmost column (recall that we allow the parallelogram to be partially filled), (3) w has a tile at NW and N (e.g., Figures 3e–3g and 3i), (4) w has a tile at NW, SW and S (e.g., Figure 3h).

3.2 An Icicle Formation Algorithm

From a high-level perspective, the construction of an icicle involves the iterative transformation of a locally uppermost fragment into a parallelogram, followed by a projection of the fragment in the DSW direction. When applying the parallelogram construction algorithm in a 2D configuration, the agent can always shift the tile at the northernmost node v of a locally westernmost column without violating connectivity (Figure 5a illustrates that connectivity



■ **Figure 5** Illustrating all scenarios in which the northernmost node of a locally westernmost column is not removable. For brevity, (a—r) only illustrate the agent’s movement (indicated by arrows) to the empty node (with a dashed outline) that is tiled next; (s) and (t) also portray the subsequent tile shifts. In (r), the agent may alternatively enter BUILDPAR if the outlined node were tiled. Note that (s) and (t) only show instances where a tile at DSW (s) and DE (t) is encountered.

is preserved in the only critical 2D case). In a 3D configuration, the situation becomes more intricate. There are multiple cases in which the tile at v must remain in its immediate neighborhood to avoid violating connectivity. Additionally, there is a case in which the tile at v cannot be moved at all unless neighboring tiles are also moved. We categorize these cases based on specific properties of node v , which we define as follows:

► **Definition 1.** Let $v \in \mathcal{T}$ be an arbitrary tiled node. Denote by $N(v)$ the neighborhood of v (excluding v), and by $N_{\mathcal{T}}(v)$ its subset of only tiled nodes. Node v is *removable*, if the tiled neighbors of v are locally connected, i.e., $G|_{N_{\mathcal{T}}(v)}$ is connected. Node v is *shiftable*, if $G|_{N_{\mathcal{T}}(v)}$ is disconnected and there exists a node $w \neq v$ (termed *bridge node of v*) for which $G|_{N_{\mathcal{T}}(v) \cup \{w\}}$ is connected. Any node that is neither removable nor shiftable is termed *unmovable*.

We now state the full icicle algorithm: The agent starts in the search phase where it repeatedly moves UW, USE, UNE, NW, SW and N until it eventually stops at some node v .

If node v is removable, the parallelogram traversal procedure BUILDPAR is entered. There are three possible outcomes: the agent returns from the procedure after finding a more western column or some tile above, after placing a tile, or at the empty node s of the fragment’s easternmost column. In the first case, the agent transitions to the search phase. In the second case, the agent first moves back to pick up the tile at node v , then moves to the next tile at s or SE , and afterwards transitions to the search phase. In the third case, the current fragment forms a correctly shaped parallelogram and the agent proceeds by executing the PROJECT procedure. During PROJECT, each tile of the fragment is projected in the DSW direction. Starting with the easternmost column, tiles are projected columnwise from east to west and within the columns from N to S (see Figures 4a–4e). Let v_0, \dots, v_k be the nodes of the currently projected column ordered from N to S. For each $i = 0, \dots, k$, the agent performs a tile shift from v_i to the first empty node w_i in direction DSW of v_i . After picking up the last tile of the column at v_k , the agent moves NW and continues the projection in the western neighboring column. In the special case of a degenerated parallelogram with a height of one, after picking up a tile, the agent moves SW and DNW instead (see Figure 4f). These additional steps ensure that connectivity is maintained during the projection. Once the last tile of the fragment is projected, the agent transitions to the search phase in the layer below.

15:10 Efficient Shape Formation by 3D Hybrid Programmable Matter

Otherwise, if the agent stops at a non-removable node v , it acts according to the case distinction outlined below, prioritized in the given order (refer to Figure 5 for a graphical overview). Subsequently, the agent transitions to the search phase, concluding our algorithm.

- ▶ **Case A.** If $v + SE$ is a bridge node of v (thereby v is shiftable and $v + SE$ is empty), then shift the tile from v to $v + SE$ (see Figures 5a–5g), and move to $v + S$ afterwards.
- ▶ **Case B.** If $v + DE$ is a bridge node of v , and at least one neighboring tile is not at DSW or SE , then shift the tile from v to $v + DE$ (see Figures 5h–5q), and move to the first tile at $v + S$, $v + SE$ or $v + NE$. Additionally, if $v + S$ is empty and both $v + SE$ and $v + NE$ are tiled, then traverse the next column starting at $v + SE$ in direction S . If during that traversal a tile at UW , USE or SW is encountered, then immediately transition to the search phase.
- ▶ **Case C.** If the only tiled neighbors of v are at DSW and SE , then move SE and observe node $w = v + SE + DSW$. If w is empty, then shift the tile from v to w (see Figure 5r), and move to $v + SE$ afterwards. Otherwise, if w is already tiled, move back to v and enter `BUILDPAR`.
- ▶ **Case D.** If the only tiled neighbors of v are at DNW , S and NE (v is unmovable, see Figure 5s), then follow these steps: First, move S until some node w is entered that has a neighboring tile at UW , USE , SW , DSW , SE or DE , or until there is no more tile in direction S . If w has a tile at UW , USE or SW , then immediately transition to the search phase. Otherwise, shift each tile in the column that is somewhere N of w in direction DE (including w if $w + DE$ is empty). To be precise, let $v_k, v_{k-1}, \dots, v_1, v$ be the nodes of the column ordered from S to N starting at $v_k = w + N$ (or $v_k = w$ if $w + DE$ is empty). Perform a tile shift from v_i to $v_i + DE$ for each i with $k \geq i > 0$. After the tile shift at v_i with $i > 0$, move $NE + DNW$ to be positioned at $v_{i-1} + DE$ (to preserve connectivity). Once the final tile is picked up, move to $v + NE$.
- ▶ **Case E.** If the only tiled neighbors of v are at DNW and S (see Figure 5t), then proceed analogously to the previous case, with the exception that tiles at DSW are disregarded. Additionally, make the following adaptations: If no tile at UW , USE , SW , SE or DE is encountered, then project the whole column (which is a parallelogram of width one) in direction DSW . Otherwise, after performing the final tile shift in direction DE , repeatedly move S (on empty nodes) and enter the first tiled node at S or SE (which must exist since we did not project).

The following remarks aim to clarify the choices made in the above case distinction: In case C, the node $v + DE$ serves as a bridge node for v ; however, the agent takes an additional step by attempting to shift the tile to $v + SE + DSW$. This decision stems from the fact that $v + DE$ is not within the bounding cylinder of tiles observable from node v . Similarly, in case D, $v + DSW$ serves as a bridge node for v . Although $v + DSW$ is within the bounding cylinder of observable tiles, it shares the same x - and y -coordinates as v . It is essential to our analysis that tiles are never placed outside of the bounding cylinder, and that, except for projections, tiles consistently advance to the east or south. In case B, the agent removes the last tile of some column c and instead of immediately transitioning to the search phase, it first traverses the next column c' in the S direction. Similarly to the `BUILDPAR` procedure, where the agent first traverses the next column c' fully in direction N whenever a more northern column of the same x -coordinate as c is found, this additional traversal is crucial for the runtime analysis. To elaborate, if column c' has multiple adjacent columns to the west, then directly entering the search phase would result in repeatedly traversing the same tiles within column c' . However, with the additional traversal in direction S in case B (and in direction N in procedure `BUILDPAR`), we can ensure that each tile of c' is visited only a constant number of times whenever the agent does not currently perform a tile shift.

4 Analysis

Due to space constraints, the proofs of our lemmas are only sketched here. Complete proofs can be found in the full version of the paper [15]. In the following, $C^i = (\mathcal{T}^i, p^i)$ denotes the configuration that results from the execution of BUILDICICLE for i steps. We start by showing that our algorithm complies with the connectivity constraint of the 3D hybrid model.

► **Lemma 2.** *If the agent disconnects $G|_{\mathcal{T}^i}$ in step i , then $G|_{\mathcal{T}^{i+4}}$ is connected, and for all $i < j < i + 4$: $G|_{\mathcal{T}^j \cup \{p^j\}}$ is connected and the agent carries a tile.*

One can show that $G|_{\mathcal{T}^i}$ can only disconnect during the projection of a parallelogram of height one, and during consecutive tile shifts in cases D and E. These are the cases for which we explicitly preserve connectivity by moving not on, but instead adjacent to tiled neighbors.

► **Lemma 3.** *If during the execution of BUILDICICLE a tile is shifted from some node v to some node w , then there are tiled nodes $u_x, u_y \in \mathcal{T}$ with $x(w) = x(u_x)$ and $y(w) = y(u_y)$.*

The lemma is proven through an extensive case distinction that includes the four conditions under which a node is tiled during the BUILDPAR procedure, as well as any tile shift that may result when the agent exits the search phase at a node v that is not removable. There are at most $2^6 = 64$ such cases, since v can have tiled neighbors in at most six directions. We argue that the 20 cases depicted in Figure 5 are complete by providing 44 distinct neighborhoods for which v is removable. Similarly, we can prove the following lemma by considering cases where a tile is picked up instead of dropped.

► **Lemma 4.** *For each $i \geq 0$ there is a tiled node $v \in \mathcal{T}^i$ with $x(v) = 0$.*

We want to measure the progress of tiles within the bounding cylinder towards the east and south by considering their x - and y -coordinates. As part of the BUILDPAR procedure and cases B, D, and E, the y -coordinate of tiles can increase when their x -coordinate decreases. Although the size of the bounding cylinder cannot increase by Lemma 3, it may decrease. In such instances, by Lemma 4, the resulting bounding cylinder always aligns with the eastern side of the initial bounding cylinder $\mathfrak{C}(\mathcal{T}^0)$. To address this, we introduce a combined representation of the x - and y -coordinates w.r.t. the bounding cylinder $\mathfrak{C}(\mathcal{T})$ for arbitrary \mathcal{T} .

Let $y_{max}^{\mathcal{T}}$ and $y_{min}^{\mathcal{T}}$ be the maximum and minimum y -coordinates within $\mathfrak{C}(\mathcal{T})$, and let $h = y_{max}^{\mathcal{T}} - y_{min}^{\mathcal{T}} + 1$ be the *height* of $\mathfrak{C}(\mathcal{T})$, i.e., the cylinder's extent along the y -axis. We define the *xy-coordinate* of some node $v \in \mathfrak{C}(\mathcal{T})$ as $xy(v) = x(v) \cdot h + y(v) - y_{min}^{\mathcal{T}}$.

Consider the following definitions, which we refer to as P1–P3, that relate to some fragment $F \subseteq \mathcal{T}$. We show that at some point any configuration contains a fragment that fulfills P1–P3. Afterwards, we show that this configuration converges to an icicle.

► **Definition 5.** *Let $F \subset \mathcal{T}$ be an arbitrary fragment.*

- P1: F is a platform, if $\{v + X \mid v \in F, X \in \{UW, USE, UNE\}\} \cap \mathcal{T}$ is an empty set.
- P2: F is covering, if for each node $v \in \mathcal{T}$ there is a node $w \in F$ with $xy(w) = xy(v)$.
- P3: F is an aligned parallelogram, if for each node $v \in F$ it holds that for all i with $xy(v) \geq i \geq 0$ there is a node $w \in F$ with $xy(w) = i$.

P1 characterizes a locally uppermost fragment, P2 a fragment covering the xy -coordinates of all tiled nodes, and P3 a fragment wherein tiles have the shape of a parallelogram aligned along the southern, eastern, and northern sides of the bounding cylinder. We can now use P1–P3 to give an alternative definition of the icicle shape.

15:12 Efficient Shape Formation by 3D Hybrid Programmable Matter

► **Definition 6.** A Configuration $C = (\mathcal{T}, p)$ is an icicle, if it contains a fragment F that satisfies P1–P3, and for any node $v \in \mathcal{T} \setminus F$ it holds that $v + \text{UNE} \in \mathcal{T}$.

Any tiled node that is not contained in the fragment F specified in Definition 6, must be somewhere $\text{DSW} = -\text{UNE}$ of F , as otherwise the number of tiles would be infinite. Hence, each node $v \in \mathcal{T} \setminus F$ is contained in a tower of tiles whose uppermost tile is contained in F , and thereby Definition 6 is equivalent to our definition of an icicle from Section 1.4.

Subsequently, we only consider configurations in which the agent leaves the search phase at some node v . This must eventually occur since moving upwards increases its z -coordinate, and moving in directions SW, NW, or N increases its xy -coordinate. Both coordinates are bounded within any finite set of tiled nodes. To simplify notation, we use $C^i = (\mathcal{T}^i, p^i)$ to represent the configuration where the agent leaves the search phase for the i -th time.

Consider the potential function $\Phi^i = \sum_{v \in \mathcal{T}^i} xy(v) + |\mathcal{P}^i|$, where \mathcal{P}^i denotes the set of all platforms, i.e., fragments satisfying P1. We first show its monotonicity and lower bound.

► **Lemma 7.** For each $i \geq 0$ it holds that $\Phi^i \geq \Phi^{i+1} \geq 0$, and if $\Phi^i = \Phi^{i+1}$, then (1) no tile was shifted between step i and $i + 1$, or (2) a fragment was projected between step i and $i + 1$.

The lemma mostly follows from the observation that within procedure BUILDPAR tiles are visited in decreasing order of their xy -coordinates, and that each tile shift in cases A–E decreases the x -coordinate of at least one tile. The number of platforms can only increase by one as a result of case D, which is compensated by two tiles with decreasing x -coordinate.

► **Lemma 8.** If $p^i \in F^i$ where F^i is a fragment in C^i that satisfies P1–P3, then $p^{i+1} \in F^{i+1}$ where F^{i+1} is a fragment in C^{i+1} that satisfies P1–P3.

The proof of the previous lemma is straight forward. Since F^i satisfies P1, the agent cannot leave F^i to a layer above. Since it satisfies P2 and cases A–E necessitate a tile below that is not covered by F^i , the agent must enter BUILDPAR within F^i . Finally, since F^i satisfies P3, the agent must project F^i directly after the BUILDPAR procedure. As a result, F^{i+1} is a direct copy of F^i in direction DSW which also satisfies P1–P3.

► **Lemma 9.** For each $i \geq 0$ there is a step $j > i$ such that (1) $\Phi^i > \Phi^j$ or (2) $p^j \in F^j$ where F^j is a fragment of configuration C^j that satisfies P1–P3.

If (2) holds in step i , the lemma follows from Lemma 8. Otherwise, by Lemma 7, either no tile was shifted or a projection was performed between step i and $i + 1$. In the first case, the agent must have progressed further west or upwards, which can happen only finitely many times since the configuration is finite. In the second case, we can show that after finitely many consecutive projections the agent is positioned in a fragment of larger size. Analogously the size of that fragment is bounded by the number of tiles, such that eventually either the potential decreases again or the latter statement (2) holds, concluding the lemma.

The initial number of platforms is at most n , and the initial xy -coordinate of any tiled node is at most n^2 . Hence, the initial potential is $\Phi^0 = \mathcal{O}(n^3)$. Consequently, Lemma 8 and Lemma 9 imply that eventually the agent is positioned in a fragment satisfying P1–P3.

For the second part of our analysis, dedicated to demonstrating convergence towards an icicle, we introduce another potential function Ψ^i . This function is defined as the number of empty nodes within the bounding box $\mathfrak{B}(\mathcal{T}^i)$ that have a tile somewhere in the DSW direction. Formally, $U^i = \{v \in \mathfrak{B}(\mathcal{T}^i) \setminus \mathcal{T}^i \mid v + k \cdot \text{DSW} \in \mathcal{T}^i \text{ for some } k > 0\}$ and $\Psi^i = |U^i|$.

► **Lemma 10.** Let $p^i \in F^i$ where F^i satisfies P1–P3. If $\Psi^i > 0$, then $\Psi^{i+1} < \Psi^i$.

The lemma is proven by showing that the projection of F^i results in at least one node in U^i to be tiled. Here, we can simply pick the node with maximum z -coordinate from U^i . Additionally, any node in U^{i+1} must already be contained in U^i , since the projection of F^i cannot create an empty node that has a tile somewhere UNE. Otherwise that tile would contradict that F^i is covering (P2) and a platform (P1).

Once the agent enters a configuration where it is positioned within a fragment satisfying P1–P3, it consistently remains within such a fragment in subsequent configurations according to Lemma 8. By Lemma 9, such a configuration must eventually be reached, and by the previous lemma, our second potential Ψ^i is strictly monotonically decreasing afterwards. It follows that the set of empty nodes within $\mathfrak{B}(\mathcal{T}^i)$ that have a tile somewhere in the DSW direction must eventually be empty. Consequently, any tiled node within the bounding box that is not contained in the singular uppermost fragment satisfying P1–P3 must possess a neighboring tile at UNE. Hence, the entire configuration satisfies Definition 6, which is captured by the following theorem, serving as the conclusion of our analysis:

► **Theorem 11.** *The sequence of configurations resulting from the execution of BUILDICICLE on any initially connected configuration $C^0 = (\mathcal{T}^0, p^0)$ with $p^0 \in \mathcal{T}^0$ converges to an icicle.*

5 Termination Criteria and Runtime

Once the agent is positioned within a fragment satisfying P1–P3, it remains within such a fragment and subsequently exclusively performs projections. In the case where the configuration is already an icicle (see Definition 6), every tiled node in the configuration must be traversed during these projections. This condition is essential for our termination check. The agent maintains a flag *term*, which it flags as true upon initiating a projection. This flag only reverts to false if the agent detects any violation of Definition 6 during the ongoing projection, i.e., whenever a tiled node v is observed for which $v + \text{UNE} \notin F$ and $v + \text{UNE} \notin \mathcal{T}$, where F is the fragment in which the projection was initiated. Once *term* still holds after a projection, the agent terminates. Note that the flag is reverted, even if $v + \text{UNE}$ is tiled immediately afterwards. As an example, if a tile shift from some node $w \in F$ to $v + \text{UNE}$ is performed as part of a projection, then node v is observed before the tile is placed at $v + \text{UNE}$. Although it is possible that the configuration is an icicle after tiling node $v + \text{UNE}$, the agent cannot verify it during that projection, as it does not traverse node v or any node DSW of v .

In general, by adhering to this termination procedure, the agent consistently performs one additional projection once the configuration converges to an icicle. Since the algorithm only terminates following a projection in which it could observe all tiled nodes and only if, in this case, Definition 6 is satisfied, the correctness of our algorithm is established.

The algorithm’s runtime can be expressed as the sum $t_{total} = t_{proj} + t_{shift} + t_{move}$, where t_{proj} accounts for all steps performed during the projection subroutine, t_{shift} for steps that are performed as part of some tile shift (outside of a projection), and t_{move} for any remaining step. We bound each term individually by $\mathcal{O}(n^3)$, which gives a runtime of $\mathcal{O}(n^3)$ in total.

► **Lemma 12.** *The total number of steps performed during projections is $t_{proj} = \mathcal{O}(n^3)$.*

The proof can be outlined as follows: Each projection takes time $\mathcal{O}(n)$. There are $\mathcal{O}(n)$ platforms initially, each of which require $\mathcal{O}(n)$ projections until the number $|\mathcal{P}|$ of platforms reduces by one. Additional platforms can only be created as a result of the execution of case D. For these platforms one can show that a single projection suffices to reduce the number of platforms. Additionally, the execution of case D decreases the x -coordinate of at least two tiles, which implies that at most $\frac{n^2}{2}$ platforms can be created.

► **Lemma 13.** *Let i be the first step following an arbitrary projection, and $j > i$ the next step in which a projection is initiated. Between step i and j at most $\mathcal{O}(n)$ steps are performed outside of tile shifts, and any tile shift from some node v to w takes $\mathcal{O}(xy(v) - xy(w))$ steps.*

The latter statement is easy to show. Especially, any tile shift in cases A–E requires only $\mathcal{O}(1)$ steps, but reduces the xy -coordinate of some tile by $\Omega(h)$. For the former statement, one must consider all cases in which the agent moves outside of tile shifts. The most challenging case is where the agent lifts the last tile of a column and then takes $\mathcal{O}(h)$ steps afterward. If that occurs in case B, then the above outlined difference between $\mathcal{O}(1)$ steps and a reduction of $\Omega(h)$ in the xy -coordinate accounts for the $\mathcal{O}(h)$ additional steps. If a column is removed in the BUILDPAR procedure, one can show that either the previous or the subsequent tile shift decreases the xy -coordinate of a tile by $\Omega(h)$, and the claim follows analogously.

As argued above, $\mathcal{O}(n^2)$ projections are performed in total, which together with Lemma 13 implies that $t_{other} = \mathcal{O}(n^3)$. The xy -coordinate of any tile is at most n^2 , non-increasing, and cannot be negative (see Lemmas 3 and 7). Together with Lemma 13, each tile contributes $\mathcal{O}(n^2)$ steps to t_{shift} , which implies that $t_{shift} = \mathcal{O}(n^3)$. This concludes our final theorem:

► **Theorem 14.** *BUILDICICLE has a runtime of $\mathcal{O}(n^3)$ steps.*

6 Experimental Analysis

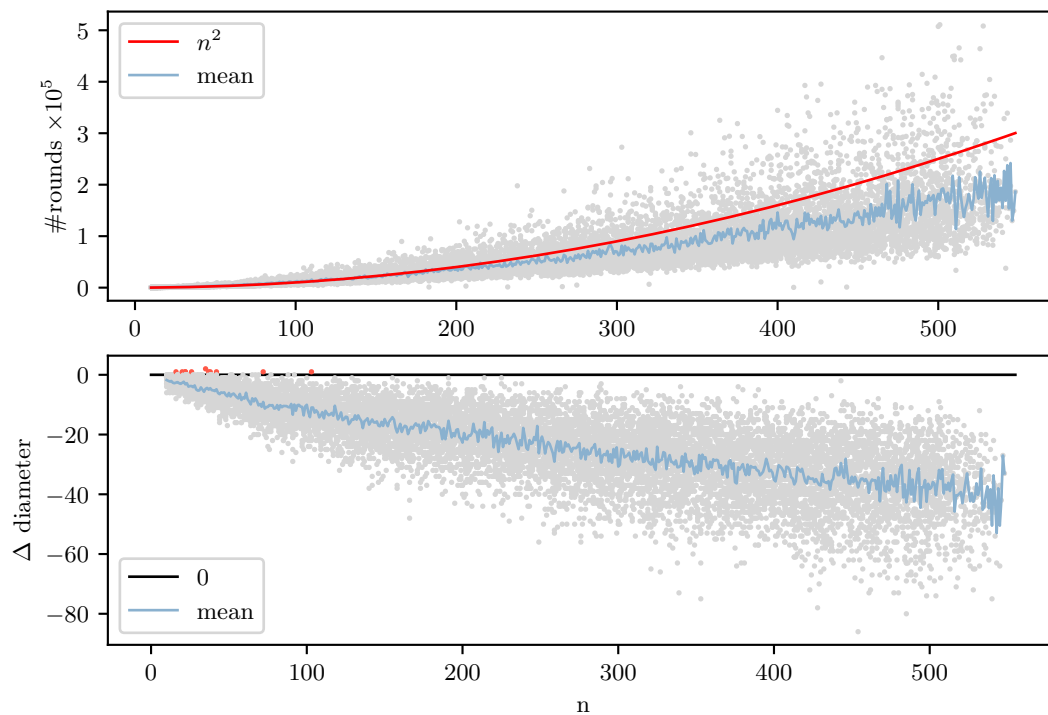
While our algorithm matches the runtime bound of the 3D line formation algorithm [16], the icicle offers distinct advantages over the line. The diameter of an icicle can be as low as $\mathcal{O}(n^{\frac{1}{3}})$, whereas a line consistently maintains a diameter of n . Unfortunately, our algorithm does not improve the diameter if the initial configuration already closely resembles a line. On the other hand, we conjecture that if the initial diameter is as low as $\mathcal{O}(n^{\frac{1}{3}})$ (which is the best case in 3D), then our algorithm can only increase the diameter by a factor of $\mathcal{O}(n^{\frac{1}{3}})$. We support our conjecture with the following simulation results on configurations where initially all tiles are contained in a sphere of radius $\mathcal{O}(n^{\frac{1}{3}})$.

We conducted a total of 12,250 simulations using the icicle formation algorithm on random configurations. For each value of n within the range $10 \leq n < 500$, we sampled 25 random configurations as follows: empty nodes were repeatedly chosen uniformly at random within a sphere of radius $4n^{\frac{1}{3}}$, and a tile was placed on each selected node until a connected component of tiled nodes with a size of at least n was formed. Subsequently, any tile outside of that component was removed, the agent was placed at a randomly chosen tile within the component, and the algorithm was simulated until termination. We measured the runtime as well as the difference in diameter, which are plotted in Figure 6.

Due to the nature of the described random generation process, configurations of size larger than 500 were also sampled, although less frequently. Specifically, we observed an average sampling rate of approx. 24.4 configurations for sizes at most 450, contrasting with approx. 15.4 configurations for sizes exceeding 450. This discrepancy contributes to the noticeable increase in variance as the configuration size approaches the 500 threshold.

The runtime remains well in the vicinity of n^2 , which can be attributed to the initial close packing of tiles in our random configurations. Instances where the diameter increases (highlighted by red dots) are infrequent, and their occurrence diminishes as the configuration size increases. This trend implies a general decrease in diameter in the average case.

We identified a configuration with an initial diameter of $\mathcal{O}(n^{\frac{1}{3}})$, where the diameter subsequently increases by a factor of $\Theta(n^{\frac{1}{3}})$. This particular configuration, which we consider to be the worst-case scenario, is discussed in Appendix B.



■ **Figure 6** The results stem from 12,250 simulations involving random configurations ranging in sizes from 10 to 550. The upper plot shows the number of steps until termination. The lower plot shows the difference in diameter between the input and output configurations. The simulations in which the diameter increases are highlighted by red dots.

7 Future Work

In this work, we introduced an algorithm capable of transforming any initially connected configuration into an icicle within $\mathcal{O}(n^3)$ steps, complemented by proofs of correctness and runtime analysis. While our algorithm’s experimental results are promising, future work should include a formal proof to substantiate the claimed upper bound of $\mathcal{O}(n^{\frac{1}{3}})$ on the increase in diameter. Additionally, the adaptability of our algorithm to the multi-agent case poses an intriguing challenge for future investigation. Given that the algorithm comprises distinct phases potentially executed in an interleaved manner, addressing its integration into a multi-agent framework presents a non-trivial research direction.

References

- 1 M. Akter, J. J. Keya, K. Kayano, A. M. R. Kabir, D. Inoue, H. Hess, K. Sada, A. Kuzuya, H. Asanuma, and A. Kakugo. Cooperative cargo transportation by a swarm of molecular machines. *Science Robotics*, 7(65), 2022. doi:10.1126/scirobotics.abm0677.
- 2 D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4), March 2006. doi:10.1007/s00446-005-0138-3.
- 3 M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, 1978. doi:10.1109/SFCS.1978.30.

- 4 J. Chao, J. Wang, F. Wang, X. Ouyang, E. Kopperger, H. Liu, Q. Li, J. Shi, J. hu, L. Wang, W. Huang, F. Simmel, and C. Fan. Solving mazes with single-molecule dna navigators. *Nature Materials*, 18, March 2019. doi:10.1038/s41563-018-0205-3.
- 5 H. Chen, C. Li, M. Mafarja, A. A. Heidari, Y. Chen, and Z. Cai. Slime mould algorithm: a comprehensive review of recent variants and applications. *International Journal of Systems Science*, 54(1), 2023. doi:10.1080/00207721.2022.2153635.
- 6 G.S. Chirikjian. Kinematics of a metamorphic robotic system. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994. doi:10.1109/ROBOT.1994.351256.
- 7 Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Amoebot - a new model for programmable matter. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, 2014. doi:10.1145/2612669.2612712.
- 8 Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication*, 2015. doi:10.1145/2800795.2800829.
- 9 Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal shape formation for programmable matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, 2016. doi:10.1145/2935764.2935784.
- 10 G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. Shape formation by programmable particles. *Distrib. Comput.*, 33(1), February 2020. doi:10.1007/s00446-019-00350-6.
- 11 P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. In *Mathematical Foundations of Computer Science 2004*, 2004.
- 12 R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, and C. Scheideler. Shape recognition by a finite automaton robot. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117, 2018. doi:10.4230/LIPIcs.MFCS.2018.52.
- 13 R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler, and T. Strothmann. Forming tile shapes with simple robots. *Natural Computing*, 19(2), June 2020. doi:10.1007/s11047-019-09774-2.
- 14 A. Heuer-Jungemann and T. Liedl. From dna tiles to functional dna materials. *Trends in Chemistry*, 1(9), 2019. doi:10.1016/j.trechm.2019.07.006.
- 15 K. Hinnenthal, D. Liedtke, and C. Scheideler. Efficient shape formation by 3d hybrid programmable matter: An algorithm for low diameter intermediate structures, 2024.
- 16 K. Hinnenthal, D. Rudolph, and C. Scheideler. Shape formation in a three-dimensional model for hybrid programmable matter. In *Proc. of the 36th European Workshop on Computational Geometry (EuroCG 2020)*, 2020.
- 17 F. Hoffmann. One pebble does not suffice to search plane labyrinths. In *International Conference on Fundamentals of Computation Theory*, 1981.
- 18 F. Hurtado, E. Molina, S. Ramaswami, and V. Sacristán. Distributed reconfiguration of 2d lattice-based modular robotic systems. *Autonomous Robots*, 38(4), April 2015. doi:10.1007/s10514-015-9421-8.
- 19 I. Kostitsyna, D. Liedtke, and C. Scheideler. Universal coating in the 3d hybrid model, 2023. doi:10.48550/arXiv.2303.16180.
- 20 I. Kostitsyna, C. Scheideler, and D. Warner. Fault-tolerant shape formation in the amoebot model. In *28th International Conference on DNA Computing and Molecular Programming (DNA 28)*, 2022. doi:10.4230/LIPIcs.DNA.28.9.
- 21 G. Li, D. St-Onge, C. Pinciroli, A. Gasparri, E. Garone, and G. Beltrame. Decentralized progressive shape formation with robot swarms. *Autonomous Robots*, 43(6), August 2019. doi:10.1007/s10514-018-9807-5.

- 22 H. Li, J. Gao, L. Cao, X. Xie, J. Fan, H. Wang, H. Wang, and Z. Nie. A dna molecular robot autonomously walking on the cell membrane to drive the cell motility. *Angewandte Chemie International Edition*, 60, September 2021. doi:10.1002/anie.202108210.
- 23 N. Nokhanji, P. Flocchini, and N. Santoro. Fully dynamic line maintenance by hybrid programmable matter. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2022. doi:10.1109/IPDPSW55747.2022.00087.
- 24 A. Padalkin, M. Kumar, and C. Scheideler. Shape formation and locomotion with joint movements in the amoebot model. *ArXiv*, abs/2305.06146, 2023.
- 25 M. J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2), June 2014. doi:10.1007/s11047-013-9379-4.
- 26 T. Peters, I. Kostitsyna, and B. Speckmann. Fast reconfiguration for programmable matter. In *37th International Symposium on Distributed Computing, DISC 2023*, 2023. doi:10.4230/LIPIcs.DISC.2023.27.
- 27 N. Tan, A. A. Hayat, M. R. Elara, and K. L. Wood. A framework for taxonomy and evaluation of self-reconfigurable robotic systems. *IEEE Access*, 8, 2020. doi:10.1109/ACCESS.2020.2965327.
- 28 P. Thalamy, B. Piranda, and J. Bourgeois. A survey of autonomous self-reconfiguration methods for robot-based programmable matter. *Robotics and Autonomous Systems*, 120, 2019. doi:10.1016/j.robot.2019.07.012.
- 29 A. J. Thubagere, W. Li, R. F. Johnson, Z. Chen, S. Doroudi, Y. L. Lee, G. Izatt, S. Wittman, N. Srinivas, D. Woods, E. Winfree, and L. Qian. A cargo-sorting dna robot. *Science*, 357(6356), 2017. doi:10.1126/science.aan6558.
- 30 T. Tucci, B. Piranda, and J. Bourgeois. A distributed self-assembly planning algorithm for modular robots. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018.
- 31 J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17(2), August 2004. doi:10.1007/s00446-003-0103-y.
- 32 H. Wang and M. Rubenstein. Shape formation in homogeneous swarms using local task swapping. *IEEE Transactions on Robotics*, 36(3), 2020. doi:10.1109/TR0.2020.2967656.
- 33 J. Werfel, K. Petersen, and R. Nagpal. Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172), 2014. doi:10.1126/science.1245842.
- 34 D. Woods, H. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, 2013. doi:10.1145/2422436.2422476.

A Deferred Pseudocode

The pseudocode for the 2D parallelogram formation algorithm, as detailed in Section 3.1, is given by Algorithm 1. Specifically, lines 12–34 within Algorithm 1 describe the BUILDPAR procedure, utilized by both the parallelogram and icicle formation algorithms. Note that the checks for tiles above (lines 13–14) can be disregarded in the 2D setting, as they only become relevant in the icicle formation algorithm. The PROJECT procedure is given in Algorithm 3, and the full icicle formation algorithm in Algorithm 2. Whenever multiple directions of movement are specified, their precedence is implicit in the provided order.

In Algorithm 1, the agent traverses a column in the S direction in lines 12–21 and the next column in the N direction in lines 26–31. Following the check for whether the empty node above the next column should be tiled (line 32), the agent recursively executes BUILDPAR starting at the N-most node of the next column (line 33). The procedure may return with the agent being in various states, such as positioned on a tiled or empty node, with or without a tile. In the main loop of the algorithm, BUILDPAR is executed repeatedly, and distinctions between these states are made to either terminate (line 4), retrieve the tile at which BUILDPAR was previously entered (lines 5–11), or enter the search phase (line 2).

In Algorithm 2, lines 16–22 are dedicated to handling case B, where a tile is shifted in the DE direction to maintain connectivity. Lines 23–29 cover case D, and lines 30–40 cover case E. These cases involve shifting multiple tiles of a column in the DE direction. For concise pseudocode, the handling of case A is delegated to the BUILDPAR procedure, and the check for a tile at $v + SE + DSW$ from case C is integrated into lines 4–5.

B Worst-Case Configuration

In the following, we present what we believe to be the worst-case configuration. Consider the configuration C depicted in Figure 7a that consists of three layers. The middle layer contains $k = \Theta(n^{\frac{1}{3}})$ fragments F_1, \dots, F_k ordered from east to west, where each F_i has size

■ **Algorithm 1** 2DPARALLELOGRAMFORMATION.

```

1 while true do
2   while  $\{p + NW, p + SW, p + N\} \cap \mathcal{T} \neq \emptyset$  do move to tile at NW, SW or N
3   firstColumn  $\leftarrow$  true; run BUILDPAR
4   if  $p \notin \mathcal{T}$  then return  $\triangleright$  terminate S of easternmost column
5   else if  $r$  carries no tile then
6     if firstColumn then
7       while  $p + N \in \mathcal{T}$  do move N
8     else
9       while  $\{p + SW, p + S\} \cap \mathcal{T} \neq \emptyset$  do move to tile at SW or S
10      while  $\{p + NW, p + SW, p + N\} \cap \mathcal{T} \neq \emptyset$  do move to tile at NW, SW or N
11      pickup tile; move to tile at S, SE or NE

  procedure BUILDPAR
12 while  $p \in T$  do
13   if  $p + UW \in \mathcal{T}$  or  $p + USE \in \mathcal{T}$  or  $p + UNE \in \mathcal{T}$  then  $\triangleright$  irrelevant in 2D
14     move to tile at UW, USE or UNE ; return
15   else if firstColumn and  $p + SW \in \mathcal{T}$  then
16     move SW; return  $\triangleright$  found more western column
17   else if  $p + NE \in \mathcal{T}$  and  $p + SE \notin \mathcal{T}$  then
18     move SE; place tile; move NW; return  $\triangleright$  place tile below eastern column
19   else if  $p + N, p + SE \in \mathcal{T}$  and  $p + NE \notin \mathcal{T}$  then
20     move NE; place tile; move SW; return  $\triangleright$  place tile above eastern column
21   move S
22 if  $p + N, p + NE, p + SE \in \mathcal{T}$  then
23   place tile; move N  $\triangleright$  place tile below current column
24 else if  $p + NE \in \mathcal{T}$  then
25   move NE; move N; firstColumn  $\leftarrow$  false  $\triangleright$  move to top of next column
26   while  $p \in \mathcal{T}$  do
27     if  $p + SW \notin \mathcal{T}$  and  $p + NW \in \mathcal{T}$  then
28       while  $p + N \in \mathcal{T}$  do move N
29       while  $p + NW \notin \mathcal{T}$  do move S
30       return  $\triangleright$  found more northern column
31     move N
32 if  $p + S, p + SE \in \mathcal{T}$  then place tile  $\triangleright$  place tile above current column
33 else move S; run BUILDPAR
34 return

```

Algorithm 2 BUILDICICLE.

```

1 while true do
2   while  $\{p + X \mid X \in \{UW, USE, UNE, NW, SW, N\}\} \cap T \neq \emptyset$  do
3     | move to tile at UW, USE, UNE, NW, SW or N
4   if  $G|_{N_{\mathcal{T}}(p)}$  or  $G|_{N_{\mathcal{T}}(p) \cup \{p+SE\}}$  is connected or  $G|_{N_{\mathcal{T}}(p) \cup \{p+SE+DSW\}}$  is connected
   with  $p + SE + DSW \in \mathcal{T}$  then
5     |  $firstColumn \leftarrow true$ ; run BUILDPAR
6     | if  $p \notin \mathcal{T}$  then move N; run PROJECT
7     | else if  $r$  carries no tile then ...  $\triangleright$  same as lines 8–15 from Algorithm 1
16  else if  $G|_{N_{\mathcal{T}}(p) \cup \{p+DE\}}$  is connected then
17    | if  $N_{\mathcal{T}}(p) = \{p + DSW, p + SE\}$  then
18      | move SE + DSW; place tile; move UNE + NW; pickup tile; move SE
19    | else move DE; place tile; move UW; pickup tile
20    | if  $p + SE, p + NE \in \mathcal{T}$  and  $p + S \notin \mathcal{T}$  then
21      | move SE; while  $\{p + UW, p + USE, p + SW, p + S\} \cap \mathcal{T} = \{p + S\}$  do move S
22    | else move to tile at S, SE or NE
23  else if  $N_{\mathcal{T}}(p) = \{p + DNW, p + S, p + NE\}$  then
24    | while  $\{p + X \mid X \in \{UW, USE, SW, DSW, SE, DE, S\}\} \cap \mathcal{T} = \{p + S\}$  do move S
25    | if  $\{p + X \mid X \in \{UW, USE, SW\}\} \cap \mathcal{T} = \emptyset$  then
26      | if  $p + DE \in \mathcal{T}$  then move N
27      | move DE; place tile; move UW; pickup tile
28      | while  $p + N \in \mathcal{T}$  do move SE + DNW; place tile; move UW; pickup tile
29      | move NE
30  else if  $N_{\mathcal{T}}(p) = \{p + DNW, p + S\}$  then
31    | while  $\{p + X \mid X \in \{UW, USE, SW, SE, DE, S\}\} \cap \mathcal{T} = \{p + S\}$  do move S
32    | if  $\{p + X \mid X \in \{UW, USE, SW\}\} \cap \mathcal{T} = \emptyset$  then
33      | if  $\{p + X \mid X \in \{SE, DE\}\} \cap \mathcal{T} = \emptyset$  then move N; run PROJECT
34      | else
35        | ...  $\triangleright$  same as lines 26–28
38      | while  $p \notin \mathcal{T}$  do
39        | | move S; if  $p + SE \in \mathcal{T}$  then move SE

```

$\mathcal{O}(i)$ and the agent's initial position is $p^0 \in F_1$. Additionally, the configuration contains a fragment F_0 of size one east of the agent's initial position. Observe that the bounding box of F_i contains no node from F_{i+1} for any i with $0 < i < k$. It follows that the agent builds and projects parallelograms in the order F_1, \dots, F_k . Since the bounding box of F_i contains p^0 for all $i > 0$, it further follows that k tiles are projected from p^0 in direction DSW. Only then, the agent traverses the lower layer and eventually finds fragment F_0 where it moves further upwards. Now consider the configuration that consists of $\Theta(n^{\frac{1}{3}})$ copies of C in direction UNE (see Figure 7b). That configuration has diameter $\mathcal{O}(n^{\frac{1}{3}})$ initially. Throughout the icicle formation algorithm, some tile at node p^0 is projected $\Theta(n^{\frac{2}{3}})$ times, which implies that the resulting icicle has depth and thereby also diameter $\Theta(n^{\frac{2}{3}})$.

Algorithm 3 PROJECT.

```

procedure PROJECT
1 if  $p + N, p + S \notin \mathcal{T}$  then  $\triangleright$  parallelogram of height one
2   do
3     while  $p \in \mathcal{T}$  do move DSW
4     place tile; while  $p + UNE \in \mathcal{T}$  do move UNE
5     pickup tile
6     if  $p + NW \in \mathcal{T}$  then move SW; move DNW else move DSW; return
7   while  $p + UNE \in \mathcal{T}$ 
8 else
9   do
10    while  $p + N \in \mathcal{T}$  do move N
11    while  $p \in \mathcal{T}$  do move DSW
12    place tile; while  $p + UNE \in \mathcal{T}$  do move UNE
13    pickup tile
14    if  $p + S \in \mathcal{T}$  then move S
15    else if  $p + NW \in \mathcal{T}$  then move NW
16    else move DSW; return
17  while  $p \in \mathcal{T}$ 
    
```

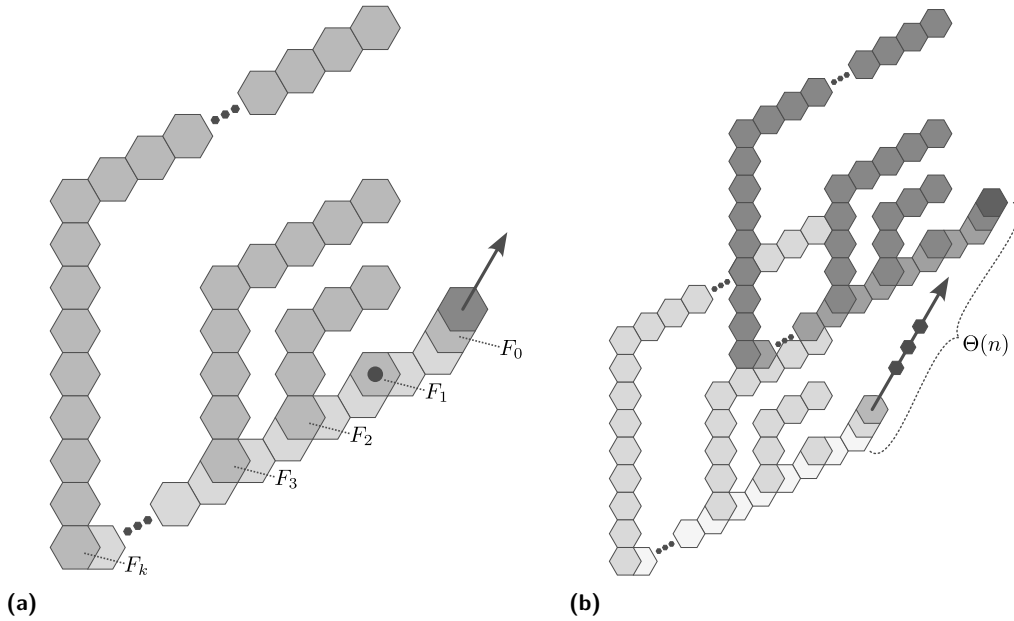


Figure 7 Illustrating what we believe to be the worst case configuration in terms of increase in diameter. In (a), the three lowest layers of the configuration are depicted in detail. The second-lowest layer contains $k = \Theta(n^{1/3})$ fragments F_i , each of size $\mathcal{O}(i)$, and the agent's initial position $p^0 \in F_1$. Observe that the bounding box of each F_i contains p^0 . The whole configuration is depicted in (b) and consists of $\Theta(n^{1/3})$ copies of the layers depicted in (a) in direction UNE (indicated by the arrows).

Temporal Graph Realization from Fastest Paths

Nina Klobas  

Department of Computer Science, Durham University, UK

George B. Mertzios  

Department of Computer Science, Durham University, UK

Hendrik Molter  

Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Paul G. Spirakis  

Department of Computer Science, University of Liverpool, UK

Abstract

In this paper we initiate the study of the *temporal graph realization* problem with respect to the fastest path durations among its vertices, while we focus on periodic temporal graphs. Given an $n \times n$ matrix D and a $\Delta \in \mathbb{N}$, the goal is to construct a Δ -periodic temporal graph with n vertices such that the duration of a *fastest path* from v_i to v_j is equal to $D_{i,j}$, or to decide that such a temporal graph does not exist. The variations of the problem on static graphs has been well studied and understood since the 1960's (e.g. [Erdős and Gallai, 1960], [Hakimi and Yau, 1965]).

As it turns out, the periodic temporal graph realization problem has a very different computational complexity behavior than its static (i. e., non-temporal) counterpart. First we show that the problem is NP-hard in general, but polynomial-time solvable if the so-called underlying graph is a tree. Building upon those results, we investigate its parameterized computational complexity with respect to structural parameters of the underlying static graph which measure the “tree-likeness”. We prove a tight classification between such parameters that allow fixed-parameter tractability (FPT) and those which imply W[1]-hardness. We show that our problem is W[1]-hard when parameterized by the *feedback vertex number* (and therefore also any smaller parameter such as *treewidth*, *degeneracy*, and *cliquewidth*) of the underlying graph, while we show that it is in FPT when parameterized by the *feedback edge number* (and therefore also any larger parameter such as *maximum leaf number*) of the underlying graph.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Mathematics of computing \rightarrow Discrete mathematics

Keywords and phrases Temporal graph, periodic temporal labeling, fastest temporal path, graph realization, temporal connectivity, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.16

Related Version *Full Version*: <https://arxiv.org/abs/2302.08860>

Funding *George B. Mertzios*: Supported by the EPSRC grant EP/P020372/1.

Hendrik Molter: Supported by the ISF, grant nr. 1456/18, and by the European Union's Horizon Europe research and innovation programme under grant agreement 949707.

Paul G. Spirakis: Supported by the EPSRC grant EP/P02002X/1.

1 Introduction

The (static) *graph realization* problem with respect to a graph property \mathcal{P} is to find a graph that satisfies property \mathcal{P} , or to decide that no such graph exists. The motivation for graph realization problems stems both from “verification” and from network design applications in engineering. In *verification* applications, given the outcomes of some experimental measurements (resp. some computations) on a network, the aim is to (re)construct an input network which complies with them. If such a reconstruction is not possible, this



© Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 16; pp. 16:1–16:18

Leibniz International Proceedings in Informatics



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

proves that the measurements are incorrect or implausible (resp. that the algorithm which made the computations is incorrectly implemented). One example of a graph realization (or reconstruction) problem is the recognition of probe interval graphs, in the context of the physical mapping of DNA, see [52, 53] and [36, Chapter 4]. In *network design* applications, the goal is to design network topologies having a desired property [4, 38]. Analyzing the computational complexity of the graph realization problems for various natural and fundamental graph properties \mathcal{P} requires a deep understanding of these properties. Among the most studied such parameters for graph realization are constraints on the distances between vertices [7, 8, 10, 16, 17, 41], on the vertex degrees [6, 22, 35, 37, 40], on the eccentricities [5, 9, 42, 51], and on connectivity [15, 29–31, 34, 37], among others.

In the simplest version of a (static) graph realization problem with respect to vertex distances, we are given a symmetric $n \times n$ matrix D and we are looking for an n -vertex undirected and unweighted graph G such that $D_{i,j}$ equals the distance between vertices v_i and v_j in G . This problem can be trivially solved in polynomial time in two steps [41]: First, we build the graph $G = (V, E)$ such that $v_i v_j \in E$ if and only if $D_{i,j} = 1$. Second, from this graph G we compute the matrix D_G which captures the shortest distances for all pairs of vertices. If $D_G = D$ then G is the desired graph, otherwise there is no graph having D as its distance matrix. Non-trivial variations of this problem have been extensively studied, such as for weighted graphs [41, 60], as well as for cases where the realizing graph has to belong to a specific graph family [7, 41]. Other variations of the problem include the cases where every entry of the input matrix D may contain a range of consecutive permissible values [7, 61, 63], or even an arbitrary set of acceptable values [8] for the distance between the corresponding two vertices.

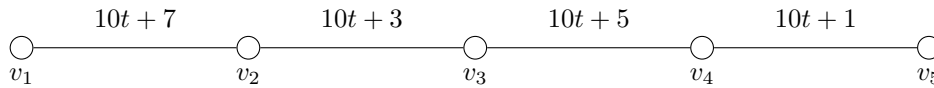
In this paper we make the first attempt to understand the complexity of the graph realization problem with respect to vertex distances in the context of *temporal graphs*, i. e., of graphs whose *topology changes over time*.

► **Definition 1** (temporal graph [43]). *A temporal graph is a pair (G, λ) , where $G = (V, E)$ is an underlying (static) graph and $\lambda : E \rightarrow 2^{\mathbb{N}}$ is a time-labeling function which assigns to every edge of G a set of discrete time-labels.*

Here, whenever $t \in \lambda(e)$, we say that the edge e is *active* or *available* at time t . In the context of temporal graphs, where the notion of vertex adjacency is time-dependent, the notions of path and distance also need to be redefined. The most natural temporal analogue of a path is that of a *temporal* (or *time-dependent*) path, which is motivated by the fact that, due to causality, entities and information in temporal graphs can “flow” only along sequences of edges whose time-labels are strictly increasing.

► **Definition 2** (fastest temporal path). *Let (G, λ) be a temporal graph. A temporal path in (G, λ) is a sequence $(e_1, t_1), (e_2, t_2), \dots, (e_k, t_k)$, where $P = (e_1, \dots, e_k)$ is a path in the underlying static graph G , $t_i \in \lambda(e_i)$ for every $i = 1, \dots, k$, and $t_1 < t_2 < \dots < t_k$. The duration of this temporal path is $t_k - t_1 + 1$. A fastest temporal path from a vertex u to a vertex v in (G, λ) is a temporal path from u to v with the smallest duration. The duration of the fastest temporal path from u to v is denoted by $d(u, v)$.*

In this paper we consider *periodic* temporal graphs, i. e., temporal graphs in which the temporal availability of each edge of the underlying graph is periodic. Many natural and technological systems exhibit a periodic temporal behavior. For example, in railway networks an edge is present at a time step t if and only if a train is scheduled to run on the respective rail segment at time t [3]. Similarly, a satellite, which makes pre-determined periodic movements,



■ **Figure 1** An example of a Δ -periodic temporal graph (G, λ, Δ) , where $\Delta = 10$ and the 10-periodic labeling $\lambda : E \rightarrow \{1, 2, \dots, 10\}$ is as follows: $\lambda(v_1v_2) = 7$, $\lambda(v_2v_3) = 3$, $\lambda(v_3v_4) = 5$, and $\lambda(v_4v_5) = 1$. Here, the fastest temporal path from v_1 to v_5 traverses the first edge v_1v_2 at time 7, second edge v_2v_3 a time 13, third edge v_3v_4 at time 15 and the last edge v_4v_5 at time 21. This results in the total duration of $21 - 7 + 1 = 15$ for the fastest temporal path from v_1 to v_5 .

can establish a communication link (i. e., a temporal edge) with another satellite whenever they are sufficiently close to each other; the existence of these communication links is also periodic. In a railway (resp. satellite) network, a fastest temporal path from u to v represents the fastest railway connection between two stations (resp. the quickest communication delay between two moving satellites). Furthermore, periodicity appears also in (the otherwise quite complex) social networks which describe the dynamics of people meeting [50, 62], as every person individually follows mostly a weekly routine.

Expanding the work on periodic temporal graphs (see [13, Class 8] and [3, 25, 58, 59]), our study represents the first attempt to understand the complexity of a graph realization problem in the context of temporal graphs. Therefore, we focus in this paper on the most fundamental case, where all edges have the same period Δ (while in the more general case, each edge e in the underlying graph has a period Δ_e). As it turns out, the periodic temporal graph realization problem with respect to a given $n \times n$ matrix D of the fastest duration times has a very different computational complexity behavior than the classic graph realization problem with respect to shortest path distances in static graphs.

Formally, let $G = (V, E)$ and $\Delta \in \mathbb{N}$, and let $\lambda : E \rightarrow \{1, 2, \dots, \Delta\}$ be an edge-labeling function that assigns to every edge of G exactly one of the labels from $\{1, \dots, \Delta\}$. Then we denote by (G, λ, Δ) the Δ -periodic temporal graph (G, L) , where for every edge $e \in E$ we have $L(e) = \{i\Delta + x : i \geq 0, x \in \lambda(e)\}$. In this case we call λ a Δ -periodic labeling of G ; see Figure 1 for an illustration. When it is clear from the context, we drop Δ from the notation and we denote the (Δ -periodic) temporal graph by (G, λ) . Given a duration matrix D , it is easy to observe that, similarly to the static case, if $D_{i,j} = 1$ then v_i and v_j must be connected by an edge. We call the graph defined by these edges the *underlying graph* of D .

Our contribution. We initiate the study of naturally motivated graph realization problems in the temporal setting. Our target is not to model unreliable communication, but instead to *verify* that particular measurements regarding fastest temporal paths in a periodic temporal graph are plausible (i. e., “realizable”). To this end, we introduce and investigate the following problem, capturing the setting described above:

SIMPLE PERIODIC TEMPORAL GRAPH REALIZATION (SIMPLE TGR)

Input: An integer $n \times n$ matrix D , a positive integer Δ .

Question: Does there exist a graph $G = (V, E)$ with vertices $\{v_1, \dots, v_n\}$ and a Δ -periodic labeling $\lambda : E \rightarrow \{1, 2, \dots, \Delta\}$ such that, for every i, j , the duration of the fastest temporal path from v_i to v_j in the Δ -periodic temporal graph (G, λ, Δ) is $D_{i,j}$?

We focus on exact algorithms. We start by showing NP-hardness of the problem (Theorem 3), even if Δ is a small constant. To establish a baseline for tractability, we show that SIMPLE TGR is polynomial-time solvable if the underlying graph is a tree (Theorem 5).

Building upon these initial results, we explore the possibilities to generalize our polynomial-time algorithm using the *distance-from-triviality* parameterization paradigm [27, 39]. That is, we investigate the parameterized computational complexity of SIMPLE TGR with respect to structural parameters of the underlying graph that measure its “tree-likeness”.

We obtain the following results. We show that SIMPLE TGR is $W[1]$ -hard when parameterized by the *feedback vertex number* of the underlying graph (Theorem 4). To this end, we first give a reduction from MULTICOLORED CLIQUE parameterized by the number of colors [26] to a variant of SIMPLE TGR where the period Δ is infinite, that is, when the labeling is non-periodic. Then we use a special gadget (the “infinity” gadget) which allows us to transfer the result to a finite period Δ . The latter construction is independent from the particular reduction we use, and can hence be treated as a reduction from the non-periodic to the periodic setting. Note that our parameterized hardness result with respect to the feedback vertex number also implies $W[1]$ -hardness for any smaller parameter, such as *treewidth*, *degeneracy*, *cliquewidth*, *distance to chordal graphs*, and *distance to outerplanar graphs*.

We complement this hardness result by showing that SIMPLE TGR is fixed-parameter tractable (FPT) with respect to the *feedback edge number* k of the underlying graph (Theorem 6). This result also implies an FPT algorithm for any larger parameter, such as the *maximum leaf number*. A similar phenomenon of getting $W[1]$ -hardness with respect to the feedback vertex number, while getting an FPT algorithm with respect to the feedback edge number, has been observed only in a few other temporal graph problems related to the connectivity between two vertices [14, 21, 32].

Our FPT algorithm works as follows on a high level. First we distinguish $O(k^2)$ vertices which we call “important vertices”. Then, we guess the fastest temporal paths for each pair of these important vertices; as we prove, the number of choices we have for all these guesses is upper bounded by a function of k . Then we also need to make several further guesses (again using a bounded number of choices), which altogether leads us to specify a small (i. e., bounded by a function of k) number of different configurations for the fastest paths between *all pairs* of vertices. For each of these configurations, we must then make sure that the labels of our solution will not allow any other temporal path from a vertex v_i to a vertex v_j have a *strictly smaller* duration than $D_{i,j}$. This naturally leads us to build one Integer Linear Program (ILP) for each of these configurations. We manage to formulate all these ILPs by having a number of variables that is upper-bounded by a function of k . Finally we use Lenstra’s Theorem [49] to solve each of these ILPs in FPT time. At the end, our initial instance is a YES-instance if and only if at least one of these ILPs is feasible.

The above results provide a fairly complete picture of the parameterized computational complexity of SIMPLE TGR with respect to structural parameters of the underlying graph which measure “tree-likeness”. To obtain our results, we prove several properties of fastest temporal paths, which may be of independent interest. Due to space constraints, proofs of results marked with \star are (partially) deferred to the full version on arXiv [46].

Related work. Graph realization problems on static graphs have been studied since the 1960s. We provide an overview of the literature in the introduction. To the best of our knowledge, we are the first to consider graph realization problems in the temporal setting. Very recently, Erlebach et al. [24] have built upon our results and, among others, studied the case where edges might appear more than once in each period. Many other connectivity-related problems have been studied in the temporal setting [2, 12, 18, 19, 23, 28, 33, 44, 48, 55, 57, 65], most of which are much more complex and computationally harder than their non-temporal counterparts, and some of which do not even have a non-temporal counterpart.

Several problems have been studied where the goal is to assign labels to (sets of) edges of a given static graph in order to achieve certain connectivity-related properties [1, 20, 45, 54]. The main difference to our problem setting is that in the mentioned works, the input is a graph and the sought labeling is not periodic. Furthermore, the investigated properties are temporal connectivity among all vertices [1, 45, 54], temporal connectivity among a subset of vertices [45], or reducing reachability among the vertices [20]. In all these cases, the duration of the temporal paths has not been considered.

Finally, there are many models for dynamic networks in the context of distributed computing [47]. These models have some similarity to temporal graphs, in the sense that in both cases the edges appear and disappear over time. However, there are notable differences. For example, one important assumption in the distributed setting can be that the edge changes are adversarial or random (while obeying some constraints such as connectivity), and therefore they are not necessarily known in advance [47].

Preliminaries and notation. We already introduced the most central notion and concepts. There are some additional definitions we need, to present our proofs and results which we give in the following.

An interval in \mathbb{N} from a to b is denoted by $[a, b] = \{i \in \mathbb{N} : a \leq i \leq b\}$; similarly, $[a] = [1, a]$. An undirected graph $G = (V, E)$ consists of a set V of vertices and a set $E \subseteq V \times V$ of edges. For a graph G , we also denote by $V(G)$ and $E(G)$ the vertex and edge set of G , respectively. We denote an edge $e \in E$ between vertices $u, v \in V$ as a set $e = \{u, v\}$. For the sake of simplicity of the representation, an edge e is sometimes also denoted by uv . A path P in G is a subgraph of G with vertex set $V(P) = \{v_1, \dots, v_k\}$ and edge set $E(P) = \{\{v_i, v_{i+1}\} : 1 \leq i < k\}$ (we often represent path P by the tuple (v_1, v_2, \dots, v_k)).

Let v_1, v_2, \dots, v_n be the n vertices of the graph G . For simplicity of the presentation (and with a slight abuse of notation) we refer during the paper to the entry $D_{i,j}$ of the matrix D as $D_{a,b}$, where $a = v_i$ and $b = v_j$. That is, we put as indices of the matrix D the corresponding vertices of G whenever it is clear from the context.

Let $P = (u = v_1, v_2, \dots, v_p = v)$ be a path from u to v in G . Recall that, in our paper, every edge has exactly one time label in every period of Δ consecutive time steps. Therefore, as we are only interested in the fastest duration of temporal paths, many times we refer to (P, λ, Δ) as any of the temporal paths from $u = v_1$ to $v = v_p$ along the edges of P , which starts at the edge v_1v_2 at time $\lambda(v_1v_2) + c\Delta$, for some $c \in \mathbb{N}$, and then sequentially visits the rest of the edges of P as early as possible. We denote by $d(P, \lambda, \Delta)$, or simply by $d(P, \lambda)$ when Δ is clear from the context, the duration of any of the temporal paths (P, λ, Δ) ; note that they all have the same duration. Many times we also refer to a path $P = (u = v_1, v_2, \dots, v_p = v)$ from u to v in G , as a temporal path in (G, λ, Δ) , where we actually mean that (P, λ, Δ) is a temporal path with P as its underlying (static) path.

We remark that a fastest path between two vertices in a temporal graph can be computed in polynomial time [11, 64]. Hence, given a Δ -periodic temporal graph (G, λ, Δ) , we can compute in polynomial-time the matrix D which consists of durations of fastest temporal paths among all pairs of vertices in (G, λ, Δ) .

2 Hardness results for Simple TGR

In this section we present our main computational hardness results. We first show that SIMPLE TGR is NP-hard even for constant Δ .

► **Theorem 3** (\star). *SIMPLE TGR is NP-hard for all $\Delta \geq 3$.*

Next, we investigate the parameterized hardness of SIMPLE TGR with respect to structural parameters of the underlying graph. We show that the problem is $W[1]$ -hard when parameterized by the feedback vertex number of the underlying graph. The *feedback vertex number* of a graph G is the cardinality of a minimum vertex set $X \subseteq V(G)$ such that $G - X$ is a forest. The set X is called a *feedback vertex set*. Note that, in contrast to the previous result (Theorem 3), the reduction we use to obtain the following result does not produce instances with a constant Δ .

► **Theorem 4** (\star). *SIMPLE TGR is $W[1]$ -hard when parameterized by the feedback vertex number of the underlying graph.*

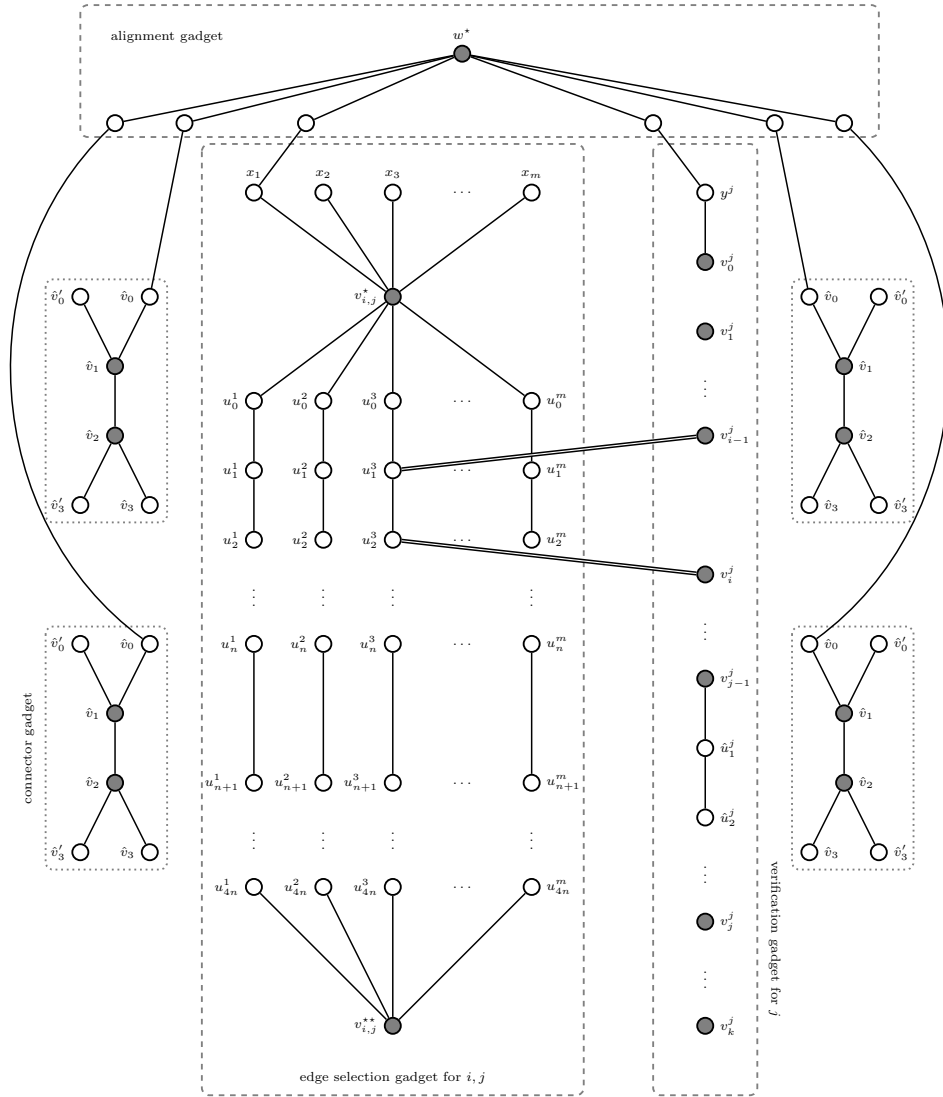
Proof. We present a parameterized reduction from the $W[1]$ -hard problem MULTICOLORED CLIQUE parameterized by the number of colors [26]. Here, given a k -partite graph $H = (W_1 \uplus W_2 \uplus \dots \uplus W_k, F)$, we are asked whether H contains a clique of size k . If $w \in W_i$, then we say that w has *color* i . W.l.o.g. we assume that $|W_1| = |W_2| = \dots = |W_k| = n$. Furthermore, for all $i \in [k]$, we assume the vertices in W_i are ordered in some arbitrary but fixed way, that is, $W_i = \{w_1^i, w_2^i, \dots, w_n^i\}$. Let $F_{i,j}$ with $i < j$ denote the set of all edges between vertices from W_i and W_j . We assume w.l.o.g. that $|F_{i,j}| = m$ for all $i < j$ (if not we can add $k \max_{i,j} |F_{i,j}|$ vertices to each W_i and use those to add up to $\max_{i,j} |F_{i,j}|$ additional isolated edges to each $F_{i,j}$). Furthermore, for all $i < j$ we assume that the edges in $F_{i,j}$ are ordered in some arbitrary but fixed way, that is, $F_{i,j} = \{e_1^{i,j}, e_2^{i,j}, \dots, e_m^{i,j}\}$.

We give a reduction to a variant of SIMPLE TGR where the period Δ is infinite (that is, the sought temporal graph is not periodic and the labeling function $\lambda : E \rightarrow \mathbb{N}$ maps to the natural numbers) and we allow D to have infinity entries, meaning that the two respective vertices are not temporally connected. Note that, given the matrix D , we can easily compute the underlying graph G , as follows. Two vertices v, v' are adjacent in G if and only if $D_{v,v'} = 1$, as having an edge between v and v' is the only way that there exists a temporal path from v to v' with duration 1. For simplicity of the presentation of the reduction, we describe the underlying graph G (which directly implies the entries of D where $D_{v,v'} = 1$) and then we provide the remaining entries of D . At the end of the proof, we show how to obtain the result for a finite Δ (by introducing a so-called “infinity gadget”) and a matrix D of durations of fastest paths which only has finite entries.

In the following, we give an informal description of the main ideas of the reduction. The construction uses several gadgets, where the main ones are an “edge selection gadget” and a “verification gadget”.

Every *edge selection gadget* is associated with a color combination i, j in the MULTICOLORED CLIQUE instance, and its main purpose is to “select” an edge connecting a vertex from color i with a vertex from color j . Roughly speaking, the edge selection gadget consists of m paths, one for every edge in $F_{i,j}$ (see Figure 2 for reference). The distance matrix D will enforce that the labels on those paths effectively order them temporally, that is, in particular, the labels on one of the paths will be smaller than the labels on all other paths. The edge corresponding to this path is selected.

We have a *verification gadget* for every color i . They interact with the edge selection gadgets as follows. The verification gadget for color i is connected to all edge selection gadgets that involve color i . More specifically, this is connected to every path corresponding to an edge at a position in the path that encodes the endpoint of color i of that edge (again, see Figure 2 for reference). Intuitively, the distances in the verification gadget are only realizable if the selected edges all have the same endpoint of color i . Hence, the distances of all verification gadgets can be realized if and only if the selected edges form a clique.



■ **Figure 2** Illustration of part of the underlying graph G and a possible labeling. Edges incident with vertices \hat{v}_1, \hat{v}_2 of connector gadgets are omitted. Gray vertices form a feedback vertex set. The double line connections, between a vertex v_{i-1}^j in the verification gadget, and u_1^3 in the edge selection gadget, and, between a vertex u_2^3 in the edge selection gadget, and v_i^j in the verification gadget, consist of $5n$ vertices $a_1^{j,i,3}, a_2^{j,i,3}, \dots, a_{5n}^{j,i,3}$ and $b_1^{j,i,3}, b_2^{j,i,3}, \dots, b_{5n}^{j,i,3}$, respectively.

Furthermore, we use an *alignment gadget* which, intuitively, ensures that the labelings of all gadgets use the same range of time labels. Finally, we use *connector gadgets* which create shortcuts between all vertex pairs that are irrelevant for the functionality of the other gadgets. This allows us to easily fill in the distance matrix with the corresponding values. We ensure that all our gadgets have a constant feedback vertex number, hence the overall feedback vertex number is quadratic in the number of colors of the MULTICOLORED CLIQUE instance and we get the parameterized hardness result.

In the following, for every gadget, we give a formal description of the underlying graph of this gadget (i. e., not the complete distance sub-matrix of the gadget). Due to space constraints, we defer the description of the distance matrix D and the formal proof of correctness for the reduction to [46].

16:8 Temporal Graph Realization from Fastest Paths

Given an instance H of MULTICOLORED CLIQUE, we construct an instance D of SIMPLE TGR (with infinity entries and no periods) as follows.

Edge selection gadget. We first introduce an *edge selection gadget* $G_{i,j}$ for color combination i, j with $i < j$. We start with describing the vertex set of the gadget.

- A set $X_{i,j}$ of vertices x_1, x_2, \dots, x_m .
- Vertex sets U_1, U_2, \dots, U_m with $4n + 1$ vertices each, that is, $U_\ell = \{u_0^\ell, u_1^\ell, u_2^\ell, \dots, u_{4n}^\ell\}$ for all $\ell \in [m]$.
- Two special vertices $v_{i,j}^*, v_{i,j}^{**}$.

The gadget has the following edges.

- For all $\ell \in [m]$ we have edge $\{x_\ell, v_{i,j}^*\}$, $\{v_{i,j}^*, u_0^\ell\}$, and $\{u_{4n}^\ell, v_{i,j}^{**}\}$.
- For all $\ell \in [m]$ and $\ell' \in [4n]$, we have edge $\{u_{\ell'-1}^\ell, u_{\ell'}^\ell\}$.

Verification gadget. For each color i , we introduce the following vertices. What we describe in the following will be used as a *verification gadget for color i* .

- We have one vertex y^i and $k + 1$ vertices v_ℓ^i for $0 \leq \ell \leq k$.
- For every $\ell \in [m]$ and every $j \in [k] \setminus \{i\}$ we have $5n$ vertices $a_1^{i,j,\ell}, a_2^{i,j,\ell}, \dots, a_{5n}^{i,j,\ell}$ and $5n$ vertices $b_1^{i,j,\ell}, b_2^{i,j,\ell}, \dots, b_{5n}^{i,j,\ell}$.
- We have a set \hat{U}_i of $13n + 1$ vertices $\hat{u}_1^i, \hat{u}_2^i, \dots, \hat{u}_{13n+1}^i$.

We add the following edges. We add edge $\{y^i, v_0^i\}$. For every $\ell \in [m]$, every $j \in [k] \setminus \{i\}$, and every $\ell' \in [5n - 1]$ we add edge $\{a_{\ell'}^{i,j,\ell}, a_{\ell'+1}^{i,j,\ell}\}$ and we add edge $\{b_{\ell'}^{i,j,\ell}, b_{\ell'+1}^{i,j,\ell}\}$.

Let $1 \leq j < i$ (skip if $i = 1$), let $e_\ell^{j,i} \in F_{j,i}$, and let $w_{\ell'}^i \in W_i$ be incident with $e_\ell^{j,i}$. Then we add edge $\{v_{j-1}^i, a_1^{i,j,\ell}\}$ and we add edge $\{a_{5n}^{i,j,\ell}, u_{\ell'-1}^\ell\}$ between $a_{5n}^{i,j,\ell}$ and the vertex $u_{\ell'-1}^\ell$ of the edge selection gadget of color combination j, i . Furthermore, we add edge $\{v_j^i, b_1^{i,j,\ell}\}$ and edge $\{b_{5n}^{i,j,\ell}, u_{\ell'}^\ell\}$ between $b_{5n}^{i,j,\ell}$ and the vertex $u_{\ell'}^\ell$ of the edge selection gadget of color combination j, i .

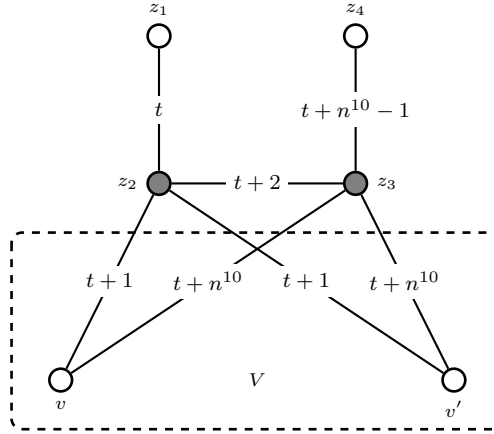
We add edge $\{v_{i-1}^i, \hat{u}_1^i\}$ and for all $\ell'' \in [13n]$ we add edge $\{\hat{u}_{\ell''}^i, \hat{u}_{\ell''+1}^i\}$. Furthermore, we add edge $\{\hat{u}_{13n+1}^i, v_i^i\}$.

Let $i < j \leq k$ (skip if $i = k$), let $e_\ell^{i,j} \in F_{i,j}$, and let $w_{\ell'}^i \in W_i$ be incident with $e_\ell^{i,j}$. Then we add edge $\{v_{j-1}^i, a_1^{i,j,\ell}\}$ and edge $\{a_{5n}^{i,j,\ell}, u_{3n+\ell'-1}^\ell\}$ between $a_{5n}^{i,j,\ell}$ and the vertex $u_{3n+\ell'-1}^\ell$ of the edge selection gadget of color combination i, j . Furthermore, we add edge $\{v_j^i, b_1^{i,j,\ell}\}$ and edge $\{b_{5n}^{i,j,\ell}, u_{3n+\ell'}^\ell\}$ between $b_{5n}^{i,j,\ell}$ and the vertex $u_{3n+\ell'}^\ell$ of the edge selection gadget of color combination i, j .

Furthermore, we use *connector gadgets*, two for each edge selection gadget, and two for every verification gadget. They consist of six vertices $\hat{v}_0, \hat{v}'_0, \hat{v}_1, \hat{v}_2, \hat{v}_3, \hat{v}'_3$ and, intuitively, are used to connect many vertex pairs by fast paths, which will make arguing about possible labelings in YES-instances much easier. Finally, we have an *alignment gadget*, which is a star with a center vertex w^* and a leaf for every other gadget. Intuitively, this gadget is used to relate labels of different gadgets to each other. A formal description of these two gadgets is given in [46].

This finishes the description of the underlying graph G . For an illustration see Figure 2. We can observe that the vertex set containing vertices $v_{i,j}^*$ and $v_{i,j}^{**}$ of each edge selection gadget, vertices v_ℓ^i with $0 \leq \ell \leq k$ of each verification gadget, vertices \hat{v}_1 and \hat{v}_2 of each connector gadget, and vertex w^* of the alignment gadget forms a feedback vertex set in G with size $O(k^2)$.

As mentioned before, due to space constraints, we defer the description of the distance matrix D and a formal correctness proof of the reduction to [46].



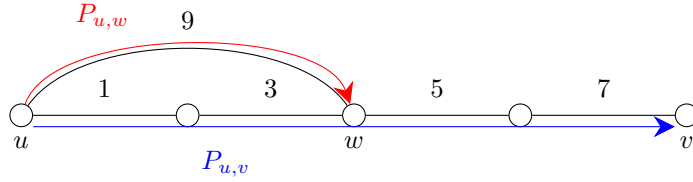
■ **Figure 3** Illustration of the infinity gadget. Gray vertices need to be added to the feedback vertex set.

Infinity gadget. Finally, we show how to get rid of the infinity entries in D and how to allow a finite Δ . To this end, we introduce the *infinity gadget*. We add four vertices z_1, z_2, z_3, z_4 to the graph and we set $\Delta = n^{11}$. Let V denote the set of all remaining vertices. We set the following durations.

- For all $v \in V$ we set $d(z_1, v) = 2$, $d(z_2, v) = d(v, z_2) = 1$, $d(z_3, v) = d(v, z_3) = 1$, and $d(z_4, v) = 2$. Furthermore, we set $d(v, z_1) = n^{11}$ and $d(v, z_4) = n^{10} - 1$.
- We set $d(z_1, z_2) = d(z_2, z_1) = 1$, $d(z_2, z_3) = d(z_3, z_2) = 1$, and $d(z_3, z_4) = d(z_4, z_3) = 1$.
- We set $d(z_1, z_3) = 3$, $d(z_3, z_1) = n^{11} - 1$, $d(z_2, z_4) = n^{10} - 2$, and $d(z_4, z_2) = n^{11} - n^{10} + 4$.
- We set $d(z_1, z_4) = n^{10}$ and $d(z_4, z_1) = 2n^{11} - n^{10} + 2$.
- For every pair of vertices $v, v' \in V$ where previously the duration of a fastest path from v to v' was specified to be infinite, we set $d(v, v') = n^{10}$.

Now we analyse which implications we get for the labels on the newly introduced edges. Assume $\lambda(\{z_1, z_2\}) = t$, then we get the following. For all $v \in V$ we have that $d(z_1, v) = 2$ and hence we get that $\lambda(\{z_2, v\}) = t + 1$. Since $d(z_1, z_4) = n^{10}$, we have that $\lambda(\{z_3, z_4\}) = t + n^{10} - 1$. From this follows that for all $v \in V$, since $d(z_4, v) = 2$, that $\lambda(\{z_3, v\}) = t + n^{10}$. Finally, since $d(z_1, z_3) = 3$, we have that $\lambda(\{z_2, z_3\}) = t + 2$. For an illustration see Figure 3. It is easy to check that all duration requirements between vertex pairs in $\{z_1, z_2, z_3, z_4\}$ are met and that all duration requirements between each vertex $v \in V$ and each vertex in $\{z_1, z_2, z_3, z_4\}$ are met. Furthermore, it is easy to check that the gadget increases the feedback vertex set by two (z_2 and z_3 need to be added).

Lastly, consider two vertices $v, v' \in V$. Note that before the addition of the infinity gadget, by construction of G we have that $d(v, v') \leq n^9 + 2$ or $d(v, v') = \infty$. Furthermore, if D is a YES-instance, we have shown in the correctness proof of the reduction that the difference between the smallest label and the largest label is at most $n^9 + 1$. This implies that for a vertex pair $v, v' \in V$ with $d(v, v') = \infty$ we have in the periodic case with $\Delta = n^{11}$, that $d(v, v') \geq n^{11} - n^9 > n^{10}$. Which means, after adding the vertices and edges of the infinity gadget, we indeed have that $d(v, v') = n^{10}$. For all vertex pairs v, v' where in the original construction we have $d(v, v') \neq \infty$, we can also see that adding the infinity gadget and setting $\Delta = n^{11}$ does not change the duration of a fastest path from v to v' , since all newly added temporal paths have duration at least n^{10} . We can conclude that the originally constructed instance D is a YES-instance if and only if it remains a YES-instance after adding the infinity gadget and setting $\Delta = n^{11}$. ◀



■ **Figure 4** An example of a temporal graph (with $\Delta \geq 9$), where the fastest temporal path $P_{u,v}$ (in blue) from u to v is of duration 7, while the fastest temporal path $P_{u,w}$ (in red) from u to a vertex w , that is on a path $P_{u,v}$, is of duration 1 and is not a subpath of $P_{u,v}$.

3 Algorithms for Simple TGR

In this section, to complement the discussed hardness aspects of SIMPLE TGR, we present some algorithmic results. We start by restricting the underlying graph G of the input matrix D of SIMPLE TGR to be a tree and get the following.

► **Theorem 5** (\star). *SIMPLE TGR can be solved in polynomial time on trees.*

The main reason, for which SIMPLE TGR is straightforward to solve on trees, is twofold:

- between any pair of vertices v_i and v_j in the tree T , there is a *unique* path P in T from v_i to v_j , and
- in any periodic temporal graph (T, λ, Δ) and any fastest temporal path $P = ((e_1, t_1), \dots, (e_i, t_i), \dots, (e_j, t_j), \dots, (e_{\ell-1}, t_{\ell-1}))$ from v_1 to v_ℓ we have that the sub-path $P' = ((e_i, t_i), \dots, (e_{j-1}, t_{j-1}))$ is also a fastest temporal path from v_i to v_j .

However, these two nice properties do not hold when the underlying graph is not a tree. For example, in Figure 4, the fastest temporal path from u to v is $P_{u,v}$ (depicted in blue) goes through w , however the sub-path of $P_{u,v}$ that stops at w is not the fastest temporal path from u to w . The fastest temporal path from u to w consists only of the single edge uw (with label 9 and duration 1, depicted in red).

Nevertheless, we prove that we can still solve SIMPLE TGR efficiently if the underlying graph is similar to a tree; more specifically we show the following result, which turns out to be non-trivial.

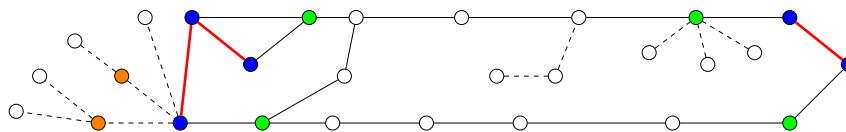
► **Theorem 6** (\star). *SIMPLE TGR is in FPT when parameterized by the feedback edge number of the underlying graph.*

From Theorem 4 and Theorem 6 we immediately get the following, which is the main result of the paper.

► **Corollary 7.** *SIMPLE TGR is:*

- *in FPT when parameterized by the feedback edge number or any larger parameter, such as the maximum leaf number.*
- *$W[1]$ -hard when parameterized by the feedback vertex number or any smaller parameter, such as: treewidth, degeneracy, cliquewidth, distance to chordal graphs, and distance to outerplanar graphs.*

Before presenting the structure of our algorithm for Theorem 6, observe that, in a static graph, the number of paths between two vertices can be upper-bounded by a function $f(k)$ of the feedback edge number k of the graph [14]. Therefore, for any fixed pair of vertices u and v , we can “guess” the edges of the fastest temporal path from u to v (by guess we mean enumerate and test all possibilities). However, for an FPT algorithm with respect to k , we cannot afford to guess the edges of the fastest temporal path for each of the $O(n^2)$ pairs of vertices. To overcome this difficulty, our algorithm follows this high-level strategy:



■ **Figure 5** An example of a graph with its important vertices: U (in blue), U^* (in green) and Z^* (in orange). Corresponding feedback edges are marked with a thick red line, while dashed edges represent the edges (and vertices) “removed” from G' at the initial step.

- We identify a small number $f(k)$ of “important vertices”.
- For each pair u, v of important vertices, we guess the edges of the fastest temporal path from u to v (and from v to u).
- From these guesses we can still not deduce the edges of the fastest temporal paths between many pairs of non-important vertices. However, as we prove, it suffices to guess only a small number of specific auxiliary structures (to be defined later).
- From these guesses we deduce fixed relationships between the labels of most of the edges of the graph.
- For all the edges, for which we have not deduced a label yet, we introduce a *variable*. With all these variables, we build an Integer Linear Program (ILP). Among the constraints in this ILP we have that, for each of the $O(n^2)$ pairs of vertices u, v in the graph, the duration of one specific temporal path from u to v (according to our guesses) is *equal* to the desired duration $D_{u,v}$, while the duration of each of the other temporal path from u to v is *at least* $D_{u,v}$.
- By making each of the above combinations of guesses, we essentially enumerate all possible ways that our instance of SIMPLE TGR has a solution, and for each of these possible ways we create an ILP. That is, our instance of SIMPLE TGR has a solution if and only if at least one of these ILPs has a feasible solution. As each ILP can be solved in FPT time with respect to k by Lenstra’s Theorem [49] (the number of variables is upper bounded by a function of k), we obtain our FPT algorithm for SIMPLE TGR with respect to k .

We now present the first part of our FPT algorithm, that is, identifying important vertices and guessing information about the fastest temporal paths. A full description of the algorithm is deferred to [46].

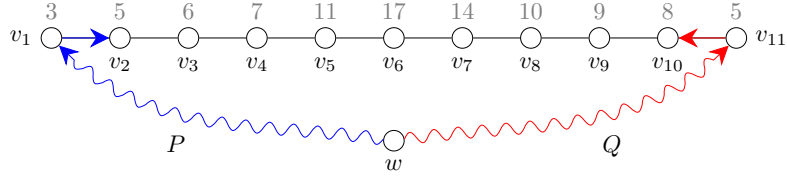
Important vertices. Let D be the input matrix of SIMPLE TGR, and let G be its underlying graph, on n vertices and m edges. From the underlying graph G of D we first create a graph G' by iteratively removing vertices of degree one from G , and denote with $Z = V(G) \setminus V(G')$, the set of removed vertices. Then we determine the set U (the “vertices of interest”), and the set U^* (the neighbors of the vertices of interest), as follows. Let T be a spanning tree of G' , with F being the corresponding feedback edge set of G' . Let $V_1 \subseteq V(G')$ be the set of leaves in the spanning tree T , $V_2 \subseteq V(G')$ be the set of vertices of degree two in T which are incident to at least one edge in F , and let $V_3 \subseteq V(G')$ be the set of vertices of degree at least 3 in T . Then $|V_1| + |V_2| \leq 2k$, since every leaf in T and every vertex in V_2 is incident to at least one edge in F , and $|V_3| \leq |V_1|$ by the properties of trees. We denote with

$$U = V_1 \cup V_2 \cup V_3$$

the set of *vertices of interest*. It follows that $|U| \leq 4k$. We set U^* to be the set of vertices in $V(G') \setminus U$ that are neighbors of vertices in U , i. e.,

$$U^* = \{v \in V(G') \setminus U : u \in U, v \in N(u)\}.$$

16:12 Temporal Graph Realization from Fastest Paths



■ **Figure 6** In the above graph vertices v_1, v_{11}, w are in U , while v_2, v_{10} are in U^* . Numbers above all v_i represent the values of the fastest temporal paths from w to each of them (i. e., the entries in the w -th row of matrix D). From the basic guesses we know the fastest temporal path P from w to v_2 (depicted in blue) and the fastest temporal path Q from w to v_{10} . From the values of durations from w to each v_i we cannot determine the fastest paths from w to all v_i . More precisely, we know that w reaches v_2, v_3, v_4, v_5 (resp. v_{10}, v_9, v_9, v_7) by first using the path P (resp. Q) and then proceeding through the vertices, but we do not know how w reaches v_6 the fastest. Therefore we have to introduce some more guesses.

Again, using the tree structure, we get that for any $u \in U$ its neighborhood is of size $|N(u)| \in O(k)$, since every neighbor of u is the first vertex of a (unique) path to another vertex in U . It follows that $|U^*| \in O(k^2)$. From the construction of Z (i. e., by exhaustively removing vertices of degree one from G), it follows that $G[Z]$ (the graph induced in G by Z) is a forest, i. e., consists of disjoint trees. Each of these trees has a unique neighbor v in G' . Denote by T_v the tree obtained by considering such a vertex v and all the trees from $G[Z]$ that are incident to v in G . We then refer to v as the *clip vertex* of the tree T_v . In the case where v is a vertex of interest we define also the set Z_v^* of *representative vertices* of T_v , as follows. We first create an empty set C_w for every vertex w that is a neighbor of v in G' . We then iterate through every vertex r that is in the first layer of the tree T_v (i. e., vertex that is a child of the root v in the tree T_v), check the matrix D and find the vertex $w \in N_{G'}(v)$ that is on the smallest duration from r . In other words, for an $r \in N_{T_v}(v)$ we find $w \in N_{G'}(v)$ such that $D_{r,w} \leq D_{r,w'}$ for all $w' \in N_{G'}(v)$. We add vertex r to C_w . In the case when there exists also another vertex $w' \in N_{G'}(v)$ for which $D_{r,w} = D_{r,w'}$, we add r also to the set $C_{w'}$. In fact, in this case $C_{w'} = C_w$. At the end we create $|N_{G'}(v)| \in O(k)$ sets C_w , whose union contains all children of v in T_v . For every two sets C_w and $C_{w'}$, where $w, w' \in N_{G'}(v)$, we have that either $C_w = C_{w'}$, or $C_w \cap C_{w'} = \emptyset$. We interpret each of these sets $\{C_w : w \in N_{G'}(v)\}$ as an *equivalence class* of the neighbors of v in the tree T_v . Now, from each equivalence class C_w we choose an arbitrary vertex $r_w \in C_w$ and put it into the set Z_v^* . We repeat the above procedure for all trees T_u with the clip vertex u from U , and define Z^* as

$$Z^* = \bigcup_{v \in U} Z_v^*. \quad (1)$$

Since $|U| \in O(k)$ and for each $u \in U$ it holds $|N_{G'}(u)| \in O(k)$, we get that $|Z^*| \in O(k^2)$. Finally, the set of *important vertices* is defined as the set $U \cup U^* \cup Z^*$. For an illustration see Figure 5.

Guesses. For every pair of important vertices $u, v \in U \cup U^* \cup Z^*$, we guess the sequence of edges in the fastest temporal path from u to v . Since $U \cup U^* \cup Z^* \in O(k^2)$ and there are $k^{O(k)}$ possibilities for a sequence of edges between a fixed vertex pair, we have $k^{O(k^5)}$ overall possible guesses. We defer further details to [46] (see guesses **G-1** to **G-6**).

With the information provided by the described guesses we are still not able to determine all fastest paths. For example consider the case depicted in Figure 6. Therefore we introduce additional guesses that provide us with sufficient information to determine all fastest paths. To do this we have to first define the following.

► **Definition 8.** Let $U \subseteq V(G')$ be a set of vertices of interest and let $u, v \in U$. A path $P = (u = v_1, v_2, \dots, v_p = v)$ of length at least 2 in graph G' , where all inner vertices are not in U , i. e., $v_i \notin U$ for all $i \in \{2, 3, \dots, p-1\}$, is called a *segment* from u to v . We denote it as $S_{u,v}$.

Note by Definition 8 that $S_{u,v} \neq S_{v,u}$. Observe that a temporal path in G' between two vertices of interest is either a segment, or it consists of a sequence of some segments. Furthermore, since we have at most $4k$ interesting vertices in G' , we can deduce the following important result.

► **Corollary 9.** There are $O(k^2)$ segments in G' .

To describe the next guesses, we introduce the following notation. Let u, v, x be three vertices in G' . We write $u \rightsquigarrow x \rightarrow v$ to denote a temporal path from u to v that passes through x , and then goes to v (via one edge). We guess the following structures.

G-7. Inner segment guess I. Let $S_{u,v} = (u = v_1, v_2, \dots, v_p = v)$ and $S_{w,z} = (w = z_1, z_2, \dots, z_r = z)$ be two segments in G' . We want to guess the fastest temporal path $v_2 \rightarrow u \rightsquigarrow w \rightarrow z_2$. We repeat this procedure for all pairs of segments. Since there are $O(k^2)$ segments in G' , there are $k^{O(k^5)}$ possible paths of this form.

Recall that $S_{u,v} \neq S_{v,u}$ for every $u, v \in U$. Furthermore note that we did not assume that $\{u, v\} \cap \{w, z\} = \emptyset$. Therefore, by repeatedly making the above guesses, we also guess the following fastest temporal paths: $v_2 \rightarrow u \rightsquigarrow z \rightarrow z_{r-1}$, $v_2 \rightarrow u \rightsquigarrow v \rightarrow v_{p-1}$, $v_{p-1} \rightarrow v \rightsquigarrow w \rightarrow z_2$, $v_{p-1} \rightarrow v \rightsquigarrow z \rightarrow z_{r-1}$, and $v_{p-1} \rightarrow v \rightsquigarrow u \rightarrow v_2$. For an example see Figure 7a.

G-8. Inner segment guess II. Let $S_{u,v} = (u = v_1, v_2, \dots, v_p = v)$ be a segment in G' , and let $w \in U \cup Z^*$. We want to guess the following fastest temporal paths $w \rightsquigarrow u \rightarrow v_2$, $w \rightsquigarrow v \rightarrow v_{p-1} \rightarrow \dots \rightarrow v_2$, and $v_2 \rightarrow u \rightsquigarrow w$, $v_2 \rightarrow v_3 \rightarrow \dots \rightarrow v \rightsquigarrow w$.

For fixed $S_{u,v}$ and $w \in U \cup Z^*$ we have $k^{O(k)}$ different possible such paths, therefore we make $k^{O(k^5)}$ guesses for these paths. For an example see Figure 7b.

G-9. Split vertex guess I. Let $S_{u,v} = (u = v_1, v_2, \dots, v_p = v)$ be a segment in G' , and let us fix a vertex $v_i \in S_{u,v} \setminus \{u, v\}$. In the case when $S_{u,v}$ is of length 4, the fixed vertex v_i is the middle vertex, else we fix an arbitrary vertex $v_i \in S_{u,v} \setminus \{u, v\}$. Let $S_{w,z} = (w = z_1, z_2, \dots, z_r = z)$ be another segment in G' . We want to determine the fastest paths from v_i to all inner vertices of $S_{w,z}$. We do this by inspecting the values in matrix D from v_i to inner vertices of $S_{w,z}$. We split the analysis into two cases.

a. There is a single vertex $z_j \in S_{w,z}$ for which the duration from v_i is the biggest. More specifically, $z_j \in S_{w,z} \setminus \{w, z\}$ is the vertex with the biggest value D_{v_i, z_j} . We call this vertex a *split vertex of v_i in the segment S_{wz}* . Then it holds that $D_{v_i, z_2} < D_{v_i, z_3} < \dots < D_{v_i, z_j}$ and $D_{v_i, z_{r-1}} < D_{v_i, z_{r-2}} < \dots < D_{v_i, z_j}$. From this it follows that the fastest temporal paths from v_i to z_2, z_3, \dots, z_{j-1} go through w , and the fastest temporal paths from v_i to $z_{r-1}, z_{r-2}, \dots, z_{j+1}$ go through z . We now want to guess which vertex w or z is on a fastest temporal path from v_i to z_j . Similarly, all fastest temporal paths starting at v_i have to go either through u or through v , which also gives us two extra guesses for the fastest temporal path from v_i to z_j . Therefore, all together we have 4 possibilities on how the fastest temporal path from v_i to z_j starts and ends. Besides that we want to guess also how the fastest temporal paths from v_i to z_{j-1}, z_{j+1} start and end. Note that one of these is the subpath of the fastest temporal path from v_i to z_j , and the ending part is uniquely determined for both of them, i. e., to reach z_{j-1} the fastest temporal path travels through w , and to reach z_{j+1} the fastest temporal path travels through z . Therefore

we have to determine only how the path starts, namely if it travels through u or v . This introduces two extra guesses. For a fixed $S_{u,v}$ and $S_{w,z}$ we find the vertex z_j in polynomial time, or determine that z_j does not exist. We then make four guesses where we determine how the fastest temporal path from v_i to z_j passes through vertices u, v and w, z and for each of them two extra guesses to determine the fastest temporal path from v_i to z_{j-1} and from v_i to z_{j+1} . We repeat this procedure for all pairs of segments, which results in producing $k^{O(k^5)}$ new guesses. Note, $v_i \in S_{u,v}$ is fixed when calculating the split vertex for all other segments $S_{w,z}$.

- b. There are two vertices $z_j, z_{j+1} \in S_{w,z}$ for which the duration from v_i is the biggest. More specifically, $z_j, z_{j+1} \in S_{w,z} \setminus \{w, z\}$ are the vertices with the biggest value $D_{v_i, z_j} = D_{v_i, z_{j+1}}$. Then it holds that $D_{v_i, z_2} < D_{v_i, z_3} < \dots < D_{v_i, z_j} = D_{v_i, z_{j+1}} > D_{v_i, z_{j+2}} > \dots > D_{v_i, z_{r-1}}$. From this it follows that the fastest temporal paths from v_i to z_2, z_3, \dots, z_j go through w , and the fastest temporal paths from v_i to $z_{r-1}, z_{r-2}, \dots, z_{j+1}$ go through z . In this case we only need to guess the following two fastest temporal paths $v_i \rightsquigarrow w \rightarrow z_2$ and $v_i \rightsquigarrow z \rightarrow z_{r-1}$. Each of these paths we then uniquely extend along the segment $S_{w,z}$ up to the vertex z_j , resp. z_{j+1} , which give us fastest temporal paths from v_i to z_j and from v_i to z_{j+1} . In this case we introduce only two more guesses. We repeat this procedure for all pairs of segments, which results in creating $k^{O(k^5)}$ new guesses.

For an example see Figure 7b.

- G-10. Split vertex guess II.** Let $w \in U \cup Z^*$ and let $S_{u,v} = (u = v_1, v_2, \dots, v_p = v)$. We want to guess a split vertex of w in $S_{u,v}$, and the fastest temporal path that reaches it. We again have two cases, first one where v_i is a unique vertex in $S_{u,v}$ that is furthest away from w , and the second one where v_i, v_{i+1} are two incident vertices in $S_{u,v}$, that are furthest away from w . All together we make two guesses for each pair $w, S_{u,v}$. We repeat this for all vertices in $U \cup Z^*$, and all segments, which produces $k^{O(k^5)}$ new guesses. For an example see Figure 7c. Detailed analysis follows arguing from above (as in **G-9**) and is deferred to [46].

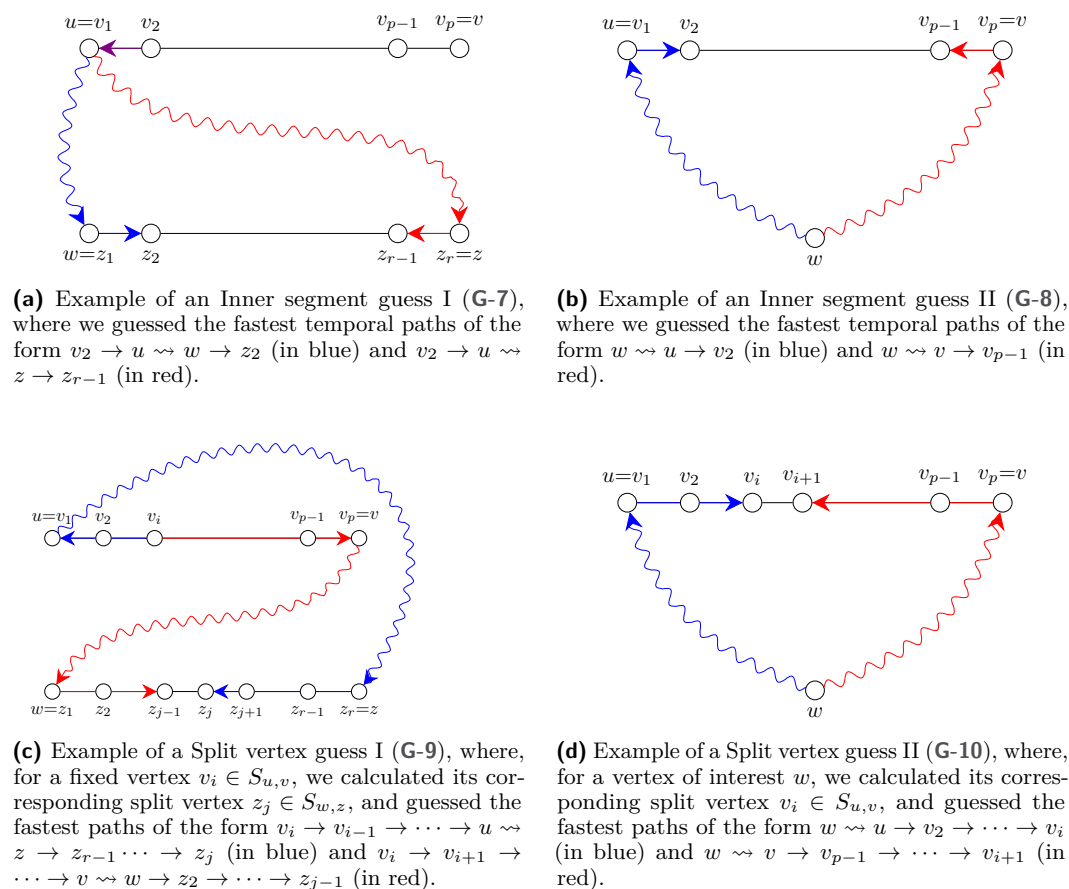
There are two more guesses **G-11** and **G-12** that are deferred to [46]. We prove in [46] that, for all guesses **G-1** to **G-12**, there are in total at most $f(k)$ possible choices, and for each one of them we create an ILP with at most $f(k)$ variables and at most $f(k) \cdot |D|^{O(1)}$ constraints. Each of these ILPs can be solved in FPT time by Lenstra's Theorem [49]. For detailed explanation and proofs of this part see [46].

4 Conclusion

We believe that our work spawns several interesting future research directions and builds a base upon which further temporal graph realization problems can be investigated.

There are several structural parameters which can be considered to obtain tractability which are either larger than or incomparable to the feedback vertex number. We believe that the *vertex cover number* or the *tree depth* are promising candidates. Furthermore, we can consider combining a structural parameter such as the *treewidth* with Δ .

There are many natural variants of our problem that are well-motivated and warrant consideration. We believe that one of the most natural generalizations of our problem is to allow more than one label per edge in every Δ -period. A well-motivated variant (especially from the network design perspective) of our problem is to consider the entries of the duration matrix D as upper-bounds on the duration of fastest paths rather than exact durations. This problem variant has very recently been studied by Mertzios et al. [56].



■ **Figure 7** Illustration of the guesses G-7, G-8, G-9, and G-10.

References

- 1 Eleni C Akrida, Leszek Gąsieniec, George B. Mertzios, and Paul G Spirakis. The complexity of optimal design of temporally connected graphs. *Theory of Computing Systems*, 61:907–944, 2017.
- 2 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Christoforos Raptopoulos. The temporal explorer who returns to the base. *Journal of Computer and System Sciences*, 120:179–193, 2021.
- 3 Emmanuel Arrighi, Niels Grüttemeier, Nils Morawietz, Frank Sommer, and Petra Wolf. Multi-parameter analysis of finding minors and subgraphs in edge-periodic temporal graphs. In *Proceedings of the 48th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 283–297, 2023.
- 4 John Augustine, Keerti Choudhary, Avi Cohen, David Peleg, Sumathi Sivasubramaniam, and Suman Sourav. Distributed graph realizations. *IEEE Transactions on Parallel and Distributed Systems*, 33(6):1321–1337, 2022.
- 5 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Efficiently realizing interval sequences. *SIAM Journal on Discrete Mathematics*, 34(4):2318–2337, 2020.
- 6 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Graph realizations: Maximum degree in vertex neighborhoods. In *Proceedings of the 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 10:1–10:17, 2020.

- 7 Amotz Bar-Noy, David Peleg, Mor Perry, and Dror Rawitz. Composed degree-distance realizations of graphs. In *Proceedings of the 32nd International Workshop on Combinatorial Algorithms (IWOCA)*, pages 63–77, 2021.
- 8 Amotz Bar-Noy, David Peleg, Mor Perry, and Dror Rawitz. Graph realization of distance sets. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 13:1–13:14, 2022.
- 9 Mehdi Behzad and James E Simpson. Eccentric sequences and eccentric sets in graphs. *Discrete Mathematics*, 16(3):187–193, 1976.
- 10 Robert E Bixby and Donald K Wagner. An almost linear-time algorithm for graph realization. *Mathematics of Operations Research*, 13(1):99–123, 1988.
- 11 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.
- 12 Arnaud Casteigts, Timothée Corsini, and Writika Sarkar. Invited paper: Simple, strict, proper, happy: A study of reachability in temporal graphs. In *Proceedings of the 24th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 3–18, 2022.
- 13 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.
- 14 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021.
- 15 Wai-Kai Chen. On the realization of a (p, s) -digraph with prescribed degrees. *Journal of the Franklin Institute*, 281(5):406–422, 1966.
- 16 Fan Chung, Mark Garrett, Ronald Graham, and David Shallcross. Distance realization problems with applications to internet tomography. *Journal of Computer and System Sciences*, 63(3):432–448, 2001.
- 17 Joseph C. Culberson and Piotr Rudnicki. A fast algorithm for constructing trees from distance matrices. *Information Processing Letters*, 30(4):215–220, 1989.
- 18 Argyrios Deligkas and Igor Potapov. Optimizing reachability sets in temporal graphs by delaying. *Information and Computation*, 285:104890, 2022.
- 19 Jessica Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021.
- 20 Jessica Enright, Kitty Meeks, and Fiona Skerman. Assigning times to minimise reachability in temporal graphs. *Journal of Computer and System Sciences*, 115:169–186, 2021.
- 21 Jessica A. Enright, Kitty Meeks, and Hendrik Molter. Counting temporal paths. In *Proceedings of the 40th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 254, pages 30:1–30:19, 2023.
- 22 Paul Erdős and Tibor Gallai. Graphs with prescribed degrees of vertices. *Mat. Lapok*, 11:264–274, 1960.
- 23 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *Journal of Computer and System Sciences*, 119:1–18, 2021.
- 24 Thomas Erlebach, Nils Morawietz, and Petra Wolf. Parameterized algorithms for multi-label periodic temporal graph realization. In *Proceedings of the 3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, pages 12:1–12:16, 2024. doi:10.4230/LIPIcs.SAND.2024.12.
- 25 Thomas Erlebach and Jakob T. Spooner. A game of cops and robbers on graphs with periodic edge-connectivity. In *Proceedings of the 46th International Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, pages 64–75, 2020.

- 26 Michael R. Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.
- 27 Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013.
- 28 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. *Theoretical Computer Science*, 806:197–218, 2020.
- 29 András Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM Journal on Discrete Mathematics*, 5(1):25–53, 1992.
- 30 András Frank. Connectivity augmentation problems in network design. *Mathematical Programming: State of the Art 1994*, 1994.
- 31 H. Frank and Wushow Chou. Connectivity considerations in the design of survivable networks. *IEEE Transactions on Circuit Theory*, 17(4):486–490, 1970.
- 32 Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Delay-robust routes in temporal graphs. In *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 30:1–30:15, 2022.
- 33 Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Temporal connectivity: Coping with foreseen and unforeseen delays. In *Proceedings of the 1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, pages 17:1–17:17, 2022.
- 34 D.R. Fulkerson. Zero-one matrices with zero trace. *Pacific Journal of Mathematics*, 10(3):831–836, 1960.
- 35 Petr A. Golovach and George B. Mertzios. Graph editing to a given degree sequence. *Theoretical Computer Science*, 665:1–12, 2017.
- 36 Martin Charles Golumbic and Ann N. Trenk. *Tolerance Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2004.
- 37 Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.
- 38 Martin Grötschel, Clyde L Monma, and Mechthild Stoer. Design of survivable networks. *Handbooks in Operations Research and Management Science*, 7:617–672, 1995.
- 39 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 162–173, 2004.
- 40 S. Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph. I. *Journal of the Society for Industrial and Applied Mathematics*, 10(3):496–506, 1962.
- 41 S. Louis Hakimi and Stephen S. Yau. Distance matrix of a graph and its realizability. *Quarterly of applied mathematics*, 22(4):305–317, 1965.
- 42 Pavol Hell and David Kirkpatrick. Linear-time certifying algorithms for near-graphical sequences. *Discrete Mathematics*, 309(18):5703–5713, 2009.
- 43 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.
- 44 Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: Temporally disjoint paths. *Autonomous Agents and Multi-Agent Systems*, 37(1):1, 2023.
- 45 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. The complexity of computing optimum labelings for temporal connectivity. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 62:1–62:15, 2022.
- 46 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Realizing temporal graphs from fastest travel times. *CoRR*, abs/2302.08860, 2023. doi:10.48550/arXiv.2302.08860.

- 47 Fabian Kuhn and Rotem Oshman. Dynamic networks: Models and algorithms. *SIGACT News*, 42(1):82–96, March 2011.
- 48 Pascal Kunz, Hendrik Molter, and Meirav Zehavi. In which graph structures can we efficiently find temporally disjoint paths and walks? In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 180–188, 2023.
- 49 Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.
- 50 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
- 51 Linda Lesniak. Eccentric sequences in graphs. *Periodica Mathematica Hungarica*, 6:287–293, 1975.
- 52 Ross M. McConnell and Jeremy P. Spinrad. Construction of probe interval models. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 866–875, 2002.
- 53 F.R. McMorris, Chi Wang, and Peisen Zhang. On probe interval graphs. *Discrete Applied Mathematics*, 88(1):315–324, 1998. Computational Molecular Biology DAM - CMB Series.
- 54 George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019.
- 55 George B. Mertzios, Hendrik Molter, Malte Renken, Paul G. Spirakis, and Philipp Zschoche. The complexity of transitively orienting temporal graphs. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 75:1–75:18, 2021.
- 56 George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Realizing temporal transportation trees. *CoRR*, abs/2403.18513, 2024. doi:10.48550/arXiv.2403.18513.
- 57 Hendrik Molter, Malte Renken, and Philipp Zschoche. Temporal reachability minimization: Delaying vs. deleting. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 76:1–76:15, 2021.
- 58 Nils Morawietz, Carolin Rehs, and Mathias Weller. A timecop’s work is harder than you think. In *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 170, pages 71–1, 2020.
- 59 Nils Morawietz and Petra Wolf. A timecop’s chase around the table. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2021.
- 60 A.N. Patrinos and S. Louis Hakimi. The distance matrix of a graph and its tree realization. *Quarterly of Applied Mathematics*, 30:255–269, 1972.
- 61 Elena Rubei. Weighted graphs with distances in given ranges. *Journal of Classification*, 33:282–297, 2016.
- 62 Piotr Sapiezynski, Arkadiusz Stopczynski, Radu Gatej, and Sune Lehmann. Tracking human mobility using wifi signals. *PloS one*, 10(7):e0130824, 2015.
- 63 H. Tamura, M. Sengoku, S. Shinoda, and T. Abe. Realization of a network from the upper and lower bounds of the distances (or capacities) between vertices. In *Proceedings of the 1993 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2545–2548, 1993.
- 64 Huanhuan Wu, James Cheng, Yiping Ke, Silu Huang, Yuzhen Huang, and Hejun Wu. Efficient algorithms for temporal path computation. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2927–2942, 2016.
- 65 Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding separators in temporal graphs. *Journal of Computer and System Sciences*, 107:72–92, 2020.


An Analysis of the Recurrence/Transience of Random Walks on Growing Trees and Hypercubes

Shuma Kumamoto ✉

Graduate School of Mathematical Science, Kyushu University, Fukuoka, Japan

Shuji Kijima ✉ 

Faculty of Data Science, Shiga University, Hikone, Japan

Tomoyuki Shirai ✉ 

Institute of Mathematics for Industry, Kyushu University, Fukuoka, Japan

Abstract

It is a celebrated fact that a simple random walk on an *infinite* k -ary tree for $k \geq 2$ returns to the initial vertex at most finitely many times during infinitely many transitions; it is called *transient*. This work points out the fact that a simple random walk on an infinitely *growing* k -ary tree can return to the initial vertex infinitely many times, it is called *recurrent*, depending on the growing speed of the tree. Precisely, this paper is concerned with a simple specific model of a *random walk on a growing graph (RWoGG)*, and shows a phase transition between the recurrence and transience of the random walk regarding the growing speed of the graph. To prove the phase transition, we develop a coupling argument, introducing the notion of *less homesick as graph growing (LHaGG)*. We also show some other examples, including a random walk on $\{0, 1\}^n$ with infinitely growing n , of the phase transition between the recurrence and transience. We remark that some graphs concerned in this paper have infinitely growing degrees.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains

Keywords and phrases Random walk, dynamic graph, recurrent, transient

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.17

Related Version *Full Version:* <https://arxiv.org/abs/2405.09102>

Funding *Shuma Kumamoto:* This work is supported by JST SPRING, Grant Number JPMJSP2136.

Shuji Kijima: This work is partly supported by JSPS KAKENHI Grant Number JP21H03396.

Tomoyuki Shirai: This work is partly supported by JSPS KAKENHI Grant Number JP20K20884, JP22H05105 and JP23H01077.

1 Introduction

The recurrence or transience is a classical and fundamental topic of random walks on *infinite* graphs, see e.g., [18]: let X_0, X_1, X_2, \dots be a random walk (or a Markov chain)¹ on an infinite state space V , e.g., $V = \mathbb{Z}$, with $X_0 = v$ for $v \in V$. For convenience, let

$$R(t) = \Pr[X_t = v] \quad (= \Pr[X_t = v \mid X_0 = v])$$

denote the probability that a random walk returns to the initial state at time step t ($t = 1, 2, \dots$), and then the initial point v is *recurrent* by the random walk if

$$\sum_{t=1}^{\infty} R(t) = \infty \tag{1}$$

¹ This paper is concerned with discrete time and space processes. We will be mainly concerned with time-*inhomogeneous* Markov chains, but here you may assume a time-homogeneous chain, i.e., the transition probability $\Pr[X_{t+1} = v \mid X_t = u]$ is independent of the time t , but depends on u, v .



holds, otherwise it is *transient*. Intuitively, (1) means that the random walk is “expected” to return to the initial state infinitely many times. It is well known that a simple random walk on \mathbb{Z}^d is recurrent for $d = 1, 2$, while it is transient for $d \geq 3$, cf. [18]. Another celebrated fact is that a simple random walk on an infinite k -ary tree is transient [28, 29].

Analysis of random walks on *dynamic graphs* has been developed in several contexts. In probability theory, random walks in random environments are a major topic, where self-interacting random walks including reinforced random walks and excited random walks have been intensively investigated as a relatively tractable non-Markovian process, see e.g., [11, 6, 17, 31, 32, 24]. The recurrence or transience of a random walk in a random environment is a major topic there, particularly random walks on growing subgraphs of \mathbb{Z}^d or on infinitely growing trees are the major targets [13, 14, 21, 1]. In distributed computing, analysis of algorithms, including random walk, on dynamic graph attracts increasing attention due to the fact that real networks are often dynamic [8, 25, 3, 30]. Searching or covering networks, related to hitting or cover times of random walks, are major topics there [9, 4, 16, 5, 26, 7, 23].

This work is concerned with the recurrence/transience of a random walk on a growing graph. We show the fact that a simple random walk on an infinitely growing complete k -ary tree can be recurrent *depending on the growing speed* of the tree, while a simple random walk on an infinite k -ary tree is transient as we mentioned above. More precisely, this paper follows the model of the random walk on growing graph (RWoGG) [23], where the network gradually grows such that the growing network keeps its shape $G(n)$ for $\mathfrak{d}(n)$ steps, and then changes the shape to $G(n+1)$ by adding some vertices to $G(n)$ (see Section 2.1 for detail). Then, we show a phase transition between the recurrence and transience of a random walk on a growing k -ary tree, regarding the growing speed of the graph. For a proof, we develop the notion of *less-homesick as graph growing* (LHaGG), which is a quite natural property of RWoGG, and gives a simple proof by a *coupling* argument, that is an elementary technique of random walks or Markov chains based on a comparison method. We also show some other examples of the phase transition, including as a random walk on $\{0, 1\}^n$ with infinitely growing n .

1.1 Existing works and contribution of the paper

The recurrence/transience of a random walk on a dynamic graph has been mainly developed in the context of random walks in random environment including reinforced random walks and excited walks. Here we briefly review some existing works concerning the recurrence of a random walk on \mathbb{Z}^d and infinite (or infinitely growing) trees, directly related to this paper.

Random walks on (asymptotically) \mathbb{Z}^d . It is a celebrated fact that the initial point, say origin $\mathbf{0}$, in the infinite integer grid \mathbb{Z}^d is recurrent when $d = 1$ and 2 by a simple random walk, and it is transient for $d \geq 3$, see e.g., [18].

Dembo et al. [14] is concerned with a random walk on an infinitely growing subgraph of \mathbb{Z}^d , and gave a phase transition, that is roughly speaking a random walk is recurrent if and only if $\sum_{t=1}^{\infty} \pi_t(\mathbf{0}) = \infty$ holds under a certain condition, where π_t denotes the stationary distribution of the transition matrix at time t . Huang [21] extended the argument of [14] and gave a similar or essentially the same phase transition for more general graphs. The proofs are based on the edge conductance and a central limit theorem, on the assumptions that every vertex of the dynamic graph has a degree at most *constant* to time (or the size of the graph), and the random walk is “lazy” such that it has at least a *constant* probability of self-loops at every vertex. Those arguments are sophisticated and enhanced using the argument of evolving set and the heat kernel by recent works [12, 15].

Random walks on infinitely growing trees. Lyons [28] studied sufficient conditions for a random walk being recurrent/transient, see also [29]. Roughly speaking, the initial point, say the root r , is recurrent if and only if the random walk is enough *homesick*, meaning that a random walk probabilistically tends to choose the direction to the root.

Amir et al. [1] introduced a random walk in changing environment model, and investigated the recurrence and transience of random walks in the model. They gave a conjecture about the conditions for the recurrence and transience regarding the limit of a graph sequence, and proved it for trees. Huang's work [21], which we mentioned above, implies that a simple random walk starting from a vertex v on growing k -ary tree is recurrent if and only if $\sum \pi_t(v) = \infty$, that is similar to or essentially the same as a main result of this paper under a certain condition. We remark that a k -ary tree with height n is not an (edge induced) subgraph of \mathbb{Z}^d for a *constant* d .

There is a lot of work on the recurrence or transience of a random walk on a growing tree, related to self-interacting random walks including reinforced random walks and excited random walks, e.g., [22, 19]. They are non-Markovian processes, and in a bit different line from [14, 1, 21] and this paper.

Contribution of this work. This paper is concerned with a specific model of dynamic graphs with an increasing number of vertices, which we will describe in Section 2.1, and gives a phase transition by the growing speed regarding a random walk being recurrent/transient. The phase transition is very similar to or essentially the same as [14, 21], while this paper contains mainly three contributions. One is the proof technique: we employ a coupling argument while the existing works are based on the conductance and a central limit theorem. The coupling arguments is a classical and elementary comparison technique of random walks, and we introduce the notion of LHaGG to use the comparison technique. Since the coupling technique is relatively simple, we can drop two assumptions in the existing works, namely a random walk being *lazy* and a growing graph having *uniformly bounded degree*, which are naturally required in the conductance argument to make the arguments simple. This paper is mainly concerned with reversible random walks of *period 2*, which contains simple random walks on undirected bipartite graphs; this is the second contribution. We also show an example of random walk on $\{0, 1\}^n$ with increasing n , where the (maximum) degree of the dynamic graph, that is n , infinitely grows; this is the third contribution.

While the coupling technique is relatively easy, it often selects the applicable target. In fact, the results by [14, 21] are widely applied to general setting as far as it satisfies appropriate assumptions, while our result is limited to specific targets. Such an argument about conductance and coupling seems known as an implicit knowledge in the literature of mixing time analysis, cf. [2, 20]. However, we emphasize that the coupling technique often gives an easy proof of an interesting phenomena, as this paper shows.

1.2 Organization

As a preliminary, we describe the model of random walk on growing graph (RWoGG) in Section 2. Section 3 introduces the notion of less homesickness as graph growing (LHaGG), and presents some general theorems for sufficient conditions of a RWoGG being recurrent/transient. Section 4 shows a phase transition between the recurrence and transience of a random walk on growing k -ary tree. Section 5 shows a phase transition for a random walk on $\{0, 1\}^n$ with increasing n .

2 Preliminaries

2.1 Model

A growing graph is a sequence of (static) graphs $\mathcal{G} = \mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, \dots$ where $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ for $t = 0, 1, 2, \dots$ denotes a graph² with a finite vertex set \mathcal{V}_t and an edge set $\mathcal{E}_t \subseteq \binom{\mathcal{V}_t}{2}$. For simplicity, this paper assumes³ $\mathcal{V}_t \subseteq \mathcal{V}_{t+1}$ and $\mathcal{E}_t \subseteq \mathcal{E}_{t+1}$. In this paper, we assume $|\mathcal{V}_\infty| = \infty$, otherwise the subject (recurrence) is trivial. A random walk on a growing graph is a Markovian series $X_t \in \mathcal{V}_t$ ($t = 0, 1, 2, \dots$).

In particular, this paper is concerned with a specific model, described as follows, cf. [23]. A *random walk on a growing graph (RWoGG)*, in this paper, is formally characterized by a 3-tuple of functions $\mathcal{D} = (\mathfrak{d}, G, P)$. The function $\mathfrak{d}: \mathbb{Z}_{>0} \rightarrow \mathbb{Z}_{\geq 0}$ denotes the duration. For convenience, let $T_n = \sum_{i=1}^n \mathfrak{d}(i)$ for $n = 1, 2, \dots$ ⁴ and $T_0 = 0$. We call the time interval $[T_{n-1}, T_n]$ *phase n* for $n = 1, 2, \dots$; thus $T_{n-1} = \sum_{i=1}^{n-1} \mathfrak{d}(i)$ is the beginning of the n -th phase, but we also say that T_{n-1} is the end of the $(n-1)$ -st phase, for convenience. The function $G: \mathbb{Z}_{>0} \rightarrow \mathfrak{G}$ represents the graph $G(n) = (V(n), E(n))$ for the phase n , where \mathfrak{G} denotes the set of all (static) graphs, i.e., our growing graph \mathcal{G} satisfies $\mathcal{G}_t = G(n)$ for $t \in [T_{n-1}, T_n]$. Similarly, the function $P: \mathbb{Z}_{>0} \rightarrow \mathfrak{M}$ is a function that represents the “transition probability” of a random walk on graph $G(n)$ where \mathfrak{M} denotes the set of all stochastic matrices.

A RWoGG X_t ($t = 0, 1, 2, \dots$) characterized by $\mathcal{D} = (\mathfrak{d}, G, P)$ is temporally a time-homogeneous finite Markov chain according to $P(n)$ with the state space $V(n)$ during the time interval $[T_{n-1}, T_n]$; precisely, a transition from X_t to X_{t+1} follows $P(n)$ for any $t \in [T_{n-1}, T_n)$. We specially remark for $t = T_n$ that $X_t \in V(n) \subseteq V(n+1)$, meaning that X_t is a state of $V(n+1)$ but actually X_t must be in $V(n)$ by the definition of the transition. Suppose $X_0 = v$ for $v \in V(1)$. We define the return probability at v by

$$R(t) = \Pr[X_t = v] \quad (= \Pr[X_t = v \mid X_0 = v]) \quad (2)$$

at each time $t = 0, 1, 2, \dots$. We say v is *recurrent* by RWoGG $\mathcal{D} = (\mathfrak{d}, G, P)$ if

$$\sum_{t=1}^{\infty} R(t) = \infty \quad (3)$$

holds, otherwise, i.e., $\sum_{t=1}^{\infty} R(t)$ is finite, v is *transient* by \mathcal{D} .

2.2 Terminology on time-homogeneous Markov chains

We here briefly introduce some terminology for random walks on static graphs, or time-homogeneous Markov chains, according to [27].

2.2.1 Ergodic random walks

Suppose that X_0, X_1, X_2, \dots is a random walk on a static graph $G = (V, E)$ characterized by a time-homogeneous transition matrix $P = (P(u, v)) \in \mathbb{R}_{\geq 0}^{V \times V}$ where $P(u, v) = \Pr[X_{t+1} = v \mid X_t = u]$. A random walk is *reversible* if there exists a positive function $\mu: V \rightarrow \mathbb{R}_{>0}$ such that $\mu(u)P(u, v) = \mu(v)P(v, u)$ hold for all $u, v \in V$. A transition matrix P is *irreducible* if

² Every static graph is simple and undirected in this paper, for simplicity of the arguments.

³ Thus, the current position does not disappear in the next step.

⁴ We do not exclude $T_{n-1} = T_n$; if $\mathfrak{d}(n) = 0$ then $T_{n-1} = T_n$.

$\forall u, v \in V, \exists t > 0, (P^t)(u, v) > 0$. The period of P is given by $\text{period}(P) = \min_{v \in V} \gcd\{t > 0 : (P^t)(v, v) > 0\}$. It is well known that $\gcd\{t > 0 : (P^t)(v, v) > 0\}$ is common for any $v \in V$ if P is irreducible.

If $\text{period}(P) = 1$ then P is said to be *aperiodic*. A transition matrix P is *ergodic* if it is irreducible and aperiodic. We say a random walk is (γ) -*lazy* if $P(v, v) \geq \gamma$ holds for any $v \in V$ for a constant γ ($0 < \gamma < 1$). A lazy random walk is clearly aperiodic. A probability distribution π over V is a *stationary distribution* if it satisfies $\pi P = \pi$. It is well known that an ergodic P has a unique stationary distribution [27]. The *mixing time* of P is given by

$$\tau(\epsilon) \stackrel{\text{def.}}{=} \min \left\{ t \mid t \in \mathbb{Z}_{>0}, \frac{1}{2} \max_{u \in V} \sum_{v \in V} |P^t(u, v) - \pi(v)| \leq \epsilon \right\} \quad (4)$$

for $\epsilon \in (0, 1)$.

2.2.2 Random walk with period 2

A *simple* random walk (or “busy” simple random walk) on an undirected graph $G = (V, E)$ is given by $P(u, v) = 1/\deg(u)$ for $\{u, v\} \in E$ where $\deg(u)$ denotes the degree of $u \in V$ on G . This paper is mainly concerned with bipartite graphs, such as trees, integer grids, and 0-1 hypercubes, and then the most targeted random walks are irreducible and reversible, but *not aperiodic*.

► **Observation 1.** *If P is reversible then its period is at most 2.*

Suppose P is irreducible and reversible, and it has period 2. Then, the underlying graph is a connected bipartite $(U, \bar{U}; E)$, where $U = \{u \in V \mid \exists t', P^{2t'}(v, u) \neq 0\}$ for any $v \in U$, $\bar{U} = \{u \mid \forall t, P^{2t}(v, u) = 0\}$, i.e., $\bar{U} = V \setminus U$, and $E = \{\{u, v\} \in V^2 \mid P(u, v) > 0\}$. Notice that E does not contain any self-loop, otherwise, P is aperiodic.

Here, we introduce some unfamiliar terminology for periodic Markov chains. We say $\hat{x} \in \mathbb{R}_{\geq 0}^V$ is *even-time distribution* if it satisfies $\sum_{v \in V} \hat{x}(v) = 1$ and $\hat{x}(u) = 0$ for any $u \in \bar{U}$. We say $\hat{\pi} \in \mathbb{R}_{\geq 0}^V$ is *even-time stationary distribution* if it is an even-time distribution and satisfies $\hat{\pi} P^2 = \hat{\pi}$.

► **Proposition 2** (limit distribution). *Suppose P is irreducible and reversible, and it has period 2. Then, P has a unique even-time stationary distribution $\hat{\pi}$, and $\lim_{t \rightarrow \infty} \hat{x} P^{2t} = \hat{\pi}$ for any even-time distribution \hat{x} .*

We define the *even mixing-time* of P by

$$\hat{\tau}(\epsilon) = \min \left\{ 2t' \mid t' \in \mathbb{Z}_{>0}, \frac{1}{2} \max_{u \in U} \sum_{v \in U} |P^{2t'}(u, v) - \hat{\pi}(v)| \leq \epsilon \right\} \quad (5)$$

for $\epsilon \in (0, 1)$. We remark that the even mixing-time of P is equal to the twice of the mixing time of $P^2[U]$, where $P^2[U]$ denotes the submatrix of P induced by U . Thus, we can use some standard arguments, e.g., coupling technique, about the even mixing-time of P . Finally, we remark on a proposition, that plays a key role in our analysis.

► **Proposition 3** (Proposition 10.25 in [27]). *If P is reversible then $\hat{\pi}(v) \leq P^{2t+2}(v, v) \leq P^{2t}(v, v)$ for any $t = 0, 1, 2, \dots$*

3 Analytical Framework: LHaGG

This section introduces the notion of less-homesickness as graph growing (LHaGG), and presents general theorems (Lemmas 5 and 6) describing some sufficient conditions of a RWoGG being recurrent or transient. See the following sections for specific RWoGGs, namely, RW on growing k -ary tree in Section 4, RW on $\{0, 1\}^n$ hypercube skeleton with increasing n in Section 5, etc.

3.1 Less-homesick as graph growing

Let $\mathcal{D} = (f, G, P)$ and $\mathcal{D}' = (f', G', P')$ be RWoGG, and let $R(t)$ and $R'(t)$ respectively denote their return probabilities to respective initial vertices at time $t = 1, 2, \dots$. We say \mathcal{D} is *less-homesick* than $\mathcal{D}' = (f', G', P')$ at time t if $R(t) \leq R'(t)$ holds.

In particular, this paper is mainly concerned with the less-homesick relationship between $\mathcal{D} = (f, G, P)$ and $\mathcal{D}' = (g, G, P)$ with the same P, G and the initial vertex v . We say \mathcal{D} is *less-homesick as graph growing (LHaGG)*⁵ if $\mathcal{D} = (f, G, P)$ is less-homesick than for any $\mathcal{D}' = (g, G, P)$ satisfying that

$$\sum_{i=1}^n f(i) \leq \sum_{i=1}^n g(i) \quad (6)$$

for any $n \in \mathbb{Z}_{>0}$. The condition (6) intuitively implies that the graph in \mathcal{D} grows faster than \mathcal{D}' . For instance, we will prove that the simple random walk on growing k -regular tree is LHaGG, in Section 4.

► **Lemma 4.** *Suppose RWoGG $\mathcal{D} = (f, G, P)$ is LHaGG. Let X_t ($t = 0, 1, 2, \dots$) be a RWoGG according to \mathcal{D} with $X_0 = v \in V(1)$. Let Y_t ($t = 0, 1, 2, \dots$) be a random walk on (a static graph) $G(n)$ according to $P(n)$ with $Y_0 = v$, where G, P and v are common with \mathcal{D} . Then, Y_t is less-homesick than X_t at any time $t \in [T_n, T_{n+1}]$, i.e., $R(t) \geq R'(t)$ holds for $t \in [T_n, T_{n+1}]$, where $R(t) = \Pr[X_t = v]$ and $R'(t) = \Pr[Y_t = v]$.*

Proof. Let

$$g(i) = \begin{cases} 0 & (i < n), \\ \sum_{j=1}^n f(j) & (i = n), \\ f(i) & (i > n). \end{cases}$$

Then, the static random walk Y_t on $G(n)$ also follows $\mathcal{D}' = (g, G, P)$ for $t \leq T_{n+1}$. Clearly, $\sum_{i=1}^n f(i) \geq \sum_{i=1}^n g(i)$ for any n . Since \mathcal{D} is LHaGG by the hypothesis, $R(t) \geq R'(t)$. ◀

We remark that if all P_n takes period 2 then $R(t) = R'(t) = 0$ for any odd t .

3.2 Recurrent

We prove the following lemma, presenting a sufficient condition for a RWoGG to be recurrent.

⁵ Strictly speaking, LHaGG should be a property of the sequence of transition matrices $P(1), P(2), P(3), \dots$. For the convenience of the notation, we say $\mathcal{D} = (f, G, P)$ is LHaGG, in this paper.

► **Lemma 5.** *Suppose that RWoGG $\mathcal{D} = (\mathfrak{d}, G, P)$ is LHaGG, and that every $P(n) = P_n$ ($n = 1, 2, \dots$) is irreducible, reversible and $\text{period}(P_n) = 2$. Let $p(n) = \hat{\pi}_n(v)$ where $\hat{\pi}_n$ denote the even-time stationary distribution of P_n . If \mathfrak{d} satisfies*

$$\sum_{n=1}^{\infty} (\mathfrak{d}(n) - 1)p(n) = \infty \quad (7)$$

then v is recurrent by \mathcal{D} .

Proof. Let $f(n) = 2\lfloor \frac{\mathfrak{d}(n)}{2} \rfloor$, i.e., $f(n) = \mathfrak{d}(n)$ if $\mathfrak{d}(n)$ is even, otherwise $f(n) = \mathfrak{d}(n) - 1$. For convenience, let $T'_n = \sum_{k=1}^n f(k)$ for $n = 1, 2, \dots$, and let $T'_0 = 0$. Let X_t (resp. X'_t) for $t = 0, 1, 2, \dots$ be a RWoGG according to $\mathcal{D} = (\mathfrak{d}, G, P)$ (resp. $\mathcal{D}' = (f, G, P)$), and let $R(t)$ (resp. $R'(t)$) denote the return probability of X_t (resp. X'_t). The hypothesis LHaGG implies $R(t) \geq R'(t)$. Let Y_t^n ($t = 0, 1, \dots, T'_n$) be a time-homogeneous random walk according to $P(n)$, and let $R''_n(t)$ ($t = 1, \dots, T'_n$) denote the return probability of Y_t^n . The hypothesis LHaGG and Lemma 4 implies

$$R'(t) \geq R''_n(t) \quad (8)$$

for $t \in (T'_{n-1}, T'_n]$. Then, we can see

$$\begin{aligned} \sum_{t=1}^{\infty} R(t) &\geq \sum_{t=1}^{\infty} R'(t) && \text{(by LHaGG)} \\ &= \sum_{n=1}^{\infty} \sum_{t=T'_{n-1}+1}^{T'_n} R'(t) \\ &\geq \sum_{n=1}^{\infty} \sum_{t=T'_{n-1}+1}^{T'_n} R''_n(t) && \text{(by (8))} \\ &= \sum_{n=1}^{\infty} \sum_{i=1}^{f(n)} R''_n(T'_{n-1} + i) && \text{(recall } T'_n = T'_{n-1} + f(n)\text{)} \\ &= \sum_{n=1}^{\infty} \sum_{i'=1}^{\frac{f(n)}{2}} R''_n(T'_{n-1} + 2i') && \text{(notice that } R''_n(T'_{n-1} + 2i' - 1) = 0\text{)} \\ &\geq \sum_{n=1}^{\infty} \sum_{i'=1}^{\frac{f(n)}{2}} p(n) && \text{(by Proposition 3)} \\ &= \frac{1}{2} \sum_{n=1}^{\infty} f(n)p(n) \\ &\geq \frac{1}{2} \sum_{n=1}^{\infty} (\mathfrak{d}(n) - 1)p(n) && (9) \end{aligned}$$

hold. If (7) holds then (9) is ∞ , meaning that v is recurrent by \mathcal{D} . ◀

It is not difficult to see that a similar proposition holds for *lazy* random walks.

3.3 Transient

This section establishes the following lemma, which suggests Lemma 5 is nearly optimal. In fact, we will provide an example of a random walk on a growing k -ary tree in Section 4, that shows a tight example of Lemma 5.

► **Lemma 6.** *Suppose that a RWoGG $\mathcal{D} = (\mathfrak{d}, G, P)$ is LHaGG, and that every $P(n) = P_n$ ($n = 1, 2, \dots$) is irreducible and reversible with $\text{period}(P_n) = 2$. Let $p(n) = \hat{\pi}_n(v)$ where $\hat{\pi}_n$ denote the even-time stationary distribution of P_n . Let $\hat{\tau}_n(\epsilon)$ denote the even mixing-time of $P(n)$, and let*

$$\mathfrak{i}(n) = \hat{\tau}_n(p(n))$$

for $n = 2, 3, \dots$. If

$$\max\{\mathfrak{d}(1), \mathfrak{i}(1)\} + \sum_{n=2}^{\infty} \max\{\mathfrak{d}(n), \mathfrak{i}(n)\} p(n-1) < \infty \quad (10)$$

holds then v is transient by \mathcal{D} .

Proof. Let

$$f(n) = \max\{\mathfrak{d}(n), \mathfrak{i}(n)\}$$

for $n = 1, 2, 3, \dots$. Let $R(t)$ and $R'(t)$ respectively denote the return probabilities of $\mathcal{D} = (\mathfrak{d}, G, P)$ and $\mathcal{D}' = (f, G, P)$. Clearly, $f(n) \geq \mathfrak{d}(n)$ for any n , LHaGG implies

$$R(t) \leq R'(t) \quad (11)$$

for any $t = 0, 1, 2, \dots$. For convenience, let

$$T'_n = \sum_{k=1}^n f(k) \quad (12)$$

for $n = 1, 2, \dots$.

We carry a tricky argument in the following: roughly speaking we compare \mathcal{D}' with P_{n-1} in the n -th round, i.e., $[T_{n-1}, T_n]$, for $n = 2, 3, \dots$. Let

$$g_{n-1}(k) = \begin{cases} f(k) & (k \leq n-2) \\ \infty & (k = n-1) \end{cases}$$

for $n = 2, 3, \dots$. Let $Z_t^{(n-1)}$ ($t = 0, 1, 2, \dots$) denote a RWoGG (g_{n-1}, G, P) , where $Z_0^{(n-1)} = v$. Let $R''_{n-1}(t)$ denote the return probability of $Z_t^{(n-1)}$. Clearly, $\sum_{i=1}^j f(i) \leq \sum_{i=1}^j g_{n-1}(i)$ holds for any j , hence the LHaGG assumption implies

$$R'(t) \leq R''_{n-1}(t) \quad (13)$$

for any $t = 0, 1, 2, \dots$ for any $n = 2, 3, \dots$.

Notice that $Z_t^{(n-1)}$ for $t \in [T_{n-2}, T_n]$ is nothing but a time-homogeneous random walk according to P_{n-1} with the “initial state” $Z_{T_{n-2}} = v$ for $n = 2, 3, \dots$. Since

$$T'_{n-1} = T'_{n-2} + f(n-1) \geq T'_{n-2} + \mathfrak{i}(n-1) = T'_{n-2} + \hat{\tau}_{n-1}(p(n-1))$$

$Z_t^{(n-1)}$ mixes well for $t > T'_{n-1}$, meaning that $|\Pr[Z_t^{(n-1)} = v] - \hat{\pi}_{n-1}(v)| \leq p(n-1)$ for any even $t \in (T'_{n-1}, T_n]$. This implies

$$R''_{n-1}(t) = \Pr[Z_t^{(n-1)} = v] \leq \hat{\pi}_{n-1}(v) + p(n-1) = 2p(n-1) \quad (14)$$

holds⁶ for $t \in (T'_{n-1}, T'_n]$, where we remark that $R''_{n-1}(t) = 0$ for any odd t . Then,

$$\begin{aligned}
\sum_{t=1}^{\infty} R(t) &\leq \sum_{t=1}^{\infty} R'(t) && \text{(by (11))} \\
&= \sum_{n=1}^{\infty} \sum_{t=T'_{n-1}+1}^{T'_n} R'(t) \\
&\leq f(1) + \sum_{n=2}^{\infty} \sum_{t=T'_{n-1}+1}^{T'_n} R'(t) \\
&\leq f(1) + \sum_{n=2}^{\infty} \sum_{t=T'_{n-1}+1}^{T'_n} R''_{n-1}(t) && \text{(by (13))} \\
&\leq f(1) + \sum_{n=2}^{\infty} \sum_{t=T'_{n-1}+1}^{T'_n} 2p(n-1) && \text{(by (14))} \\
&= f(1) + 2 \sum_{n=2}^{\infty} f(n)p(n-1)
\end{aligned}$$

holds. Now it is easy to see that (10) implies $\sum_{t=1}^{\infty} R(t) < \infty$, meaning that v is transient by \mathcal{D} . \blacktriangleleft

It is not difficult to see that a similar proposition holds for *lazy* random walks.

4 Random Walk on a Growing Complete k -ary Tree

Lyons gave sufficient conditions that a random walk on an infinite tree gets recurrent or transient at the root (initial point), cf. [28, 29], as a consequence, it is a celebrated fact that a simple random walk on an infinite k -ary tree is transient. This section shows that a simple random walk on a *moderately* growing complete k -ary tree is recurrent at the root.

4.1 Result summary

Let k be an integer greater than one, and let $G_n = (V_n, E_n)$ denote a *complete k -ary tree* with height n for $n = 1, 2, \dots$, i.e., $|V_n| = \sum_{i=0}^n k^i = \frac{k^{n+1}-1}{k-1}$, every internal node (including the root) has exactly k children, and every leaf places the same height n . Let $r \in V_n$ denote the root, that is the unique vertex of height 0. For convenience, let $h(v)$ denote the height of vertex $v \in V_n$, i.e., $h(r) = 0$, and $h(v) = n$ if and only if v is a leaf of G_n . Let

$$U_n = \{v \in V_n \mid h(v) \equiv 0 \pmod{2}\} \tag{15}$$

denote the vertices of even heights, and thus $\bar{U}_n = V_n \setminus U_n$ is the vertices of odd heights. Clearly, $G_n = (U_n, \bar{U}_n; E_n)$ is a bipartite graph. See [10] for a standard terminology about a complete k -ary tree, e.g., parent, child, root, internal node, leaf, height.

⁶ We remark this argument requires only *point-wise additive error* bound, instead of total variation. Clearly, point-wise additive error is upper bounded by total variation. We here use the mixing time for total variation just because it has been better analyzed than the other.

17:10 Recurrence/Transience of Random Walks on Growing Trees and Hypercubes

Next, we define a transition probability of a random walk over G_n according to [28, 29]. Let λ be a fixed positive real⁷, and we define a transition probability on the k -ary tree G_n with height n by

$$P_n(u, v) = \begin{cases} \frac{1}{k} & \text{if } u = r \text{ and } v \text{ is a child of } u, \\ \frac{1}{\lambda+k} & \text{if } u \neq r \text{ and } v \text{ is a child of } u, \\ \frac{\lambda}{\lambda+k} & \text{if } u \text{ is an internal node and } v \text{ is the parent of } u, \\ 1 & \text{if } u \text{ is a leaf and } v \text{ is the parent of } u, \\ 0 & \text{otherwise,} \end{cases} \quad (16)$$

for $u, v \in V_n$. Notice that (16) denotes a *simple random walk* over T_n when $\lambda = 1$. We also remark that λ and k are constants to n . As a consequence of [28], we know the following fact about a random walk on an infinite k -ary tree T_∞ .

► **Proposition 7** ([28, 29]). *If $\lambda \geq k$ (resp. $\lambda < k$) then the root r is recurrent (resp. transient) by P_∞ .*

Then, we are concerned with a RWaGG $\mathcal{D}_T = (\mathfrak{d}, G, P)$ starting from the root r where $G(n) = G_n$ and $P(n) = P_n$. Our goal of the section is to establish the following theorem.

► **Theorem 8.** *Let $k \geq 2$ and $\lambda > 0$ be constants to n . Then, the root r is recurrent by \mathcal{D}_T if*

$$\sum_{n=1}^{\infty} \mathfrak{d}(n) \left(\frac{\lambda}{k}\right)^n = \infty \quad (17)$$

holds, otherwise, transient.

For instance, Theorem 8 implies the following corollary, about a simple random walk on an infinitely growing k -ary tree.

► **Corollary 9.** *Let $\lambda = 1$, i.e., every P_n denotes a simple random walk on the complete k -ary tree T_n . If $\mathfrak{d}(n) = \Omega(k^n/(n \log n))$ then r is recurrent by \mathcal{D}_T . If $\mathfrak{d}(1) < \infty$ and $\mathfrak{d}(n) = O(k^n/(n(\log n)^{1+\epsilon}))$ for $n \geq 2$ with a constant $\epsilon > 0$ then r is transient by \mathcal{D}_T .*

Proof. Suppose $\mathfrak{d}(n) \geq ck^n/(n \log n)$ for some constant $c > 0$. Then, $\sum_{n=1}^{\infty} \mathfrak{d}(n) \left(\frac{1}{k}\right)^n \geq \sum_{n=1}^{\infty} c \frac{k^n}{n \log n} \left(\frac{1}{k}\right)^n = c \sum_{n=1}^{\infty} \frac{1}{n \log n} \geq c \int_2^{\infty} \frac{1}{x \log x} = c[\log \log n]_2^{\infty} = \infty$, and Theorem 8 implies that r is recurrent.

Suppose $\mathfrak{d}(n) \leq c'k^n/(n(\log n)^{1+\epsilon})$ for some constant $c' > 0$. Then, $\sum_{n=1}^{\infty} \mathfrak{d}(n) \left(\frac{1}{k}\right)^n \leq \mathfrak{d}(1) + \sum_{n=2}^{\infty} c' \frac{k^n}{n(\log n)^{1+\epsilon}} \left(\frac{1}{k}\right)^n \leq \mathfrak{d}(1) + c' \frac{1}{2(\log 2)^{1+\epsilon}} + c' \int_2^{\infty} \frac{1}{x(\log x)^{1+\epsilon}} dx = \mathfrak{d}(1) + c' \frac{1}{2(\log 2)^{1+\epsilon}} + c'k \left[-\frac{1}{\epsilon(\log x)^\epsilon} \right]_2^{\infty} < \infty$, and Theorem 8 implies that r is transient. ◀

4.2 Proof of Theorem 8

We prove Theorem 8. As a preliminary step, we remark on the following two facts.

► **Lemma 10.** *(i) Every P_n ($n = 1, 2, \dots$) is reversible: precisely, let*

$$\phi(v) = \begin{cases} \frac{k}{\lambda+k} & \text{if } h(v) = 0 \text{ (i.e., } v = r), \\ \lambda^{-h(v)} & \text{if } 0 < h(v) < n, \\ \frac{\lambda}{\lambda+k} \lambda^{-n} & \text{if } h(v) = n \text{ (i.e., } v \text{ is a leaf)}. \end{cases} \quad (18)$$

⁷ For simplicity of notation, Lyons [28] and Lyons and Peres [29] assume $\lambda > 1$, but many arguments are naturally extended to $\lambda > 0$ by modifications with some bothering notations.

Then, the detailed balance equation

$$\phi(u)P_n(u, v) = \phi(v)P_n(v, u)$$

holds for any $u, v \in V_n$. (ii) Every P_n is irreducible and $\text{period}(P_n) = 2$. Thus the even-time stationary distribution of P_n is

$$\hat{\pi}_n(v) = \frac{\phi(v)}{\sum_{u \in U_n} \phi(u)} \quad (19)$$

for any $v \in U_n$.

Let $p(n) = \hat{\pi}_n(r)$, then

$$p(n) = \begin{cases} \frac{\frac{k}{\lambda+k}}{\frac{k}{\lambda+k} + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \binom{k}{\lambda}^{2i}} & \text{if } n \text{ is odd,} \\ \frac{\frac{k}{\lambda+k}}{\frac{k}{\lambda+k} + \sum_{i=1}^{\frac{n}{2}-1} \binom{k}{\lambda}^{2i} + \frac{\lambda}{\lambda+k} \binom{k}{\lambda}^n} & \text{if } n \text{ is even} \end{cases} \quad (20)$$

by (18) and (19) considering the fact $|\{v \in V_n \mid h(v) = i\}| = k^i$ for $i = 0, 1, \dots, n$.

► **Lemma 11.** *If $\lambda < k$, then*

$$\frac{k-\lambda}{k} \left(\frac{\lambda}{k}\right)^{n+1} \leq p(n) \leq \left(\frac{\lambda}{k}\right)^{n-1}. \quad (21)$$

Proof. Firstly, we prove the upper bound of (21). When n is odd,

$$p(n) = \frac{\frac{k}{\lambda+k}}{\frac{k}{\lambda+k} + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \binom{k}{\lambda}^{2i}} \leq \frac{1}{\frac{k}{\lambda+k} + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \binom{k}{\lambda}^{2i}} \leq \frac{1}{\binom{k}{\lambda}^{2\lfloor \frac{n}{2} \rfloor}} = \left(\frac{\lambda}{k}\right)^{2\lfloor \frac{n}{2} \rfloor} = \left(\frac{\lambda}{k}\right)^{n-1}$$

and we obtain the upper bound in the case. When n is even, similarly,

$$\begin{aligned} p(n) &= \frac{\frac{k}{\lambda+k}}{\frac{k}{\lambda+k} + \sum_{i=1}^{\frac{n}{2}-1} \binom{k}{\lambda}^{2i} + \frac{\lambda}{\lambda+k} \binom{k}{\lambda}^n} = \frac{1}{1 + \frac{\lambda+k}{k} \sum_{i=1}^{\frac{n}{2}-1} \binom{k}{\lambda}^{2i} + \binom{k}{\lambda}^n} \\ &\leq \frac{1}{\binom{k}{\lambda}^n} \leq \left(\frac{\lambda}{k}\right)^{n-1} \end{aligned}$$

and we obtain the upper bound. Then, we prove the lower bound of (21). When n is odd,

$$p(n) = \frac{\frac{k}{\lambda+k}}{\frac{k}{\lambda+k} + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \binom{k}{\lambda}^{2i}} \geq \frac{\frac{k}{\lambda+k}}{1 + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \binom{k}{\lambda}^{2i}} = \frac{\frac{k}{\lambda+k}}{\frac{((\frac{k}{\lambda})^2)^{\lfloor \frac{n}{2} \rfloor + 1} - 1}{(\frac{k}{\lambda})^2 - 1}} = \frac{\frac{k}{\lambda+k}}{\frac{(\frac{k}{\lambda})^{n+2} - 1}{(\frac{k}{\lambda})^2 - 1}}$$

and we obtain the lower bound in the case. When n is even,

$$p(n) = \frac{\frac{k}{\lambda+k}}{\frac{k}{\lambda+k} + \sum_{i=1}^{\frac{n}{2}-1} \binom{k}{\lambda}^{2i} + \frac{\lambda}{\lambda+k} \binom{k}{\lambda}^n} \geq \frac{\frac{k}{\lambda+k}}{1 + \sum_{i=1}^{\frac{n}{2}-1} \binom{k}{\lambda}^{2i} + \binom{k}{\lambda}^n} = \frac{\frac{k}{\lambda+k}}{\frac{(\frac{k}{\lambda})^{n+2} - 1}{(\frac{k}{\lambda})^2 - 1}}$$

holds. In both cases,

$$\begin{aligned} p(n) &\geq \frac{\frac{k}{\lambda+k}}{\frac{(\frac{k}{\lambda})^{n+2} - 1}{(\frac{k}{\lambda})^2 - 1}} = \frac{k}{\lambda+k} \left(\left(\frac{k}{\lambda}\right)^2 - 1 \right) \frac{1}{\left(\frac{k}{\lambda}\right)^{n+2} - 1} \\ &\geq \frac{k}{\lambda+k} \left(\left(\frac{k}{\lambda}\right)^2 - 1 \right) \frac{1}{\left(\frac{k}{\lambda}\right)^{n+2}} = \frac{k-\lambda}{k} \left(\frac{\lambda}{k}\right)^{n+1} \end{aligned}$$

holds and we obtain the lower bound. ◀

17:12 Recurrence/Transience of Random Walks on Growing Trees and Hypercubes

The following lemma is a key of the proof of Theorem, 8.

► **Lemma 12.** *If $\lambda < k$ then \mathcal{D}_T is LHaGG.*

Proof. Let f and g satisfy $\sum_{i=1}^n f(i) \leq \sum_{i=1}^n g(i)$ for any $n = 1, 2, \dots$, and let X_t and Y_t ($t = 0, 1, 2, \dots$) respectively follow (f, G, P) and (g, G, P) , i.e., the tree of (f, G, P) grows faster than (g, G, P) . Let $X_0 = Y_0 = r$, and we prove $\Pr[X_t = r] \leq \Pr[Y_t = r]$ for any $t = 1, 2, \dots$ (recall Section 3.1 for LHaGG).

We construct a coupling of $\mathbf{X} = \{X_t\}_{t \geq 0}$ and $\mathbf{Y} = \{Y_t\}_{t \geq 0}$ such that $h(X_t) \geq h(Y_t)$ holds for any $t = 1, 2, \dots$. The proof is an induction concerning t . Clearly, $h(X_0) = h(Y_0) = 0$. Inductively assuming $h(X_t) \geq h(Y_t)$, we prove $h(X_{t+1}) \geq h(Y_{t+1})$. If $h(X_t) > h(Y_t)$ then $h(X_t) \geq h(Y_t) - 2$ since every P_n is period(P_n) = 2 for $n = 1, 2, \dots$. It is easy to see that $h(X_{t+1}) \geq h(X_t) - 1 \geq h(Y_t) + 1 \geq h(Y_{t+1})$, and we obtain $h(X_{t+1}) \geq h(Y_{t+1})$ in the case.

Suppose $h(X_t) = h(Y_t)$. We consider four cases: (i) $X_t = Y_t = r$, (ii) both X_t and Y_t are internal nodes, (iii) both X_t and Y_t are leaves, i.e., both trees of (f, G, P) and (g, G, P) take the same height at time t , (iv) X_t is not a leaf but Y_t is a leaf, i.e., the tree of (f, G, P) is higher than that of (g, G, P) at time t . In the case (i),

$$\Pr[h(X_{t+1}) = h(X_t) + 1] = \Pr[h(Y_{t+1}) = h(Y_t) + 1] = 1$$

hold, and hence we can couple them to satisfy $h(X_{t+1}) = h(Y_{t+1})$. In the case (ii), since both X_t and Y_t are internal nodes,

$$\Pr[h(X_{t+1}) = h(X_t) - 1] = \Pr[h(Y_{t+1}) = h(Y_t) - 1] = \frac{\lambda}{k + \lambda}$$

and

$$\Pr[h(X_{t+1}) = h(X_t) + 1] = \Pr[h(Y_{t+1}) = h(Y_t) + 1] = \frac{k}{k + \lambda}$$

hold, and hence we can couple them to satisfy $h(X_{t+1}) = h(Y_{t+1})$. In the case (iii), since both X_t and Y_t are leaves,

$$\Pr[h(X_{t+1}) = h(X_t) - 1] = \Pr[h(Y_{t+1}) = h(Y_t) - 1] = 1$$

holds, and hence we can couple them to satisfy $h(X_{t+1}) = h(Y_{t+1})$. In the case (iv), since X_t is not a leaf but Y_t is a leaf,

$$\Pr[h(X_{t+1}) = h(X_t) - 1] = \frac{\lambda}{k + \lambda} \leq \Pr[h(Y_{t+1}) = h(Y_t) - 1] = 1$$

holds, and hence we can couple them to satisfy $h(X_{t+1}) \geq h(Y_{t+1})$.

Now we obtain a coupling of $\mathbf{X} = \{X_t\}_{t \geq 0}$ and $\mathbf{Y} = \{Y_t\}_{t \geq 0}$ such that $h(X_t) \geq h(Y_t)$ hold for any $t = 1, 2, \dots$, which implies that $h(Y_t) = 0$ as long as $h(X_t) = 0$. This means that $\Pr[X_t = r] \leq \Pr[Y_t = r]$ for any $t = 1, 2, \dots$. We obtain the claim. ◀

By Lemma 5 with Lemma 12, we get a sufficient condition for recurrence in Theorem 8. On the other hand, we cannot directly apply Lemma 6 to the sufficient condition for transient in Theorem 8, because the “mixing time” of P_n is proportional to k^n , see e.g., [27]. Then, we estimate $R(t)$ by another random walk.

Let $Z_t = h(X_t)$, where X_t is a random walk on a growing k -ary tree $\mathcal{D}_T = (\mathfrak{d}, G, P)$. Then Z_t is a RWoGG $\mathcal{D}_L = (\mathfrak{d}, L, Q)$ where $L(n) = (\{0, 1, \dots, n\}, \{\{i, i+1\} \mid i = 0, 1, 2, \dots, n-1\})$ is a path graph of length n , and the transition probability matrix $Q(n) = Q_n$ is given by

$$\begin{cases} Q_n(0, 1) = 1, \\ Q_n(i, i+1) = \frac{k}{\lambda+k} \quad \text{for } i = 1, 2, \dots, n-1, \\ Q_n(i, i-1) = \frac{\lambda}{\lambda+k} \quad \text{for } i = 1, 2, \dots, n-1, \\ Q_n(n, n-1) = 1. \end{cases}$$

The following Lemmas 13 and 14 are easy to observe.

► **Lemma 13.** *Let X_t (resp. Z_t) follow $\mathcal{D}_T = (f, G, P)$ (resp. $\mathcal{D}_L = (f, L, Q)$). Let $R(t) = \Pr[X_t = r]$ (resp. $R'(t) = \Pr[Z_t = r]$), and let $\hat{\pi}_n$ (resp. $\hat{\pi}'_n$) denote the even-time stationary distribution of P_n (resp. Q_n). Then, $R(t) = R'(t)$ for any $t = 1, 2, \dots$, as well as $\hat{\pi}_n(r) = \hat{\pi}'_n(r)$.*

► **Lemma 14.** *If $\lambda < k$ then \mathcal{D}_L is LHaGG.*

The following lemma about the mixing time of Q_n is easily obtained by a standard coupling argument for the mixing time, and we here omit the proof.

► **Lemma 15.** *Let $\hat{\tau}'_n(\epsilon)$ denote the even mixing-time of Q_n then $\hat{\tau}'_n(\epsilon) \leq n^2 \log \epsilon^{-1}$.*

Then, we can prove the condition for \mathcal{D}_L being transient from Lemma 6.

► **Lemma 16.** *If $\lambda < k$ and $\sum_{n=1}^{\infty} \mathfrak{d}(n)(\frac{\lambda}{k})^n < \infty$ then 0 is transient by \mathcal{D}_L .*

Proof. Let $p(n) = \hat{\pi}_n(r)$ and $p'(n) = \hat{\pi}'_n(r)$, then $p(n) = p'(n)$ by Lemma 13. By Lemma 15, $\hat{\tau}'_n = \hat{\tau}'_n(p(n-1)) \leq n^2 \log(p(n-1)) \leq n^2 \log((\frac{\lambda}{k})^n) \leq c'n^3$, and hence $\sum_{n=1}^{\infty} \hat{\tau}'_n p(n-1) \leq \sum_{n=1}^{\infty} n^3 c'(\frac{\lambda}{k})^{n-1} < \infty$. If $\sum_{n=1}^{\infty} \mathfrak{d}(n)(\frac{\lambda}{k})^n < \infty$, then $\sum_{n=1}^{\infty} \max\{\mathfrak{d}(n), \hat{\tau}'_n\} p(n-1) \leq \sum_{n=1}^{\infty} (\mathfrak{d}(n) + \hat{\tau}'_n)(\frac{\lambda}{k})^{n-1} < \infty$, which implies $\sum_{t=1}^{\infty} R'(t) < \infty$ by Lemma 6 with Lemma 14. ◀

Now, we are ready to prove Theorem 8.

Proof of Theorem 8. First, we consider the (interesting) case $\lambda < k$.

(Recurrent) Assuming $\sum_{n=1}^{\infty} \mathfrak{d}(n)(\frac{\lambda}{k})^n = \infty$, we prove $\sum_{n=1}^{\infty} (\mathfrak{d}(n) - 1)p(n) = \infty$. Notice that $\sum_{n=1}^{\infty} p(n) \leq c \sum_{n=1}^{\infty} (\frac{\lambda}{k})^n = \frac{c}{1-\frac{\lambda}{k}} < \infty$. Let $C = \sum_{n=1}^{\infty} p(n)$, then $\sum_{n=1}^{\infty} (\mathfrak{d}(n) - 1)p(n) = \sum_{n=1}^{\infty} \mathfrak{d}(n)p(n) - C \geq \sum_{n=1}^{\infty} \mathfrak{d}(n)c(\frac{\lambda}{k})^n - C$, which is ∞ from the assumption. Thus, r is recurrent by Lemma 5.

(Transient) By Lemma 13, $\sum_{t=1}^{\infty} R(t) = \sum_{t=1}^{\infty} R'(t)$. If $\sum_{n=1}^{\infty} \mathfrak{d}(n)(\frac{\lambda}{k})^n < \infty$ then $\sum_{t=1}^{\infty} R'(t) < \infty$ by Lemma 16, meaning that r is transient.

In the case of $\lambda \geq k$, it is always recurrent. The proof follows that of Lemma 5, but here we omit the proof. ◀

5 Random Walk on $\{0, 1\}^n$ with Increasing n

5.1 Main result

This section shows an interesting example. Let $C_n = (V_n, E_n)$ where

$$V_n = \{0, 1\}^n \tag{22}$$

$$E_n = \left\{ \{\mathbf{u}, \mathbf{v}\} \in \binom{V_n}{2} \mid \|\mathbf{u} - \mathbf{v}\|_1 = 1 \right\} \tag{23}$$

17:14 Recurrence/Transience of Random Walks on Growing Trees and Hypercubes

for $n = 1, 2, \dots$. Let $\mathbf{0} \in V_n$ denote the (common) origin vertex $(0, \dots, 0)$ for each n . Let

$$P_n(\mathbf{u}, \mathbf{v}) = \begin{cases} \frac{1}{n} & \text{if } \|\mathbf{u} - \mathbf{v}\|_1 = 1, \\ 0 & \text{otherwise,} \end{cases} \quad (24)$$

for $\mathbf{u}, \mathbf{v} \in V_n$. Then, we are concerned with $\mathcal{D}_C = (\mathfrak{d}, G, P)$ starting from $\mathbf{0}$ where $G(n) = C_n$ and $P(n) = P_n$.

► **Theorem 17.** *If \mathcal{D}_C satisfies*

$$\sum_{n=1}^{\infty} \frac{\mathfrak{d}(n)}{2^n} = \infty \quad (25)$$

then $\mathbf{0}$ is recurrent, otherwise $\mathbf{0}$ is transient.

The following lemma is not very difficult, but nontrivial.

► **Lemma 18.** *\mathcal{D}_C is LHaGG.*

Proof. Let f and g satisfy $\sum_{i=1}^n f(i) \leq \sum_{i=1}^n g(i)$ for any $n = 1, 2, \dots$, and let X_t and Y_t ($t = 0, 1, 2, \dots$) respectively follow (f, G, P) and (g, G, P) , i.e., the box of (f, G, P) grows faster than (g, G, P) . Let n_t (resp. n'_t) denote the dimension of (f, G, P) (resp. (g, G, P)) at time t , and then notice that $n_t \geq n'_t$ hold for any $t = 0, 1, \dots$ by the assumption that (f, G, P) grows faster. Let $X_0 = Y_0 = \mathbf{0}$, and we prove $\Pr[X_t = \mathbf{0}] \leq \Pr[Y_t = \mathbf{0}]$ for any $t = 1, 2, \dots$

Let $h(\mathbf{u}) = |\{i \in \{1, \dots, n\} \mid u_i = 1\}|$ for $\mathbf{u} = (u_1, \dots, u_n) \in V_n$. We construct a coupling of $\mathbf{X} = \{X_t\}_{t \geq 0}$ and $\mathbf{Y} = \{Y_t\}_{t \geq 0}$ such that $h(X_t) \geq h(Y_t)$ holds for any $t = 1, 2, \dots$. The proof is an induction concerning t . Clearly, $h(X_0) = h(Y_0) = 0$. Inductively assuming $h(X_t) \geq h(Y_t)$, we prove $h(X_{t+1}) \geq h(Y_{t+1})$. If $h(X_t) > h(Y_t)$ then $h(X_t) \geq h(Y_t) - 2$ since every P_n is period(P_n) = 2 for $n = 1, 2, \dots$. It is easy to see that $h(X_{t+1}) \geq h(X_t) - 1 \geq h(Y_t) + 1 \geq h(Y_{t+1})$, and we obtain $h(X_{t+1}) \geq h(Y_{t+1})$ in the case. Suppose $h(X_t) = h(Y_t)$. Then,

$$\begin{aligned} \Pr[h(X_{t+1}) = h(X_t) - 1] &= \frac{h(X_t)}{n_t} \leq \frac{h(Y_t)}{n'_t} = \Pr[h(Y_{t+1}) = h(Y_t) - 1], & \text{and} \\ \Pr[h(X_{t+1}) = h(X_t) + 1] &= 1 - \frac{h(X_t)}{n_t} \geq 1 - \frac{h(Y_t)}{n'_t} = \Pr[h(Y_{t+1}) = h(Y_t) + 1] \end{aligned}$$

hold, which implies that a coupling exists such that $h(X_{t+1}) \geq h(Y_{t+1})$.

Now we obtain a coupling of $\mathbf{X} = \{X_t\}_{t \geq 0}$ and $\mathbf{Y} = \{Y_t\}_{t \geq 0}$ satisfying $h(X_t) \geq h(Y_t)$ for any $t = 1, 2, \dots$, which implies that $h(Y_t) = 0$ as long as $h(X_t) = 0$. This means that $\Pr[X_t = \mathbf{0}] \leq \Pr[Y_t = \mathbf{0}]$ for any $t = 1, 2, \dots$. We obtain the claim. ◀

The following two lemmas are well known.

► **Lemma 19.** *Let $\hat{\tau}_n(\epsilon)$ denote the mixing time of P_n . Then, $\hat{\tau}_n(\epsilon) = O(n \log(n/\epsilon))$.*

► **Lemma 20.** $p(n) = \frac{1}{2^n} = 2^{-n+1}$.

Proof of Theorem 17. (Recurrence) By Lemma 18, \mathcal{D}_C is LHaGG. Since $p(n) = 2^{-n+1}$ by Lemma 20, Lemma 5 implies that if $\sum_{n=1}^{\infty} \frac{\mathfrak{d}(n)}{2^n} = \infty$ then $\mathbf{0}$ is recurrent.

(Transience) By Lemma 19, $\mathfrak{t}(n) = \hat{\tau}_n(p(n-1)) \leq n \log \frac{n}{p(n-1)} \leq n \log(n2^n) \leq c'n^2 \log n$, and hence $\sum_{n=1}^{\infty} \mathfrak{t}(n)p(n-1) \leq \sum_{n=1}^{\infty} c'n^2 \log n \frac{1}{2^{n-2}} < \infty$. If $\sum_{n=1}^{\infty} \frac{\mathfrak{d}(n)}{2^n} < \infty$, then $\sum_{n=1}^{\infty} \max\{\mathfrak{d}(n), \mathfrak{t}(n)\}p(n-1) \leq \sum_{n=1}^{\infty} (\mathfrak{d}(n) + \mathfrak{t}(n)) \frac{1}{2^n} < \infty$, which implies $\sum_{t=1}^{\infty} R(t) < \infty$ by Lemma 6 with Lemma 18. ◀

5.2 An Interesting fact: every finite point becomes recurrent

We can easily observe the following fact from Theorem 17.

► **Corollary 21.** *If $\mathfrak{d}(n) = \Omega(2^n/n)$ then $\mathbf{0}$ is recurrent. If $\mathfrak{d}(n) = O(2^n/n^{1+\epsilon})$ then $\mathbf{0}$ is transient.*

Notice that the maximum degree of $G(n)$ is unbounded asymptotic to n , clearly. Nevertheless, we can see the following interesting fact.

► **Proposition 22.** *If $\mathfrak{d}(n) = \Omega(n2^n)$ then \mathcal{D}_C starting from $\mathbf{0}$ visits $\mathbf{v} \in V_m$ infinitely many times for any $m < \infty$.*

Proof. Notice that $\hat{\tau}_n(2^{-n-1}) = O(n \log(n2^{n+1})) = O(n^2 \log n)$ by Lemma 19. Thus, in the n -th phase, i.e., $[T_{n-1}, T_n]$, the random walk X visits $\mathbf{v} \in V_n$ with probability at least 2^{-n-1} in every $O(n^2 \log n)$ steps (even if $\mathbf{v} \in \bar{U}_n$, here we omit the proof). Thus the probability that X never visit \mathbf{v} during the n -th phase is at most $(1 - 2^{-n-1})^{n2^{n+1}/n^2 \log n} \leq \exp(-\frac{1}{n \log n})$. This implies that the probability that X never visits $\mathbf{v} \in V_m$ forever is at most $\prod_m \exp(-\frac{1}{n \log n}) = \exp(-\sum_{n=m}^{\infty} \frac{1}{n \log n}) \leq \exp(-\int_m^{\infty} \frac{1}{x \log x} dx) = \exp(-[\log \log x]_m^{\infty}) = \exp(-\infty) = 0$. This means that the RWoGG X visits $\mathbf{v} \in V_m$ at least once in finite steps with probability 1.

Once we know that X visits \mathbf{v} in a finite steps, the claim is trivial thanks to the vertex transitivity of the hypercube skeleton. ◀

We think that the hypothesis of Proposition 22 can be relaxed from $\Omega(n2^n)$ to $\Omega(2^n/n)$, but we are not sure.

6 Concluding Remark

In this paper, we have developed a coupling method to prove the recurrence and transience of a RWoGG, by introducing the notion of LHaGG. Then, we showed the phase transition between the recurrence and transience of random walks on a growing k -ary tree (Theorem 8) and on a growing hypercube (Theorem 17). We also have other examples of LHaGG, such as growing integer grids and growing level trees (see a full paper version). It is a future work to develop an extended technique to prove the phase transitions for more general growing trees and integer grids.

References

- 1 Gideon Amir, Itai Benjamini, Ori Gurel-Gurevich, and Gady Kozma. Random walk in changing environment. *Stochastic Processes and their Applications*, 130(12):7463–7482, 2020. doi:10.1016/j.spa.2020.08.003.
- 2 V.S. Anil Kumar and H. Ramesh. Coupling vs. conductance for the Jerrum–Sinclair chain. *Random Structures & Algorithms*, 18(1):1–17, 2001. doi:10.1002/1098-2418(200101)18:1<1::AID-RSA1>3.0.CO;2-7.
- 3 John Augustine, Gopal Pandurangan, and Peter Robinson. Distributed algorithmic foundations of dynamic networks. *SIGACT News*, 47(1):69–98, March 2016. doi:10.1145/2902945.2902959.
- 4 Chen Avin, Michal Koucký, and Zvi Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, ICALP 2008, pages 121–132, 2008.

- 5 Chen Avin, Michal Koucký, and Zvi Lotker. Cover time and mixing time of random walks on dynamic graphs. *Random Structures & Algorithms*, 52(4):576–596, 2018. doi:10.1002/rsa.20752.
- 6 Itai Benjamini and David Wilson. Excited Random Walk. *Electronic Communications in Probability*, 8:86–92, 2003. doi:10.1214/ECP.v8-1072.
- 7 Leran Cai, Thomas Sauerwald, and Luca Zanetti. Random walks on randomly evolving graphs. In Andrea Werneck Richa and Christian Scheideler, editors, *Structural Information and Communication Complexity*, SIROCCO 2020, pages 111–128, 2020.
- 8 Colin Cooper. Random walks, interacting particles, dynamic networks: Randomness can be helpful. In Adrian Kosowski and Masafumi Yamashita, editors, *Structural Information and Communication Complexity*, SIROCCO 2011, pages 1–14, 2011.
- 9 Colin Cooper and Alan Frieze. Crawling on simple models of web graphs. *Internet Mathematics*, 1(1):57–90, 2003.
- 10 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Second edition, 2001.
- 11 Burgess Davis. Reinforced random walk. *Probability Theory and Related Fields*, 84(2):203–229, 1990. doi:10.1007/BF01197845.
- 12 Amir Dembo, Ruojun Huang, Ben Morris, and Yuval Peres. Transience in growing subgraphs via evolving sets. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 53(3):1164–1180, 2017. doi:10.1214/16-AIHP751.
- 13 Amir Dembo, Ruojun Huang, and Vladas Sidoravicius. Monotone interaction of walk and graph: recurrence versus transience. *Electronic Communications in Probability*, 19:1–12, 2014. doi:10.1214/ECP.v19-3607.
- 14 Amir Dembo, Ruojun Huang, and Vladas Sidoravicius. Walking within growing domains: recurrence versus transience. *Electronic Journal of Probability*, 19:1–20, 2014. doi:10.1214/EJP.v19-3272.
- 15 Amir Dembo, Ruojun Huang, and Tianyi Zheng. Random walks among time increasing conductances: heat kernel estimates. *Probability Theory and Related Fields*, 175(1):397–445, October 2019. doi:10.1007/s00440-018-0894-1.
- 16 Oksana Denysyuk and Luís Rodrigues. Random walks on evolving graphs with recurring topologies. In Fabian Kuhn, editor, *Distributed Computing*, DISC 2014, pages 333–345, 2014.
- 17 Dmitry Dolgopyat, Gerhard Keller, and Carlangelo Liverani. Random walk in Markovian environment. *The Annals of Probability*, 36(5):1676–1710, 2008. doi:10.1214/07-AOP369.
- 18 Richard Durrett. *Probability: theory and examples*. Cambridge University Press, Fifth edition, 2019.
- 19 Daniel Figueiredo, Giulio Iacobelli, Roberto Oliveira, Bruce Reed, and Rodrigo Ribeiro. On a random walk that grows its own tree. *Electronic Journal of Probability*, 26:1–40, 2021. doi:10.1214/20-EJP574.
- 20 Venkatesan Guruswami. Rapidly mixing markov chains: A comparison of techniques (a survey), 2016. arXiv:1603.01512.
- 21 Ruojun Huang. On random walk on growing graphs. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 55(2):1149–1162, 2019. doi:10.1214/18-AIHP913.
- 22 Giulio Iacobelli, Daniel R. Figueiredo, and Giovanni Neglia. Transient and slim versus recurrent and fat: Random walks and the trees they grow. *Journal of Applied Probability*, 56(3):769–786, 2019. doi:10.1017/jpr.2019.43.
- 23 Shuji Kijima, Nobutaka Shimizu, and Takeharu Shiraga. How many vertices does a random walk miss in a network with moderately increasing the number of vertices? In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2021, pages 106–122, 2021. doi:10.1137/1.9781611976465.8.
- 24 Elena Kosygina and Martin PW Zerner. Excited random walks: results, methods, open problems. *Bulletin of the Institute of Mathematics Academia Sinica (New Series)*, 8(1), 2013.

- 25 Fabian Kuhn and Rotem Oshman. Dynamic networks: models and algorithms. *SIGACT News*, 42(1):82–96, March 2011. doi:10.1145/1959045.1959064.
- 26 Ioannis Lamprou, Russell Martin, and Paul Spirakis. Cover time in edge-uniform stochastically-evolving graphs. *Algorithms*, 11(10), 2018. doi:10.3390/a11100149.
- 27 David A. Levin and Yuval Peres. *Markov chains and mixing times*. American Mathematical Society, Second edition, 2017.
- 28 Russell Lyons. Random walks and percolation on trees. *The Annals of Probability*, 18(3):931–958, 1990.
- 29 Russell Lyons and Yuval Peres. *Probability on Trees and Networks*. Cambridge University Press, USA, 2017.
- 30 Othon Michail and Paul G. Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2):72, January 2018. doi:10.1145/3156693.
- 31 Laurent Saloff-Coste and Jessica Zúñiga. Merging for time inhomogeneous finite Markov chains, Part I: Singular values and stability. *Electronic Journal of Probability*, 14:1456–1494, 2009. doi:10.1214/EJP.v14-656.
- 32 Laurent Saloff-Coste and Jessica Zúñiga. Merging for time inhomogeneous finite Markov chains, Part II: Nash and log-Sobolev inequalities. *The Annals of Probability*, 39(3):1161–1203, 2011.

Reconfiguration and Locomotion with Joint Movements in the Amoebot Model

Andreas Padalkin ✉ 
Paderborn University, Germany

Manish Kumar ✉ 
Bar-Ilan University, Israel

Christian Scheideler ✉ 
Paderborn University, Germany

Abstract

We are considering the geometric amoebot model where a set of n amoebots is placed on the triangular grid. An amoebot is able to send information to its neighbors, and to move via expansions and contractions. Since amoebots and information can only travel node by node, most problems have a natural lower bound of $\Omega(D)$ where D denotes the diameter of the structure. Inspired by the nervous and muscular system, Feldmann et al. have proposed the *reconfigurable circuit extension* and the *joint movement extension* of the amoebot model with the goal of breaking this lower bound.

In the joint movement extension, the way amoebots move is altered. Amoebots become able to push and pull other amoebots. Feldmann et al. demonstrated the power of joint movements by transforming a line of amoebots into a rhombus within $O(\log n)$ rounds. However, they left the details of the extension open. The goal of this paper is therefore to formalize the joint movement extension. In order to provide a proof of concept for the extension, we consider two fundamental problems of modular robot systems: *reconfiguration* and *locomotion*.

We approach these problems by defining meta-modules of rhombical and hexagonal shapes, respectively. The meta-modules are capable of movement primitives like sliding, rotating, and tunneling. This allows us to simulate reconfiguration algorithms of various modular robot systems. Finally, we construct three amoebot structures capable of locomotion by rolling, crawling, and walking, respectively.

2012 ACM Subject Classification Computing methodologies → Cooperation and coordination; Theory of computation → Computational geometry

Keywords and phrases programmable matter, modular robot system, reconfiguration, locomotion

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.18

Related Version *Full Version*: <https://arxiv.org/abs/2305.06146> [45]

Preliminary Results: https://eurocg2024.math.uoi.gr/data/uploads/paper_36.pdf [46]

Funding This work has been supported by the DFG Project SCHE 1592/10-1 and the Israel Science Foundation under Grants 867/19 and 554/23.

1 Introduction

Programmable matter consists of homogeneous nano-robots that are able to change the properties of the matter in a programmable fashion, e.g., the shape, the color, or the density [57]. We are considering the geometric amoebot model [14, 15, 16] where a set of n nano-robots (called *amoebots*) is placed on the triangular grid. An amoebot is able to send information to its neighbors, and to move by first expanding into an unoccupied adjacent node, and then contracting into that node. Since amoebots and information can only travel node by node, most problems have a natural lower bound of $\Omega(D)$ where D denotes the



© Andreas Padalkin, Manish Kumar, and Christian Scheideler;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 18; pp. 18:1–18:20

Leibniz International Proceedings in Informatics



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

diameter of the structure. Inspired by the *nervous* and *muscular system*, Feldmann et al. [23] proposed the *reconfigurable circuit extension* and the *joint movement extension* with the goal of breaking this lower bound.

In the reconfigurable circuit extension, the amoebot structure is able to interconnect amoebots by *circuits*. Each amoebot can send a primitive signal on circuits it is connected to. The signal is received by all amoebots connected to the same circuit. Among others, Feldmann et al. [23] solved the leader election problem, compass alignment problem, and chirality agreement problem within $O(\log n)$ rounds w.h.p.¹ These problems will be important to coordinate the joint movements. Afterward, Padalkin et al. [47] explored the structural power of the circuits by considering the spanning tree problem and symmetry detection problem. Both problems can be solved within polylogarithmic time w.h.p.

In the joint movement extension, the way amoebots move is altered. In a nutshell, an expanding amoebot is capable of pushing other amoebots away from it, and a contracting amoebot is capable of pulling other amoebots towards it. Feldmann et al. [23] demonstrated the power of joint movements by transforming a line of amoebots into a rhombus within $O(\log n)$ rounds. However, they left the details of the extension open. The goal of this paper is therefore to formalize the joint movement extension. In order to provide a proof of concept for the extension, we consider two fundamental problems of modular robot systems (MRS): reconfiguration and locomotion. We study these problems from a centralized view to explore the limits of the extension.

In the *reconfiguration* problem, an MRS has to reconfigure its structure into a given shape. Examples of reconfiguration algorithms in the original amoebot model can be found in [17, 18, 35, 40]. However, all of these are subject of the aforementioned natural lower bound. To our knowledge, polylogarithmic time solutions were found for two types of MRSs: in the nubot model [62] and crystalline atom model [6]. We will show that in the joint movement extension, the amoebots are able to simulate the reconfiguration algorithm for the crystalline atom model, and with that break the lower bound.

In the *locomotion* problem, an MRS has to move along an even surface as fast as possible. We might also ask the MRS to transport an object along the way. In the original amoebot model, one would use the spanning tree primitive to move along the surface [14]. However, we only obtain a constant velocity with that. Furthermore, the original amoebot model does not allow us to transport any objects. In terrestrial environments, there are three basic types of locomotion: rolling, crawling, and walking [26, 33]. For each of these locomotion types, we will present an amoebot structure.

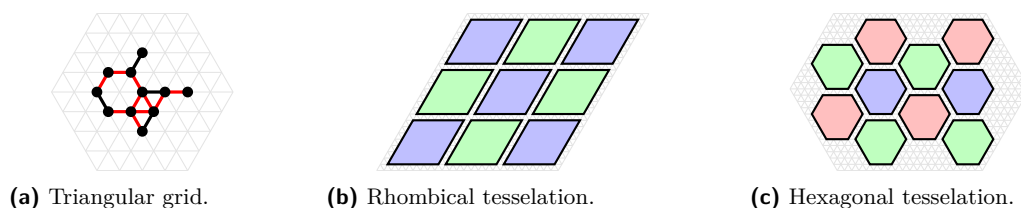
2 Preliminaries

In this section, we introduce the geometric amoebot model [15, 16] and formalize the joint movement extension.

2.1 Geometric Amoebot Model

In this section, we introduce the *geometric amoebot model* [15]. We slightly deviate from the original model to make it more suitable to our extension. A set of n amoebots is placed on the infinite regular triangular grid graph $G_{\Delta} = (V, E)$ (see Figure 1a). An amoebot is an

¹ An event holds with high probability (w.h.p.) if it holds with probability at least $1 - 1/n^c$, where the constant c can be made arbitrarily large.



■ **Figure 1** Domains. The figures show the domains we are working with. The tessellations will be explained in Section 3. Note that there are spaces between the meta-modules since a node cannot be occupied by more than one amoebot. However, the spaces do not contain any nodes.

anonymous, randomized finite state machine in the form of a line segment. The endpoints may either occupy the same node or two adjacent nodes. If the endpoints occupy the same node, the amoebot has length 0 and is called *contracted* and otherwise, it has length 1 and is called *expanded*. Every node of G_Δ is occupied by at most one amoebot. Two endpoints of different amoebots that occupy adjacent nodes in G_Δ are connected by *bonds* (red edges). Let the *amoebot structure* $S \subseteq V$ be the set of nodes occupied by the amoebots. We say that S is *connected* iff G_S is connected, where $G_S = G_\Delta|_S$ is the graph induced by S . Initially, S is connected. An amoebot can move through *contractions* and *expansions*. We refer to [15] for more details.

2.2 Joint Movement Extension

In the *joint movement extension* [23], the way the amoebots move is altered. The idea behind the extension is to allow amoebots to push and pull other amoebots. In the following, we formalize the joint movement extension.

We assume the fully synchronous activation model, i.e., the time is divided into synchronous rounds, and every amoebot is active in each round. Furthermore, we make the idealistic assumption that all movements start at the same time and are performed at the same speed. W.l.o.g., we assume that all movements happen within the time period $[0, 1]$.

Joint movements are performed in two steps. In the first step, the amoebots remove bonds from G_S as follows. Each amoebot can decide to release an arbitrary subset of its currently incident bonds in G_S . A bond is removed iff either of the amoebots at the endpoints releases the bond. However, an expanded amoebot cannot release the bond connecting its occupied nodes. Let $E_L \subseteq E_S$ be the set of all line segments (amoebots) of length 1, $E_R \subseteq E_S$ be the set of the remaining bonds, and $G_R = (S, E_L \cup E_R)$ be the resulting graph. We require that G_R is connected since otherwise, disconnected parts might float apart. We say that a *connectivity conflict* occurs iff G_R is not connected. Whenever a connectivity conflict occurs, the amoebot structure transitions into an undefined state such that we become unable to make any statements about the structure.

In the second step, each amoebot may perform one of the following movements. A contracted amoebot may expand on one of the axes as follows (see blue amoebot in Figure 2a). At $t = 0$, the amoebot can reorientate itself and reassign each of its incident bonds to one of its endpoints. Bonds assigned to an endpoint will stay with that endpoint as the amoebot expands. At $t \in [0, 1]$, the amoebot has a length of t . In the process, the incident bonds do not change their orientations or lengths. As a result, the expanding amoebot pushes all connected amoebots. An expanded amoebot may contract analogously by reversing the contraction (see green amoebot in Figure 2b). Thereby, it pulls all connected amoebots.

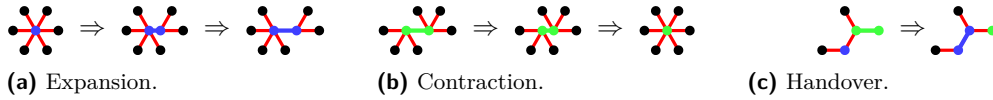
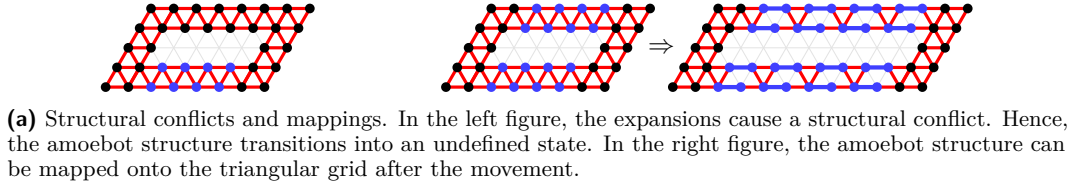
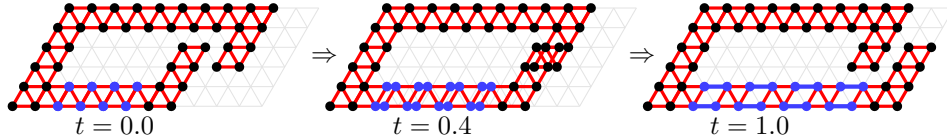


Figure 2 Movements in the extension. Red lines indicate bonds. Blue amoebots are expanding. Green amoebots are contracting. The first two figures show a movement in 0.5 time steps.



(a) Structural conflicts and mappings. In the left figure, the expansions cause a structural conflict. Hence, the amoebot structure transitions into an undefined state. In the right figure, the amoebot structure can be mapped onto the triangular grid after the movement.



(b) Collision. Initially, we have a valid amoebot structure given $(t = 0)$. At $t = 1$, the amoebot structure could be mapped on G_Δ . However, for $t \in [0.25, 0.75]$, parts of the structure collide. Hence, the amoebot structure transitions into an undefined state.

Figure 3 Joint movements. Red lines indicate bonds. Blue amoebots are expanding horizontally.

Furthermore, a contracted amoebot x occupying node u and an expanded amoebot y occupying nodes v and w may perform a handover if there is a bond b between u and v , as follows (see Figure 2c where x is marked in blue and y is marked in green). At an arbitrary $t \in [0, 1]$, we switch the association of the endpoint v from y to x such that x becomes an expanded amoebot with endpoints occupying nodes u and v , y becomes a contracted amoebot with both endpoints occupying w , and x and y are connected by bond $\{v, w\}$. We have to include the handover to ensure universality of the model since otherwise, it would not be possible to move through a narrow tunnel. In theory, all movement primitives presented in Section 3 can be realized without handovers. However, for reasons of clarity and comprehensibility, we will still make use of the handovers.

The amoebots may not be able to perform their movements. We distinguish between two cases. First, the amoebots may not be able to perform their movements while maintaining their relative positions (see Figure 3a). We call that a *structural conflict*. Second, parts of the structure may collide into each other. More precisely, a *collision* occurs if there is a $t \in [0, 1]$ such that two non-adjacent bonds intersect at some point (see Figure 3b). Whenever either a structural conflict or a collision occurs, the amoebot structure transitions into an undefined state such that we become unable to make any statements about the structure.

Otherwise, at $t = 1$, we map the amoebots on the triangular grid G_Δ in compliance with the orientations of all bonds and line segments (see Figure 3a). The mapping is unique except for translations since G_R is connected. We choose any of these mappings as our next amoebot structure. Let S' be the set of nodes occupied the amoebots, E'_L be the set of all line segments (amoebots) of length 1 after all movements were completed, and $G_M = (S', E'_L \cup E_R)$ be the resulting graph. Afterwards, the amoebots reestablish all possible bonds, i.e., we obtain the graph $G_{S'} = G_\Delta|_{S'}$ induced by S' . Unless stated otherwise, each arrow in the figures indicates a single round. The left side shows the structure after the removal of bonds (first step), and the right side the structure after the execution of movements (second step). We chain multiple rounds if we do not change bonds.

In this paper, we assume that we have a centralized scheduler. The scheduler knows the current state of the amoebot structure at all times. At the beginning of each synchronous round, it decides for each amoebot (i) which bonds to release and (ii) which movements to perform. We leave the design of distributed solutions for future work.

2.3 Problem Statement and Our Contribution

In this paper, we formalize the joint movement extension proposed by Feldmann et al. [23]. In the following, we provide a proof of concept. For that, we focus on two fundamental problems of MRSs: reconfiguration and locomotion. We study these problems from a centralized view to explore the limits of the extension.

In the *reconfiguration* problem, an MRS has to reconfigure its structure into a given configuration. For that, we define meta-modules of rhombical and hexagonal shape. We show that these meta-modules are able to perform various movement primitives of other MRSs, e.g., crystalline atoms, and rectangular/hexagonal metamorphic robots. This allows us to simulate the reconfiguration algorithms of those models.

In the *locomotion* problem, an MRS has to move along an even surface as fast as possible. We might also ask the MRS to transport an object along the way. We present three amoebot structures that are able to move by rolling, crawling, and walking, respectively. We analyze their velocities and compare them to other structures of similar models.

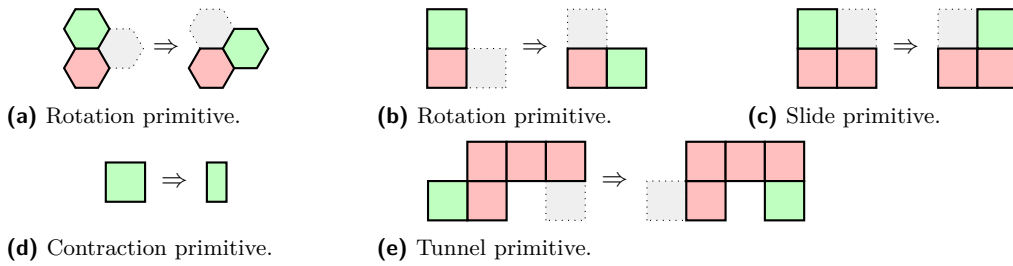
2.4 Related Work

MRSs can be classified into various types, e.g., lattice-type, chain-type, and mobile-type [1, 9, 66, 67]. We refer to the cited papers for examples. We will focus on lattice-type MRSs. These in turn can be characterized by three properties: (i) the lattice, (ii) the connectivity constraint, and (iii) the allowed movement primitives [2].

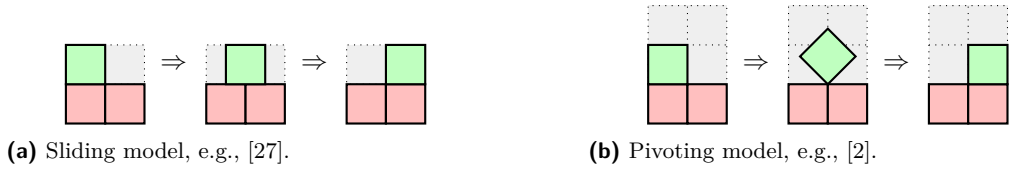
Various MRSs have been defined for different lattices, e.g., [12, 62] utilize the triangular grid, and [3, 22, 51] utilize the Cartesian grid. We will build meta-modules for the Cartesian and the triangular grid. Note that some MRSs were also physically realized. Examples for MRSs using the triangular grid are hexagonal metamorphic robots [12], HexBots [52], fractal machines [42], and catoms [34]. Examples for MRSs using the Cartesian grid are CHOBIE II [56], EM-Cubes [7], M-Blocks [49], pneumatic cellular robots [28], and XBots [61].

We can identify four types of connectivity constraints: (i) the structure is connected at all times, (ii) the structure is connected except for moving robots, (iii) the structure is connected before and after movements, and (iv) there are no connectivity constraints. Our joint movement extension falls into the first category. Other examples are crystalline atoms [50], telecubes [55, 59], and prismatic cubes [60]. Examples for the second category are the sliding cube model [10, 24], rectangular [22] and hexagonal metamorphic robots [12], and for the third category the nubot model [62] and the line pushing model [3]. An example for the last category is the variant of the amoebot model considered by Dufoulon et al. [21].

The most common movement primitives for MRSs on the Cartesian grid are the rotation and slide primitives (see Figures 4b and 4c), and for MRSs on the triangular grid the rotation primitive (see Figure 4a). Some models may constrain these movement primitives (see Figure 5). We refer to [2, 27] for a deeper discussion about possible constraints. One way to bypass such constraints is to construct meta-modules, e.g., see [19, 27, 43]. Our meta-modules implement all aforementioned movement primitives without any constraints, i.e., they perform them in place.



■ **Figure 4** Examples of movement primitives. The green modules are moving, respectively. Models that utilize the Cartesian grid usually assume square modules. In Section 3.1, we utilize rhombical modules instead.



■ **Figure 5** Constrained movement primitives. The green modules are moving. The gray cells must be empty. Both primitives have the same result while the pivoting model requires more free space than the sliding model.

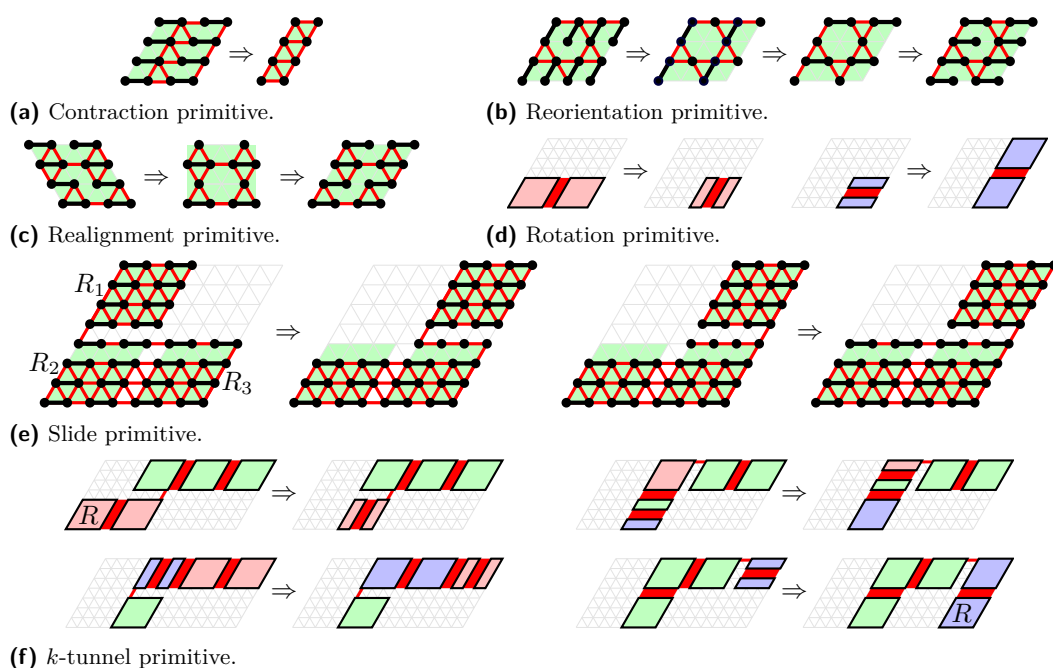
From all aforementioned models, crystalline atoms [50], telecubes [55, 59], and prismatic cubes [60] are the closest to our joint movement extension. In these MRSs, each robot has the shape of a unit square and is able to expand an arm from each side by half a unit. Adjacent robots are able to attach and detach their arms. Similar to our joint movement extension, a robot can move attached robots by expanding or contracting its arms. In contrast to amoebots, a line of robots cannot reconfigure to any other shape since each pair of robots can only have a single point of contact [5]. In order to allow arbitrary reconfigurations, the robots are combined into meta-modules of square shape that are capable of various movement primitives, e.g., the rotation, slide, contraction, and tunnel primitives (see Figures 4b–4e). Our rhombical meta-modules can implement all these movement primitives. The rhombical shape has two advantages compared to the square shape. First, we can implement further movement primitives that provide us with a simple way to construct a walking structure. Second, we can utilize rhombical meta-modules as a basis for hexagonal meta-modules.

3 Meta-Modules

In this section, we will combine multiple amoebots to meta-modules. In other models for programmable matter and modular robots, meta-modules have proven to be very useful. For example, they allow us to bypass restrictions on the reconfigurability [19, 58] and to simulate (reconfiguration) algorithms for other models [4, 48]. In the subsections, we will present meta-modules of rhombical and hexagonal shape, respectively.

3.1 Rhombical Meta-Modules

Let ℓ be a positive even integer. Our *rhombical meta-module* consists of $\ell^2/2$ uniformly oriented expanded amoebots that we arrange into a rhombus of side length $\ell - 1$ (see Figure 6). We obtain a parallelogram of side lengths $\ell - 1$ and $\ell/2 - 1$ if we *contract* all amoebots (see Figure 6a). Note that we have to remove some bonds to perform the contraction. We can *expand* the parallelogram again by reversing the contractions.



■ **Figure 6** Movement primitives for rhombical meta-modules. Red meta-modules perform a pull operation, and blue meta-modules a push operation.

► **Lemma 1.** *Our implementation of the contraction and expansion primitive requires a single round, respectively.*

There are exactly two possibilities to arrange the uniformly oriented expanded amoebots in a rhombus. By reorienting the amoebots in pairs with the help of handovers, we can *reorientate* all amoebots within a rhombus (see Figure 6b).

► **Lemma 2.** *Our implementation of the reorientation primitive requires 3 rounds.*

Furthermore, there are three possibilities to align the sides of a rhombus to the axes of the triangular grid. By sliding each second row along its axis to the other side, we can *realign* the other axis a rhombus is aligned to (see Figure 6c). Note that in combination with the reorientation of the amoebots within a rhombus, we are able to align a rhombus with any two axes of the triangular grid.

► **Lemma 3.** *Our implementation of the realignment primitive requires 2 rounds.*

We can arrange the meta-modules on a rhombical tessellation of the plane if they are all aligned to the same axes (see Figure 1b). Note that due to the triangular grid, the meta-modules are not connected diagonally everywhere. Hence, we will only consider meta-modules connected if their sides are connected. In the following, we introduce two movement primitives: the slide and k -tunnel primitive. Our implementations of these primitives are similar to the ones for crystalline robots (e.g., [5]) and teletubes (e.g., [59]).

In the *slide primitive*, we move a meta-module R_1 along two adjacent substrate meta-modules R_2 and R_3 (see Figure 4c). We realize the primitive as follows (see Figure 6e). We assume that all amoebots are orientated into the movement direction. Otherwise, we apply the reorientation primitive. With respect to Figure 6e, let L_1 denote the uppermost layer of R_2 and R_3 , and L_2 the second uppermost layer of R_2 and R_3 . Our slide primitive consists

of two rounds. In the first round, we contract all amoebots in L_1 after removing all bonds between L_1 and R_1 except the last one in the movement direction, and all bonds between L_1 and L_2 except the last one in the opposite direction. This moves R_1 into its target position. In the second round, we restore R_2 and R_3 . For that, we expand L_1 again after removing all bonds between L_1 and R_1 , and between L_1 and L_2 except the last ones in the movement direction, respectively. This ensures that R_1 stays in place.

► **Lemma 4.** *Our implementation of the slide primitive requires 2 rounds.*

In the k -tunnel primitive, we move a meta-module R through a simple path of meta-modules with k corners to the other end (see Figure 4e). However, we do not move R directly through the path. Instead, we make use of the following two basic operations. First, by contracting two adjacent meta-modules into parallelograms, we *pull* one meta-module into the other (see the first round in Figure 6f). Second, by expanding two adjacent contracted meta-modules, we *push* one meta-module out again (see the fourth round in Figure 6f). Note that two adjacent contracted meta-modules form a rhombus consisting of ℓ^2 contracted amoebots. Since the amoebots do not have any orientation, we can perform the push operation into any direction in parallel to the axes the meta-modules are aligned to.

Now, consider a line of at least 4 meta-modules with two contracted meta-modules at one end. By expanding those two meta-modules and contracting the two meta-modules at the other end, we transfer a meta-module from one end to the other end without changing the length of the line (see the third round in Figure 6f). If we have only a line of 3 meta-modules, it suffices to contract and expand one meta-module, respectively (see the second round in Figure 6f). Note that we have to remove most of the bonds along the line to permit the line to move freely. The pull and push operations allow us to transfer R from one corner to the next corner in a single round.

► **Lemma 5.** *Our implementation of the k -tunnel primitive requires $k + 1$ rounds.*

In particular, note that a 1-tunnel allows us to move a meta-module around another one (see Figure 6d). In other models for modular robot systems, this simple case is known as the *rotation primitive*.

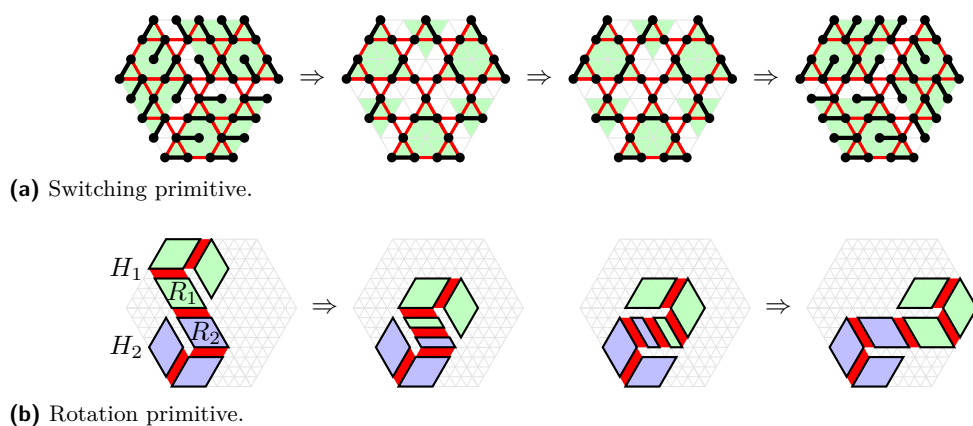
► **Lemma 6.** *Our implementation of the rotation primitive requires 2 rounds.*

3.2 Hexagonal Meta-Modules

Let ℓ be an even integer as before. Our *hexagonal meta-module* consists of three rhombical meta-modules of side length $\ell - 1$ (see Figure 7a). We arrange them into a hexagon of alternating side lengths ℓ and $\ell - 1$.

There are two possibilities to arrange the rhombical meta-modules in the hexagon (see Figure 7a). We can *switch* between them as follows. Each rhombical meta-module within the hexagonal meta-module can be split into an equilateral triangle of side length $\ell - 1$, and an equilateral triangle of side length $\ell - 2$. The idea is to apply a similar technique as in the realignment primitive for rhombical meta-modules. In the first round, each rhombical meta-module contracts each second row without shifting the other rows. In the second round, the rhombical meta-modules interchange the smaller triangles through handovers. In the third round, each resulting rhombical meta-module expands each second row into the opposite direction (of the contracted amoebots of its bigger triangle) – again without shifting the other rows.

► **Lemma 7.** *Our implementation of the switching primitive requires 3 rounds.*



■ **Figure 7** Movement primitives for hexagonal meta-modules.

We can arrange the meta-modules on a hexagonal tessellation of the plane (see Figure 1c). In the following, we introduce the rotation primitive for hexagonal meta-modules.

In the *rotation primitive*, we move a hexagonal meta-module H_1 around another hexagonal meta-module H_2 as follows (see Figure 7b). We arrange H_2 such that a rhombical meta-module R_2 is adjacent to both the old and new position of H_1 , and H_1 such that the rhombical meta-module R_1 adjacent to R_2 is aligned to the same axes as R_2 . We contract R_1 and R_2 , and then expand them into the direction of the new position of H_1 (compare to Section 3.1). This movement primitive requires two rounds. Note that additional steps may be necessary to switch or reorientate the rhombical meta-modules beforehand.

► **Lemma 8.** *Our implementation of the rotation primitive requires 2 rounds.*

4 Reconfiguration

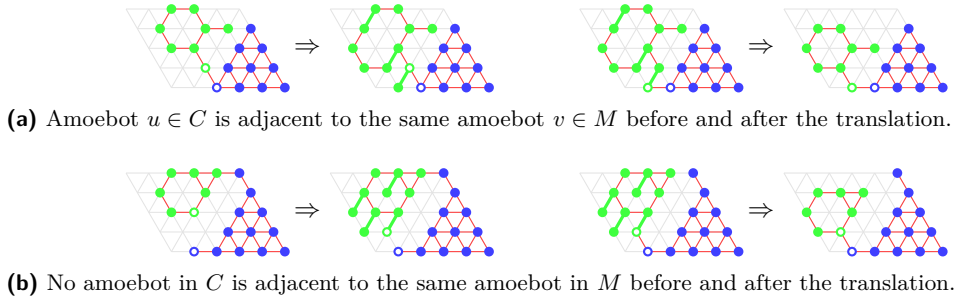
In this section, we discuss possible reconfiguration algorithms. For that, we look at reconfiguration algorithms for other lattice-type MRSs and discuss whether amoebots are capable of simulating these. In particular, we consider the polylogarithmic time solutions for the nubot model [62] and the crystalline atom model [6]. The first subsection deals with the former model while the second subsection deals with reconfiguration algorithms for our meta-modules including the one for the latter model.

4.1 Nubot Model

Similar to the amoebot model, the *nubot model* [62] considers robots on the triangular grid where at most one robot can be positioned on each node. Adjacent robots can be connected by rigid bonds². In the joint movement extension, the rigid bonds correspond to E_R .

Robots are able to appear, disappear, and rotate around adjacent robots. A rotating robot may push and pull other robots into its movement direction. Hence, each rotation results in a translation of a set of connected robots (including the rotating robot) into the movement direction by the distance of 1. The set depends on the bonds and the movement direction, and may include robots not connected by rigid bonds to the rotating robot. In

² Woods et al. [62] distinguish between rigid and flexible bonds. We ignore that since the flexible bonds are not necessary to achieve the polylogarithmic time reconfiguration algorithm.



■ **Figure 8** Simulation of the nubot model. Let d denote the northeastern direction, d' the southwestern direction, and d'' the northwestern direction. The blue amoebots indicate set M . The green amoebots indicate set C . The amoebots marked by a white dot indicate amoebots u and v .

contrast to the joint movement extension, there are no collisions by definition of the set. However, a movement may not be performed due to structural conflicts. In this case, the nubot model does not perform the movement.

► **Lemma 9.** *An amoebot structure of contracted amoebots is able to translate a set of robots in a constant number of joint movements if the translation is possible in the nubot model.*

Proof. Let d denote the movement direction, d' the opposite direction, and d'' any other cardinal direction. Let M denote the set of amoebots that has to be moved. Instead of moving M into direction d , we will move the remaining amoebot structure into direction d' . Note that M divides the remaining amoebot structure into connected components.

▷ **Claim 10.** Each node x in direction d' of an amoebot not in M is either occupied by an amoebot of the same connected component or unoccupied.

Proof. Trivially, x cannot be occupied by an amoebot of another connected component. Further, x cannot be occupied by an amoebot in M since otherwise, the resulting amoebot structure would not be free of collisions. ◁

For each connected component C , we perform the following steps in parallel. If there is an amoebot $u \in C$ that is adjacent to the same amoebot $v \in M$ before and after the translation, we proceed as follows (see Figure 8a). Let A denote the row of amoebots in C through u into direction d'' . Note that A may only contain u . In the first move cycle, we remove all bonds between C and M except for the bond between u and v , and each amoebot in A expands into direction d' . In the second move cycle, we remove all bonds between C and M except for the new bond between u and v , and each amoebot in A contracts. Note that both movements are possible due to Claim 10.

If there is no amoebot in C that is adjacent to the same amoebot in M before and after the translation, we proceed as follows (see Figure 8b). Let B denote all amoebots in C that have an unoccupied node in direction d' . In the first move cycle, each amoebot in B expands into direction d' . There has to be an amoebot $u \in B$ that becomes adjacent to an amoebot $v \in M$. Otherwise, the resulting amoebot structure would not be connected. In the second move cycle, we remove all bonds between C and M except for the bond between u and v , and each amoebot in B contracts. Note that both movements are possible due to Claim 10. ◀

Woods et al. [62] showed that in the nubot model, the robots are able to self-assemble arbitrary shapes/patterns in an amount of time equal to the worst-case running time for a Turing machine to compute a pixel in the shape/pattern plus an additional factor which is

polylogarithmic in its size. While we are able to perform the translations, we do not have the means to let amoebots appear and disappear in the amoebot model. This prevents us from simulating the reconfiguration algorithm by Woods et al. [62].

4.2 Reconfiguration Algorithms for Meta-Modules

Naturally, our meta-modules allow us to simulate reconfiguration algorithms for lattice-type MRSs of similar shape if we can implement the same movement primitives. This leads us to the following results.

► **Theorem 11.** *There is a centralized reconfiguration algorithm for m rhombical meta-modules that requires $O(\log m)$ rounds and performs $\Theta(m \log m)$ moves overall.*

Proof. Aloupis et al. [6] proposed a reconfiguration algorithm for crystalline atoms. It requires $O(\log m)$ rounds and performs $\Theta(m \log m)$ moves overall. The idea is to transform the initial shape to a canonical shape using a divide and conquer approach. The target shape is reached by reversing that procedure. We refer to [6] for the details. The algorithm utilizes the contraction, slide and tunnel primitives which our rhombical meta-modules are capable of (see Section 3.1). Hence, they can simulate this reconfiguration algorithm. ◀

► **Theorem 12.** *There is a centralized reconfiguration algorithm for m hexagonal meta-modules that requires $O(m)$ rounds. Each module has to perform at most $O(m)$ moves.*

Proof. Hurtado et al. [27] proposed a reconfiguration algorithm for hexagonal robots. It requires $O(m)$ rounds and each module has to perform at most $O(m)$ moves. The idea is to compute a spanning tree and to move the robots along the boundary of the tree to a leader module where the robots form a canonical shape, e.g., a line. The target shape is reached by reversing that procedure. We refer to [27] for the details. The algorithm utilizes the rotation primitive which our hexagonal meta-modules are capable of (see Section 3.2). Hence, they can simulate this reconfiguration algorithm. ◀

5 Locomotion

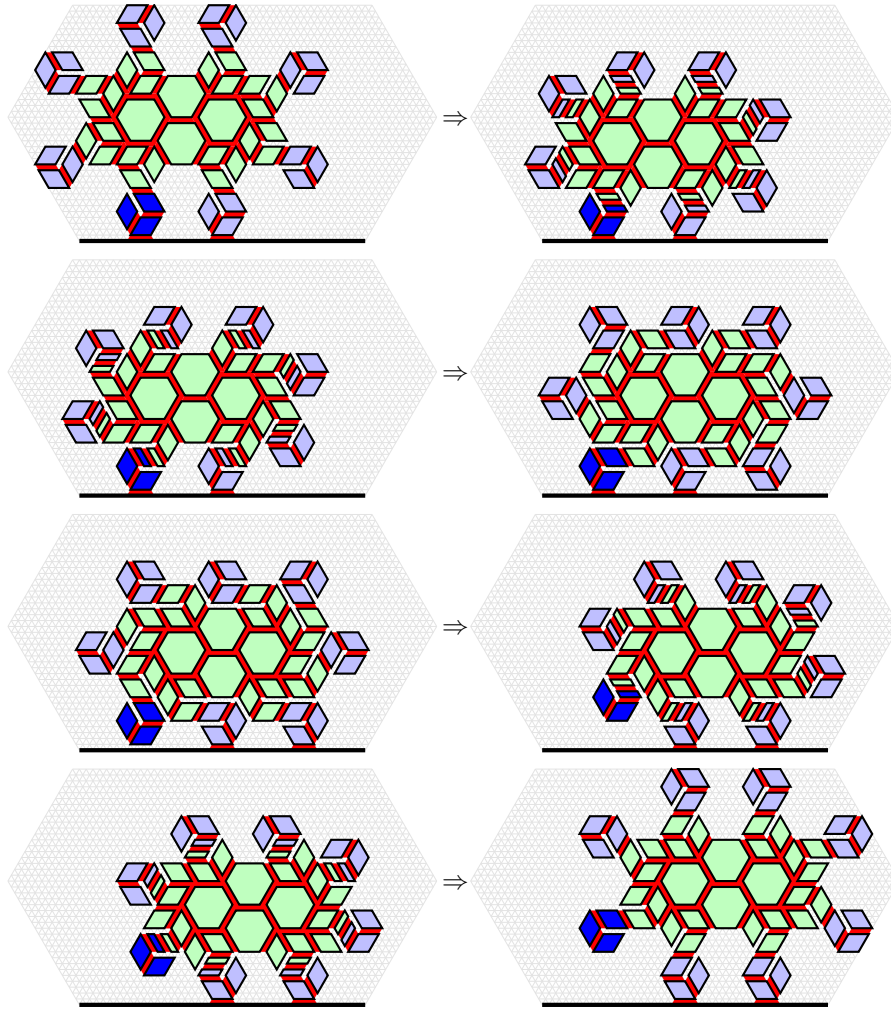
In this section, we consider amoebot structures capable of locomotion along an even surface. There are three basic types of terrestrial locomotion: rolling, crawling, and walking [26, 33]. We can find biological and artificial examples for each of those. In the following subsections, we will present an amoebot structure for each type and analyze their velocity. In the last subsection, we discuss the transportation of objects.

5.1 Rolling

Animals and robots that move by rolling either rotate their whole body or parts of it. Rolling is rather rare in nature. Among others, spiders, caterpillars, and shrimps are known to utilize rolling as a secondary form of locomotion during danger [8]. Bacterial flagella are an example for a creature that rotates a part of its body around an axle [39].

In contrast, rolling is commonly used in robotic systems mainly in the form of wheels. An example of a robot system rolling as a whole are chain-type MRSs that can roll by forming a loop, e.g., Polypod [63], Polybot [64, 65], CKBot [41, 53], M-TRAN [68, 38], and SMORES [29, 30]. Further, examples for rolling robots can be found in [8, 9].

Our rolling amoebot structure imitates a continuous track that rotates around a set of wheels. Continuous tracks are deployed in various fields, e.g., construction, agriculture, and military. We build our *continuous track structure* from hexagonal meta-modules of



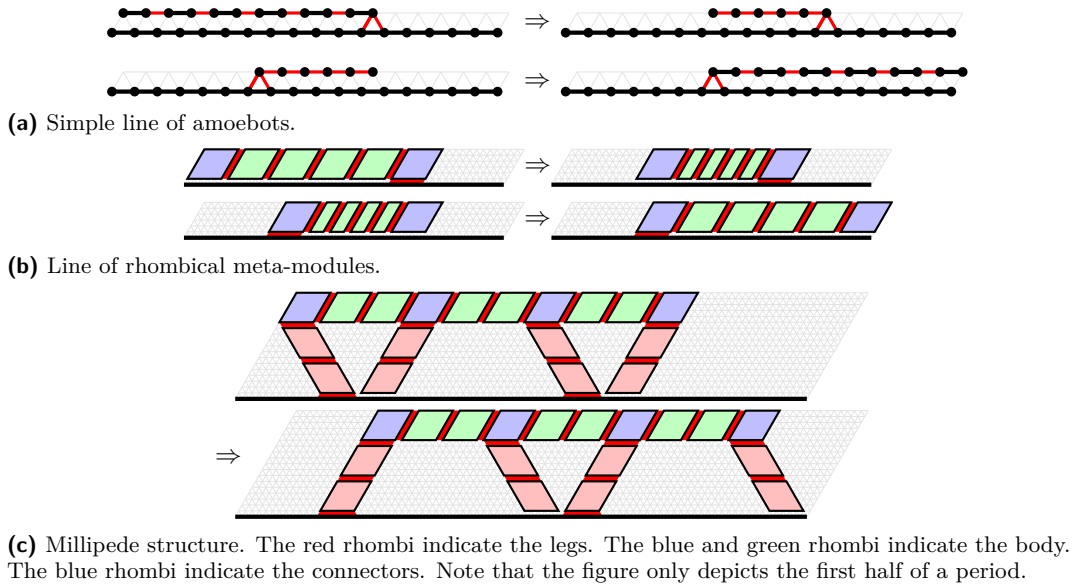
■ **Figure 9** Continuous track structure. The blue meta-modules rotate clockwise around the green meta-modules. We highlight one of the rotating meta-modules in a darker blue.

alternating side lengths ℓ and $\ell - 1$ (see Figure 9). The structure consists of two parts: a connected substrate structure (green meta-modules), and a closed chain of meta-modules rotating along the outer boundary of the substrate (blue meta-modules).

The continuous track structure moves as follows. The rotating meta-modules that are in contact with the surface release all such bonds with the surface if they rotate away from the surface (see the dark blue meta-module in the third round). Otherwise, they keep these bonds such that the substrate structure is pushed forwards. Note that we have to apply the switching primitive between the rotations. We obtain the initial structure after two rotations. In doing so, the structure has moved a distance of $2 \cdot \ell$. By performing the movements periodically, we obtain the following theorem.

► **Theorem 13.** *Our continuous track structure composed of hexagonal meta-modules of alternating side lengths ℓ and $\ell - 1$ moves a distance of $2 \cdot \ell$ within each period of constant length.*

Butler et al. [10] have proposed another rolling structure for the sliding-cube model that we are able to simulate. The structure resembles a swarm of caterpillars where caterpillars climb over each other from the back to the front [20]. However, due to stalling times, this structure is slower than our continuous track structure. We refer to [10] for the details.



■ **Figure 10** Worm and millipede structures.

5.2 Crawling

Crawling locomotion is used by limbless animals. According to [31], crawling can be classified into three types: worm-like locomotion, caterpillar-like locomotion, and snake-like locomotion. We will explain the earthworm-like locomotion below and refer to [31] for the other two types. Due to the advantage of crawling in narrow spaces, various crawling structures have been developed for MRSs, e.g., crystalline atoms [36, 50, 51], catoms [11, 13], polypod [63], polybot [65, 69], M-TRAN [38, 68], and origami robots [32].

Our crawling amoebot structure imitates earthworms. An earthworm is divided into a series of segments. It can individually contract and expand each of its segments. Earthworms move by peristaltic crawling, i.e., they propagate alternating waves of contractions and expansions of their segments from the anterior to the posterior part. The friction between the contracted segments and the surface gives the worm grip. This anchors the worm as other segments expand or contract. The waves of contractions pull the posterior parts to the front, and the waves of expansions push the anterior parts to the front. [31, 44]

The simplest amoebot structure that imitates an earthworm is a simple line of n expanded amoebots along the surface (see Figure 10a). Each amoebot can be seen as a segment of the worm. Instead of propagating waves of contractions and expansions, we contract and expand the whole structure at once. During each contraction (expansion), we release all bonds between the amoebot structure and the surface except for the ones at the head (tail) of the structure that serve as an anchor. As a result, the contraction (expansion) pulls (pushes) the structure to the front. The simple line has moved by a distance of n along the surface after performing a contraction and an expansion. This is the fastest way possible to move along a surface since we accumulate the movements of all amoebots into the same direction. By performing the movements periodically, we obtain the following theorem.

► **Theorem 14.** *A simple line of n expanded amoebots moves a distance of n every 2 rounds.*

However, in practice, the contractions and expansions of the whole structure yield high forces acting on the connections within the amoebot structure. We can address this problem by thickening the worm structure. This increases the expansion of the structure and with

that its stability. Consider a line of r rhombical meta-modules of side length $\ell - 1$ (see Figure 10b). Each module can be seen as a segment of the worm structure. Recall that we can contract a rhombical meta-module into a parallelogram (see Figure 6a). The line of rhombical meta-modules moves in the same manner as the simple line except for the following two points. First, we utilize the whole meta-module at the front and at the end as an anchor to increase the grip, respectively. Second, only the middle $r - 2$ meta-modules participate in the contractions and expansions. The line of rhombical meta-modules moves a distance of $\frac{r-2}{2} \cdot \ell$ along the surface after performing a contraction and an expansion. By performing the movements periodically, we obtain the following theorem.

► **Theorem 15.** *A line of r rhombical meta-modules of side length $\ell - 1$ moves a distance of $\frac{r-2}{2} \cdot \ell$ every 2 rounds.*

Another problem in practice is friction between the structure and the surface and with that the wear of the structure. The worm structure is therefore poorly scalable in its length such that other types for locomotion are more suitable for large amounts of amoebots.

Most of the cited MRSs at the beginning of this section are very similar to our construction. The construction for crystalline robots is the closest one. Each of these consists of a line of (meta-)modules that are able to contract. Katoy et al. [36] also propose a “walking” structure (see Figure 12). However, the locomotion is still caused by the contraction of the body instead of motions of the legs. So, it is rather a caterpillar-like crawling than a walking movement.

5.3 Walking

A wide variety of animals are capable of walking locomotion, e.g., mammals, reptiles, birds, insects, millipedes, and spiders. Just as wide is the variety of differences, e.g., they differ in the number of legs, in the structure of the legs, and in their gait. Walking structures have been built for chain-type MRSs, e.g., M-TRAN [68, 38] and polybot [65, 69].

Our walking amoebot structure imitates millipedes. Millipedes have flexible, segmented bodies with tens to hundreds of legs that provide morphological robustness [54]. They move by propagating leg-density waves from the posterior to the anterior [25, 37]. We build our *millipede structure* from rhombical meta-modules of side length $\ell - 1$ (see Figure 10c). Let p denote the number of legs. The body and each leg consists of a line of rhombical meta-modules. All legs have the same size. Let q denote the number of rhombical modules in each leg. We attach each leg to a meta-module of the body and orientate them alternating to the front and to the back. We call those meta-modules *connectors*. In order to prevent the legs from colliding, we place q meta-modules between two connectors. Altogether, the millipede structure consists of $(2p - 1) \cdot q + p$ meta-modules.

In order to move the legs back and forth, we simply apply the realignment primitive (see Figure 6c) on all meta-modules within the legs (see Figure 10c). We achieve forward motion by releasing all bonds between the surface and the legs moving forwards. In one step, the body moves a distance of $q \cdot \ell$. Note that the number of legs has no impact on the velocity. By continuously repeating these leg movements, we achieve a motion similar to the leg-density waves of millipedes. Note that we reach the initial amoebot structure after two leg movements. Hence, we obtain the following theorem.

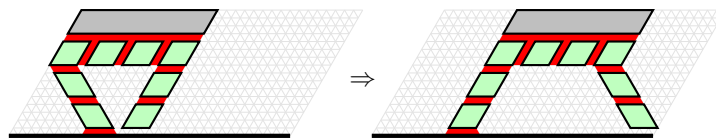
► **Theorem 16.** *Our millipede structure composed of rhombical meta-modules of side length $\ell - 1$ with p legs composed of q rhombical meta-modules moves a distance of $2 \cdot q \cdot \ell$ within each period of constant length.*

In practice, we can reduce the friction by additionally lifting the legs moving forwards (see Figure 13). For that, it suffices to partially contract all meta-modules of the body except for the connectors connected to a leg moving backwards. After the movement, we lower the lifted legs back to the ground. For that, we reverse the contractions within the body.

5.4 Transportation

Another important aspect of locomotion is the transportation of objects. The continuous track and worm structure are unsuitable for the transportation of objects due to their unstable top. In the worm structure, we can circumvent that problem by increasing the number of meta-modules not participating in the contractions and expansions. However, this decreases the velocity of the worm structure and increases the friction under the object.

In contrast, the millipede structure provides a rigid transport surface for the transportation of objects (see Figure 11). In practice, a high load introduced by the object on the structure may lead to problems. However, it was shown that millipedes have a large payload-to-weight ratio since they have to withstand high loads while burrowing in leaf litter, dead wood, or soil [25, 54]. The millipede is able to distribute the weight evenly on its legs. Consequently, the millipede structure is well suited for the transportation of objects.



■ **Figure 11** Transportation of objects by the millipede structure. The green rhombi indicate the millipede structure. The gray parallelogram indicates the object.

6 Conclusion and Future Work

In this paper, we have formalized the joint movement extension that were proposed by Feldmann et al. [23]. We have constructed meta-modules of rhombical and hexagonal shape that are able to perform various movement primitives. This allows us to simulate reconfiguration algorithms of various MRSs. However, our meta-modules are more flexible, e.g., we can move a hexagonal meta-module through two others. Such new movement primitives may lead to faster reconfiguration algorithms, e.g., sublinear solutions for hexagonal meta-modules or even arbitrary amoebot structures. Furthermore, we have presented three amoebot structures capable of moving along an even surface. In future work, movement on uneven surfaces can be considered.

References

- 1 Hossein Ahmadzadeh, Ellips Masehian, and Masoud Asadpour. Modular robotic systems: Characteristics and applications. *J. Intell. Robot. Syst.*, 81(3-4):317–357, 2016.
- 2 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmovic, Robin Y. Flatland, Matias Korman, Belén Palop, Irene Parada, André van Renssen, and Vera Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The $O(1)$ musketeers. *Algorithmica*, 83(5):1316–1351, 2021.
- 3 Abdullah Almethen, Othon Michail, and Igor Potapov. Distributed transformations of hamiltonian shapes based on line moves. *Theor. Comput. Sci.*, 942:142–168, 2023.

- 4 Greg Aloupis, Nadia M. Benbernou, Mirela Damian, Erik D. Demaine, Robin Y. Flatland, John Iacono, and Stefanie Wuhrer. Efficient reconfiguration of lattice-based modular robots. *Comput. Geom.*, 46(8):917–928, 2013.
- 5 Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D. Demaine, Robin Y. Flatland, Stefan Langerman, Joseph O’Rourke, Suneeta Ramaswami, Vera Sacristán Adinolfi, and Stefanie Wuhrer. Linear reconfiguration of cube-style modular robots. *Comput. Geom.*, 42(6-7):652–663, 2009.
- 6 Greg Aloupis, Sébastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristán Adinolfi, and Stefanie Wuhrer. Reconfiguration of cube-style modular robots using $O(\log n)$ parallel moves. In *ISAAC*, volume 5369 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2008.
- 7 Byoung Kwon An. Em-cube: cube-shaped, self-reconfigurable robots sliding on structure surfaces. In *2008 IEEE International Conference on Robotics and Automation, ICRA 2008, May 19-23, 2008, Pasadena, California, USA*, pages 3149–3155. IEEE, 2008. doi:10.1109/ROBOT.2008.4543690.
- 8 Rhodri H Armour and Julian FV Vincent. Rolling in nature and robotics: A review. *Journal of Bionic Engineering*, 3(4):195–208, 2006.
- 9 Alberto Brunete, Avinash Ranganath, Sergio Segovia, Javier Perez De Frutos, Miguel Hernando, and Ernesto Gambao. Current trends in reconfigurable modular robots design. *International Journal of Advanced Robotic Systems*, 14(3):1729881417710457, 2017.
- 10 Zack J. Butler, Keith Kotay, Daniela Rus, and Kohji Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *Int. J. Robotics Res.*, 23(9):919–937, 2004.
- 11 Jason Campbell and Padmanabhan Pillai. Collective actuation. *Int. J. Robotics Res.*, 27(3-4):299–314, 2008.
- 12 Gregory S. Chirikjian. Kinematics of a metamorphic robotic system. In *ICRA*, pages 449–455. IEEE Computer Society, 1994.
- 13 David Johan Christensen and Jason Campbell. Locomotion of miniature catom chains: Scale effects on gait and velocity. In *ICRA*, pages 2254–2260. IEEE, 2007.
- 14 Joshua J. Daymude, Kristian Hinnenthal, Andréa W. Richa, and Christian Scheideler. Computing by programmable particles. In *Distributed Computing by Mobile Entities*, volume 11340 of *Lecture Notes in Computer Science*, pages 615–681. Springer, 2019.
- 15 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The canonical amoebot model: Algorithms and concurrency control. In *DISC*, volume 209 of *LIPICs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 16 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: amoebot - a new model for programmable matter. In *SPAA*, pages 220–222. ACM, 2014.
- 17 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *NANOCOM*, pages 21:1–21:2. ACM, 2015.
- 18 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Universal shape formation for programmable matter. In *SPAA*, pages 289–299. ACM, 2016.
- 19 Daniel J. Dewey, Michael P. Ashley-Rollman, Michael DeRosa, Seth Copen Goldstein, Todd C. Mowry, Siddhartha S. Srinivasa, Padmanabhan Pillai, and Jason Campbell. Generalizing metamodules to simplify planning in modular robotic systems. In *IROS*, pages 1338–1345. IEEE, 2008.
- 20 Shlomi Dolev, Sergey Frenkel, Michael Rosenblit, Ram Prasad Narayanan, and K Muni Venkateswarlu. In-vivo energy harvesting nano robots. In *2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE)*, pages 1–5, 2016.
- 21 Fabien Dufoulon, Shay Kutten, and William K. Moses Jr. Efficient deterministic leader election for programmable matter. In *PODC*, pages 103–113. ACM, 2021.

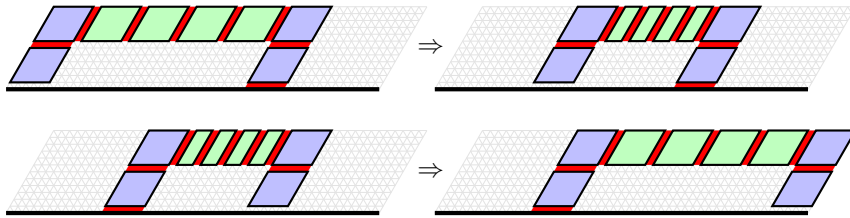
- 22 Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Motion planning for metamorphic systems: feasibility, decidability, and distributed reconfiguration. *IEEE Trans. Robotics*, 20(3):409–418, 2004.
- 23 Michael Feldmann, Andreas Padalkin, Christian Scheideler, and Shlomi Dolev. Coordinating amoebots via reconfigurable circuits. *J. Comput. Biol.*, 29(4):317–343, 2022.
- 24 Robert Fitch and Zack J. Butler. Million module march: Scalable locomotion for large self-reconfiguring robots. *Int. J. Robotics Res.*, 27(3-4):331–343, 2008.
- 25 Anthony Garcia, Gregory Krummel, and Shashank Priya. Fundamental understanding of millipede morphology and locomotion dynamics. *Bioinspiration & Biomimetics*, 16(2):026003, December 2020.
- 26 Shigeo Hirose. Three basic types of locomotion in mobile robots. In *Fifth International Conference on Advanced Robotics' Robots in Unstructured Environments*, pages 12–17. IEEE, 1991.
- 27 Ferran Hurtado, Enrique Molina, Suneeta Ramaswami, and Vera Sacristán Adinolfi. Distributed reconfiguration of 2d lattice-based modular robotic systems. *Auton. Robots*, 38(4):383–413, 2015.
- 28 Norio Inou, Hisato Kobayashi, and Michihiko Koseki. Development of pneumatic cellular robots forming a mechanical structure. In *ICARCV*, pages 63–68. IEEE, 2002.
- 29 Gangyuan Jing, Tarik Tosun, Mark Yim, and Hadas Kress-Gazit. An end-to-end system for accomplishing tasks with modular robots. In *Robotics: Science and Systems*, 2016.
- 30 Gangyuan Jing, Tarik Tosun, Mark Yim, and Hadas Kress-Gazit. Accomplishing high-level tasks with modular robots. *Auton. Robots*, 42(7):1337–1354, 2018.
- 31 Zoltán Juhász and Ambrus Zelei. Analysis of worm-like locomotion. *Periodica Polytechnica Mechanical Engineering*, 57(2):59–64, 2013.
- 32 Manivannan Sivaperuman Kalairaj, Catherine Jiayi Cai, Pavitra S, and Hongliang Ren. Untethered origami worm robot with diverse multi-leg attachments and responsive motions under magnetic actuation. *Robotics*, 10(4):118, 2021.
- 33 Matthew Kehoe and Davide Piovesan. Taxonomy of two dimensional bio-inspired locomotion systems. In *EMBC*, pages 3703–3706. IEEE, 2019.
- 34 Brian T. Kirby, Jason Campbell, Burak Aksak, Padmanabhan Pillai, James F. Hoburg, Todd C. Mowry, and Seth Copen Goldstein. Catoms: Moving robots without moving parts. In *AAAI*, pages 1730–1731. AAAI Press / The MIT Press, 2005.
- 35 Irina Kostitsyna, Christian Scheideler, and Daniel Warner. Fault-tolerant shape formation in the amoebot model. In *DNA*, volume 238 of *LIPICs*, pages 9:1–9:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- 36 Keith Kotay, Daniela Rus, and Marsette Vona. Using modular self-reconfiguring robots for locomotion. In *ISER*, volume 271 of *Lecture Notes in Control and Information Sciences*, pages 259–269. Springer, 2000.
- 37 Shigeru Kuroda, Nariya Uchida, and Toshiyuki Nakagaki. Gait switching with phase reversal of locomotory waves in the centipede scolopocryptops rubiginosus. *Bioinspiration & Biomimetics*, 17(2):026005, March 2022.
- 38 Haruhisa Kurokawa, Eiichi Yoshida, Kohji Tomita, Akiya Kamimura, Satoshi Murata, and Shigeru Kokaji. Self-reconfigurable M-TRAN structures and walker generation. *Robotics Auton. Syst.*, 54(2):142–149, 2006.
- 39 Michael LaBarbera. Why the wheels won't go. *The American Naturalist*, 121(3):395–408, 1983.
- 40 Giuseppe Antonio Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. Shape formation by programmable particles. *Distributed Comput.*, 33(1):69–101, 2020.
- 41 Daniel Mellinger, Vijay Kumar, and Mark Yim. Control of locomotion with shape-changing wheels. In *ICRA*, pages 1750–1755. IEEE, 2009.

- 42 Satoshi Murata, Haruhisa Kurokawa, and Shigeru Kokaji. Self-assembling machine. In *ICRA*, pages 441–448. IEEE Computer Society, 1994.
- 43 An Nguyen, Leonidas J Guibas, and Mark Yim. Controlled module density helps reconfiguration planning. In *Workshop on the Algorithmic Foundations of Robotics*, pages TH15–TH27, 2001.
- 44 Hayato Omori, Takeshi Hayakawa, and Taro Nakamura. Locomotion and turning patterns of a peristaltic crawling earthworm robot composed of flexible units. In *IROS*, pages 1630–1635. IEEE, 2008.
- 45 Andreas Padalkin, Manish Kumar, and Christian Scheideler. Reconfiguration and locomotion with joint movements in the amoebot model. *CoRR*, abs/2305.06146, 2023.
- 46 Andreas Padalkin, Manish Kumar, and Christian Scheideler. Reconfiguration and locomotion with joint movements in the amoebot model. *40th European Workshop on Computational Geometry*, 2024.
- 47 Andreas Padalkin, Christian Scheideler, and Daniel Warner. The structural power of reconfigurable circuits in the amoebot model. In *DNA*, volume 238 of *LIPICs*, pages 8:1–8:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- 48 Irene Parada, Vera Sacristán, and Rodrigo I. Silveira. A new meta-module for efficient reconfiguration of hinged-units modular robots. In *ICRA*, pages 5197–5202. IEEE, 2016.
- 49 John Romanishin, Kyle Gilpin, and Daniela Rus. M-blocks: Momentum-driven, magnetic modular robots. In *IROS*, pages 4288–4295. IEEE, 2013.
- 50 Daniela Rus and Masette Vona. Self-reconfiguration planning with compressible unit modules. In *ICRA*, pages 2513–2520. IEEE Robotics and Automation Society, 1999.
- 51 Daniela Rus and Masette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Auton. Robots*, 10(1):107–124, 2001.
- 52 Hossein Sadjadi, Omid Mohareri, Mohammad Amin Al-Jarrah, and Khaled Assaleh. Design and implementation of hexbot: A modular self-reconfigurable robotic system. *J. Frankl. Inst.*, 349(7):2281–2293, 2012.
- 53 Jimmy Sastra, Sachin Chitta, and Mark Yim. Dynamic rolling for a modular loop robot. *Int. J. Robotics Res.*, 28(6):758–773, 2009.
- 54 Qi Shao, Xuguang Dong, Zhonghan Lin, Chao Tang, Hao Sun, Xin-Jun Liu, and Huichan Zhao. Untethered robotic millipede driven by low-pressure microfluidic actuators for multi-terrain exploration. *IEEE Robotics Autom. Lett.*, 7(4):12142–12149, 2022.
- 55 John W. Suh, Samuel B. Homans, and Mark Yim. Telecubes: Mechanical design of a module for self-reconfigurable robotics. In *ICRA*, pages 4095–4101. IEEE, 2002.
- 56 Yosuke Suzuki, Norio Inou, Michihiko Koseki, and Hitoshi Kimura. Reconfigurable modular robots adaptively transforming a mechanical structure (numerical expression of transformation criteria of "chobie ii" and motion experiments). In *DARS*, pages 393–403. Springer, 2008.
- 57 Tommaso Toffoli and Norman Margolus. Programmable matter: Concepts and realization. *Int. J. High Speed Comput.*, 5(2):155–170, 1993.
- 58 Sergei Vassilvitskii, Jeremy Kubica, Eleanor Gilbert Rieffel, John W. Suh, and Mark Yim. On the general reconfiguration problem for expanding cube style modular robots. In *ICRA*, pages 801–808. IEEE, 2002.
- 59 Sergei Vassilvitskii, Mark Yim, and John W. Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *ICRA*, pages 117–122. IEEE, 2002.
- 60 Michael Philetus Weller, Brian T. Kirby, H. Benjamin Brown, Mark D. Gross, and Seth Copen Goldstein. Design of prismatic cube modules for convex corner traversal in 3d. In *IROS*, pages 1490–1495. IEEE, 2009.
- 61 Paul J. White and Mark Yim. Scalable modular self-reconfigurable robots using external actuation. In *IROS*, pages 2773–2778. IEEE, 2007.
- 62 Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *ITCS*, pages 353–354. ACM, 2013.
- 63 Mark Yim. New locomotion gaits. In *ICRA*, pages 2508–2514. IEEE Computer Society, 1994.

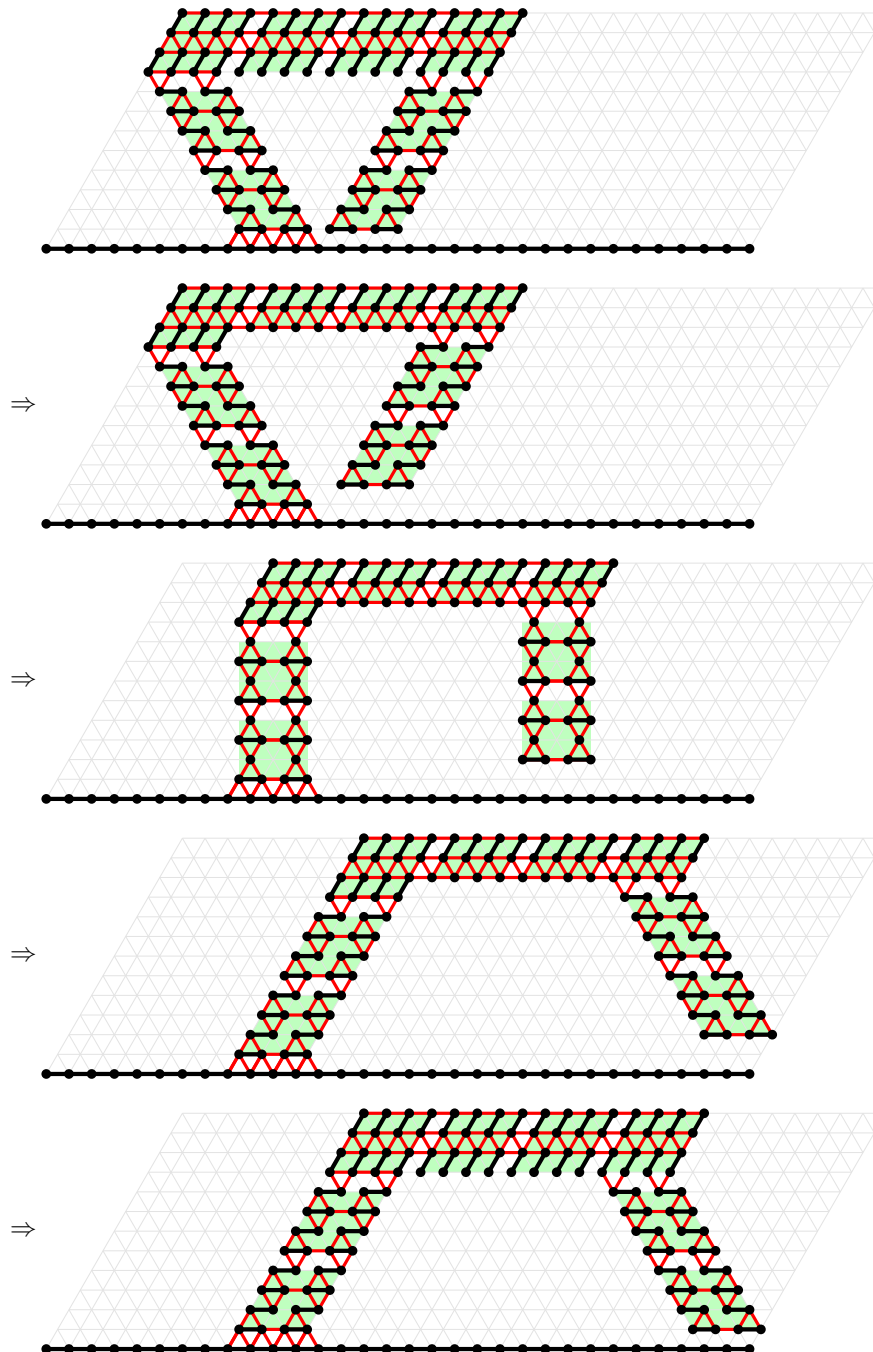
- 64 Mark Yim, David Duff, and Kimon Roufas. Polybot: A modular reconfigurable robot. In *ICRA*, pages 514–520. IEEE, 2000.
- 65 Mark Yim, Kimon Roufas, David Duff, Ying Zhang, Craig Eldershaw, and Samuel B. Homans. Modular reconfigurable robots in space applications. *Auton. Robots*, 14(2-3):225–237, 2003.
- 66 Mark Yim, Paul J. White, Michael Park, and Jimmy Sastra. Modular self-reconfigurable robots. In *Encyclopedia of Complexity and Systems Science*, pages 5618–5631. Springer, 2009.
- 67 Mark Yim, Ying Zhang, and David Duff. Modular robots. *IEEE Spectrum*, 39(2):30–34, 2002.
- 68 Eiichi Yoshida, Satoshi Murata, Akiya Kamimura, Kohji Tomita, Haruhisa Kurokawa, and Shigeru Kokaji. Evolutionary synthesis of dynamic motion and reconfiguration process for a modular robot M-TRAN. In *CIRA*, pages 1004–1010. IEEE, 2003.
- 69 Ying Zhang, Mark Yim, Craig Eldershaw, Dave Duff, and Kimon Roufas. Scalable and reconfigurable configurations and locomotion gaits for chain-type modular reconfigurable robots. In *CIRA*, pages 893–899. IEEE, 2003.

A Omitted Figures

This section contains omitted figures.



■ **Figure 12** Caterpillar structure. This structure is a replication of the structure by Katoy et al. [36] in the joint movement extension.



■ **Figure 13** Movement of a millipede. For the sake of simplicity, we only show two legs.

Complexity of Boolean Automata Networks Under Block-Parallel Update Modes

Kévin Perrot

Université publique, Marseille, France
Aix-Marseille Univ, CNRS, LIS, Marseille, France

Sylvain Sené

Université publique, Marseille, France
Aix-Marseille Univ, CNRS, LIS, Marseille, France

Léah Tapin

Aix-Marseille Univ, CNRS, LIS, Marseille, France

Abstract

Boolean automata networks (*aka* Boolean networks) are space-time discrete dynamical systems, studied as a model of computation and as a representative model of natural phenomena. A collection of simple entities (the automata) update their 0-1 states according to local rules. The dynamics of the network is highly sensitive to update modes, i.e., to the schedule according to which the automata apply their local rule. A new family of update modes appeared recently, called block-parallel, which is dual to the well studied block-sequential. Although basic, it embeds the rich feature of update repetitions among a temporal updating period, allowing for atypical asymptotic behaviors. In this paper, we prove that it is able to breed complex computations, squashing almost all decision problems on the dynamics to the traditionally highest (for reachability questions) class PSPACE. Despite obtaining these complexity bounds for a broad set of local and global properties, we also highlight a surprising gap: bijectivity is still coNP.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Problems, reductions and completeness

Keywords and phrases Boolean networks, finite dynamical systems, block-parallel update schedule

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.19

Related Version *Full Version*: <https://arxiv.org/abs/2402.06294>

Acknowledgements This work received support from ANR-18-CE40-0002 FANs, STIC AmSud CAMA 22-STIC-02 (Campus France MEAE) and HORIZON-MSCA-2022-SE-01-101131549 ACAN-COS projects.

1 Introduction

Automata networks are distributed models of computation, defined locally by means of individual entities (called automata) interacting with each other over discrete time, and collectively performing global computations. The model originates from the seminal work of McCulloch and Pitts on neural networks [32] (with local threshold Boolean functions). It raised fundamental complexity and computability questions on their dynamics, with notable considerations of feedback shift registers [26, 14], and perceptrons [39]. The Boolean case serves as a framework for biological modelling, as proposed by Kauffman and Thomas on gene regulation [28, 42], and repeatedly confirmed since the 1990s [33, 1, 19, 43] (where limit dynamics receive biological interpretations matching experiments).

Our contribution is at the frontier between theoretical computer science, discrete mathematics, and systems biology. When working on Boolean automata networks, it is utmost important to define the way automata update their state over time (namely the update mode), in order to obtain a discrete dynamical system. Indeed:



© Kévin Perrot, Sylvain Sené, and Léah Tapin;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 19; pp. 19:1–19:19

Leibniz International Proceedings in Informatics



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

- Even if the fixed points obtained under the parallel update mode are fixed points obtained under any other update mode [20], specific update modes may generate additional fixed points (e.g., block-parallel).
- The limit cycles which are not fixed points, obtained under a given update mode, are not necessarily conserved under another update mode [11, 23, 5, 22, 6].

In other words, a Boolean automata network may admit a large number of distinct dynamics (depending on the update mode), which requires a strong attention, in particular when it is employed as a phenomenological model in systems biology. In the context of gene regulatory networks, chromatin dynamics has emerged as a full-fledged research track to understand the temporality of mRNA transcriptional machinery (which has no clear biological answer at present) [24, 7, 25, 16]. From a theoretical standpoint, advances on chromatin dynamics tend to show that genetic expression is neither purely asynchronous nor purely synchronous, hence supporting studies of in-between update modes.

In this line, this paper aims at studying the peculiar role and impact of block-parallel update modes, shown to have relevant features from both formal and applied standpoints [12, 36], in the sense that (i) they can generate fixed points which are not fixed points of the dynamical system obtained when the underlying network evolves synchronously, and (ii) they can implement specific biological timers which are intrinsically governed by phenomena exogenous to regulatory control. We take the lens of complexity theory, and provide ground results on classical decision problems related to fixed points and limit cycles, reachability, *etc.* These new complexity bounds highlight that most decision problems known to be NP-complete under block-sequential update modes, such as the image/preimage problems, and fixed point problems [17, 8, 35], are PSPACE-complete under block-parallel update modes. It suggests that the “expressivity” of such update modes comes at a high cost in terms of simulation, which strengthens the need for structural results. However, there are unexpected exceptions, related to bijectivity and steadiness.

In Section 2, we define formally the model and present known results. Section 3 exposes our results. Classical problems on computing images, preimages, fixed points and limit cycles are characterized: they all jump from NP (under block-sequential update modes) to PSPACE (under block-parallel update modes). Then we prove a general bound on the recognition of functional subdynamics. Regarding global properties, recognizing bijective dynamics remains coNP-complete, and recognizing constant dynamics becomes PSPACE-complete. The case of identity recognition is much subtler, and we provide three incomparable bounds: a trivial coNP-hardness one, a tough ModP-hardness, and a $\text{FP}^{\text{PSPACE}}$ -completeness result derived from the recent literature. In Section 4, we summarize the results and expose perspectives.

2 Definitions and state of the art

We denote the set of integers by $\llbracket n \rrbracket = \{0, \dots, n-1\}$, the Booleans by $\mathbb{B} = \{0, 1\}$, the i -th component of a vector $x \in \mathbb{B}^n$ by $x_i \in \mathbb{B}$, and the restriction of x to domain $I \subset \llbracket n \rrbracket$ by $x_I \in \mathbb{B}^{|I|}$. For two graphs $G = (V(G), A(G))$ and $H = (V(H), A(H))$, we denote by $G \sim H$ when they are isomorphic, i.e., when there is a bijection $\pi : V(G) \rightarrow V(H)$ such that $(x, y) \in A(G) \iff (\pi(x), \pi(y)) \in A(H)$. We denote by $G \sqsubset H$ when G is a subgraph of H , i.e., when G' such that $G' \sim G$ can be obtained from H by vertex and arc deletions.

Boolean automata network. A *Boolean automata network* (BAN) is a discrete dynamical system on \mathbb{B}^n . A configuration $x \in \mathbb{B}^n$ associates to each of the n automata among $\llbracket n \rrbracket$ a Boolean state among \mathbb{B} . The individual dynamics of a each automaton $i \in \llbracket n \rrbracket$ is described

by a local function $f_i : \mathbb{B}^n \rightarrow \mathbb{B}$ giving its new state according to the current configuration. To get a dynamics, one needs to settle the order in which the automata update their state by application of their local function. That is, an *update schedule* must be given. The most basic is the parallel update schedule, where all automata update their state synchronously at each step, formally as $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ defined by $\forall x \in \mathbb{B}^n : f(x) = (f_0(x), f_1(x), \dots, f_{n-1}(x))$. In this work, we concentrate on the block-parallel update schedule, motivated by the biological context of gene regulatory networks, where each automaton is a gene and the dynamics give clues on cell phenotypes. Not all automata will be update simultaneously as in the parallel update mode. They will instead be grouped by subsets. For simplicity in defining the local functions of a BAN, we extend the $f_i : \mathbb{B}^n \rightarrow \mathbb{B}$ notation to subsets $I \subseteq \llbracket n \rrbracket$ as $f_I : \mathbb{B}^n \rightarrow \mathbb{B}^{|I|}$. We also denote $f_{(I)} : \mathbb{B}^n \rightarrow \mathbb{B}^n$ the update of automata from subset I , defined as:

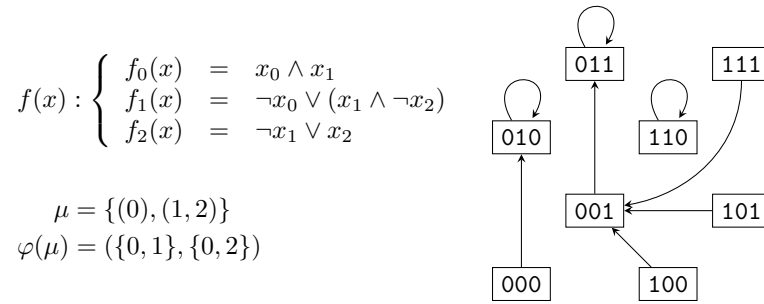
$$\forall i \in \llbracket n \rrbracket : f_{(I)}(x)_i = \begin{cases} f_i(x) & \text{if } i \in I \\ x_i & \text{otherwise.} \end{cases}$$

Block-sequential update schedule. A *block-sequential* update schedule is an *ordered partition* of $\llbracket n \rrbracket$, given as a sequence of subsets $(W_i)_{i \in \llbracket \ell \rrbracket}$ where $W_i \subseteq \llbracket n \rrbracket$ is a *block*. The automata within a block are updated simultaneously, and the blocks are updated sequentially. During one iteration (*step*) of the network, the state of each automaton is updated exactly once. The update of each block is called a *substep*. This update mode received great attention on many aspects. The concept of the *update digraph* is introduced in [4] and characterized in [3] to capture equivalence classes of block-sequential update schedules (leading to the same dynamics). Conversions between block-sequential and parallel update schedules are investigated in [37] (how to parallelize a block-sequential update schedule), [22] (the preservation of cycles throughout the parallelization process), and [9] (the cost of sequentialization of a parallel update schedule).

Block-parallel update schedule. A *block-parallel* update schedule is a *partitioned order* of $\llbracket n \rrbracket$, given as a set of subsets $\mu = \{S_k\}_{k \in \llbracket s \rrbracket}$ where $S_k = (i_0^k, \dots, i_{n_k-1}^k)$ is a sequence of $n_k > 0$ elements of $\llbracket n \rrbracket$ for all $k \in \llbracket s \rrbracket$, called an *o-block* (shortcut for *ordered-block*). Each automaton appears in exactly one o-block. It follows an idea dual to the block-sequential update mode: the automata within an o-block are updated sequentially, and the o-blocks are updated simultaneously. The o-block sequences are taken circularly at each substep, until we reach the end of each o-block simultaneously (which happens after the least common multiple (lcm) of their sizes). The set of block-parallel update modes of size n is denoted BP_n . Formally, the update of f under $\mu \in \text{BP}_n$ is given by $f_{\{\mu\}} : \mathbb{B}^n \rightarrow \mathbb{B}^n$ defined, with $\ell = \text{lcm}(n_1, \dots, n_s)$, as $f_{\{\mu\}}(x) = f_{(W_{\ell-1})} \circ \dots \circ f_{(W_1)} \circ f_{(W_0)}(x)$, where for all $i \in \llbracket \ell \rrbracket$ we define $W_i = \{i_{i \bmod n_k}^k \mid k \in \llbracket s \rrbracket\}$. In order to compute the set of automata updated at each substep, it is possible to convert a block-parallel update schedule into a sequence of blocks of length ℓ (which is usually not a block-sequential update schedule, because repetitions of automaton update may appear [36]). We defined this map as φ :

$$\varphi(\{S_k\}_{k \in \llbracket s \rrbracket}) = (W_i)_{i \in \llbracket \ell \rrbracket} \text{ with } W_i = \{i_{i \bmod n_k}^k \mid k \in \llbracket s \rrbracket\}.$$

An example is given on Figure 1. The parallel update schedule corresponds to the block-parallel update schedule $\mu_{\text{par}} = \{(i) \mid i \in \llbracket n \rrbracket\} \in \text{BP}_n$, with $\varphi(\mu_{\text{par}}) = (\llbracket n \rrbracket)$, i.e., a single block containing all automata is updated at each step (there is only one substep).



■ **Figure 1** Example of an automata network of size $n = 3$ with a block-parallel update mode $\mu \in \text{BP}_n$. Local functions (upper left), conversion of μ to a sequence of blocks (lower left), and dynamics of $f_{\{\mu\}}$ on configuration space \mathbb{B}^3 (right). One step is composed of two substeps: the first substep updates the block $\{0, 1\}$, the second substep updates the block $\{0, 2\}$. As an example, in computing the image of configuration 111, the first substep (update of automata 0 and 1) gives 101, and the second substep (update of automata 0 and 2) gives 001.

Block-parallel update schedules have been introduced in [12], motivated by applications to gene regulatory networks, and their ability to generate new stable configurations (compared to block-sequential update schedules). A first theoretical study has been conducted in [36], providing counting formulas and enumeration algorithms, subject to equivalence relations on the produced dynamics.

Fixed point and limit cycle. A BAN f of size n under block-parallel update schedule $\mu \in \text{BP}_n$ defines a deterministic discrete dynamical system $f_{\{\mu\}}$ on configuration space \mathbb{B}^n . Since the space is finite, the orbit of any configuration is ultimately periodic. For $p \geq 1$, a sequence of configurations x^0, \dots, x^{p-1} is a *limit cycle* of length p when $\forall i \in \llbracket p \rrbracket : f_{\{\mu\}}(x^i) = x^{i+1 \bmod p}$. For $p = 1$ we call $x \in \mathbb{B}^n$ such that $f_{\{\mu\}}(x) = x$ a *fixed point*.

Complexity. To be given as input to a decision problem, a BAN is encoded as a tuple of n Boolean circuits, one for each local function $f_i : \mathbb{B}^N \rightarrow \mathbb{B}$ for $i \in \llbracket n \rrbracket$. This encoding can be seen as Boolean formulas for each automaton, and easily implements high-level descriptions with if-then-else statements (used intensively in our constructions).

The computational complexity of finite discrete dynamical systems has been explored on the related models of finite cellular automata [40] and reaction networks [13]. Regarding automata networks, fixed points received early attention in [2] and [17], with existence problems complete for NP. Because of the fixed point invariance for block-sequential update schedules [38], the focus switched to limit cycles [6, 8], with problems reaching the second level of the polynomial hierarchy. The interplay of different update schedules has been investigated in [6]. Finally, let us mention the general complexity lower bounds, established for any first-order question on the dynamics, under the parallel update schedule [18].

3 Computational complexity under block-parallel updates

Computational complexity is important to anyone willing to use algorithmic tools in order to study discrete dynamical systems. Lower bounds inform on the best worst case time or space one can expect with an algorithm solving some problem. The n local functions of a BAN are encoded as Boolean circuits, which is a convenient formalism corresponding to the high level descriptions one usually employs. The update mode is given as a list of lists of integers, each of them being encoded either in unary or binary (this makes no difference, because the encoding of local functions already has a size greater than n).

In this section we characterize the computational complexity of typical problems arising in the framework of automata networks. We will see that almost all problems reach PSPACE-completeness. The intuition behind this fact is that the description of a block-parallel update mode may expend (through φ) to an exponential number of substeps, during which a linear bounded Turing machine may be simulated via iterations of a circuit. We first recall this folklore building block and present a general outline of our constructions (Subsection 3.1). Then we start with results on computing images, preimages, fixed points and limit cycles (Subsection 3.2), before studying reachability and global properties of the function $f_{\{\mu\}}$ computed by an automata network f under block-parallel update schedule μ (Subsection 3.3).

3.1 Outline of the PSPACE-hardness constructions

We will design polynomial time many-one reductions from the following PSPACE-complete decision problem, which appears for example in [21].

Iterated Circuit Value Problem (**Iter-CVP**)

Input: a Boolean circuit $C : \mathbb{B}^n \rightarrow \mathbb{B}^n$, a configuration $x \in \mathbb{B}^n$, and $i \in \llbracket n \rrbracket$.

Question: does $\exists t \in \mathbb{N} : C^t(x)_i = 1$?

► **Theorem 1** (folklore). **Iter-CVP** is PSPACE-complete.

Before presenting the general outline of our constructions, we need a technical lemma related to the generation of primes (proof in Appendix A).

► **Lemma 2.** For all $n \geq 2$, a list of distinct prime integers p_1, p_2, \dots, p_{k_n} such that $2 \leq p_i < n^2$ and $2^n < \prod_{i=1}^{k_n} p_i < 2^{2n^2}$ can be computed in time $\mathcal{O}(n^2)$, with $k_n = \lfloor \frac{n^2}{2 \ln(n)} \rfloor$.

Our constructions of automata networks and block-parallel update schedules for the computational complexity lower bounds are based on the following.

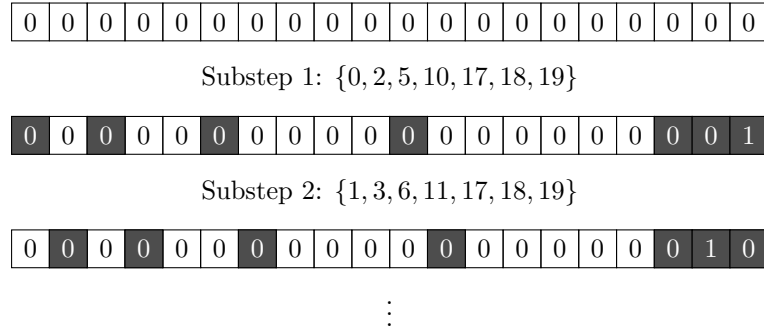
► **Definition 3.** For any $n \geq 2$, let p_1, p_2, \dots, p_{k_n} be the k_n primes given by Lemma 2, and denote $q_j = \sum_{i=1}^j p_i$ their cumulative series for j from 0 to k_n . Define the automata network g_n on q_{k_n} automata $\llbracket q_{k_n} \rrbracket$ with constant 0 local functions, where the components are grouped in o -blocks of length p_i , that is with $\mu_n = \bigcup_{i \in \llbracket k_n \rrbracket} \{(q_i, q_i + 1, \dots, q_{i+1} - 1)\}$.

► **Lemma 4.** For any $n \geq 2$, one can compute g_n and μ_n in time $\mathcal{O}(n^4)$, and $|\varphi(\mu_n)| > 2^n$.

Proof. The time bound comes from Lemma 2 and the fact that q_{k_n} is in $\mathcal{O}(n^4)$. The number of blocks in $\varphi(\mu_n)$ is the least common multiple of its o -block sizes, which is the product $\prod_{i=1}^{k_n} p_i$, hence from Lemma 2 we conclude that it is greater than 2^n . ◀

The general idea is now to add some automata to g_n and place them within singletons in μ_n , i.e., each of them in a new o -block of length 1. We propose an example implementing a binary counter on n bits.

► **Example 5.** Given $n \geq 2$, consider g_n and μ_n given by Lemma 4. Construct f from g_n by adding n Boolean components $\{q_{k_n}, \dots, q_{k_n+n}\}$, whose local functions increment a binary counter on those n bits, until it freezes to $2^n - 1$ (all bits in state 1). Construct μ' from μ_n as $\mu' = \mu_n \cup \bigcup_{i \in \llbracket n \rrbracket} \{(q_{k_n} + i)\}$, so that the counter components are updated at each substep. Observe that the pair f, μ' can be still be computed from n in time $\mathcal{O}(n^4)$. Figure 2 illustrates an example of orbit for $n = 3$, and one can notice that $f_{\{\mu'\}}$ is a constant function sending any $x \in \mathbb{B}^n$ to $0^{q_{k_n}} 1^n$.



■ **Figure 2** Substeps leading to the image of configuration $0^{q_{k_n}}010$ in $f_{\{\mu'\}}$ from Example 5 for $n = 3$ ($k_n = 4$ and $q_{k_n} = 2 + 3 + 5 + 7 = 17$). The last 3 bits implement a binary counter, freezing at 7 (111). Above each substep the block of updated automata is given.

Remark that we will prove complexity lower bounds by reduction from **Iter-CVP**, where n will be the number of inputs and outputs of the circuit to be iterated, hence the integer n itself will be encoded in unary. As a consequence, the construction of Example 5 is computed in polynomial time.

3.2 Images, preimages, fixed points and limit cycles

We start the study of the computational complexity of automata networks under block-parallel update schedules with the most basic problem of computing the image $f_{\{\mu\}}(x)$ of some configuration x through $f_{\{\mu\}}$ (i.e., one step of the evolution), which is already PSPACE-hard. We conduct this study as decision problems. It is actually hard to compute even a single bit of $f_{\{\mu\}}(x)$. The fixed point verification problem is a particular case of computing an image, which is still PSPACE-hard (unlike block-sequential update schedules for which this problem is in P). Recall that the encoding of μ (with integers in unary or binary) has no decisive influence on the input size, this latter being characterized by the circuits sizes and in particular their number of inputs, denoted n , which is encoded in unary.

Block-parallel step bit (**BP-Step-Bit**)
 Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in \llbracket n \rrbracket}$ as circuits, $\mu \in \text{BP}_n$, $x \in \mathbb{B}^n$, $j \in \llbracket n \rrbracket$.
 Question: does $f_{\{\mu\}}(x)_j = 1$?

Block-parallel step (**BP-Step**)
 Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in \llbracket n \rrbracket}$ as circuits, $\mu \in \text{BP}_n$, $x, y \in \mathbb{B}^n$.
 Question: does $f_{\{\mu\}}(x) = y$?

Block-parallel fixed point verification (**BP-Fixed-Point-Verif**)
 Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in \llbracket n \rrbracket}$ as circuits, $\mu \in \text{BP}_n$, $x \in \mathbb{B}^n$.
 Question: does $f_{\{\mu\}}(x) = x$?

This first set of problems is related to the image of a given configuration x , which allows the reasonings to concentrate on the dynamics of substeps for that single configuration x , regardless of what happens for other configurations. Note that n will be the size of the **Iter-CVP** instance, while the size of the automata network will be $q_{k_n} + \ell' + n + 1$.

► **Theorem 6.** *BP-Step-Bit, BP-Step and BP-Fixed-Point-Verif are PSPACE-complete.*

Proof. The problems **BP-Step-Bit**, **BP-Step** and **BP-Fixed-Point-Verif** are in PSPACE, with a simple algorithm obtaining $f_{\{\mu\}}(x)$ by computing the least common multiple of o-block sizes and then using a pointer for each block throughout the computation of that number of substeps (each substep evaluates local functions in polynomial time).

We give a single reduction for the hardness of **BP-Step-Bit**, **BP-Step** and **BP-Fixed-Point-Verif**, where we only need to consider the dynamics of the substeps starting from one configuration x . Given an instance of **Iter-CVP** with a circuit $C : \mathbb{B}^n \rightarrow \mathbb{B}^n$, a configuration $\tilde{x} \in \mathbb{B}^n$ and $i \in \llbracket n \rrbracket$, we apply Lemma 4 to construct g_n, μ_n on automata set $P = \llbracket q_{k_n} \rrbracket$. Automata from P have constant 0 local functions, and the number of substeps is $\ell = |\varphi(\mu_n)| > 2^n$ thanks to the prime's lcm. We define a BAN f by adding:

- $\ell' = \lceil \log_2(\ell) \rceil$ automata numbered $B = \{q_{k_n}, \dots, q_{k_n} + \ell' - 1\}$, implementing a counter that increments modulo ℓ at each substep, and remains fixed when x_B encodes an integer greater or equal to ℓ (case not considered in this proof);
- n automata numbered $D = \{q_{k_n} + \ell', \dots, q_{k_n} + \ell' + n - 1\}$, whose local functions iterate $C : \mathbb{B}^n \rightarrow \mathbb{B}^n$ while the counter is smaller than $\ell - 1$, and go to state \tilde{x} when the counter reaches $\ell - 1$, i.e., with

$$f_D(x) = \begin{cases} C(x_D) & \text{if } x_B < \ell - 1, \\ \tilde{x} & \text{otherwise; and} \end{cases}$$

- 1 automaton numbered $R = \{q_{k_n} + \ell' + n\}$, whose local function $f_R(x) = x_R \vee x_{q_{k_n} + \ell' + i}$ records whether a state 1 appeared at automaton in relative position i within D .

We also add singletons to μ_n for each of these additional automata, by setting

$$\mu' = \mu_n \cup \bigcup_{j \in B \cup D \cup R} \{(j)\}.$$

Now, consider the dynamics of substeps in computing the image of configuration $x = 0^{q_{k_n}} 0^{\ell'} \tilde{x} 0$. During the first $\ell - 1$ substeps:

- automata P have constant 0 local function;
- automata B increment a counter from 0 to $\ell - 1$;
- automata D iterate circuit C from \tilde{x} ; and
- automaton R records whether the i -th bit of D has been in state 1 during some iteration.

During the last substep, automata B go back to 0^n because of the modulo, and automata D go back to state \tilde{x} . Since the number of substeps ℓ is greater than 2^n (Lemma 4), the iterations of C search the whole orbit of \tilde{x} , and at the end of the step automaton R has recorded whether the **Iter-CVP** instance is positive (went to state 1) or negative (still in state 0). The images are respectively $y_- = 0^{q_{k_n}} 0^{\ell'} \tilde{x} 0$ or $y_+ = 0^{q_{k_n}} 0^{\ell'} \tilde{x} 1$. This concludes the reductions, to **BP-Step-Bit** by asking whether automaton R (numbered $q_{k_n} + 2n$) is in state 1, to **BP-Step** by asking whether the image of x is y_+ , and to **BP-Fixed-Point-Verif** because $y_- = x$ (coPSPACE-hardness). ◀

As a corollary, the associated functional problem of computing $f_{\{\mu\}}$ is computable in polynomial space and is PSPACE-hard for polynomial time Turing reductions (not for many-one reductions, as there is no concept of negative instance for total functional problems). Deciding whether a given configuration y has a preimage through $f_{\{\mu\}}$ is also PSPACE-complete (see Appendix A for details).

Now, we study the computational complexity of problems related to the existence of fixed points and limit cycles in an automata network under block-parallel update schedule. Again, we need to consider the image of all configurations, and have no control on neither the start

configuration x nor the end configuration y during the dynamics of substeps. In particular, the counter may be initialized to any value, and the bit R may already be set to 1. We adapt the previous reductions accordingly.

Block-parallel fixed point (**BP-Fixed-Point**)

Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in \llbracket n \rrbracket}$ as circuits, $\mu \in \text{BP}_n$.

Question: does $\exists x \in \mathbb{B}^n : f_{\{\mu\}}(x) = x$?

Block-parallel limit cycle of length k (**BP-Limit-Cycle- k**)

Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in \llbracket n \rrbracket}$ as circuits, $\mu \in \text{BP}_n$.

Question: does $\exists x \in \mathbb{B}^n : f_{\{\mu\}}^k(x) = x$?

Block-parallel limit cycle (**BP-Limit-Cycle**)

Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in \llbracket n \rrbracket}$ as circuits, $\mu \in \text{BP}_n$, $k \in \mathbb{N}_+$.

Question: does $\exists x \in \mathbb{B}^n : f_{\{\mu\}}^k(x) = x$?

On limit cycles we have a family of problems (one for each integer k), and a version where k is part of the input (encoded in binary). It makes no difference on the complexity.

► **Theorem 7.** **BP-Fixed-Point**, **BP-Limit-Cycle- k** for any $k \in \mathbb{N}_+$ and **BP-Limit-Cycle** are PSPACE-complete.

Proof. These problems still belong to PSPACE, because they amount to enumerating configurations and computing images by $f_{\{\mu\}}$, which can be performed from **BP-Step** (Theorem 6).

We start with the hardness proof for the fixed point existence problem, and we will then adapt it to limit cycle existence problems. Given an instance $C : \mathbb{B}^n \rightarrow \mathbb{B}^n$, $\tilde{x} \in \mathbb{B}^n$, $i \in \llbracket n \rrbracket$ of **Iter-CVP**, we construct the same block-parallel update schedule μ' as in the proof of Theorem 6, and modify the local functions of automata B and R as follows:

- automata B increment a counter modulo ℓ at each substep, and go to 0 when the counter is greater than (or equal to) $\ell - 1$; and
- automaton R records whether a state 1 appears at the i -th bit of x_D , and flips when the counter is equal to $\ell - 1$, i.e.,

$$f_R(x) = \begin{cases} x_R \vee x_{q_{k_n} + \ell' + i} & \text{if } x_B < \ell - 1, \\ \neg x_R & \text{otherwise.} \end{cases}$$

Recall that automata D iterate the circuit when $x_B < \ell - 1$ and go to \tilde{x} otherwise, and that the number ℓ of substeps is larger than 2^n .

If the **Iter-CVP** instance is positive, then configuration $x = 0^{q_{k_n}} 0^{\ell'} \tilde{x} 0$ is a fixed point of $f_{\{\mu'\}}$. Indeed, during the ℓ -th and last substep, the primes P are still in state $0^{q_{k_n}}$, the counter B goes back to 0 (state $0^{\ell'}$), the circuit D goes back to \tilde{x} , and automaton R has recorded the 1 which is flipped into state 0.

Conversely, if there is a fixed point configuration x , then the counter must be at most $\ell - 1$ because of the modulo ℓ increment. Furthermore, automata D will encounter one substep during which it goes to \tilde{x} , hence the resulting configuration on D will be in the orbit of \tilde{x} , i.e., x_D is in the orbit of \tilde{x} . Finally, automaton R will also encounter exactly one substep during which it is flipped (when $x_B \geq \ell - 1$). As a consequence, in order to go back to its initial value x_R , the state of R must be flipped during another substep, which can only happen when it is in state 0 and automaton $q_{k_n} + \ell' + i$ is in state 1. We conclude that the i -th bit of a configuration in the orbit of \tilde{x} is in state 1 during some iteration of the circuit C , meaning that the **Iter-CVP** instance is positive. Remark that in this case, configuration $0^{q_{k_n}} 0^{\ell'} \tilde{x} 0$ is one of the fixed points.

For the limit cycle existence problems, we modify the construction to let the counter go up to $k\ell - 1$. Precisely:

- $\ell' = \lceil \log_2(k\ell) \rceil$ automata B implement a binary counter which is incremented at each substep, and goes to 0 when $x_B \geq k\ell - 1$;
- n automata D iterate the circuit C if $x_B < \ell - 1$, else go to state \tilde{x} (no change); and
- 1 automaton R records whether a state 1 appears in the i -th bit of x_D , and flips when the counter is equal to $\ell - 1$.

The reasoning is identical to the case $k = 1$, except that the counter needs k times ℓ substeps, i.e., k steps, in order to go back to its initial value. As a consequence, there is no x and $k' < k$ such that $f_{\{\mu\}}^{k'}(x) = x$, and the dynamics has no limit cycle of length smaller than k . Remark that when the **Iter-CVP** instance is positive, configurations $(0^{q_{k_n}} B_i \tilde{x} 0)_{i \in \llbracket k \rrbracket}$ with B_i the ℓ' -bits encoding of $i\ell$ form one of the limit cycles of length k . Also remark that the encoding of k in binary within the input has no consequence, neither on the PSPACE algorithm, nor on the polynomial time many-one reduction. ◀

Remark that our construction also applies to the notion of limit cycle x^0, \dots, x^{p-1} where it is furthermore required that all configurations are different (this corresponds to having the minimum length p): the problem is still PSPACE-complete.

3.3 Reachability and general complexity bounds

In this part, we settle the computational complexity of the classical reachability problem, which is unsurprisingly still PSPACE-hard by reduction from another model of computation (see Appendix A for details). In light of what precedes, one may be inclined to think that any problem related to the dynamics of automata networks under block-parallel update schedules is PSPACE-hard. We prove that this is partly true with a general complexity bound theorem on subdynamics existing within $f_{\{\mu\}}$, based on our previous results on fixed points and limit cycles. However, we will also prove that a Rice-like complexity lower bound analogous to the main results of [18], i.e., which would state that any non-trivial question on the dynamics (on the functional graph of $f_{\{\mu\}}$) expressible in first order logics is PSPACE-hard, does not hold (unless a collapse of PSPACE to the first level of the polynomial hierarchy). Indeed, we will see that deciding the bijectivity $(\forall x, y \in \mathbb{B}^n : f_{\{\mu\}}(x) = f_{\{\mu\}}(y) \implies x = y)$ is complete for coNP. We conclude the section with a discussion on reversible dynamics.

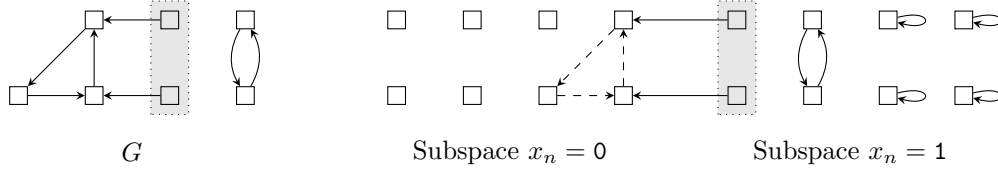
From the fixed point and limit cycle theorems in Section 3.2, we now derive that any particular subdynamics is hard to identify within $f_{\{\mu\}}$ under block-parallel update schedule. A functional graph is a directed graph of out-degree exactly one, and we assimilate $f_{\{\mu\}}$ to its functional graph. We define a family of problems, one for each functional graph G to find as a subgraph of $f_{\{\mu\}}$, and prove that the problem is always PSPACE-hard. Since $\text{PSPACE} = \text{coPSPACE}$, checking the existence of a subdynamics is as hard as checking the absence of a subdynamics, even though the former is a local property whereas the latter is a global property at the dynamics scale. This is understandable in regard of the fact that PSPACE scales everything to the global level (one can search the whole dynamics in PSPACE), because verifying that a given set of configurations (a certificate) gives the subgraph G is difficult (Theorem 6).

Block-parallel G as subdynamics (**BP-Subdynamics- G**)

Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in \llbracket n \rrbracket}$ as circuits, $\mu \in \text{BP}_n$.

Question: does $G \sqsubset f_{\{\mu\}}$?

19:10 Complexity of Boolean Automata Networks Under Block-Parallel Update Modes



■ **Figure 3** Construction of g in the proof of Theorem 8. Subspace $x_n = 0$ contains a copy of f with a potential limit cycle dashed. Subspace $x_n = 1$ implements G' , and wires configurations of U (grey area) to the potential limit cycle in the copy of f (remaining configurations are fixed points).

Remark that asking whether G appears as a subgraph or as an induced subgraph makes no difference when G is functional (has out-degree exactly one), because $f_{\{\mu\}}$ is also a functional graph: it is necessarily induced since there is no arc to delete.

► **Theorem 8.** *BP-Subdynamics- G is PSPACE-complete for any functional graph G .*

Proof. A polynomial space algorithm for **BP- G -Subdynamics** consists in enumerating all subsets $S \subseteq \mathbb{B}^n$ of size $|S| = |V(G)|$, and test for each whether the restriction of $f_{\{\mu\}}$ to S is isomorphic to G (functional graphs are planar hence isomorphism can be decided in logarithmic space [10]).

For the PSPACE-hardness, the idea is to choose a fixed point or limit cycle in G , and make it the decisive element whose existence or not lets G be a subgraph of the dynamics or not. Since G is a functional graph, it is composed of fixed points and limit cycles, with hanging trees rooted into them (the trees are pointing towards their root). Let $G(v)$ denote the unique out-neighbor of $v \in V(G)$.

Let us first assume that G has a limit cycle of length $k \geq 2$, or a fixed point with a tree of height greater or equal to 1 hanging (the case where G has only isolated limit cycles is treated thereafter). A fixed point is assimilated to a limit cycle of length $k = 1$. Let G' be the graph G without this limit cycle of size k , and let U be the vertices of G' without out-neighbor (if $k = 1$ then $U \neq \emptyset$). We reduce from **Iter-CVP**, and first compute the f, μ of size n obtained by the reduction from Theorem 7 for the problem **BP-Limit-Cycle- k** . We have that $f_{\{\mu\}}$ has a limit cycle of length k on configurations $(0^{qk_n} B_i \tilde{x} 0)_{i \in \llbracket k \rrbracket}$ (or configuration $0^{qk_n} 0^\ell \tilde{x} 0$ for $k = 1$) if and only if the **Iter-CVP** instance is positive.

We construct g on $n + 1$ automata, and the update schedule μ' being the union of μ with a singleton o-block for the new automaton. We assume that $n \geq |V(G)| - k$, otherwise we pad f, μ to that size (with identity local functions for the new automata). The idea is that g will consist in a copy of f on the subspace $x_n = 0$, and a copy of G' on the subspace $x_n = 1$ where the images of the configurations corresponding to the vertices of U will be configurations of the potential limit cycle of $f_{\{\mu\}}$ (in the other subspace $x_n = 0$). Other configurations in the subspace $x_n = 1$ will be fixed points. Figure 3 illustrates the construction. Recall that G is fixed, and consider a mapping $\alpha : V(G) \rightarrow \{0, 1\}^n$ such that vertices of the limit cycle of length k are sent to the configurations $(0^{qk_n} B_i \tilde{x} 0)_{i \in \llbracket k \rrbracket}$ respectively (or $0^{qk_n} 0^\ell \tilde{x} 0$ for $k = 1$). We define:

$$g(x) = \begin{cases} f(x_{\llbracket n \rrbracket})0 & \text{if } x_n = 0, \\ \alpha(G(v))0 & \text{if } x_n = 1 \text{ and } \exists v \in U : \alpha(v) = x_{\llbracket n \rrbracket}, \\ \alpha(G(v))1 & \text{if } x_n = 1 \text{ and } \exists v \in G' \setminus U : \alpha(v) = x_{\llbracket n \rrbracket}, \\ x & \text{otherwise.} \end{cases}$$

The obtained dynamics $g_{\{\mu'\}}$ has one copy of $f_{\{\mu\}}$ (in subspace $x_n = 0$), with a copy of G' (in subspace $x_n = 1$) which becomes a copy of G if configurations $(0^{qk_n} B_i \tilde{x} 0)_{i \in \llbracket k \rrbracket}$ (or

$0^{q_{k_n}} 0^{\ell} \tilde{x} 0$ in the case $k = 1$) form a limit cycle of length k . Moreover, it becomes a copy of G only if so by our assumption on the limit cycle or fixed point of G , because the remaining configurations in subspace $x_n = 1$ are all isolated fixed points. This concludes the reduction.

For the case where G is made of k isolated fixed points, we reduce from **BP-Fixed-Point** and construct an automata network with k copies of the dynamics of f , by adding $\lceil \log_2(k) \rceil$ automata with identity local functions. ◀

When the property of being a functional graph is dropped, that is when the out-degree of G is at most one (otherwise any instance is trivially negative), problem **BP-Subdynamics-G** is subtler. Indeed, one can still ask for the existence of fixed points, limit cycles and any functional subdynamics PSPACE-complete by Theorem 8, but new problems arise, some of which are provably complete only for coNP. The symmetry of existence versus non existence is broken. In what follows, we settle that deciding the bijectivity of $f_{\{\mu\}}$ is coNP-complete, and then discuss the complexity of decision problems which are subsets of bijective networks, such as the problem of deciding whether $f_{\{\mu\}}$ is the identity. We conclude the section by proving that it is nevertheless PSPACE-complete to decide whether $f_{\{\mu\}}$ is a constant map. These results hint at the subtleties behind a full characterization of the computational complexity of **BP-Subdynamics-G** for all graphs of out-degree at most one.

Block-parallel bijectivity (**BP-Bijectivity**)
 Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in \llbracket n \rrbracket}$ as circuits, $\mu \in \text{BP}_n$.
 Question: is $f_{\{\mu\}}$ bijective?

Remark that, because the space of configurations is finite, injectivity, surjectivity and bijectivity are equivalent properties of $f_{\{\mu\}}$.

► **Lemma 9.** *Let $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ a BAN and $\mu \in \text{BP}_n$ a block-parallel update mode. Then $f_{\{\mu\}}$ is bijective if and only if $f_{(W)}$ is bijective for every block W of $\varphi(\mu)$.*

Proof. The right to left implication is obvious since $f_{\{\mu\}}$ is a composition of bijections $f_{(W)}$. We prove the contrapositive of the left to right implication, assuming the existence of a block W in $\varphi(\mu)$ such that $f_{(W)}$ is not bijective. Let W_ℓ be the first such block in the sequence $\varphi(\mu)$, so there exist $x, y \in \mathbb{B}^n$ such that $x \neq y$ but $f_{(W_\ell)}(x) = f_{(W_\ell)}(y) = z$. By minimality of ℓ , the composition $g = f_{(W_{\ell-1})} \circ \dots \circ f_{(W_0)}$ is bijective, hence there also exist $x', y' \in \mathbb{B}^n$ with $x' \neq y'$ such that $g(x') = x$ and $g(y') = y$. That is, after the ℓ -th substep the two configurations x' and y' have the same image z , and we conclude that $f_{\{\mu\}}(x') = f_{\{\mu\}}(y') = f_{(W_{\ell-1})} \circ \dots \circ f_{(W_{\ell+1})}(z)$ therefore $f_{\{\mu\}}$ is not bijective. ◀

Lemma 9 shows that bijectivity can be decided at the local level of circuits (not iterated), which can be checked in coNP and gives Theorem 10.

► **Theorem 10.** **BP-Bijectivity** is coNP-complete.

Proof. A coNP algorithm can be established from Lemma 9, because it is equivalent to check the bijectivity at all substeps. A non-deterministic algorithm can guess a temporality $t \in \llbracket |\varphi(\mu)| \rrbracket$ (in binary) within the substeps, two configurations x, y , and then check in polynomial time that they certify the non-bijectivity of that substep as follows. First, construct W the t -th block of $\varphi(\mu)$, by computing t modulo each o-block size to get the automata from that o-block. Second, check that $f_{(W)}(x) = f_{(W)}(y)$.

The coNP-hardness is a direct consequence of that complexity lower bound for the particular case of the parallel update schedule [35, Theorem 5.17]. ◀

19:12 Complexity of Boolean Automata Networks Under Block-Parallel Update Modes

We now turn our attention to the recognition of identity dynamics.

Block-parallel identity (**BP-Identity**)
 Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in [n]}$ as circuits, $\mu \in \text{BP}_n$.
 Question: does $f_{\{\mu\}}(x) = x$ for all $x \in \mathbb{B}^n$?

This problem is in PSPACE, and is coNP-hard by reduction from the same problem in the parallel case [35, Theorem 5.18]. However, it is neither obvious to design a coNP-algorithm to solve it, nor to prove PSPACE-hardness by reduction from **Iter-CVP**.

► **Open problem 11.** **BP-Identity** is coNP-hard and in PSPACE. For which complexity class is it complete?

A major obstacle to the design of an algorithm, or of a reduction from **Iter-CVP** to **BP-Identity**, lies in the fact that, by Theorem 10, “hard” instances of the latter are bijective networks (because non-bijective instances can be recognized in our immediate lower bound coNP, and they are all negative instances of **BP-Identity**). A reduction would therefore be related to the lengths of cycles in the dynamics of substeps, and whether they divide the least common multiple of α -block sizes (for $x \in \mathbb{B}^n$ such that $f(x) = x$) or not ($f(x) \neq x$).

Nonetheless, we are able to prove another lower bound, related to the hardness of computing the number of models of a given propositional formula. The canonical ModP-complete problem takes as input a formula ψ and two integers k, i encoded in unary, and consists in deciding whether the number of models of ψ is congruent to k modulo the i -th prime number (which can be computed in polytime). It generalizes classes Mod $_k$ P (such as the parity case Mod $_2$ P = \oplus P), and it is notable that #P polytime truth-table reduces to ModP [29].

► **Theorem 12.** **BP-Identity** is ModP-hard (for polytime many-one reduction).

Proof. Given a formula ψ on n variables, m and i in unary, we apply Lemma 4 to construct g_n, μ_n on automata set $P = \llbracket q_{k_n} \rrbracket$. Automata from P have identity local functions, and the number of substeps is $\ell = |\varphi(\mu_n)| > 2^n$. Let p_i be the i -th prime number. We add:

- $\ell' = \lceil \log_2(\ell) \rceil$ automata numbered $B = \{q_{k_n}, \dots, q_{k_n} + \ell' - 1\}$, implementing a ℓ' bits binary counter that increments modulo ℓ at each substep, except for configurations with a counter greater or equal to ℓ which are left unchanged.
- $\ell'' = \lceil \log_2(p_i) \rceil$ automata numbered $R = \{q_{k_n} + \ell', \dots, q_{k_n} + \ell' + \ell'' - 1\}$, whose local functions are:

$$f_R(x) = \begin{cases} x_R - m + 1 \pmod{p_i} & \text{if } x_B = 0 \text{ and } x_B \text{ satisfies } \psi \\ x_R - m \pmod{p_i} & \text{if } x_B = 0 \text{ and } x_B \text{ does not satisfy } \psi \\ x_R + 1 \pmod{p_i} & \text{if } 0 < x_B < 2^n \text{ and } x_B \text{ satisfies } \psi \\ x_R & \text{otherwise.} \end{cases}$$

We also add singletons to μ_n for each of these additional automata, with $\mu' = \mu_n \cup \bigcup_{j \in B \cup R} \{(j)\}$. The resulting dynamics of $f_{\{\mu'\}}$ proceeds as follows.

Configurations x such that $x_B \geq \ell$ verify $f_{\{\mu'\}}(x) = x$, because all local functions are identities in this case. For configurations x such that $x_B < \ell$, during the dynamics of substeps from x to $f_{\{\mu'\}}(x)$, the counter x_B takes exactly once the values from 0 to $\ell - 1$, with $f_{\{\mu'\}}(x)_B = x_B$ (it goes back to its initial value). Meanwhile, at each substep with $x_B < 2^n$, the record of automata R is incremented if and only if x_B satisfies ψ , with a subtraction of m when $x_B = 0$. Since $\ell > 2^n$ each valuation of ψ is checked exactly once,

and x_R gets added the number of models of ψ minus m , modulo p_i (when $2^n \leq x_B < \ell$ automata R are left unchanged). Consequently, we have $f_{\{\mu'\}}(x)_R = x_R$ if and only if it has been incremented m times modulo p_i , i.e., f, μ' is a positive instance of **BP-Identity** if and only if ψ, m, i is a positive instance of **Mod-SAT** (the number of models of ψ is congruent to k modulo p_i). ◀

Our attempts to prove PSPACE-hardness failed, for the following reasons. To get bijective circuits one could reduce from reversible Turing machines (RTM) and problem **Reversible Linear Space Acceptance** [31]. A natural strategy would be to simulate a RTM for an exponential number of substeps, and then simulate it backwards for that same number of substeps, while ending in the exact same configuration (identity map) if and only if the simulation did not halt or was not in the orbit of the given input w . The difficulty with this approach is that the dynamics of substeps must not be the identity map when a *conjunction* of two temporally separated events happens: first that the simulation has halted, and second that the starting configuration was w . It therefore requires to remember at least one bit of information, which is subtle in the reversible setting. Indeed, the constructions of [31] and [34] consider only starting configurations of the Turing machine in the initial state and with blank tapes. However, in the context of Boolean automata networks, any configuration must be considered (hence any configuration of the simulated Turing machine).

Regarding iterated circuits simulating reversible cellular automata (for which the whole configuration space is usually considered), the literature focuses on decidability issues [27, 41], but a recent contribution fits our setting and we derive the following. $\text{FP}^{\text{PSPACE}}$ is the class of functions computable in polynomial time with an oracle in PSPACE.

▶ **Theorem 13** ([15, Theorem 5.7]). *There is a one-dimensional reversible cellular automaton for which simulating any given number of iterations, with periodic boundary conditions, is complete for $\text{FP}^{\text{PSPACE}}$*

▶ **Corollary 14.** *Given $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in \llbracket n \rrbracket}$ as circuits, $\mu \in \text{BP}_n$ such that $f_{\{\mu\}}$ is bijective, $x \in \mathbb{B}^n$ and $t \in \llbracket |\varphi(\mu)| \rrbracket$ in binary, computing the configuration at the t -th substep is complete for $\text{FP}^{\text{PSPACE}}$.*

Proof. For a fixed reversible cellular automaton (of any dimension), given a configuration of size n and a time t , one can compute in polynomial time a block-parallel update schedule μ and circuits for the local functions of a Boolean automata network of large enough size (to encode the CA's state space in binary), such that:

- $|\varphi(\mu)| > t$ (by Lemma 4; these automata are left aside with identity local functions),
- one substep of $f_{\{\mu\}}$ simulates one step of the CA; and
- $f_{\{\mu\}}$ is bijective (because the CA is reversible, padding with identity).

This gives a functional Turing many-one reduction from Theorem 13. ◀

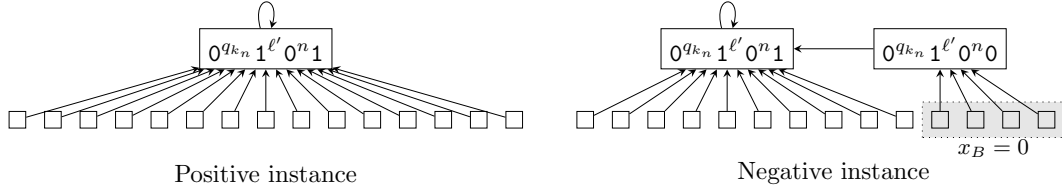
Intuitively, the dynamics of substeps embeds complexity. The relationship to the complexity of computing the configuration after the whole step composed of $|\varphi(\mu)|$ substeps (image through $f_{\{\mu\}}$), in order to reach **BP-Identity**, is not obvious.

Being a constant map is another global property of the dynamics, which turns out to be PSPACE-complete to recognize for BANs under block-parallel update schedules.

Block-parallel constant (**BP-Constant**)

Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in \llbracket n \rrbracket}$ as circuits, $\mu \in \text{BP}_n$.

Question: does there exist $y \in \mathbb{B}^n$ such that $f_{\{\mu\}}(x) = y$ for all $x \in \mathbb{B}^n$?



■ **Figure 4** Illustration of the dynamics obtained for the reduction to **BP-Constant** in the proof of Theorem 15. Configurations x with the counter automata B initialized to $x_B = 0$ either go to $0^{q_{k_n}} 1^{\ell'} 0^n 1$ (left, positive instance), or to $0^{q_{k_n}} 1^{\ell'} 0^n 0$ (right, negative instance). Only the bit of automata R changes.

▶ **Theorem 15.** **BP-Constant** is PSPACE-complete.

Proof. To decide **BP-Constant**, one can simply enumerate all configurations and compute their image (Theorem 6) while checking that it always gives the same result.

For the PSPACE-hardness proof, we reduce from **Iter-CVP**. Given a circuit $C : \mathbb{B}^n \rightarrow \mathbb{B}^n$, a configuration \tilde{x} and $i \in \llbracket n \rrbracket$, we apply Lemma 4 to construct g_n, μ_n on automata set $P = \llbracket q_{k_n} \rrbracket$. Automata from P have constant 0 local functions, and the number of substeps is $\ell = |\varphi(\mu_n)| > 2^n$. We add (Figure 4 illustrates the obtained dynamics):

- $\ell' = \lceil \log_2(\ell) \rceil$ automata numbered $B = \{q_{k_n}, \dots, q_{k_n} + \ell' - 1\}$, implementing a ℓ' -bits binary counter that increments at each substep, and sets all automata from B in state 1 when the counter is greater or equal to $\ell - 1$;
- n automata numbered $D = \{q_{k_n} + \ell', \dots, q_{k_n} + \ell' + n - 1\}$, whose local functions are given below; and
- 1 automaton numbered $R = \{q_{k_n} + \ell' + n\}$, whose local function is given below.

$$f_D(x) = \begin{cases} C(\tilde{x}) & \text{if } x_B = 0 \\ C(x_D) & \text{if } 0 < x_B < \ell - 1 \\ 0^n & \text{otherwise} \end{cases} \quad f_R(x) = \begin{cases} \tilde{x}_i & \text{if } x_B = 0 \\ x_R \vee x_{q_{k_n} + \ell' + i} & \text{if } 0 < x_B < \ell \\ 1 & \text{otherwise} \end{cases}$$

We also add singletons to μ_n for these additional automata, via $\mu' = \mu_n \cup \bigcup_{j \in B \cup D \cup R} \{(j)\}$.

For any configuration x with a counter not initialized to 0, i.e., with $x_B \neq 0$, the counter will reach and remain in the all 1 state before the last substep, therefore automata from D will be updated to 0^n and automaton R will be updated to 1. We conclude that $f_{\{\mu'\}}(x) = 0^{q_{k_n}} 1^{\ell'} 0^n 1$. For configurations x with $x_B = 0$, substeps proceed as follows:

- automata B count until $\ell - 1$ at the penultimate substep (recall that $\ell = |\varphi(\mu_n)| = |\varphi(\mu'_n)|$), which finally brings them all in state 1 during the last substep;
- automata D iterate the circuit C , starting from $C(\tilde{x})$ during the first substep; and
- automaton R records whether a 1 appears or not in the whole orbit of \tilde{x} (recall that $\ell = |\varphi(\mu'_n)| > 2^n$), starting from \tilde{x} itself during the first substep (even though $x_D \neq \tilde{x}$) and without encountering the “1 otherwise” case.

We conclude that the image of x on automata P is $0^{q_{k_n}}$, on B is $1^{\ell'}$, on D is 0^n , and on R it depends whether the **Iter-CVP** instance is positive (automaton R in state 1) or negative (automaton R in state 0). This completes the reduction: the image is always $0^{q_{k_n}} 1^{\ell'} 0^n 1$ if and only if the **Iter-CVP** instance is positive. ◀

4 Conclusion and perspectives

Block-sequential update schedules have a number of substeps limited by the fact that every automaton is updated only once. Block-parallel update schedules overcome this restriction, thus significantly raising the n (number of automata) upper bound for the number of substeps (Lemma 4 gives a backbone construction with more than 2^n substeps). This greatly increases the expressiveness of block-parallel dynamics, and we have demonstrated that this gain in computational power comes along with higher complexity costs. A fundamental point is that computing a single transition becomes PSPACE-hard in this context (Theorem 6), whereas it is feasible in polynomial time for all block-sequential update schedules [37]. We derive multiple consequences on the PSPACE-completeness of classical decision problems related to the existence of preimages, fixed points, limit cycles, and the recognition of constant dynamics. These problems are NP-complete (existence problems), or coNP-complete (global dynamical properties) for block-sequential modes (see [35]), hence one might be tempted to extrapolate to the following conjecture, which is false (unless a drastic complexity collapse).

► **Conjecture 16 (false).** *If a problem is NP-hard or coNP-hard and in PSPACE for block-sequential update schedules then it is PSPACE-complete for block-parallel update schedules.*

The recognition of bijective dynamics disproves Conjecture 16: according to Lemma 9, a single substep is necessary and sufficient to break the bijectivity of the automata network's dynamics, hence bringing the question to the circuit level (of substeps), in coNP. It also prevents to level the Rice-like complexity lower bound theorem presented in [18], to PSPACE-hardness. Recognition problems are nonetheless still NP-hard or coNP-hard for non-trivial first order questions, because parallel is a particular case of block-parallel.

The reachability problem, which is PSPACE-complete for block-sequential modes, remains PSPACE-complete for block-parallel modes (Theorem 18). Intuitively, on the one hand the idea of reachability can be embedded in a single transition step of block-parallel update, because it may have an exponential number of substeps. On the other hand, the sequence of reachability problems at the level of substeps combines into a reachability problem at the level of steps which is still in PSPACE.

The recognition of identity dynamics is not fully characterized (Open problem 11 and Theorem 12). A fine interplay between computing in a reversible setting (since non-bijective dynamics can be identified in NP) and the length of limit cycles in the dynamics of substeps (to loop back to the starting configuration and be the identity map) is still to be discovered. Computing the interaction graph (feasible in DP, just above NP and coNP) may give some insights but, contrary to block-sequential modes having identity dynamics if and only if the interaction graph is made of n positive loops, it is possible to design more complex identity dynamics under block-parallel update schedules.

After determining the complexity of recognizing preimages, image points or fixed points in Subsection 3.2, the next logical step would be the complexity of counting them. This is not an easy step to make from the constructions presented in the present work, which are not parcimonious (for the definition of #PSPACE, see [30]).

An important remark for the community is that, while all the proofs in this paper were written with Boolean automata networks in mind, the results also hold for non-Boolean automata networks.

Another avenue of research could be questions about the existence of a block-parallel update schedules verifying a certain property, as in [8, 6] for block-sequential update schedules. Given that the fixed point invariance is broken under block-parallel update schedules, it opens the way for more questions. The ability to create new fixed points (how and when does it happen?) is in itself a meaningful track of research.

References

- 1 T. Akutsu, S. Miyano, and S. Kuhara. Identification of genetic networks from a small number of gene expression patterns under the Boolean network model. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 17–28. World Scientific, 1999.
- 2 N. Alon. Asynchronous threshold networks. *Graphs and Combinatorics*, 1:305–310, 1985. doi:10.1007/BF02582959.
- 3 J. Aracena, J. Demongeot, É. Fanchon, and M. Montalva. On the number of different dynamics in Boolean networks with deterministic update schedules. *Mathematical Biosciences*, 242:188–194, 2013. doi:10.1016/j.mbs.2013.01.007.
- 4 J. Aracena, É. Fanchon, M. Montalva, and M. Noual. Combinatorics on update digraphs in Boolean networks. *Discrete Applied Mathematics*, 159:401–409, 2011.
- 5 J. Aracena, E. Goles, A. Moreira, and L. Salinas. On the robustness of update schedules in Boolean networks. *Biosystems*, 97:1–8, 2009.
- 6 J. Aracena, L. Gómez, and L. Salinas. Limit cycles and update digraphs in Boolean networks. *Discrete Applied Mathematics*, 161:1–12, 2013.
- 7 A. Benecke. Chromatin code, local non-equilibrium dynamics, and the emergence of transcription regulatory programs. *The European Physical Journal E*, 19:353–366, 2006.
- 8 F. Bridoux, C. Gaze-Maillet, K. Perrot, and S. Sené. Complexity of Limit-Cycle Problems in Boolean Networks. In *Proceedings of SOFSEM'2021*, volume 12607 of *LNCS*, pages 135–146. Springer, 2021.
- 9 F. Bridoux, P. Guillon, K. Perrot, S. Sené, and G. Theyssier. On the cost of simulating a parallel boolean automata network by a block-sequential one. In *Proceedings of TAMC'2017*, volume 10185 of *LNCS*, pages 112–128, 2017. doi:10.1007/978-3-319-55911-7_9.
- 10 S. Datta, N. Limaye, P. Nimbhorkar, T. Thierauf, and F. Wagner. Planar Graph Isomorphism is in Log-Space. In *Algebraic Methods in Computational Complexity*, volume 9421, pages 1–32. Schloss Dagstuhl, 2010. doi:10.4230/DagSemProc.09421.6.
- 11 J. Demongeot, A. Elena, and S. Sené. Robustness in regulatory networks: a multi-disciplinary approach. *Acta Biotheoretica*, 56:27–49, 2008.
- 12 J. Demongeot and S. Sené. About block-parallel Boolean networks: a position paper. *Natural Computing*, 19:5–13, 2020.
- 13 A. Dennunzio, E. Formenti, L. Manzoni, and A. E. Porreca. Complexity of the dynamics of reaction systems. *Information and Computation*, 267:96–109, 2019.
- 14 B. Elspas. The theory of autonomous linear sequential networks. *IRE Transactions on Circuit Theory*, 6:45–60, 1959.
- 15 David Eppstein. The Complexity of Iterated Reversible Computation. *TheoretCS*, 2, 2023. doi:10.46298/theoretics.23.10.
- 16 B. Fierz and M. G. Poirier. Biophysics of chromatin dynamics. *Annual Review of Biophysics*, 48:321–345, 2019.
- 17 P. Floréen and P. Orponen. On the computational complexity of analyzing Hopfield nets. *Complex Systems*, 3:577–587, 1989.
- 18 G. Gamard, P. Guillon, K. Perrot, and G. Theyssier. Rice-like theorems for automata networks. In *Proceedings of STACS'21*, volume 187 of *LIPICs*, pages 32:1–32:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 19 C. E. Giacomantonio and G. J. Goodhill. A Boolean model of the gene regulatory network underlying mammalian cortical area development. *PLoS Computational Biology*, 6:e1000936, 2010.
- 20 E. Goles and S. Martínez. *Neural and automata networks: dynamical behavior and applications*, volume 58 of *Mathematics and Its Applications*. Kluwer Academic Publishers, 1990.
- 21 E. Goles, P. Montealegre, V. Salo, and I. Törmä. PSPACE-completeness of majority automata networks. *Theoretical Computer Science*, 609:118–128, 2016. doi:10.1016/j.tcs.2015.09.014.

- 22 E. Goles and M. Noul. Block-sequential update schedules and Boolean automata circuits. In *Proceedings of AUTOMATA'2010*, pages 41–50. DMTCS, 2010.
- 23 E. Goles and L. Salinas. Comparison between parallel and serial dynamics of Boolean networks. *Theoretical Computer Science*, 396:247–253, 2008.
- 24 J. C. Hanse and J. Ausio. Chromatin dynamics and the modulation of genetic activity. *Trends in Biochemical Sciences*, 17:187–191, 1992.
- 25 M. R. Hübner and D. L. Spector. Chromatin dynamics. *Annual Review of Biophysics*, 39:471–489, 2010.
- 26 D. A. Huffman. Canonical forms for information-lossless finite-state logical machines. *IRE Transactions on Information Theory*, 5:41–59, 1959.
- 27 J. Kari. Reversible cellular automata. In *Proceedings of DLT'2005*, volume 3572 of *LNCS*, pages 57–68. Springer, 2005. doi:10.1007/11505877_5.
- 28 S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22:437–467, 1969.
- 29 J. Köbler and S. Toda. On the power of generalized Mod-classes. *Mathematical Systems Theory*, 29:33–46, 1996. doi:10.1007/BF01201812.
- 30 R. E. Ladner. Polynomial Space Counting Problems. *SIAM Journal on Computing*, 18(6):1087–1097, 1989. doi:10.1137/0218073.
- 31 K.-J. Lange, P. McKenzie, and A. Tapp. Reversible Space Equals Deterministic Space. *Journal of Computer and System Sciences*, 60(2):354–367, 2000. doi:10.1006/jcss.1999.1672.
- 32 W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Journal of Mathematical Biophysics*, 5:115–133, 1943.
- 33 L. Mendoza and E. R. Alvarez-Buylla. Dynamics of the genetic regulatory network for *Arabidopsis thaliana* flower morphogenesis. *Journal of Theoretical Biology*, 193:307–319, 1998.
- 34 K. Morita. *Theory of Reversible Computing*. Springer, first edition, 2017. doi:10.1007/978-4-431-56606-9.
- 35 K. Perrot. *Études de la complexité algorithmique des réseaux d'automates*. Habilitation thesis, Université d'Aix-Marseille, 2022.
- 36 K. Perrot, S. Sené, and L. Tapin. Combinatorics of block-parallel automata networks. In *Proceedings of SOFSEM'24*, *LNCS*. Springer, 2024. To appear.
- 37 P. Perrotin and S. Sené. Turning block-sequential automata networks into smaller parallel networks with isomorphic limit dynamics. In *Proceedings of CiE'23*, *LNCS*. Springer, 2023. To appear.
- 38 F. Robert. *Discrete iterations: a metric study*, volume 6 of *Springer Series in Computational Mathematics*. Springer, 1986.
- 39 F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.
- 40 K. Sutner. On the Computational Complexity of Finite Cellular Automata. *Journal of Computer and System Sciences*, 50(1):87–97, 1995. doi:10.1006/jcss.1995.1009.
- 41 K. Sutner. The complexity of reversible cellular automata. *Theoretical Computer Science*, 325(2):317–328, 2004. doi:10.1016/j.tcs.2004.06.011.
- 42 R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42:563–585, 1973.
- 43 D. J. Wooten, S. M. Groves, D. R. Tyson, Q. Liu, J. S. Lim, R. Albert, C. F. Lopez, J. Sage, and V. Quaranta. Systems-level network modeling of small cell lung cancer subtypes identifies master regulators and destabilizers. *PLoS Computational Biology*, 15:e1007343, 2019.

A Omitted proofs

Proof of Lemma 2. By the prime number theorem, there are approximately $\frac{N}{\ln(N)}$ primes lower than N . As a consequence, distinct prime integers p_1, p_2, \dots, p_{k_n} with $k_n = \lfloor \frac{n^2}{\ln(n^2)} \rfloor$ can be computed in time $\mathcal{O}(n^2)$ using Atkin sieve algorithm. Since $2 \leq p_i < n^2$, we have $2^{k_n} \leq \prod_{i=1}^{k_n} p_i < n^{2k_n}$. It holds that $2^{k_n} = 2^{\lfloor \frac{n^2}{2 \ln(n)} \rfloor} > 2^n$, and $n^{2k_n} \leq n^{\frac{n^2}{\ln(n^2)}}$ with

$$\log_2 \left(n^{\frac{n^2}{\ln(n^2)}} \right) = \frac{\frac{n^2}{\ln(n^2)}}{\log_n(2)} = \frac{n^2}{\ln(2)}$$

meaning that $n^{2k_n} \leq 2^{\frac{n^2}{\ln(2)}} < 2^{2n^2}$. ◀

Block-parallel preimage (BP-Preimage)
 Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in [n]}$ as circuits, $\mu \in \text{BP}_n$, $y \in \mathbb{B}^n$.
 Question: does $\exists x \in \mathbb{B}^n : f_{\{\mu\}}(x) = y$?

► **Theorem 17.** **BP-Preimage** is PSPACE-complete.

The difficulty in this reduction is that we need to take into account the image of every configuration x . We modify the preceding construction by setting automata D to \tilde{x} when the counter B encodes 0.

Proof. The algorithm for **BP-Preimage** computes the image of each configuration (enumerated in polynomial space with a simple counter) using the same procedure as **BP-Step** (Theorem 6), and decides whether there is some x such that $f_{\{\mu\}}(x) = y$.

Given an instance $C : \mathbb{B}^n \rightarrow \mathbb{B}^n$, $\tilde{x} \in \mathbb{B}^n$, $i \in [n]$ of **Iter-CVP**, we construct the same block-parallel update schedule μ' as in the proof of Theorem 6, and modify the local functions of automata D and R as follows:

$$f_D(x) = \begin{cases} C(\tilde{x}) & \text{if } x_B = 0 \\ C(x_D) & \text{if } 0 < x_B < \ell - 1 \\ 0^n & \text{otherwise} \end{cases} \quad f_R(x) = \begin{cases} \tilde{x}_i & \text{if } x_B = 0 \\ x_R \vee x_{q_{k_n} + \ell' + i} & \text{otherwise} \end{cases}$$

The purpose is that D iterates the circuit from \tilde{x} when the counter is initialized to 0, and that R records whether the i -th bit of D has been in state 1 (including the initial substep). We set $y = 0^{q_{k_n}} 0^\ell 0^n 1$.

If the **Iter-CVP** instance is positive, then we have $f_{\{\mu'\}}(0^{q_{k_n}} 0^\ell 0^n 0) = y$ (automata B go back to $0^{q_{k_n}}$, automata D iterate circuit C from \tilde{x} and end in state 0^n , and automaton R has recorded that the i -th bit of D has been to state 1).

Conversely, if there is a configuration x such that $f_{\{\mu'\}}(x) = y$, then the automata from the counter B must have started in state $x_B = 0^{q_{k_n}}$, because of the increment modulo ℓ which is the number of substeps. We deduce that D iterate circuit C for the whole orbit of \tilde{x} and end in state 0^n , and that automaton R records the answer to the **Iter-CVP** instance. Since it ends in state $y_R = 1$ by our assumption that $f_{\{\mu'\}}(x) = y$, we conclude that it is positive. ◀

Block-parallel reachability (BP-Reachability)
 Input: $(f_i : \mathbb{B}^n \rightarrow \mathbb{B})_{i \in [n]}$ as circuits, $\mu \in \text{BP}_n$, $x, y \in \mathbb{B}^n$.
 Question: does $\exists t \in \mathbb{N} : f_{\{\mu\}}^t(x) = y$?

► **Theorem 18.** *BP-Reachability is PSPACE-complete.*

Proof. The problem belongs to PSPACE, because it can naively be solved by simulating the dynamics of $f_{\{\mu\}}$ starting from configuration x , for 2^n time steps.

Reachability problems in cellular automata and related models are known to be PSPACE-complete on finite configurations [40]. We reduce from the reachability problem for reaction systems, which can be seen as a particular case of Boolean automata networks, and is also known to be PSPACE-complete [13]. Given a reaction system (S, A) where S is a finite set of entities, and A is a set of reactions of the form (R, I, P) where R are the reactants, I the inhibitors and P the products, we construct the BAN of size $n = |S|$ with local functions:

$$\forall i \in \llbracket n \rrbracket : f_i(x) = \bigvee_{\substack{(R,I,P) \in A \\ \text{such that } i \in P}} \left(\bigwedge_{j \in R} x_j \wedge \bigwedge_{k \in I} \neg x_k \right).$$

A configuration $x \in \mathbb{B}^n$ of the BAN corresponds to a state of the reaction system with each automaton indicating the presence or absence of its corresponding entity. The parallel evolution of f (under μ_{par}) is in direct correspondance with the evolution of the reaction system. ◀

Space and Move-Optimal Arbitrary Pattern Formation on Infinite Rectangular Grid by Oblivious Robot Swarm

Avishek Sharma ✉ 

Department of Mathematics, Jadavpur University, India

Satakshi Ghosh ✉ 

Department of Mathematics, Jadavpur University, India

Pritam Goswami ✉ 

Department of Mathematics, Jadavpur University, India

Buddhadeb Sau ✉ 

Department of Mathematics, Jadavpur University, India

Abstract

Arbitrary Pattern Formation (APF) is a fundamental coordination problem in swarm robotics. It requires a set of autonomous robots (mobile computing units) to form an arbitrary pattern (given as input) starting from any initial pattern. This problem has been extensively investigated in continuous and discrete scenarios, with this study focusing on the discrete variant. A set of robots is placed on the nodes of an infinite rectangular grid graph embedded in the euclidean plane. The movements of each robot is restricted to one of the four neighboring grid nodes from its current position. The robots are autonomous, anonymous, identical, and homogeneous, and operate Look-Compute-Move cycles. In this work, we adopt the classical *OBLLOT* robot model, meaning the robots have no persistent memory or explicit communication methods, yet they possess full and unobstructed visibility. This work proposes an algorithm that solves the APF problem in a fully asynchronous scheduler assuming the initial configuration is asymmetric. The considered performance measures of the algorithm are space and number of moves required for the robots. The algorithm is asymptotically move-optimal. Here, we provide a definition of space complexity that takes the visibility issue into consideration. We observe an obvious lower bound \mathcal{D} of the space complexity and show that the proposed algorithm has the space complexity $\mathcal{D} + 4$. On comparing with previous related works, we show that this is the first proposed algorithm considering *OBLLOT* robot model that is asymptotically move-optimal and has the least space complexity which is almost optimal.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed algorithms, Oblivious robots, Optimal algorithms, Swarm robotics, Space optimization, and Rectangular grid

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.20

Related Version *Full Version:* <https://doi.org/10.48550/arXiv.2309.11190> [14]

Funding The first and the third authors are supported by the University Grants Commission (UGC), India. The second author is supported by the West Bengal State Government Fellowship Scheme. The fourth author is supported by the Science and Engineering Research Board (SERB), India

1 Introduction

Swarm robotics involves a group of simple computing units referred to as *robots* that operate autonomously without having any centralized control. Moreover, the robots are generally anonymous (no unique identifier), homogeneous (all robots execute the same algorithm), and identical (physically indistinguishable). Generally on activation, a robot first takes a snapshot of its surroundings. This phase is called the LOOK phase. Then based on the snapshot an



© Avishek Sharma, Satakshi Ghosh, Pritam Goswami, and Buddhadeb Sau;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 20; pp. 20:1–20:17

Leibniz International Proceedings in Informatics



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

inbuilt algorithm determines a destination point. This phase is called the COMPUTE phase. Finally, in the MOVE phase it moves towards the computed destination. These three phases together are called a LOOK-COMPUTE-MOVE (LCM) cycle of a robot.

Through collaborative efforts, these robot swarms can accomplish different tasks such as gathering at a specific point, configuring into predetermined patterns, navigating networks, etc. Presently, the field of robotics research is witnessing significant enthusiasm for swarm robots. The inherent decentralized characteristics of these algorithms provide swarm robots with a notable advantage, as distributed algorithms are both easily scalable and more resilient in the face of errors. Furthermore, swarm robots boast a multitude of real-world applications, including but not limited to tasks like area coverage, patrolling, network maintenance, etc.

In order to accomplish specific tasks, robots require some computational capabilities, which can be determined by various factors such as memory, communication, etc. With respect to memory and communication, the literature identifies two primary robot models. The first one is called the classical *OBLLOT* model. In this model, the robots are devoid of persistent memory and communication abilities. Another robot model is the *LUMI* model where the robots are equipped with a finite number of lights that can take a finite number of different colors. These colors serve as persistent memory (as a robot can see its own color) and communication architecture (as the colors of lights are visible to all other robots). The responsibility for activating robots rests with an entity referred to as the *Scheduler*. Within the existing literature, three primary types of schedulers emerge: *Fully-Synchronous* (*FSYNC*), *Semi-Synchronous* (*SSYNC*), and *Asynchronous* (*ASYNC*). In the case of fully synchronous and semi-synchronous schedulers, time is partitioned into rounds of uniform length. The duration of the LOOK, COMPUTE, and MOVE phases for all activated robots are identical. Under a fully-synchronous scheduler, all robots become active at the onset of each round, but in a semi-synchronous setup, not all robots may activate simultaneously in a given round. In an asynchronous scheduler, round divisions are absent. At any given moment, a robot can be either idle or engaged in any of the LOOK, COMPUTE, or MOVE phases. The duration of these phases and the spans of robot idleness are finite but unbounded.

The primary focus of this study is to solve the Arbitrary Pattern Formation (APF) problem on an infinite rectangular grid while minimizing spatial utilization. The APF problem involves a group of robots situated within an environment, aiming to create a designated pattern. This pattern is conveyed to each robot as a set of points within a coordinate system as an input. This problem has been extensively studied in the euclidean plane ([2, 3, 4, 6, 7, 8, 16, 17]) and also on a continuous circle [13]. Bose et al. [1] first proposed this problem on a rectangular grid. The rectangular grid is a natural discretization of the plane. To the best of our knowledge, on the discrete domain, this problem has been studied in [1, 5, 9, 10, 11, 12, 15]. In this paper, the focus is placed on an environment characterized by an infinite rectangular grid. In the upcoming subsection, we delve into the reasons behind the introduction of spatial constraint in the context of this problem.

1.1 Motivation

In the majority of previous studies, the implementation of this problem on a grid necessitates a substantial allocation of space (space of a configuration formed by a set of robots is the dimension of the smallest enclosing square of the configuration), even when both the initial and target configurations have minimal spatial requirements. This promptly gives rise to a lot of problems. To begin with, in the scenario where the grid is of bounded dimensions, it is possible that certain patterns cannot be formed, even if robots are initially located within the bounded grid and the target pattern could potentially fit within the grid. This limitation

arises due to the existence of intermediate configurations that demand a spatial extent that cannot be accommodated within the confined grid. Moreover, when the spatial demand for an APF algorithm on a grid increases, the count of patterns that can be formed within a bounded grid becomes noticeably fewer compared to the count of patterns formable on the same grid with a lower space requirement. To be more specific, patterns that are “big enough” can not be formed if the space requirement is “big” on a bounded grid. So, the requirement of large space compromises better utilization of the space.

Moreover, even if complete visibility is entertained for theoretical considerations, this assumption does not hold practical validity within an unbounded environment. In the context of a bounded region, it can be applied with the premise that the environment is finite, and the entire environment falls within the visibility range of each robot. However, introducing the concept of an infinite grid disrupts this assumption. In situations where the grid lacks bounds, it is possible that due to substantial spatial requirements, certain robots might stray beyond the visibility range of others. To the best of our knowledge, there remains an absence of work that addresses the APF challenge within the constraints of limited visibility, an asynchronous scheduler, and the absence of any global coordinate agreement. Thus in this paper, the problem of APF on a grid with minimal spatial requirement has been considered.

1.2 Related Work

In the discrete setting, the problem is first studied in [1]. Here, the authors solved the problem deterministically on an infinite rectangular grid with *OBLLOT* robots in an asynchronous scheduler. Later in [5], the authors studied the problem on a regular tessellation graph. In [1], authors count the total required moves asymptotically and also give an asymptotic lower bound for the move complexity, i.e., total number of moves required to solve the problem. In [5], authors did not count the total number of moves required for their proposed algorithm. In [9], the authors provided two deterministic algorithms for solving the problem in an asynchronous scheduler. The first algorithm of [9] solves the APF problem for the *OBLLOT* model. The move complexity of this algorithm matches the asymptotic lower bound given in [1]. Thus, this algorithm is asymptotically move-optimal. The second algorithm of [9] solves the problem for the *LUMI* model, and this algorithm is asymptotically move-optimal. Further authors showed that the algorithm is time-optimal, i.e., the number of epochs (a time interval in which each robot activates at least once) to complete the algorithm is asymptotically optimal. In [11], the authors provided a deterministic algorithm for solving the problem with opaque (non-transparent) point robots in the *LUMI* model with an asynchronous scheduler assuming one-axis agreement. In [10], the authors proposed two randomized algorithms for solving the APF problem in an asynchronous scheduler. The second algorithm works for the *OBLLOT* model. This algorithm is asymptotically move-optimal and time-optimal. The randomization in this algorithm is only used to break any present symmetry in the initial configuration. If the initial configuration is asymmetric then the algorithm is deterministic. The first algorithm works for opaque point robots with the *LUMI* model. This algorithm is also asymptotically move-optimal and time-optimal. In [12], the authors solve the problem with opaque fat robots (robots having nontrivial dimension) with the *LUMI* model in an asynchronous scheduler assuming one-axis agreement. In [15], the authors provide an asymptotically move-optimal algorithm solving this problem with robots in the *LUMI* model. The work also considered a special requirement and showed that the algorithm is space-optimal. In the next section, we formally state the space complexity of an algorithm and discuss the space complexity of the mentioned works.

1.3 Space Complexity of APF Algorithms in Rectangular Grid

In [15], the authors considered the total space required to execute an algorithm. In Definition 1, we define the space complexity of an algorithm executed by a set of robots on a rectangular grid. Before that let's define the dimension of a rectangle, vertices of which are on some grid nodes, as $m \times n$ if the rectangle has m horizontal grid lines and n vertical grid lines.

► **Definition 1.** *The space complexity of an algorithm executed by a set of robots on a rectangular grid is the minimum dimension of the squares (whose sides are parallel with the grid lines) such that no robot steps out of the square throughout the execution of the algorithm.*

Let the smallest enclosing rectangle (SER), the sides of which are parallel to grid lines, of the initial configuration and pattern configuration formed by the robots, respectively, have dimensions $m \times n$ ($m \geq n$) and $m' \times n'$ ($m' \geq n'$). Let $\mathcal{D} = \max\{m, n, m', n'\}$. Then the minimum space complexity for an algorithm to solve the APF problem is \mathcal{D} . Definition 1 assigns a real number to the space complexity that makes it easy to compare different APF algorithms. But consider an APF algorithm that takes a space enclosed by an axis aligned rectangle of dimension $p \times q$. if $M = \max\{m, m'\}$ and $N = \max\{n, n'\}$, then the APF algorithm is better (as far as space is concerned) if p is closer to M and q is closer to N .

Space Complexity of the Previous APF Algorithms

(*OBLLOT model APF algorithms*) The algorithm proposed in [1] has space-complexity at least $2\mathcal{D}$ in the worst case as one of the leaders, named tail moves far away from the rest of the configuration. The first algorithm proposed in [9] is for the *OBLLOT* model. It requires the robots to form a compact line. The space complexity of these algorithms is \mathcal{D}^2 in the worst case. The second randomized algorithm in [10] is for the *OBLLOT* model. In this algorithm, the leader robot moves upwards far away from the rest of the configuration. Thus, it has a space complexity of at least $30\mathcal{D}$ in the worst case.

(*LUMI model APF algorithms*) The second algorithm proposed in [9] is for the *LUMI* model. This algorithm requires a step-looking configuration where each robot occupies a unique vertical line. Therefore, the space complexity of the algorithm can be \mathcal{D}^2 in the worst case. This algorithm needs each robot to have a light with three distinct colors. The first randomized algorithm in [10] for *LUMI* model has space-complexity at least $\mathcal{D} + 2$. The authors also did not count the number of lights and colors required for the robots. With a closer look, we observe that this algorithm uses at least 31 distinct colors. Further, deterministic APF algorithms proposed in [11, 12] solved it for obstructed visibility. These works also need the robots to form a compact line, hence the space complexity of these algorithms is \mathcal{D}^2 in the worst case. The proposed algorithm in [15] has space-complexity $\mathcal{D} + 1$ and it requires three distinct colors.

We say that the first algorithm proposed in [10] and algorithm proposed in [15] are *almost* space-optimal, as the space-complexity is of the form $\mathcal{D} + c$, \mathcal{D} is a lower bound of the space-complexity and c is a constant independent of \mathcal{D} . If we consider the rectangle to measure the space, then a rectangle of dimension $M \times N$ is minimally required to solve the APF problem. The first algorithm in [10] and the algorithm in [15] takes space enclosed by rectangle of dimension $(M + 2) \times (N + 2)$ and $(M + 1) \times N$ respectively. We can consider these algorithms as so far the best APF algorithms as far as space complexity is concerned. For the rest of the algorithms one dimension of the rectangle that encloses the required space shoots up twice (algorithm in [1]) or 30 times (2^{nd} algorithm in [10]) or squares (algorithm

in [9, 11, 12]). For the rectangle version, if an APF algorithm takes a space of enclosing rectangle of dimension $(M + c_1) \times (N + c_2)$, where c_1 and c_2 are constants independent of M and N , then the algorithm is said to be almost optimal. The challenge of this work is to reconfigure the (oblivious and silent) robots in an optimal space avoiding the occurrence of symmetric configurations and collision among robots while keeping the number of movements asymptotically optimal.

Our Contribution

First a deterministic algorithm for solving APF in an infinite discrete line is presented. Then exploiting that algorithm this manuscript presents a deterministic algorithm for solving APF in an infinite rectangular grid which is almost space-optimal as well as asymptotically move-optimal. Precisely, the space complexity for the algorithm is $\mathcal{D} + 4$ and this algorithm takes a space enclosing the rectangle of dimension $(M + 4) \times (N + 1)$. The move-complexity of the algorithm is $O(k\mathcal{D})^1$, where k is the number of robots. The robot model is the classical *OBLLOT* model and the scheduler is fully asynchronous. To the best of our knowledge so far, this is the first deterministic algorithm solving APF problem in the *OBLLOT* robot model that has the least space-complexity and optimal move-complexity (See Table 1 for comparison with the previous works). The architecture of the description of the algorithm and correctness proof are motivated from [1].

■ **Table 1** Comparison table.

Work	Model	Visibility	Deterministic/ Randomised	Space complexity
[1]	<i>OBLLOT</i>	Unobstructed	Deterministic	$\geq 2\mathcal{D}$
1 st algorithm in [9]	<i>OBLLOT</i>	Unobstructed	Deterministic	\mathcal{D}^2
2 nd algorithm in [10]	<i>OBLLOT</i>	Unobstructed	Randomised ²	$\geq 30\mathcal{D}$
2 nd algorithm in [9]	<i>LUMI</i>	Unobstructed	Deterministic	\mathcal{D}^2
1 st algorithm in [10]	<i>LUMI</i>	Obstructed	Randomised	$\geq \mathcal{D} + 2$
[11]	<i>LUMI</i>	Obstructed	Deterministic	\mathcal{D}^2
[12]	<i>LUMI</i>	Obstructed (fat robot)	Deterministic	\mathcal{D}^2
[15]	<i>LUMI</i>	Unobstructed	Deterministic	$\mathcal{D} + 1$
Algorithm in this work	<i>OBLLOT</i>	Unobstructed	Deterministic	$\mathcal{D} + 4$

2 Model and Problem Statement

Robot

The robots are assumed to be identical, anonymous, autonomous, and homogeneous. Robots are oblivious, i.e., they do not have any persistent memory to remember previous configurations or past actions. Robots do not have any explicit means of communication with other robots. The robots are modeled as points on an infinite rectangular grid graph embedded on

¹ In [10], the authors provides this tight lower bound

² The randomisation is only used to break any symmetry present in the initial configuration

a plane. Initially, robots are positioned on distinct grid nodes. A robot chooses the local coordinate system such that the axes are parallel to the grid lines and the origin is its current position. Robots do not agree on a global coordinate system. The robots do not have a global sense of clockwise direction. A robot can only rest on a grid node. Movements of the robots are restricted to the grid lines, and through a movement, a robot can choose to move to one of its four adjacent grid nodes.

Look-Compute-Move Cycle

A robot has two states: sleep/idle state and active state. On activation, a robot operates in Look-Compute-Move (LCM) cycles, which consist of three phases. In the Look phase, a robot takes a snapshot of its surroundings and gets the position of all the robots. We assume that the robots have full, unobstructed visibility. In the Compute phase, the robots run an inbuilt algorithm that takes the information obtained in the Look phase and obtains a position. The position can be its own or any of its adjacent grid nodes. In the Move phase, the robot either stays still or moves to the adjacent grid node as determined in the Compute phase.

Scheduler

The robots work asynchronously. There is no common notion of time for robots. Each robot independently gets activated and executes its LCM cycle. The time length of LCM cycles, Compute phases, and Move phases of robots may be different. Even the length of two LCM cycles for one robot may be different. The gap between two consecutive LCM cycles, or the time length of an LCM cycle for a robot, is finite but can be unpredictably long. We consider the activation time and the time taken to complete an LCM cycle to be determined by an adversary. In a fair adversarial scheduler, a robot gets activated infinitely often.

Grid Terrain and Configurations

Let \mathcal{G} be an infinite rectangular grid graph embedded on \mathbb{R}^2 . The \mathcal{G} can be formally defined as a geometric graph embedded on a plane as $\mathcal{P} \times \mathcal{P}$, which is the cartesian product of two infinite (from both ends) path graphs \mathcal{P} . Suppose a set of $k > 2$ robots is placed on \mathcal{G} . Let f be a function from the set of vertices of \mathcal{G} to $\mathbb{N} \cup \{0\}$, where $f(v)$ is the number of robots on the vertex v of \mathcal{G} . Then the pair (\mathcal{G}, f) is said to be a *configuration* of robots on \mathcal{G} . For the initial configuration (\mathcal{G}, f) , we assume $f(v) \leq 1$ for all v .

Symmetries

Let (\mathcal{G}, f) be a configuration. A *symmetry* of (\mathcal{G}, f) is an automorphism ϕ of the graph \mathcal{G} such that $f(v) = f(\phi(v))$ for each node v of \mathcal{G} . A symmetry ϕ of (\mathcal{G}, f) is called *trivial* if ϕ is an identity map. If there is no non-trivial symmetry of (\mathcal{G}, f) , then the configuration (\mathcal{G}, f) is called an *asymmetric* configuration and otherwise a *symmetric* configuration. Note that any automorphism of $\mathcal{G} = \mathcal{P} \times \mathcal{P}$ can be generated by three types of automorphisms, which are translations, rotations, and reflections. Since there are only a finite number of robots, it can be shown that (\mathcal{G}, f) cannot have any translation symmetry. Reflections can be defined by an axis of reflection that can be horizontal, vertical, or diagonal. The angle of rotation can be of 90° or 180° , and the center of rotation can be a grid node, the midpoint of an edge, or the center of a unit square. We assume the initial configuration to be asymmetric. The necessity of this assumption is discussed after the problem statement.

Problem Statement

Suppose a swarm of robots is placed in an infinite rectangle grid such that no two robots are on the same grid node and the configuration formed by the robots is asymmetric. The Arbitrary Pattern Formation (APF) problem asks to design a distributed deterministic algorithm following which the robots autonomously can form any arbitrary but specific (target) pattern, which is provided to the robots as an input, without scaling it. The target pattern is given to the robots as a set of vertices in the grid with respect to a cartesian coordinate system. We assume that the number of vertices in the target pattern is the same as the number of robots present in the configuration. The pattern is considered to be formed if a configuration is formed and that is the same with target pattern up to translations, rotations, and reflections. The algorithm should be *collision-free*, i.e., no two robots should occupy the same node at any time, and two robots must not cross each other through the same edge.

Admissible Initial Configurations

We assume that in the initial configuration there is no multiplicity point, i.e., no grid node that is occupied by multiple robots. This assumption is necessary because all robots run the same deterministic algorithm, and two robots located at the same point have the same view. Thus, it is deterministically impossible to separate them afterward. Next, suppose the initial configuration has a reflectional symmetry with no robot on the axis of symmetry or a rotational symmetry with no robot on the point of rotation. Then it can be shown that no deterministic algorithm can form an asymmetric target configuration from this initial configuration. However, if the initial configuration has reflectional symmetry with some robots on the axis of symmetry or rotational symmetry with a robot at the point of rotation, then symmetry may be broken by a specific move of such robots. But making such a move may not be very easy as the robots' moves are restricted to their adjacent grid nodes only. In this work, we assume the initial configuration to be asymmetric.

3 Space-optimal Arbitrary Pattern Formation on a Grid Line

In this section, we solve this problem on a discrete straight line. Suppose we have an infinite path graph $\mathcal{P} = \{(i, i + 1) \mid i \in \mathbb{Z}\}$ embedded on a straight line. Suppose k robots are placed on \mathcal{P} at distinct nodes. A configuration is defined similarly as done in the previous section by considering $\mathcal{G} = \mathcal{P}$. The target pattern is given as a set of k distinct positive integers.

Leader Election and Global Coordinate Setup

We assume the initial configuration of robots does not have reflectional symmetry. First, we set up a global coordinate system that can be agreed upon by all the robots. Suppose \mathcal{C} is a configuration having no reflectional symmetry. For a configuration, we define the smallest enclosing line segment (SEL) to be the smallest line segment in length that contains all the robots in the configuration. Let $\mathcal{L} = AB$ be the SEL of the configuration \mathcal{C} . Consider two binary strings of length $|AB|$ (the length of a line segment is the number of grid points on the line segment) called λ_A and λ_B with respect to the endpoints of \mathcal{L} . Let $\lambda_A = \{a_i\}_{i=1}^{|AB|}$ such that $a_i = 1$ if and only if the node on the AB line segment having distance $i - 1$ from A is occupied by a robot. Similarly, we define λ_B . Since \mathcal{C} has no reflectional symmetry, λ_A and λ_B are different. Therefore one of them is lexicographically smaller than the other.

Suppose λ_A is lexicographically smaller than λ_B . Then A is considered as the origin and \overrightarrow{AB} is considered as the positive (right) direction. Also, the robot located at A is said to be *head* and the robot located at B is said to be *tail*. We denote $\mathcal{C} \setminus \{\text{tail}\}$ as \mathcal{C}' .

Target Embedding

Next we embed the pattern in the following way. Considering the integers given in the target pattern on the number line proceed similarly as done above for \mathcal{C} . Let \mathcal{C}_{target} be the target configuration and $A'B'$ be the SEL of \mathcal{C}_{target} . Consider two binary strings $\lambda_{A'}$ and $\lambda_{B'}$. If both the strings are equal then the target pattern has a reflectional symmetry. In this case, embed the pattern such that all the target positions are on the right side of the origin except the left most one which is on the origin. If the strings are different then we suppose $\lambda_{A'}$ is the lexicographically smaller one. In this case, embed the pattern such that $A = A'$ and all the target positions are on the right side of the origin. After embedding, the farthest target position from the origin is said to be the *tail-target* and denoted as t_{target} . We define, $\mathcal{C}'_{target} = \mathcal{C}_{target} \setminus \{t_{target}\}$.

Proposed APF algorithm a Line

Next, we describe our proposed algorithm APFLINE. If in a snapshot of a robot, another robot is seen on an edge then the robot discards the snapshot and goes to sleep. Therefore, for simplicity, we assume that any snapshot taken by a robot contains a still configuration \mathcal{C} . The head never moves in the algorithm. Firstly, if $\mathcal{C}' = \mathcal{C}'_{target}$ then the tail moves to t_{target} . Otherwise, if t_{target} is at the right of the tail, then the tail moves right and the other robots remain static. If $\mathcal{C}' \neq \mathcal{C}'_{target}$, and the tail is at the t_{target} or to the right of the t_{target} , then inner robots move to make $\mathcal{C}' = \mathcal{C}'_{target}$. Let r_i be the i^{th} robot from the left and t_i be the i^{th} target position from the left. We try to design the algorithm such that r_i moves to t_i . The r_1 robot is the head and it is already on t_1 . If t_i is towards the left of r_i and the left adjacent grid node is empty, then an inner robot r_i moves towards the left. If for each inner robot r_j which is not currently on t_j , t_j is at the right of the r_j , then an inner robot r_i moves right if t_i is at the right of the r_i and the right adjacent grid node is empty (The pseudo-code of the algorithm is given in Algorithm 1).

Algorithm 1 APFLINE (for a generic robot r).

```

1 if  $\mathcal{C}' = \mathcal{C}'_{target}$  then
2   | tail moves towards  $t_{target}$ ;
3 else
4   | if  $t_{target}$  is at the right of the tail then
5   |   | tail moves towards right;
6   | else
7   |   | if  $r = r_i$  is an inner robot then
8   |   |   | if  $t_i$  is at the left of  $r_i$  then
9   |   |   |   | if left adjacent grid node is empty then
10  |   |   |   |   |  $r$  moves towards left;
11  |   |   |   | else if for each inner robot  $r_j$  which is not currently on  $t_j$ ,  $t_j$  is at the right of the  $r_j$ 
12  |   |   |   |   | then
13  |   |   |   |   |   | if  $t_i$  is at the right of  $r_i$  then
14  |   |   |   |   |   |   | if right adjacent grid node is empty then
   |   |   |   |   |   |   |   |  $r$  moves towards right;

```

► **Theorem 2.** *From any asymmetric initial configuration, the algorithm APFLINE can form any target pattern on an infinite grid line within finite time under an asynchronous scheduler.*

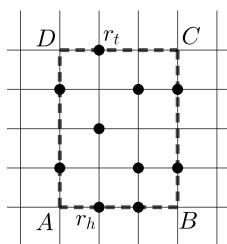
Proof. See the full version [14]. ◀

4 The Proposed Apf Algorithm on a Rectangular Grid

4.1 Agreement of a Global Coordinate System and Target Embedding

Let \mathcal{C} be an asymmetric configuration. Consider the smallest enclosing rectangle (SER) containing all the robots where the sides of the rectangle are parallel to the grid lines. Let $\mathcal{R} = ABCD$ be the SER of the configuration, a $m \times n$ rectangle with $|AD| = m \geq n = |AB|$. The length of the sides of \mathcal{R} is considered to be the number of grid points on that side. If all the robots are on a grid line, then \mathcal{R} is just a line segment. In this case, \mathcal{R} is considered a $m \times 1$ “rectangle” with $A = B$, $D = C$, and $AB = CD = 1$.

For a side, say AB , of \mathcal{R} we define a binary string, denoted as λ_{AB} , as follows. Let $(A = A_1, A_2, \dots, A_m = D)$ be the sequence of grid points on the AD line segment and $(B = B_1, B_2, \dots, B_m = C)$ be the sequence of grid points on the BC line segment. Scan the line segment AB from A to B . Then scan the line segments A_iB_i one by one in the increasing order of i . The direction of scanning the line segment A_iB_i is set as follows: Scan it from B_i to A_i if i is even and scan it from A_i to B_i if i is odd. While scanning, for each grid point put 0 or 1 according to whether it is empty or occupied, respectively (See λ_{AB} in Fig. 1).



■ **Figure 1** $ABCD$ is the SER of the configuration. $\lambda_{AB} = 01101101010011010100$ is the largest lexicographic string, and r_h and r_t are respectively the head and tail robots of the configuration.

If $m > n > 1$, then for each corner point A , B , C , and D , consider the binary strings λ_{AB} , λ_{BA} , λ_{CD} and λ_{DC} , respectively. If $m = n > 1$, then for each corner point, we have to associate two binary strings with respect to the two sides adjacent to the corner point. Then we have eight binary strings λ_{AB} , λ_{BA} , λ_{AD} , λ_{DA} , λ_{BC} , λ_{CB} , λ_{DC} and λ_{CD} . If any two strings of them are equal then it can be shown that \mathcal{C} has a (reflectional or rotational) symmetry. Since \mathcal{C} is asymmetric, we can find a unique lexicographically largest string (See Fig. 1). Let λ_{AB} be the lexicographically largest string, and then A is considered the *leading corner* of the configuration. The leading corner is taken as the origin, and \overrightarrow{AB} is as the x -axis, and \overrightarrow{AD} is as the y -axis.

If \mathcal{R} is an $m \times 1$ rectangle, then λ_{AB} and λ_{BA} are the same string. Then we have two strings to compare. Since the configuration is asymmetric, these two strings must be distinct. Then we shall have a leading corner, say $A = B$. For this case, A is considered as the origin, and \overrightarrow{AD} as the y -axis. There will be no agreement of the x -axis in this case but since all the robots are on the y -axis, so x -coordinate of the positions of the robots are 0 at this time.

If \mathcal{C} is asymmetric then a unique string can be elected and hence, all robots can agree on a global coordinate system. By “up” (“down”) and “right” (“left”), we shall refer to the positive (“negative”) directions of the x -axis and y -axis of the coordinate system, respectively. The robot responsible for the first 1 in this string is considered the *head* robot of \mathcal{C} and the robot responsible for the last 1 is considered the *tail* of \mathcal{C} . The robot other than the head and tail is termed the *inner robot*. We define, $\mathcal{C}' = \mathcal{C} \setminus \{\text{tail}\}$ and $\mathcal{C}'' = \mathcal{C} \setminus \{\text{head}, \text{tail}\}$.

Target Pattern Embedding

Here we discuss how robots are supposed to embed the target pattern when they agree on a global coordinate system. The target configuration \mathcal{C}_{target} is given with respect to some arbitrary coordinate system. Let the $\mathcal{R}' = A'B'C'D'$ be the SER of the target pattern, an $m' \times n'$ rectangle with $|A'D'| \geq |A'B'| > 1$. We associate binary strings similarly for \mathcal{R}' as done for \mathcal{R} . Let $\lambda_{A'B'}$ be the lexicographically largest (but may not be unique because the \mathcal{C}_{target} can be symmetric) among all other strings for \mathcal{R}' . The first target position on this string $\lambda_{A'B'}$ is said to be *head-target* and denoted as h_{target} and the last target position is said to be *tail-target* and denoted as t_{target} . The rest of the target positions are called *inner target* positions. Then the target pattern is to be embedded such that A' is the origin, $\overrightarrow{A'B'}$ direction is along the positive x -axis, and $\overrightarrow{A'D'}$ direction is along the positive y -axis. Next, let us consider the case when $|A'B'| = 1$, that is when the SER of the target pattern is a line $A'D'$. Let $\lambda_{A'D'}$ be the lexicographically largest string between $\lambda_{A'D'}$ and $\lambda_{D'A'}$. Then the target is embedded in such a way that A' is at the origin and $\overrightarrow{A'D'}$ direction is along the positive y -axis. The positive x -axis direction can be decided randomly by the robot which first moves out of that line making the SER a rectangle. We define, $\mathcal{C}'_{target} = \mathcal{C}_{target} \setminus \{t_{target}\}$ and $\mathcal{C}''_{target} = \mathcal{C}_{target} \setminus \{h_{target}, t_{target}\}$.

4.2 Outline of the Proposed Algorithm

The algorithm is logically divided into seven phases³. A robot infers which phase it is in from the configuration visible at that time. It does so by checking which conditions in Table 2 are fulfilled. We assume that in a visible configuration, no robot is seen on an edge. We maintain such assumption by an additional condition that, if a robot sees a configuration where a robot is on an edge then discard the snapshot and go to sleep.

A Preview of the Algorithm

- Firstly the tail robot moves upwards to reach a horizontal line such that neither the horizontal line nor other horizontal lines above it contain any robot or target position (Phase I).
- Next the head robot moves left to reach the origin (Phase II).
- Then the tail robot moves a few steps upwards to remove the chance of occurrence of symmetry during the later inner robot movements (phase I).
- Then the tail robot moves rightwards to reach a vertical line such that neither the vertical line nor any vertical line to the right of it contains any robot or target positions (Phase III).

³ The phases are assigned numerical names, yet the sequence of these numerals doesn't precisely correspond to the sequence of their execution during algorithm execution.

- After that a spanning line is considered (Figure 2) and inner robots carefully move along this line (Function **Rearrange**) to take their respective target position avoiding collision or forming any symmetric configuration (Phase IV).
- After that the tail moves horizontally to reach the vertical line that contains t_{target} (Phase V).
- Then the head robot moves horizontally to reach h_{target} (Phase VI).
- After that the tail moves vertically to reach t_{target} (Phase VII).

■ **Table 2** Set of conditions on an asymmetric configuration \mathcal{C} having SER $ABCD$ such that the origin is at A .

C_0	$\mathcal{C} = \mathcal{C}_{target}$
C_1	$\mathcal{C}' = \mathcal{C}'_{target}$
C_2	$\mathcal{C}'' = \mathcal{C}''_{target}$
C_3	x -coordinate of the tail = x -coordinate of t_{target}
C_4	There is neither any robot except the tail nor any target positions on or above H_t , where H_t is the horizontal line containing the tail
C_5	y -coordinate of the tail is odd
C_6	SER of \mathcal{C} is not a square
C_7	There is neither any robot except the tail nor any target positions on or at the right of V_t , where V_t is the vertical line containing the tail
C_8	The head is at origin
C_9	If the tail and the head are relocated respectively at C and A , then the new configuration remains asymmetric
C_{10}	\mathcal{C}' has a symmetry with respect to a vertical line

4.3 Detail Discussion of the Phases

Phase I

A robot infers itself in Phase I if $\neg(C_4 \wedge C_5 \wedge C_6) \wedge \neg(C_1 \wedge C_3)$ is true. In this phase, the tail moves upward and all other robots remain static. The aim of this phase is to make $C_4 \wedge C_5 \wedge C_6$ true.

Phase II

A robot infers itself in Phase II if $(C_4 \wedge C_5 \wedge C_6 \wedge \neg C_8) \wedge ((C_2 \wedge \neg C_3) \vee \neg C_2)$ is true. In this phase, the head moves towards the left, and other robots remain static. This phase aims to make C_8 true.

Phase III

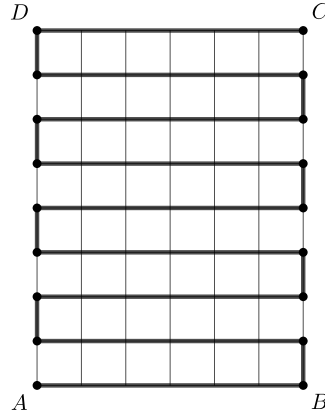
A robot infers itself in Phase III if $C_4 \wedge C_5 \wedge C_6 \wedge C_8 \wedge \neg C_2 \wedge \neg C_7$ is true. The aim of this phase is to make C_7 true. In this phase, there are two cases to consider. The robots will check whether C_{10} is true or not. If C_{10} is false, then robots check whether C_9 is true or not. If C_9 is not true then the tail moves upward. Otherwise, the tail moves right or upwards in accordance with $m > n + 1$ or $m = n + 1$ (dimension of the current SER is $m \times n$ with $m \geq n$). If C_{10} is true, then the tail moves left or upwards in accordance with $m > n + 1$ or $m = n + 1$. Other robots remain static in both cases.

Phase IV

A robot infers itself in Phase IV if $C_4 \wedge C_5 \wedge C_6 \wedge C_7 \wedge C_8 \wedge \neg C_2$ is true. In this phase, the inner robots execute function **Rearrange** to make C_2 true.

Function Rearrange

In this function inner robots move to take their respective target positions. Let \mathcal{C} be the current configuration. Let $ABCD$ be the SER of \mathcal{C} . According to the assumption exactly two nonadjacent vertices are occupied by robots in rectangle $ABCD$. Specifically, these two robots are the head and the tail of the configuration. Let the head and tail be located at A and C respectively. Consider the path \mathcal{P} starting from A to C as illustrated in bold edges in Fig. 2. Inner robots adopt algorithm APFLINE considering this path as the line. Here, we define a robot r' at the **left (right)** side of r if r' is closer to the head (tail) than r in \mathcal{P} . Let us order the target positions. Denote t_{target} as t_1 , then the next closest target position from the head in \mathcal{P} as t_2 . Similarly, denote the i^{th} closest target positions in \mathcal{P} from the head as t_i . Note that, t_k is the t_{target} . Similarly order all the robots, $\{r_i\}_{i=1}^k$, where r_1 is the head and r_i ($i > 1$) is the i^{th} closest robot from the head on \mathcal{P} .



■ **Figure 2** Path joining the nodes A and C mentioned in bold edges.

If t_i is at the **left** of r_i and there are no other robots in the sub-path of \mathcal{P} starting from the position of r_i to t_i , then r_i moves to t_i . The movement strategy is described as follows. If r_i and t_i are at the same vertical (or, horizontal) line then r_i moves through the vertical (or, horizontal) line joining r_i and t_i . Suppose, r_i and t_i are not at the same vertical line or horizontal line. If the downward adjacent vertex of r_i is at the **right** of t_i then r_i moves downwards. If the downward adjacent vertex is at the **left** of t_i , then r_i moves to its **left** adjacent node on \mathcal{P} .

If there is no robot r_j such that t_j is at the **left** of r_j , then movements of an inner robot towards **right** start. If t_i is at the **right** of r_i , and there are no other robots in the sub-path of \mathcal{P} starting from the position of r_i to t_i , then r_i moves to t_i . The movement strategy is described as follows. If r_i and t_i are at the same vertical (or, horizontal) line then r_i moves through the vertical (or, horizontal) line joining r_i and t_i . Suppose, r_i and t_i are not at the same vertical line or horizontal line. If the upward adjacent vertex of r_i is at the **left** of t_i then r_i moves upwards. If the upward adjacent vertex is at the **right** of t_i , then r_i moves to its **right** adjacent on node \mathcal{P} (pseudo code of the function **Rearrange** is given Algorithm 2).

Phase VI

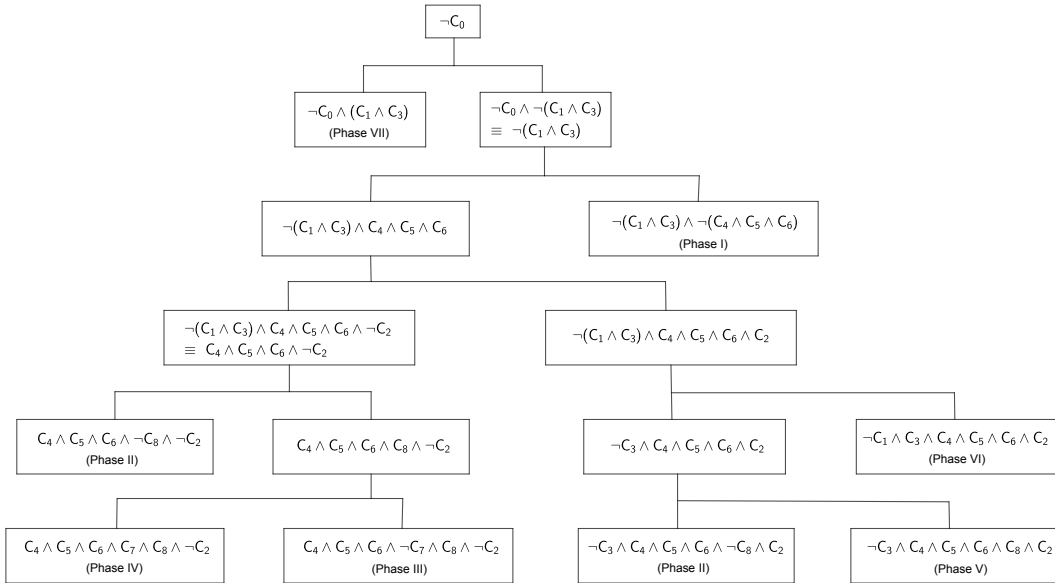
A robot infers itself in Phase VI if $\neg C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6$ is true. In this phase, the head moves horizontally to reach h_{target} . After the completion of this phase, $\neg C_0 \wedge C_1 \wedge C_3$ becomes true.

Phase VII

A robot infers itself in Phase VII if $\neg C_0 \wedge C_1 \wedge C_3$ is true. In this phase, the tail moves vertically to reach t_{target} .

4.4 Correctness and Performance of the Proposed Algorithm

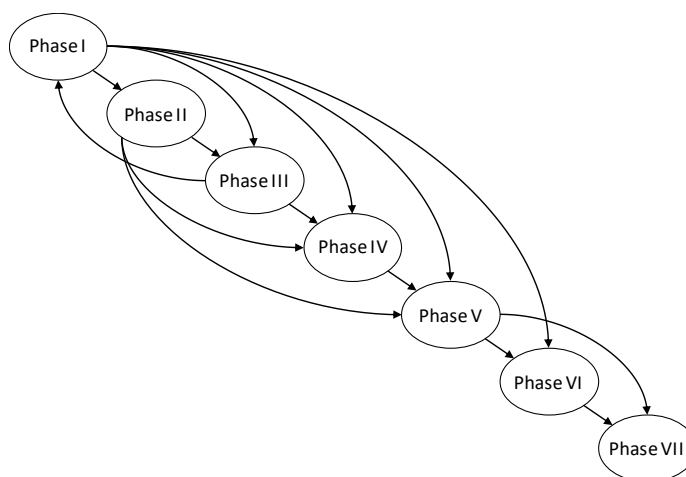
In this section, we prove the correctness of the proposed algorithm. First, we show that any initial asymmetric configuration for which C_0 is not true falls in one of the seven phases (See Figure 4). Then we show that from any asymmetric initial configuration, the algorithm allows the configuration to satisfy $C_0 = \text{true}$ after passing through several phases (See Figure 5). The correctness proof details are omitted from this paper due to space constraint. See the full version of the paper in [14].



■ **Figure 4** For any configuration with $C_0 = \text{false}$ belongs to one of the seven phases.

► **Theorem 3.** *The proposed algorithm can form any pattern consisting of k points by a set of k oblivious asynchronous robots if the initial configuration formed by the robots is an asymmetric configuration and has no multiplicity point.*

Recall the Definition 1 of the space complexity of an algorithm executed by a set of robots on an infinite rectangular grid. In Theorem 4, we calculate the space complexity of the proposed algorithm. The move complexity is recorded in the Theorem 5. Proofs of these theorems are available in full version of the paper [14].



■ **Figure 5** Phase transition digraph.

► **Theorem 4.** Let $\mathcal{D} = \max\{m, n, m', n'\}$ where $m \times n$ ($m \geq n$) is the dimension of the SER of the initial configuration and $m' \times n'$ ($m' \geq n'$) is the dimension of the SER of the target configuration. Then the space complexity of the proposed algorithm is at most $\mathcal{D} + 4$. More precisely, if $M = \max\{m, m'\}$ and $N = \max\{n, n'\}$, then the proposed algorithm takes the space enclosed by a rectangle of dimension $(M + 4) \times (N + 1)$.

► **Theorem 5.** The proposed algorithm requires each robot to make $O(\mathcal{D})$ movements, hence the move-complexity of the proposed algorithm is $O(k\mathcal{D})$.

5 Conclusion

This work first provides an algorithm that solves the APF problem in an infinite line by a robot swarm. Then adopting the method, it provides another algorithm that solves the APF in an infinite rectangular grid by a robot swarm. The robots are autonomous, anonymous, identical, and homogeneous. The robot model used here is the classical *OBLLOT* model. The robots work under a fully asynchronous scheduler. The proposed algorithm is almost space-optimal (Theorem 4) and asymptotically move-optimal (Theorem 5).

A few limitations of this work are the following. Here we assume that the initial configuration is asymmetric. Finding complete characterization of the initial configurations from which APF can be solved deterministically is an interesting future direction. Next, the version of the APF problem under consideration does not permit multiple points in the target configuration. More precisely, the number of target positions in the target pattern is equal to the number of robots within the system. Solving a more generalized version of the problem that allows target patterns with target positions less than the total number of robots, is a possible future direction. Next, the proposed algorithm is almost space optimal, so finding out the exact lower bound when starting from an asymmetric initial configuration is an interesting direction. Also, this does not consider time-optimality, so considering all the three parameters space, move and time at the same time can be an interesting future work.

References

- 1 Kaustav Bose, Ranendu Adhikary, Manash Kumar Kundu, and Buddhadeb Sau. Arbitrary pattern formation on infinite grid by asynchronous oblivious robots. *Theoretical Computer Science*, 815:213–227, 2020. doi:10.1016/j.tcs.2020.02.016.

- 2 Kaustav Bose, Archak Das, and Buddhadeb Sau. Pattern formation by robots with inaccurate movements. In Quentin Bramas, Vincent Gramoli, and Alessia Milani, editors, *25th International Conference on Principles of Distributed Systems, OPODIS 2021, December 13-15, 2021, Strasbourg, France*, volume 217 of *LIPICs*, pages 10:1–10:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.OPODIS.2021.10.
- 3 Quentin Bramas and Sébastien Tixeuil. Probabilistic asynchronous arbitrary pattern formation (short paper). In Borzoo Bonakdarpour and Franck Petit, editors, *Stabilization, Safety, and Security of Distributed Systems - 18th International Symposium, SSS 2016, Lyon, France, November 7-10, 2016, Proceedings*, volume 10083 of *Lecture Notes in Computer Science*, pages 88–93, 2016. doi:10.1007/978-3-319-49259-9_7.
- 4 Quentin Bramas and Sébastien Tixeuil. Arbitrary pattern formation with four robots. In Taisuke Izumi and Petr Kuznetsov, editors, *Stabilization, Safety, and Security of Distributed Systems - 20th International Symposium, SSS 2018, Tokyo, Japan, November 4-7, 2018, Proceedings*, volume 11201 of *Lecture Notes in Computer Science*, pages 333–348. Springer, 2018. doi:10.1007/978-3-030-03232-6_22.
- 5 Serafino Cicerone, Alessia Di Fonso, Gabriele Di Stefano, and Alfredo Navarra. Arbitrary pattern formation on infinite regular tessellation graphs. *Theor. Comput. Sci.*, 942:1–20, 2023. doi:10.1016/j.tcs.2022.11.021.
- 6 Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Embedded pattern formation by asynchronous robots without chirality. *Distributed Comput.*, 32(4):291–315, 2019. doi:10.1007/s00446-018-0333-7.
- 7 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, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, volume 6343 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2010. doi:10.1007/978-3-642-15763-9_26.
- 8 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theor. Comput. Sci.*, 407(1-3):412–447, 2008. doi:10.1016/j.tcs.2008.07.026.
- 9 Satakshi Ghosh, Pritam Goswami, Avisek Sharma, and Buddhadeb Sau. Move optimal and time optimal arbitrary pattern formations by asynchronous robots on infinite grid. *International Journal of Parallel, Emergent and Distributed Systems*, 0(0):1–23, 2022. doi:10.1080/17445760.2022.2124411.
- 10 Rory Hector, Gokarna Sharma, Ramachandran Vaidyanathan, and Jerry L. Trahan. Optimal arbitrary pattern formation on a grid by asynchronous autonomous robots. In *2022 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022, Lyon, France, May 30 - June 3, 2022*, pages 1151–1161. IEEE, 2022. doi:10.1109/IPDPS53621.2022.00115.
- 11 Manash Kumar Kundu, Pritam Goswami, Satakshi Ghosh, and Buddhadeb Sau. Arbitrary pattern formation by asynchronous opaque robots on infinite grid. *CoRR*, abs/2205.03053, 2022. doi:10.48550/arXiv.2205.03053.
- 12 Manash Kumar Kundu, Pritam Goswami, Satakshi Ghosh, and Buddhadeb Sau. Arbitrary pattern formation by opaque fat robots on infinite grid. *International Journal of Parallel, Emergent and Distributed Systems*, 37(5):542–570, 2022. doi:10.1080/17445760.2022.2088750.
- 13 Brati Mondal, Pritam Goswami, Avisek Sharma, and Buddhadeb Sau. Arbitrary pattern formation on a continuous circle by oblivious robot swarm. In *Proceedings of the 25th International Conference on Distributed Computing and Networking, ICDCN 2024, Chennai, India, January 4-7, 2024*, pages 94–103. ACM, 2024. doi:10.1145/3631461.3631545.
- 14 Avisek Sharma, Satakshi Ghosh, Pritam Goswami, and Buddhadeb Sau. Space and move-optimal arbitrary pattern formation on infinite rectangular grid by oblivious robot swarm. *CoRR*, abs/2309.11190, 2023. doi:10.48550/arXiv.2309.11190.
- 15 Avisek Sharma, Satakshi Ghosh, Pritam Goswami, and Buddhadeb Sau. Space and move-optimal arbitrary pattern formation on a rectangular grid by robot swarms. In *Proceedings of*

- the 25th International Conference on Distributed Computing and Networking, ICDCN 2024, Chennai, India, January 4-7, 2024*, pages 65–73. ACM, 2024. doi:10.1145/3631461.3631542.
- 16 Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999. doi:10.1137/S009753979628292X.
 - 17 Masafumi Yamashita and Ichiro Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theoretical Computer Science*, 411(26):2433–2453, 2010. doi:10.1016/j.tcs.2010.01.037.

Gathering in Carrier Graphs: Meeting via Public Transportation System

Haozhi Zheng ✉ 

Graduate School of Science and Technology, Nara Institute of Science and Technology, Japan

Ryota Eguchi ✉ 

Graduate School of Science and Technology, Nara Institute of Science and Technology, Japan

Fukuhito Ooshita ✉ 

Faculty of Engineering, Fukui University of Technology, Japan

Michiko Inoue ✉ 

Graduate School of Science and Technology, Nara Institute of Science and Technology, Japan

Abstract

The *gathering problem* requires multiple mobile agents in a network to meet at a single location. This paper investigates the gathering problem in *carrier graphs*, a subclass of *recurrence of edge* class of *time-varying graphs*. By focusing on three subclasses of single carrier graphs – *circular*, *simple*, and *arbitrary* – we clarify the conditions under which the problem can be solved, considering prior knowledge endowed to agents and obtainable online information, such as the count and identifiers of agents or sites. We propose algorithms for solvable cases and analyze the complexities and we give proofs for the impossibility for unsolvable cases. We also consider general carrier graphs with multiple carriers and propose an algorithm for arbitrary carrier graphs. To the best of our knowledge, this is the first work that investigates the gathering problem in carrier graphs.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Gathering, Carrier Graph, Time-varying Graph, Mobile agent

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.21

Funding This work was supported by JSPS KAKENHI Grant Numbers 22K11903, 23H03347. The second author was supported by JSPS KAKENHI Grant Number 22H03569.

1 Introduction

Imagine a group of friends attending a music festival in a large park. They have agreed to meet somewhere to enjoy the festival together. However, they become separated in the crowd and have no means of wireless communication. They decide to use the festival shuttle bus, a public transportation carrier that travels around the park on a fixed schedule, to meet at an undetermined location. The challenge of determining when to get on and off the bus and how to ensure that everyone is at the same place is formalized as a *gathering problem*, particularly, gathering problem in a *carrier graph*.

The *gathering problem* requires multiple mobile agents in a network to meet at a single location. The gathering facilitates information sharing among agents working on collaborative tasks. Distributed algorithms for this problem including the *rendezvous* problem (gathering for two agents) have been well studied, especially for static graphs [1, 4, 10, 17].

Recently, distributed algorithms for *highly dynamic graphs* have been intensively studied. These are dynamic graphs whose dynamics are not restricted locally in time or space, that is, graphs are continuously changing over time. Casteigts et al. integrated several concepts of dynamic graphs investigated separately as *time-varying graphs* and sorted them into 15 classes [3]. One meaning class is *connectivity over time COT* (or *temporally connected* [8]) that is a class of graphs where any pair of two nodes are connected over time in both directions (an



© Haozhi Zheng, Ryota Eguchi, Fukuhito Ooshita, and Michiko Inoue; licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 21; pp. 21:1–21:17

Leibniz International Proceedings in Informatics



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

agent can reach from one side to another side when it adequately waits and moves in some intermediate nodes). Not surprisingly, due to its weak connectivity assumption, most results for \mathcal{COT} are negative and gathering algorithms achieve only weak properties [2, 16].

As subclasses of \mathcal{COT} , more constrained (or more easily handled) classes are defined. Classes *Constant connectivity* \mathcal{CC} and *T-interval connectivity* \mathcal{INT}_T are graphs that guarantee connectivity for any moment. A graph in \mathcal{INT}_T keeps the same spanning tree for any period of T consecutive time steps, and \mathcal{CC} is the same as 1-interval connectivity \mathcal{INT}_1 . These classes have inclusion relation $\mathcal{COT} \supset \mathcal{CC} = \mathcal{INT}_1 \supset \mathcal{INT}_2 \supset \dots$. For 1-interval connected graphs, especially 1-interval connected rings, gathering, and related problems have been well studied [14, 9, 15, 18].

Another interesting subclasses of \mathcal{COT} are *recurrence of edges* \mathcal{RE} , *time-bounded recurrence of edges* \mathcal{BRE} and *periodicity of edges* \mathcal{PE} . These are graphs where any edge recurrently appears if it appears at least once. The recurrence of edges is bounded in time in \mathcal{BRE} and it is periodic in \mathcal{PE} . The inclusion relation among these classes is $\mathcal{COT} \supset \mathcal{RE} \supset \mathcal{BRE} \supset \mathcal{PE}$. The gathering problem has not been well studied in these classes. There is one work on gathering problem considering \mathcal{COT} , \mathcal{CC} , \mathcal{RE} , \mathcal{BRE} and static graphs [2].

For a class of *carrier graphs*, that is a subclass of \mathcal{PE} , the *exploration* problem has been studied [6, 11]. The carrier graph (C-graph) models a system where one or more *carriers* periodically visit sites in the system by following their routes. Agents can move sites with a carrier when some carrier comes to their current site. Practical examples of this model include public transportation systems like buses [20], planes [12] and satellites [19]. Furthermore, this model also finds relevance in the context of ad-hoc data-routing schemes [13, 21]. The exploration requires an agent to visit all the nodes in a network. This is closely related to the gathering since agents need to explore a network to achieve gathering.

This paper considers the gathering problem in carrier graphs. This can be seen as a problem where people try to meet at some station (unknown in advance) in a public transportation system. We consider several assumptions on prior knowledge of agents such as the counts of agents or sites, and, on acquirable information at sites such as identifiers or the number of agents at the site or the site identifier, and for each assumption, clarify the solvability of the problem and propose algorithms for solvable cases.

1.1 Related Works

The gathering problems targeting time-varying graphs are well studied for a family of constantly connected graphs, especially 1-interval connected graphs [2, 14, 15, 18]. Di Luna et al. first investigated the gathering problem in 1-interval connected rings [14]. They first showed gathering at a single node is impossible in 1-interval connected rings and considered the *weak gathering* that allows gathering at the same node, or the two end nodes of the same edge. The feasible initial configurations (initial configurations from which the problem is solvable) were clarified and gathering algorithms were proposed under several assumptions on chirality and cross-detection. Michail et al. expanded targeting graphs beyond rings and examined the solvability of the weak gathering problem for the class of 1-interval connected graphs and initial configurations [15]. Shibata et al. considered the *partial gathering* problem where each agent is required to gather with a group of at least g agents for a given g for 1-interval connected graphs [18]. They clarified the solvability and proposed algorithms for solvable cases under several assumptions on g .

Ooshita and Datta considered weak gathering on rings in \mathcal{COT} [16]. They proved that in \mathcal{COT} , weak gathering is impossible when agents cannot leave information at nodes or when all agents should terminate. They also proposed a weak gathering algorithm without termination when agents can leave information at nodes.

Bournat et al. had a unique work on the gathering in time-varying graphs [2]. They considered not only the constant connectivity class \mathcal{CC} but also the family of recurrence of edges classes \mathcal{RE} (recurrence of edges) and \mathcal{BRE} (time-bounded recurrence of edges). They clarified the solvability for four variants of the gathering problem, gathering (all agents gather in bounded time), eventual gathering (all agents gather in finite time), weak gathering (all agents but at most one gather in bounded time, a different definition from [14]) and eventual weak gathering (all agents but at most one gather in finite time), and proposed a single algorithm that solves the strongest feasible variant for \mathcal{COT} , \mathcal{CC} , \mathcal{RE} , \mathcal{BRE} and static graphs. This is the only work that considers the gathering problem for the family of recurrence of edge classes, and there is no work on the gathering problem targeting carrier graphs.

The carrier graph model was introduced as a subclass of time-varying graphs by Flocchini et al. [6]. This work considers anonymous systems (sites possess no IDs) and non-anonymous systems (sites possess unique IDs), and for both systems established the necessary and sufficient conditions for exploration. To showcase the time complexity differences across various settings, they introduced three carrier graph classes – circular, simple, and arbitrary. The agents in this research are constrained to move exclusively with the carrier and lack the capability to wait on sites, analogous to the context of low-earth orbiting satellite systems.

In contrast, many real-world public transportation systems allow agents to stay at a station, enabling them to await a potentially distinct carrier. Ilcinkas and Wade extended this perspective by allowing agents to leave the carrier and wait on sites [11]. They demonstrated that this added capability enables agents to reduce the number of moves in the worst case. Additionally, this ability allows the agent agents not only to achieve exploration but also to map the whole carrier graph.

Flocchini et al. studied the mapping of carrier graphs with “black holes” which are sites that destroy agents [5, 7]. Their investigations delved into collectively mapping the graph by multiple agents with the ability to leave messages at sites. The goal was to collaboratively construct a map of the carrier graph while minimizing agent loss.

1.2 Our Contributions

This paper considers the gathering problem for carrier graphs. We examine the solvability and propose algorithms (if solvable) for three classes of *circular*, *simple*, and *arbitrary* carrier graphs with a single carrier. The solvability and the time complexity of the gathering problem under several conditions are summarized in Table 1, where p , P , k , and n denote the period, an upper bound of the period, the number of agents, and the number of sites, respectively. Note that some of the results, e.g. for circular or simple graphs with knowledge of P , are automatically derived from the results for the superclass, e.g. for arbitrary graphs with knowledge of P . We also prove the impossibility for unsolvable cases.

Furthermore, we prove the impossibility of gathering in C-graphs with multiple carriers. We also propose a gathering algorithm that terminates in at most $2M + (2p - 2)(m - 1) + 2p$ rounds using existing exploration algorithms, where m and M denote the number of carriers and the termination time of the exploration algorithm, respectively.

2 Preliminaries

2.1 Carrier graphs

We consider a system composed of a set S of n sites and a set C of m carriers. The sites have unique identifiers (IDs) or no ID depending on assumptions, and carriers move among the sites. Each carrier c has a unique identifier $id(c)$ and an ordered sequence of sites

■ **Table 1** Time complexity of gathering algorithms on single carrier graphs.

Assumptions		Graph Class		
prior knowledge	observation ability	Circular	Simple	Arbitrary
P	-	$p + P^*$	$p + P^*$	$p + P^*$
k	-	p^*	p^*	p^*
n	-	$2p^*$	$p + n(n-1)^*$	Impossible
n	agent ID	$2p^*$	$p + n(n-1)^*$	Impossible
n	site ID	$2p^*$	$2p$	$2p$
-	agent ID	$3p$	$4p - 1$	Impossible
-	site ID	$2p^*$	$2p + 1^*$	Impossible
-	agent ID & site ID	$2p^*$	$2p + 1^*$	Impossible

*) algorithms *with simultaneous termination*

 ■ **Table 2** Possibilities of gathering algorithms on general carrier graphs.

Assumptions		Possibility
prior knowledge	observation ability	
P	-	Possible
k	site ID	Impossible
n	-	Impossible
n	site ID	Possible

$\pi(c) = \langle s_0, s_1, \dots, s_{p(c)-1} \rangle, s_i \in S$, called a *route*, where the positive integer $p(c)$ is called a *period* of the route. The carrier c starts at site s_0 at time 0 and then moves to the next site along the route at each time unit in a cyclic manner (moving from $s_{p(c)-1}$ to s_0). We use $s_{p(c)}$ as $s_{p(c)} = s_0$ for convenience. Letting $\pi(c)[j]$ denote a site where c is located at time j , $\pi(c)[j] = s_i$ holds where $i \equiv_{p(c)} j$. A set of all the sites appear in $\pi(c)$ is called a *domain* of c , denoted as $S(c) = \bigcup_{0 \leq i \leq p(c)-1} \{s_i\}$ where $\bigcup_{c \in C} S(c) = S$ holds. We have $|S(c)| \leq p(c)$ since the same site can be visited several times along the route.

Each route $\pi(c)$ defines an arc-labelled multi-graph $\vec{G}(c) = (S(c), \vec{E}(c))$, where $\vec{E}(c) = \{(s_i, s_{i+1}, i) : 0 \leq i < p(c)\}$. The set of all routes of carriers is denoted by $R = \{\pi(c) : c \in C\}$, and a period of R is defined as $p(R) = \max \{p(c) : c \in C\}$. When no ambiguity arises, we will simply denote $p(R)$ as p . The arc-labelled multi-graph $\vec{G}(C) = (S, \vec{E})$, where $\vec{E} = \bigcup_{c \in C} \vec{E}(c)$, is called *carrier graph*, or shortly, *C-graph*. Especially, a carrier graph with only one carrier is called a *single carrier graph*, or shortly, *SC-graph*.

For any C-graph $\vec{G}(C)$, we define a static and undirected *meeting graph* $H(C)$ that has C as a set of nodes. In the meeting graph $H(C)$, there is an edge between two nodes c and c' if and only if there exists a site s such that both $s \in S(c)$ and $s \in S(c')$ hold. A C-graph $\vec{G}(C)$ is said to be connected if and only if $H(C)$ is connected. In this paper, we will always consider *connected* C-graphs.

We classify routes by their property into *circular*, *simple*, and *arbitrary* as follows.

► **Definition 1.** A route $\pi(c) = \langle s_0, s_1, \dots, s_{p(c)-1} \rangle$ is **simple** if $\vec{G}(c)$ contains no self-loop or multiple arcs, that is $s_i \neq s_{i+1}$, for $0 \leq i < p(c)$, and $(x, y, i), (x, y, j) \in \vec{E}(c)$ only if $i = j$.

► **Definition 2.** A simple route $\pi(c) = \langle s_0, s_1, \dots, s_{p(c)-1} \rangle$ is **circular** if it contains no repeated sites, that is $|S(c)| = p(c)$.

A carrier graph $\vec{G}(C) = (S, \vec{E})$ is said to be *circular* and *simple* if every route $\pi(c) \in R$ is circular and simple, respectively. Let \mathcal{C}_C , \mathcal{C}_S , and \mathcal{C}_A denote classes of circular, simple, and arbitrary C-graphs, respectively. Obviously, we have $\mathcal{C}_C \subset \mathcal{C}_S \subset \mathcal{C}_A$.

We have the following lemma as $|S(c)| \leq p$.

► **Lemma 3.** *Every site is visited by some carrier at least once for any time interval $[t, t']$ ($t' - t \geq p - 1$).*

2.2 Mobile Agents

There are k mobile computational entities $a_0, a_1, \dots, a_{k-1} \in A$, called *agents*, in the system, with unique identifiers as $id(a_i) \in \mathbb{N}, 0 \leq i \leq k - 1$. The prior knowledge of the system for agents depends on the assumption considered.

Agents operate in a LOOK-COMPUTE-MOVE manner in each synchronous round j , which is the interval between time $j - 1$ and time j where $j \in \mathbb{N}^+$. At the beginning of each round, an agent gets information on the current site (LOOK operation). The number of agents at the same site can be observed by default, while the ability to observe agents' IDs and the site's ID is endowed by certain assumptions. The agent then determines whether it will move or stay at the current site (COMPUTE operation). Then the agent performs the move depending on the decision (MOVE operation). Agents' memory is persistent across rounds (non-oblivious). We assume that agents cannot observe other agents' memory.

An agent can stay at the current site or move with one of the carriers. An agent a_i can move with or switch to a carrier c only when it is placed at the same site as c at the same time. Agent a_i at site s at time t will be at site s' at time $t + 1$ if it moves with or switches to carrier c , where $(s, s', i) \in \vec{E}$ and $t \equiv_{p(c)} i$ hold. Otherwise, agent a_i stays at site s .

2.3 The Gathering Problem

The goal of the *gathering* problem is to gather all the agents within finite time, that is to let every agent $a \in A$ move to the same site $s \in S$ and terminate, within finite time, regardless of the starting position. Moreover, the gathering problem *with simultaneous termination* requires all the agents to terminate an algorithm simultaneously at the same site.

We assume that an agent starts its execution either spontaneously upon encountering a carrier or upon encountering another moving agent. To evaluate the time complexity of an algorithm, we measure the number of rounds from time 0 (representing the initial configuration) to the point at which all agents have terminated their operations. We designate the initiation of execution by the first agent as round 1, and subsequently, time 1 shows the configuration after the execution of round 1.

3 Gathering with One Carrier

This section explores the gathering problem on SC-graphs. The process of congregating all agents in an SC-graph is straightforward: agents are instructed to move with the carrier whenever they encounter it. Consequently, all agents are assured to be located at the same site at any round $t \geq p$. However, the challenge is how each agent decides when to terminate its execution. We will showcase the impossibility of gathering under certain assumptions. Next, we will show several feasible assumptions under which we propose algorithms to solve the gathering problem. We will prove their correctness and assess their time complexity.

3.1 Impossibilities

At first, we show the impossibility of gathering for arbitrary SC-graphs.

► **Theorem 4.** *There does not exist any algorithm that solves the gathering problem in arbitrary SC-graphs when the agents have no prior knowledge even when they can observe IDs of agents and the current site ID.*

Proof. For contradiction, suppose that there exists an algorithm \mathcal{A} that solves the gathering problem in the settings of the statement of the theorem.

Consider a SC-graph G with n sites s_0, \dots, s_{n-1} . Assume that a single carrier c has a route $\pi(c) = \langle s_0, \dots, s_{n-1} \rangle$ and n agents a_0, \dots, a_{n-1} exist at sites s_0, \dots, s_{n-1} initially. By algorithm \mathcal{A} , there exists T such that all agents gather and terminate at some site $s \in S(c)$ in T rounds.

We consider another SC-graph G' with $n' + 1$ sites $s'_0, \dots, s'_{n'}$. Assume that a single carrier c' has a route $\pi(c') = \langle s'_0, \dots, s'_0, s'_1, \dots, s'_{n'} \rangle$ where c' visits s'_0 repeatedly T times and then visits $s'_1, \dots, s'_{n'}$. Also assume that n' agents $a'_0, \dots, a'_{n'-1}$ exist at sites $s'_1, \dots, s'_{n'}$ initially. Note that the agents do not observe a carrier during the first T rounds. By algorithm \mathcal{A} , there exists T' such that all agents gather and terminate at some site $s' \in \pi(c')$ in T' rounds.

Lastly, we construct another SC-graph G'' from the above two graphs G and G' . The sites of G'' include $s_0, \dots, s_{n-1}, s'_0, \dots, s'_{n'}$. Assume that n agents a_0, \dots, a_{n-1} are initially placed at the same positions as G and n' agents $a'_0, \dots, a'_{n'-1}$ are initially placed at the same positions as G' . In this graph, we consider a single carrier c'' such that c'' moves similarly to G during the first T rounds and moves similarly to G' after that. That is, $\pi(c'')[j] = \pi(c)[j]$ for $0 \leq t < T$ and $\pi(c'')[j] = \pi(c')[j]$ for $T \leq t < T'$. Let us consider the behavior of agents a_0, \dots, a_{n-1} in graph G'' . During the first T rounds, carrier c'' moves similarly to G and consequently agents a_0, \dots, a_{n-1} behave similarly to G . Hence, agents a_0, \dots, a_{n-1} terminate at site s in T rounds. Next consider the behavior of agents $a'_0, \dots, a'_{n'-1}$. During the first T rounds, none of them observes a carrier, which is the same as G' . During time T to T' , carrier c'' moves similarly to G' . Hence, agents $a'_0, \dots, a'_{n'-1}$ behave similarly to G' and terminate at site s' in T' rounds. This implies that agents terminate at different two nodes s and s' . This is a contradiction. ◀

► **Theorem 5.** *There does not exist any algorithm that solves the gathering problem in arbitrary SC-graphs when the agents only have knowledge of n , even when they can observe IDs of agents.*

Proof. For contradiction, suppose that there exists an algorithm \mathcal{A} that solves the gathering problem in the settings of the statement of the theorem. In the proof, we assume $n = 2k + 2$.

Consider a SC-graph G_0 with n sites s_0, \dots, s_{n-1} . Assume that a single carrier c_0 has a route $\pi(c_0) = \langle s_0, \dots, s_{n-1} \rangle$ and k agents a_0, \dots, a_{k-1} exist at sites s_0, \dots, s_{k-1} initially. By algorithm \mathcal{A} , there exists T such that all agents gather and terminate in T rounds. Let $S_{half} = \{s_0, \dots, s_k\}$. The following claim shows that we can change the movement of the carrier so that agents gather and terminate in T rounds without going out from S_{half} .

▷ **Claim 6.** There exists a SC-graph G_1 with n sites s_0, \dots, s_{n-1} such that (1) the single carrier c_1 visits only sites in S_{half} during the first T rounds, and (2) when k agents a_0, \dots, a_{k-1} are initially placed at the same positions as G_0 , they gather and terminate in T rounds.

Proof. First, we introduce some terms. The state of an agent is a tuple of its ID and the values of all variables. A state of a site is a tuple of states of agents if agents exist, or empty otherwise. We say a site is occupied if some agents exist on the site. For G_i ($i \in \{0, 1\}$), we define $\phi_i^t(s)$ as the state of site s at time t in G_i . Let $S = \{s_0, \dots, s_{n-1}\}$.

In the following, we inductively prove that, by defining $\pi(c_1)$ carefully, for $0 \leq t \leq T$, (a) $\pi(c_1)[t] \in S_{half}$ holds, (b) every agent exists on a node in S_{half} at time t , and (c) there exists a bijection $f^t : S \rightarrow S$ that maps site s of G_0 to site $f^t(s)$ of G_1 so that $\phi_1^t(f^t(s)) = \phi_0^t(s)$ holds for any $s \in S$ and $\pi(c_1)[t] = f^t(\pi(c_0)[t])$ holds. Clearly, this implies the claim.

For the base case, consider time 0. We define $\pi(c_1)[0] = \pi(c_0)[0] = s_0 \in S_{half}$. This implies condition (a) for $t = 0$. Since all agents in G_1 are initially placed at the same positions as G_0 , conditions (b) and (c) hold for $t = 0$ by defining $f^0(s) = s$ for any $s \in S$.

To prove inductive cases, for $t = \ell < T$, assume that conditions (a) to (c) hold. That is, $\phi_1^\ell(f^\ell(s)) = \phi_0^\ell(s)$ holds for any $s \in S$ and $\pi(c_1)[\ell] = f^\ell(\pi(c_0)[\ell])$ holds. Let $v_0 = \pi(c_0)[\ell]$, $w_0 = \pi(c_0)[\ell + 1]$, and $v_1 = \pi(c_1)[\ell] = f^\ell(v_0)$. We define $\pi(c_1)[\ell + 1]$ as follows: $\pi(c_1)[\ell + 1] = f^\ell(w_0)$ if $f^\ell(w_0) \in S_{half}$ holds, and otherwise $\pi(c_1)[\ell + 1] = w \in S_{half}$ such that w is not occupied in G_1 at time ℓ . In the latter case, from condition (b), w_0 is not occupied in G_0 at time ℓ , and hence we use some non-occupied site $w \in S_{half}$ instead of $f^\ell(w_0) \notin S_{half}$. From $k < |S_{half}|$, such w definitely exists. This definition implies conditions (a) and (b) for $r = \ell + 1$.

In the following, we prove condition (c) for $t = \ell + 1$. Let $w_1 = \pi(c_1)[\ell + 1]$ and define u_0 as a site satisfying $w_1 = f^\ell(u_0)$. We define bijection $f^{\ell+1}$ as follows:

- If $w_1 = f^\ell(w_0)$ holds, we define $f^{\ell+1}(s) = f^\ell(s)$ for any $s \in S$.
- If $w_1 \neq f^\ell(w_0)$ holds, we define $f^{\ell+1}$ by exchanging sites mapped from w_0 and u_0 , that is, $f^{\ell+1}(w_0) = f^\ell(u_0) = w_1$, $f^{\ell+1}(u_0) = f^\ell(w_0)$, and $f^{\ell+1}(s) = f^\ell(s)$ for any $s \in S \setminus \{w_0, u_0\}$.

In both cases, $w_1 = f^{\ell+1}(w_0)$ and hence $\pi(c_1)[\ell + 1] = f^{\ell+1}(\pi(c_0)[\ell + 1])$ hold.

In the rest, we prove $\phi_1^{\ell+1}(f^{\ell+1}(s)) = \phi_0^{\ell+1}(s)$ for any $s \in S$. We first consider an arbitrary site $x_0 \in S \setminus \{w_0, u_0\}$. Let $x_1 = f^{\ell+1}(x_0) = f^\ell(x_0)$. In this case, x_0 (resp., x_1) is not the destination of a carrier in G_0 (resp., G_1) during the round between time ℓ to $\ell + 1$. If x_0 is not occupied in G_0 at time ℓ , x_1 is also not occupied in G_1 from $\phi_1^\ell(x_1) = \phi_0^\ell(x_0)$. If x_0 is occupied, from $\phi_1^\ell(x_1) = \phi_0^\ell(x_0)$ and $\pi(c_1)[\ell] = f^\ell(\pi(c_0)[\ell])$, agents on x_1 in G_1 and agents on x_0 in G_0 observe the same site state and existence of a carrier, agents on x_1 in G_1 behave as the same as those on x_0 in G_0 . This implies $\phi_1^{\ell+1}(f^{\ell+1}(x_0)) = \phi_1^{\ell+1}(x_1) = \phi_0^{\ell+1}(x_0)$.

Next consider the case of $x_0 = w_0$ (including the case of $u_0 = w_0$). In this case, during the round between time ℓ to $\ell + 1$, carrier c_0 moves from v_0 to w_0 in G_0 , and carrier c_1 moves from $v_1 = f^\ell(v_0)$ to $w_1 = f^{\ell+1}(w_0)$ in G_1 . We can observe $\phi_1^\ell(w_1) = \phi_0^\ell(w_0)$. Indeed this is trivial if $w_1 = f^\ell(w_0)$, and otherwise w_0 and w_1 are not occupied at time ℓ and hence $\phi_1^\ell(w_1) = \phi_0^\ell(w_0)$ holds. Similarly to the previous discussion, agents on v_1 (resp., w_1) in G_1 and those on v_0 (resp., w_0) in G_0 behave similarly. Consequently, carrier c_1 carries agents (if any) to w_1 such that the agents have the same states as G_0 . Hence, $\phi_1^{\ell+1}(f^{\ell+1}(w_0)) = \phi_1^{\ell+1}(w_1) = \phi_0^{\ell+1}(w_0)$ holds.

Lastly, consider the case of $x_0 = u_0 \neq w_0$. In this case, u_0 in G_0 and $f^{\ell+1}(u_0)$ in G_1 are not occupied at time $\ell + 1$. This implies $\phi_1^{\ell+1}(f^{\ell+1}(u_0)) = \phi_0^{\ell+1}(u_0)$. \triangleleft

Next we consider another SC-graph G'_0 with n sites s'_0, \dots, s'_{n-1} . Assume that a single carrier c'_0 has a route $\pi(c'_0) = \langle s'_0, \dots, s'_0, s'_1, s'_2, \dots, s'_{n-1} \rangle$ where c'_0 visits s'_0 repeatedly T times and then visits s'_1, \dots, s'_{n-1} . Also assume that k agents a'_0, \dots, a'_{k-1} exist at sites s'_1, \dots, s'_k initially. Note that the agents do not observe a carrier during the first T rounds. By algorithm \mathcal{A} , there exists T' such that all agents gather and terminate in T' rounds. Let $S'_{half} = \{s'_0, \dots, s'_k\}$. Similarly to Claim 6, we can prove the following claim.

\triangleright **Claim 7.** There exists a SC-graph G'_1 with n sites s'_0, \dots, s'_{n-1} such that (1) the single carrier c'_1 visits only sites in S'_{half} during the first T' rounds, and (2) when k agents a'_0, \dots, a'_{k-1} are initially placed at the same positions as G'_0 , they do not observe a carrier during the first T rounds, and gather and terminate in T' rounds.

Algorithm 1 Gather-With-Number-Of-Agents.

```

1:  $k \leftarrow$  total number of agents
2: for each round do
3:    $agents \leftarrow$  the number of agents at the same site,  $carrier \leftarrow$  carrier at the same site
4:   if  $agents = k$  then
5:     Terminate
6:   end if
7:   if  $carrier \neq \emptyset$  then
8:     Move with  $carrier$ 
9:   end if
10: end for

```

Lastly, we construct a SC-graph G_2 from the above two graphs G_1 and G'_1 . The set of sites of G_2 is $S_{half} \cup S'_{half}$. Assume that initially k agents a_0, \dots, a_{k-1} are located at sites s_0, \dots, s_{k-1} like G_1 and other k agents a'_0, \dots, a'_{k-1} are located at sites s'_1, \dots, s'_k like G'_1 . We consider a single carrier c_2 such that c_2 moves similarly to G_1 during the first T rounds and moves similarly to G'_1 after that up to round T' (and it can move arbitrarily). That is, $\pi(c_2)[j] = \pi(c_1)[j]$ for $0 \leq j < T$ and $\pi(c_2)[j] = \pi(c'_1)[j]$ for $T \leq j < T'$. Let us consider the behavior of agents a_0, \dots, a_{k-1} in graph G_2 . During the first T rounds, carrier c_2 moves similarly to G_1 and consequently agents a_0, \dots, a_{k-1} behave similarly to G_1 . Hence, agents a_0, \dots, a_{k-1} terminate at a node in S_{half} in T rounds. Next consider the behavior of agents a'_0, \dots, a'_{k-1} . During the first T rounds, none of them observes a carrier, which is the same as G'_1 . During time T to T' , carrier c_2 moves similarly to G'_1 . Hence, agents a'_0, \dots, a'_{k-1} behave similarly to G'_1 and terminate at a node in S'_{half} in T' rounds. This implies that agents terminate at different two sites s and s' . This is a contradiction. \blacktriangleleft

3.2 Possibilities

In this subsection, we present algorithms for SC-graphs for several scenarios on the initial knowledge of agents and/or the information acquired in the LOOK operations.

3.2.1 With Prior Knowledge of agents amount k

We start with a simple case where each agent knows the number k of agents. This allows the agents to determine when to terminate as they can observe the count of agents at the same site. Each agent moves with a carrier whenever it encounters the carrier, and terminates once it can see k agents.

► **Theorem 8.** *Algorithm 1 solves the gathering problem with simultaneous termination within p rounds for arbitrary SC-graphs if agents know the total number k of agents.*

Proof. In this algorithm, any agent moves with the carrier whenever the carrier arrives at its current site. Through this process, all k agents inevitably gather in p rounds and terminate simultaneously, as a consequence of lines 5, 6, 8, 9, and Lemma 3. \blacktriangleleft

3.2.2 With Prior Knowledge of Period p

We present the algorithm when the agents initially know an upper bound of the period of a given SC-graph in Algorithm 2. Let P denote an upper bound on the period p of the single carrier. The idea of the algorithm is simple: The agents maintain a variable *steps* initially set to 0, and increment the variable in each round. When the carrier appears in the

Algorithm 2 Gather-With-Period.

```

1:  $P \leftarrow$  an upper bound on the carrier's period,  $steps \leftarrow 0$ ,  $pre-agents \leftarrow 1$ 
2: for each round do
3:    $agents \leftarrow$  the number of agents at the same site,  $carrier \leftarrow$  carrier at the same site
4:   if  $steps = P$  then
5:     Terminate
6:   else if  $agents = pre-agents$  then
7:      $steps \leftarrow steps + 1$ 
8:   else
9:      $steps \leftarrow 0$ 
10:  end if
11:   $pre-agents \leftarrow agents$ 
12:  if  $carrier \neq \emptyset$  then
13:    Move with  $carrier$ 
14:  end if
15: end for

```

current site, the agent moves sites along the carrier. The value of $steps$ is reset to 0 when the number of agents in the carrier increases, which is detected by comparing the number of agents at the current round and the number of agents in the previous round. The number of agents in the previous round is maintained by another variable $pre-agents$. Finally, the agents terminate when the value of $steps$ is equal to the value of the upper bound of the period. The algorithm finishes gathering within $p + P$ rounds.

► **Lemma 9.** *All the agents operating Algorithm 2 are located at the same site and have identical $steps$ values at the end of any round $t \geq p$.*

Proof. As illustrated in lines 15 and 16, an agent moves with the carrier upon encounter and remains on the carrier until termination. From this observation, we can see that all agents are present at the same site after round p , by Lemma 3. Let $t_0 (\leq p)$ be the round when the carrier encounters the last agent. Each agent operates $steps \leftarrow 0$ at round t_0 , and increments it by one in each round after round t_0 . That is all the agents have identical $step$ values at the end of each round $t \geq p > t_0$. ◀

► **Theorem 10.** *Algorithm 2 solves the gathering problem for arbitrary SC-graphs within $p + P$ rounds if agents know an upper bound P of the period p of the carrier.*

Proof. In the case of $k = 1$, the variable $steps$ of the only agent increases every round until $steps = P$. Then the agent terminates at the P -th round. In the case of $k > 1$, all k agents congregate at the same location on the carrier with identical $steps$ values by the end of any round $t \leq p$ from Lemma 9. Hence, at a certain round $r \leq p + P$, all k agents terminate their execution at the same site simultaneously. ◀

Algorithm 2 can solve the problem when the initial knowledge of the agents is n (the number of sites) in circular SC-graphs. This is because we have $n = p$ by the definition of the circular route (in Definition 2). Therefore, by substituting n for p , the agents can gather within $2p$ rounds.

► **Corollary 11.** *The gathering problem can be solved for circular SC-graphs within $2p$ rounds if agents know the number n of sites.*

In simple SC-graphs, the agent can obtain an upper bound $n(n - 1)$ of the period, since $p \leq n(n - 1)$ holds from the definition of the simple route (in Definition 1). Therefore, the agents can gather within $p + n(n - 1)$ rounds.

Algorithm 3 Gather-With-Site-ID-and- n .

```

1:  $n \leftarrow$  total number of sites,  $sites \leftarrow \emptyset$ ,  $explore-mode \leftarrow true$ 
2: for each round do
3:    $current \leftarrow$  current site ID,  $sites \leftarrow \{current\} \cup sites$ ,  $carrier \leftarrow$  carrier at the same site
4:   if  $|sites| = n$  then
5:      $explore-mode \leftarrow false$ 
6:   end if
7:   if  $carrier \neq \emptyset$  then
8:     if  $explore-mode = false \wedge current = \min(sites)$  then
9:       Terminate
10:    else
11:      Move with  $carrier$ 
12:    end if
13:  end if
14: end for

```

► **Corollary 12.** *The gathering problem can be solved for simple SC-graphs within $p + n(n - 1)$ rounds if agents know the number n of sites.*

The same strategy also can be applied to the situation where agents can observe the ID of the current site in each LOOK operation in circular and simple SC-graphs. In this case, agents find the period p while executing Algorithm 2. Each agent checks site IDs and the number of moves when it moves with the carrier. When it encounters the same site (arc in the case of simple SC-graphs) twice, it detects the end of the first cycle and gets the period p . Let $t_0(\leq p)$ be the round when the carrier encounters the last agent. The value of *steps* reaches to p at round $t_0 + p \leq 2p$, while the last agent gets the value p at round $t_0 + p$ in circular SC-graphs and at $t_0 + p + 1$ in simple SC-graphs. The strategy can work well if agents decide termination when *steps* value becomes $p + 1$ in circular and simple SC-graphs. The modified algorithm solves the gathering problem within $2p$ and $2p + 1$ rounds in circular and simple SC-graphs.

► **Corollary 13.** *The gathering problem can be solved for circular and simple SC-graphs within $2p$ and $2p + 1$ rounds if agents can observe the current site's ID.*

Algorithms derived from Algorithm 2 also solve the problem *with simultaneous termination* since Lemma 9 guarantees that the *steps* value is identical for all the agents before terminating.

3.2.3 With Prior Knowledge of n and Ability to Observe Site ID

Another straightforward approach directs the agents to gather at the site with the smallest ID. Agents determine the minimum site ID while exploring the entire graph with knowledge of the total number n of sites and the ability to observe the current site's ID.

► **Theorem 14.** *Algorithm 3 solves the gathering problem for arbitrary SC-graphs within $2p$ rounds if agents can obtain the current site's ID and know the total number n of sites.*

Proof. In this algorithm, the operation is organized into two distinct phases denoted by the parameter *explore-mode*. Initially, each agent operates in *explore-mode*, where the agent explores the entire graph with the carrier. The agent exits *explore-mode* by round p when all sites in the graph have been visited, as indicated by the condition $|sites| = n$. Subsequently, the agent moves with the carrier up to p rounds toward the site with the smallest ID, where it finally terminates. Since all the agents terminate the same site with the smallest ID, the gathering is achieved within $2p$ rounds. ◀

Algorithm 4 Gather-With-Agent-ID-Circular.

```

1:  $leader \leftarrow \perp$ ,  $landmark \leftarrow \perp$ ,  $leader-acknowledged \leftarrow false$ 
2: for each round do
3:    $agents \leftarrow \{\text{IDs at the same site}\}$ ,  $carrier \leftarrow \text{carrier at the same site}$ 
4:   if  $carrier \neq \emptyset$  then
5:     if  $leader = \perp \vee \min(agents) < leader$  then
6:        $leader \leftarrow \min(agents)$ ,  $leader-acknowledged \leftarrow false$ 
7:     end if
8:     if  $leader = id$  then
9:       if  $landmark \neq \perp \wedge landmark \in agents$  then ▷ found the landmark
10:        Terminate
11:       else if  $landmark = \perp \wedge |agent| \geq 2$  then
12:          $landmark \leftarrow \min(agents \setminus \{id\})$ 
13:       end if
14:       Move with  $carrier$ 
15:     else if  $leader-acknowledged = false \wedge leader \in agents$  then
16:        $leader-acknowledged \leftarrow true$ 
17:       Stay at the site ▷ start to wait for a leaderless carrier
18:     else if  $leader \notin agents$  then
19:       Move with  $carrier$  ▷ head to the leader
20:     else
21:       Terminate ▷ found the leader
22:     end if
23:   end if
24: end for

```

Algorithm 3 does not attain *gathering with simultaneous termination* due to the lack of information about other agents' exploration progress.

3.2.4 With Ability to Observe Other Agents' ID

An alternative approach involves designating a special agent as a leader, who terminates first. Then, the remaining agents move to and terminate at the leader's position. To determine the leader agent, agents observe each other's unique ID. The leader's termination serves as a signal for the other agents to gather at the designated position.

The algorithm for gathering in circular SC-graphs relies on the election of two special agents, namely the *leader* and the *landmark*. All agents will learn the leader's ID when the election is over since the leader will meet every other agent during the leader election process. The leader then moves to the location of the landmark and terminates with the landmark. Subsequently, other agents observing a carrier without a leader will recognize that it is time to head toward the leader. They will terminate upon encountering the leader again. The following theorem holds for Algorithm 4. The proof is in the Appendix.

► **Theorem 15.** *Algorithm 4 solves the gathering problem within $3p$ rounds for circular SC-graphs when $k > 1$, if agents can obtain the IDs of other agents at the same site.*

Proof. We show that Algorithm 4 makes all agents gather within $3p$ rounds. The proof proceeds with showing some claims.

▷ **Claim 16.** There are no terminated agents at round $t < p$.

21:12 Gathering in Carrier Graphs: Meeting via Public Transportation System

Proof. If an agent terminates as a leader (line 10), it first sets some ID to *landmark* variable and meets again its landmark to terminate. If an agent terminates as non-leader (line 21), it first meets with the leader and meets the leader again to terminate. Both cases require agents to encounter the carrier at least twice. To encounter the carrier twice, any agent needs $p + 1$ rounds. \triangleleft

▷ **Claim 17.** By round p , the agents riding the carrier include the agent with the smallest agent ID, and it moves with the carrier until termination.

Proof. In the first cycle of the carrier's move, every agent first encounters the carrier and moves with the carrier as a leader if it has the smallest ID among agents at the site (line 14). This implies the agent a_{min} with the smallest ID becomes a leader (sets its own *id* to *leader* variable) when it first encounters the carrier at round t_0 ($\leq p$) and solely moves with the carrier until termination at round $p + 1$ or later by Claim 16. \triangleleft

We then show that a_{min} again meets its landmark by round $2p$.

▷ **Claim 18.** The agent a_{min} arrives at the landmark's site by round $t = 2p$.

Proof. When a_{min} first meets other agents, it sets one agent ID a_{land} to its *landmark*. We first show that a_{min} first meets a_{land} by round p . When a_{min} first encounters the carrier, if there are other agents at the same site, a_{min} selects a_{land} among these agents, that is, a_{min} first meets a_{land} by p . Otherwise, since one agent moves with the carrier after the carrier encounters some agents, this means that a_{min} is the first agent that the carrier encounters and will meet a_{land} at its initial location by round p . Let t ($\leq p$) be the round when a_{min} meets a_{land} first time. Since a_{land} does not become a leader, it remains to stay at the site. Since a_{min} moves with the carrier until termination by Claim 17, it again meets a_{land} at time $t + p$ ($\leq 2p$). \triangleleft

After reaching the landmark's location, the leader terminates its execution. Subsequently, any agents situated at distinct sites observe the leaderless carrier and move along it. Following an additional p rounds from the leader's termination, the carrier returns to the leader's site, bringing all agents to this site.

In summary, the algorithm guarantees the gathering of all agents to a solitary site. This gathering is accomplished within $3p$ rounds for circular SC-graphs. \blacktriangleleft

The leader election process on circular SC-graphs does not work for simple SC-graphs as it is, since a site may appear multiple times along a simple route. Fortunately, a simple route includes no repeated arcs. We can use an arc as a landmark. Algorithm 5 for simple SC-graphs has a similar approach to that for circular SC-graphs. It elects the smallest agent as the leader, and the leader sets *landmark* by placing two agents *pre-landmark* and *post-landmark* at both endpoints of the landmark when it first meets two other agents. The leader's termination is triggered upon observing *pre-landmark* and *post-landmark* in this order in consecutive two rounds.

In Algorithm 5, two variables *pre-landmark* and *post-landmark* are used to indicate a landmark. An agent sets a landmark when first meets two other agents. To do so, agents move with the carrier while there are two or fewer agents together before setting a landmark (while *leader-acknowledged* is false). The leader terminates if it again meets *pre-landmark* and *post-landmark* in this order in consecutive rounds. Other agents behave similarly to Algorithm 4. They stay at the site to wait for a leaderless carrier, but if it is selected as a post-landmark (*leader-acknowledged* is false and it has the second smallest ID among agents

Algorithm 5 Gather-With-Agent-ID-Simple.

```

1: leader  $\leftarrow \perp$ , pre-landmark  $\leftarrow \perp$ , post-landmark  $\leftarrow \perp$ 
2: leader-acknowledged  $\leftarrow false$ , visit-pre-landmark  $\leftarrow false$ , post-landmark-move  $\leftarrow false$ 
3: for each round do
4:   agents  $\leftarrow$  {IDs at the same site}, carrier  $\leftarrow$  carrier at the same site
5:   if carrier  $\neq \emptyset$  then
6:     if leader =  $\perp \vee \min(\text{agents}) < \text{leader}$  then
7:       leader  $\leftarrow \min(\text{agents})$ , leader-acknowledged  $\leftarrow false$ 
8:     end if
9:     if leader = id then
10:      if pre-landmark  $\neq \perp \wedge \text{pre-landmark} \in \text{agents}$  then
11:        visit-pre-landmark  $\leftarrow true$ 
12:      else if visit-pre-landmark then
13:        if post-landmark  $\in \text{agents}$  then ▷ found the landmark
14:          Terminate
15:        else
16:          visit-pre-landmark  $\leftarrow false$ 
17:        end if
18:      else if pre-landmark =  $\perp \wedge |\text{agents}| \geq 3$  then
19:        pre-landmark  $\leftarrow$  3rd-min(agents), post-landmark  $\leftarrow$  2nd-min(agents)
20:      end if
21:      Move with carrier
22:    else if leader-acknowledged = false  $\wedge |\text{agents}| \geq 3 \wedge \text{leader} \in \text{agents}$  then
23:      leader-acknowledged  $\leftarrow true$ 
24:      if 2nd-min(agents) = id then
25:        post-landmark-move  $\leftarrow true$ 
26:        Move with carrier
27:      else
28:        Stay at the site ▷ start to wait for a leaderless carrier
29:      end if
30:    else if post-landmark-move = true then
31:      post-landmark-move  $\leftarrow false$ 
32:      Stay at the site ▷ start to wait for a leaderless carrier
33:    else if leader-acknowledged = false  $\vee \text{leader} \notin \text{carrier}$  then
34:      Move with carrier ▷ elect or head to the leader
35:    else
36:      Terminate ▷ found the leader
37:    end if
38:  end if
39: end for

```

in the current site) it has one more move. After the leader terminates, the other agents see a leaderless carrier and head to the leader with the carrier. The following theorem holds for Algorithm 5.

► **Theorem 19.** *Algorithm 5 solves the gathering problem within $4p - 1$ rounds for simple SC-graphs when $k > 2$ if agents can obtain the IDs of other agents at the same site.*

Proof. We will see a difference from Algorithm 4. Let a_{min} denote the agent with the smallest ID.

▷ **Claim 20.** The agent a_{min} arrives at the landmark's arc by round $3p - 1$.

Proof. When a_{min} first meets two other agents, it sets two agent IDs a_{pre_land} and a_{post_land} to its *pre-landmark* and *post-landmark*, respectively. We first show that a_{min} first meets a_{pre_land} and a_{post_land} by round $2p - 2$. The agent a_{min} meets at least one agent by round p

21:14 Gathering in Carrier Graphs: Meeting via Public Transportation System

as in Algorithm 4. However, if a_{min} meets only one agent a' at that time, it has to move more with the carrier to meet more agents (a' accompanies a_{min} since its *leader-acknowledged* is *false*). The worst case is that a_{min} is initially located at $\pi(c)[p-1]$, and one agent is located at $\pi(c)[0]$ and the other agents are located at $\pi(c)[p-3]$. In this case, The second-smallest agent a' once set its landmark at $(\pi(c)[p-3], \pi(c)[p-2])$ at rounds $p-2$ and $p-3$ and all the agents except a_{min} and a' stay one of the endpoints. Thus, a_{min} takes another $p-2$ moves to meet the pre-landmark of a' at $\pi(c)[p-3]$ and sets a_{min} 's landmark at $(\pi(c)[p-3], \pi(c)[p-2])$ at round $2p-2$ and $2p-3$. Then a_{min} takes another p moves to again meet its *pre-landmark* and *post-landmark* in this order. That is, a_{min} again arrives at the landmark's arc at time $3p-1$. \triangleleft

After reaching the landmark's location, the leader terminates its execution. Subsequently, any agents situated at distinct sites observe the leaderless carrier and move along it. Following an additional p rounds from the leader's termination, the carrier returns to the leader's site, bringing all agents to this site.

In summary, the algorithm guarantees the gathering of all agents to a solitary site. This gathering is accomplished within $4p-1$ rounds for circular SC-graphs. \blacktriangleleft

Given the inherently asynchronous nature of the termination process, achieving the goal of *gather with simultaneous termination* remains unattainable.

4 Gathering on Multi-Carrier C-Graphs

4.1 Impossibility

According to Flocchini et al. [6], exploration on anonymous C-graphs (sites' IDs not observable) is impossible without knowledge of P , even if n and k are given, and is impossible without knowledge of either P or n when the site ID is visible, even if k is given. Thus, gathering on anonymous C-graphs is impossible without prior knowledge of P as shown by Theorem 21.

► **Theorem 21.** *There is no algorithm that solves the problem of gathering on C-graphs that agents terminate before every site is visited.*

Proof. Assume algorithm \mathcal{A} solves the gathering problem without making all agents visit every site. Let G_1 be a C-graph that a site $s_1 \in S(G_1)$ will not be visited by an agent A_1 operating \mathcal{A} starting from a certain initial configuration. Similarly, let G_2 be a copy of G_1 so that an agent A_2 will not visit a site $s_2 \in S(G_2)$ in an execution starting from the same initial configuration as before. Then, let G_3 be a C-graph that is a combination of G_1 and G_2 . That is, $S(G_3)$ has all the carriers and sites of G_1 and G_2 , and all the carriers and agents are located at the same sites as before at the round 0. Let a carrier c move periodically between s_1 and s_2 , so that the C-graph G_3 becomes connected. Because A_1 will terminate without visiting s_1 and so for A_2 and s_2 , the two agents will never meet. This is a contradiction. \blacktriangleleft

In non-anonymous C-graphs, solving the gathering problem is trivial because agents must explore, as indicated by Theorem 21. This exploration leads agents to move to the site with the smallest ID.

Algorithm 6 Gather-With-Multiple-Carriers.

- 1: For each agent i , explores and maps the graph $\vec{G}(C_i)$ by \mathcal{A}_E
 - 2: $p \leftarrow p(C_i)$, $m \leftarrow |C_i|$, $n \leftarrow |S(C_i)|$
 - 3: $destination \leftarrow \min(C_i)$
 - 4: Compute the foremost path to $destination$ then move to it
 - 5: Wait until round $M + (2p - 2)(m - 1)$
 - 6: Wait until an agent comes for at most M rounds
 - 7: Operate Algorithm 2 with period p
-

4.2 Gathering on Anonymous C-graphs

When gathering on an anonymous C-graph with multiple carriers, we adopt a two-tiered approach. Initially, agents gather in the meeting graph $H(C)$, which means arriving at the same carrier's route. Then, agents use algorithms designed for SC-graphs to gather on that carrier. If agents have learned about the identities (IDs), routes, and timetables of all carriers, as well as the topology of the C-graph, agents can directly gather at the carrier with the smallest ID with at most $(2p - 2)(m - 1)$ rounds as Lemma 22 shows. Another $2p$ round is then required for operating Algorithm 2 to gather all the agents on the same site since agents know the period of the carrier of the smallest ID.

► **Lemma 22.** *The foremost path (the path that an agent arrives at its destination at the earliest time) from any site to any carrier costs at most $(2p - 2)(m - 1)$ rounds.*

Proof. The basic scenario involves two connected carriers c_1 and c_2 . Since there are at most p sites in each carrier, the longest path from c_1 to c_2 is $p - 1$. In the worst case, the agent at a shared site needs to wait for the carrier's coming for $p - 1$ rounds. Therefore, the accumulative time for the foremost path is at most $2p - 2$.

Since the longest foremost path involves at most m carriers with $m - 1$ times carrier switching, the foremost path costs at most $(2p - 2)(m - 1)$ rounds. ◀

For an unknown C-graph, agents can learn the required knowledge by assigning names to the visited sites so that each agent will make a private map of the C-graph after visiting every site. Suppose we have an algorithm \mathcal{A}_E that maps the C-graph in M rounds, the gathering problem can be solved in at most $M + (2p - 2)(m - 1) + 2p$ rounds.

► **Theorem 23.** *The Algorithm 6 solves the gathering problem for C-graphs in $2M + (2p - 2)(m - 1) + 2p$ rounds with a mapping algorithm \mathcal{A}_E , which maps the C-graph in M rounds.*

Proof. The topology and the dynamics are known to the agent by round M as the exploration ends. Therefore, every agent will be at a site belonging to the route of the carrier with the smallest ID by the $M + (2p - 2)(m - 1)$ -th round. That allows Algorithm 2 to solve the gathering problem in another $2p$ rounds.

In a scenario where each agent starts up at the beginning of round 0, agents agree on the global time so that every agent begins Algorithm 2 at round $2M + (2p - 2)(m - 1)$ that ensures the gathering at round $2M + (2p - 2)(m - 1) + 2p$.

Otherwise, when each agent starts up separately, there is a delay d between the first and last started agents. As we suppose an agent will start its gathering process after encountering another moving agent, the delay d can be at most M rounds. The Line 6 eliminates the effects of the delay and leads to a gathering time of $2M + (2p - 2)(m - 1) + 2p$ rounds. ◀

Ilcinkas and Wade [11] propose an exploration algorithm EXPLORE-WITH-WAIT that maps a C-graph. Given the a priori knowledge of an upper bound $B = \mathcal{O}(p)$ on the maximum period p , the worst-case time complexity is $\Theta(np)$. Subsequently, the time complexity of a EXPLORE-WITH-WAIT version of Algorithm 6 is $\Theta(np)$ as $m = \mathcal{O}(n)$.

5 Conclusion

In this study, we started an exploration of a variant of the gathering problem: gathering in carrier graphs, a particular class of time-varying graphs.

Throughout our investigation, we analyzed several factors that affect the feasibility and the time complexity of gathering in single carrier graphs. Then, we extended our algorithms to solve the gathering problem in general carrier graphs.

Open questions remaining include exploring the impact of communication abilities on the gathering problem in C-graphs, identifying additional factors that may influence the feasibility of gathering, and extending our findings to encompass a wider array of dynamic graph classes, such as bounded-recurrent-edge graphs.

References

- 1 Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55 of *International series in operations research and management science*. Kluwer, 2003. doi:10.1007/b10080.
- 2 Marjorie Bournat, Swan Dubois, and Franck Petit. Gracefully degrading gathering in dynamic rings. In *Stabilization, Safety, and Security of Distributed Systems - 20th International Symposium*, volume 11201 of *Lecture Notes in Computer Science*, pages 349–364. Springer, 2018. doi:10.1007/978-3-030-03232-6_23.
- 3 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *Int. J. Parallel Emergent Distributed Syst.*, 27(5):387–408, 2012. doi:10.1080/17445760.2012.668546.
- 4 Yoann Dieudonné, Andrzej Pelc, and David Peleg. Gathering despite mischief. *ACM Trans. Algorithms*, 11(1):1:1–1:28, 2014. doi:10.1145/2629656.
- 5 Paola Flocchini, Matthew Kellett, Peter C. Mason, and Nicola Santoro. Finding good coffee in paris. In Evangelos Kranakis, Danny Krizanc, and Flaminia L. Luccio, editors, *Fun with Algorithms - 6th International Conference*, volume 7288 of *Lecture Notes in Computer Science*, pages 154–165. Springer, 2012. doi:10.1007/978-3-642-30347-0_17.
- 6 Paola Flocchini, Bernard Mans, and Nicola Santoro. On the exploration of time-varying networks. *Theoretical Computer Science*, 469:53–68, 2013. doi:10.1016/j.tcs.2012.10.029.
- 7 Paola Flocchini, Nicola Santoro, Peter C. Mason, and Matthew Kellett. Black hole search in the network and subway models. *Theoretical Computer Science*, 50:158–184, 2012. doi:10.1007/s00224-011-9341-8.
- 8 Tsuyoshi Gotoh, Paola Flocchini, Toshimitsu Masuzawa, and Nicola Santoro. Tight bounds on distributed exploration of temporal graphs. In Pascal Felber, Roy Friedman, Seth Gilbert, and Avery Miller, editors, *23rd International Conference on Principles of Distributed Systems*, volume 153 of *LIPICs*, pages 22:1–22:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.OPODIS.2019.22.
- 9 Tsuyoshi Gotoh, Yuichi Sudo, Fukuhito Ooshita, and Toshimitsu Masuzawa. Dynamic ring exploration with (h, S) view. *Algorithms*, 13(6):141, 2020. doi:10.3390/A13060141.
- 10 Jion Hirose, Junya Nakamura, Fukuhito Ooshita, and Michiko Inoue. Gathering with a strong team in weakly byzantine environments. In *Proceedings of the 22nd International Conference on Distributed Computing and Networking*, ICDCN '21, pages 26–35. Association for Computing Machinery, 2021. doi:10.1145/3427796.3427799.
- 11 David Ilcinkas and Ahmed M. Wade. Exploration of carrier-based time-varying networks: The power of waiting. *Theoretical Computer Science*, 841:50–61, 2020. doi:10.1016/j.tcs.2020.07.003.
- 12 Ari Keränen and Jörg Ott. DTN over aerial carriers. In Martin May, Gunnar Karlsson, and Ellen W. Zegura, editors, *Proceedings of the 4th ACM workshop on Challenged networks*, pages 67–76. ACM, 2009. doi:10.1145/1614222.1614234.

- 13 Cong Liu and Jie Wu. Scalable routing in cyclic mobile networks. *IEEE Trans. Parallel Distributed Syst.*, 20(9):1325–1338, 2009. doi:10.1109/TPDS.2008.218.
- 14 Giuseppe Antonio Di Luna, Paola Flocchini, Linda Pagli, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Gathering in dynamic rings. *Theor. Comput. Sci.*, 811:79–98, 2020. doi:10.1016/J.TCS.2018.10.018.
- 15 Othon Michail, Paul G. Spirakis, and Michail Theofilatos. Beyond rings: Gathering in 1-interval connected graphs. *Parallel Process. Lett.*, 31(4):2150020:1–2150020:31, 2021. doi:10.1142/S0129626421500201.
- 16 Fukuhito Ooshita and Ajoy K. Datta. Brief announcement: Feasibility of weak gathering in connected-over-time dynamic rings. In *Stabilization, Safety, and Security of Distributed Systems - 20th International Symposium*, volume 11201 of *Lecture Notes in Computer Science*, pages 393–397. Springer, 2018. doi:10.1007/978-3-030-03232-6_27.
- 17 Andrzej Pelc. Deterministic rendezvous algorithms. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 423–454. Springer, 2019. doi:10.1007/978-3-030-11072-7_17.
- 18 Masahiro Shibata, Yuichi Sudo, Junya Nakamura, and Yonghwan Kim. Partial gathering of mobile agents in dynamic rings. In Colette Johnen, Elad Michael Schiller, and Stefan Schmid, editors, *Stabilization, Safety, and Security of Distributed Systems - 23rd International Symposium*, volume 13046 of *Lecture Notes in Computer Science*, pages 440–455. Springer, 2021. doi:10.1007/978-3-030-91081-5_29.
- 19 Shin-Ywan Wang, J. Leigh Torgerson, Joshua Schoolcraft, and Yan Brenman. The deep impact network experiment operations center monitor and control system. In *2009 Third IEEE International Conference on Space Mission Challenges for Information Technology*, pages 34–40, 2009. doi:10.1109/SMC-IT.2009.13.
- 20 Xiaolan Zhang, Jim Kurose, Brian Neil Levine, Donald F. Towsley, and Honggang Zhang. Study of a bus-based disruption-tolerant network: mobility modeling and impact on routing. In Evangelos Kranakis, Jennifer C. Hou, and Ram Ramanathan, editors, *Proceedings of the 13th Annual International Conference on Mobile Computing and Networking*, pages 195–206. ACM, 2007. doi:10.1145/1287853.1287876.
- 21 Wenrui Zhao and M.H. Ammar. Message ferrying: proactive routing in highly-partitioned wireless ad hoc networks. In *The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 308–314, 2003. doi:10.1109/FTDCS.2003.1204352.


Brief Announcement: Collision-Free Robot Scheduling*

Duncan Adamson 

Leverhulme Research Centre for Functional Materials Design, University of Liverpool, UK

Nathan Flaherty 

Leverhulme Research Centre for Functional Materials Design, University of Liverpool, UK

Igor Potapov  

Department of Computer Science, University Of Liverpool, UK

Paul G. Spirakis  

Department of Computer Science, University Of Liverpool, UK

Abstract

Robots are becoming an increasingly common part of scientific work within laboratory environments. In this paper, we investigate the problem of designing *schedules* for completing a set of tasks at fixed locations with multiple robots in a laboratory. We represent the laboratory as a graph with tasks placed on fixed vertices and robots represented as agents, with the constraint that no two robots may occupy the same vertex, or traverse the same edge, at the same time. Each schedule is partitioned into a set of timesteps, corresponding to a walk through the graph (allowing for a robot to wait at a vertex to complete a task), with each timestep taking time equal to the time for a robot to move from one vertex to another and each task taking some given number of timesteps during the completion of which a robot must stay at the vertex containing the task. The goal is to determine a set of schedules, with one schedule for each robot, minimising the number of timesteps taken by the schedule taking the greatest number of timesteps within the set of schedules.

We show that the problem of finding a task-fulfilling schedule in at most L timesteps is NP-complete for many simple classes of graphs. Explicitly, we provide this result for complete graphs, bipartite graphs, star graphs, and planar graphs. Finally, we provide positive results for line graphs, showing that we can find an optimal set of schedules for k robots completing m tasks of equal length of a path of length n in $O(kmn)$ time, and a k -approximation when the length of the tasks is unbounded.

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

Keywords and phrases Graph Exploration, Scheduling, NP-Completeness, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.22

Related Version *Full Version:* <https://arxiv.org/abs/2402.12019> [1]

1 Introduction

In this paper, we are interested in the scheduling of robots within chemistry labs, motivated by a significant and expanding body of work concerning robotic chemists. Initial work on these systems focused on building robots performing reactions within fixed environments [4, 3], however recently Burger et al. [2] have presented a robot capable of moving within a laboratory and completing tasks throughout the space. The works of Burger et al. [2] and Liu et al. [5] provide the main motivation for this work, namely the problem of moving robots within a laboratory environment (as presented by Burger et al. [2]) while avoiding collisions (as investigated in the manufacturing context by Liu et al. [5]).

* A full version of this paper is available on arXiv [1]



2 Preliminaries

In this problem, we consider a set of agents, which we call *robots*, moving on a given graph $G = (V, E)$ and completing a set of tasks $\mathcal{T} = \{t_1, t_2, \dots, t_m\}$. As mentioned in our introduction, this problem originates in the setting of lab spaces, particularly in the chemistry setting. As such, our definitions of robots and tasks are designed to mimic those found in real-world problems. We associate each task with a vertex on which it is located and the duration required to complete the task. We do not allow tasks to be moved by a robot, a task can only be completed by a single robot remaining at the station for the entire task duration, and any robot may complete any number of tasks, with no restrictions on which task a robot can complete. This requirement reflects the motivation from chemistry, where tasks reflect reactions that must be done within an exact time frame and at a fixed workstation.

Formally, we define a task t_i as a tuple (d_i, v_i) where d_i is the *duration* of the task, and v_i is the vertex at which the task is located. We use $|t_i|$ to denote the duration of the task t_i . In general, the reader may assume that for a graph $G = (V, E)$ containing the vertex set $V = \{v_1, v_2, \dots, v_n\}$, the notation i_t is used to denote the index of the vertex at which task $t = (d, v_{i_t})$ is located. This will be specified throughout the paper where relevant.

To complete tasks, we assign each robot a *schedule*, composed of an alternating sequence of walks and tasks. We note that each schedule can begin and end with either a walk and a walk, a walk and a task, a task and a walk, or a task and a task. We treat each schedule as a set of commands to the robot, directing it within a given time frame. In this way, we partition the schedule into a set of time steps, with each time step allowing a robot to move along one edge or complete some fraction of a task, with a task t requiring exactly $|t|$ time steps to complete. We call the time span of a schedule the total number of timesteps required to complete it. We denote the time span of the schedule C containing the walks w_1, w_2, \dots, w_ℓ and tasks t_1, t_2, \dots, t_m by $|C| = \left(\sum_{i \in [1, \ell]} |w_i|\right) + \left(\sum_{j \in [1, m]} |t_j|\right)$. Given a walk w directly following the task t in the schedule C , we require that the first edge traversed in w begins at the vertex v_{i_t} on which t is located. Similarly, we require that the task t' following the walk w' in the schedule C is located on the last vertex in the last edge in w' .

The *walk representation* $\mathcal{W}(C)$ of a schedule C is an ordered sequence of edges formed by replacing the task $t_i = (d, v_i)$ in C with a walk of length $|t_i| = d$ consisting only of the edge (v_i, v_i) , then concatenate the walks together in order. Note that $|\mathcal{W}(C)| = |C|$. For a given robot R assigned schedule C , in timestep j R is located on the vertex $v \in V$ that is the end vertex of the i^{th} edge in $\mathcal{W}(C)$, i.e. the vertex v such that $\mathcal{W}(C)[i] = (u, v)$. We require the first vertex in the walk representation of any schedule C assigned to robot R to be the *starting vertex* of R , i.e. some predetermined vertex representing where R starts on the graph. If the schedule C containing the task t is assigned to robot R , we say that t is *assigned* to R .

Given a set of schedules $\mathcal{C} = (C_1, C_2, \dots, C_k)$ for a set of k robots R_1, R_2, \dots, R_k , and set of tasks $\mathcal{T} = (t_1, t_2, \dots, t_m)$. we say that \mathcal{C} is *task completing* if for every task $t \in \mathcal{T}$ there exists exactly one schedule C_i such that $t \in C_i$. We call \mathcal{C} *collision-free* if there is no timestep where any pair of robots occupy the same vertex or traverse the same edge. Formally, \mathcal{C} is collision-free if, for every C_i, C_j where $i \neq j$ and time-step $s \in [1, |C_i|]$, $\mathcal{W}(C_i)[s] = (v, u)$ and $\mathcal{W}(C_j)[s] = (v', u')$ satisfies $u \neq u'$ and $(v, u) \neq (u', v')$.

For the remainder of this paper, we assume every robot in the graph is assigned exactly 1 schedule. Given 2 sets of schedules \mathcal{C} and \mathcal{C}' , we say \mathcal{C} is *faster* than \mathcal{C}' if $\max_{C_i \in \mathcal{C}} |C_i| < \max_{C'_j \in \mathcal{C}'} |C'_j|$. Given a graph $G = (V, E)$, a set of k robots R_1, R_2, \dots, R_k starting on vertices sv_1, sv_2, \dots, sv_k , and a set of tasks \mathcal{T} , a *fastest* task-completing, collision-free set of k schedules is the set of schedules \mathcal{C} such that any other set of task-completing, collision-free schedules is no faster than \mathcal{C} . Note that there may be multiple such sets of schedules.

► **Problem 1** (k -ROBOT SCHEDULING). Given a graph $G = (V, E)$, set of k robots R_1, R_2, \dots, R_k starting on vertices sv_1, sv_2, \dots, sv_k , and set of tasks \mathcal{T} , what is the fastest task-completing, collision-free set of k -schedules $\mathcal{C} = (C_1, C_2, \dots, C_k)$ such that C_i can be assigned to $R_i, \forall i \in [1, k]$?

3 Results

Hardness Results

We have found that the ROBOT SCHEDULING problem is NP-Hard, and that hardness remains even when we restrict the class of graphs we consider, the known results are shown in Table 1.

■ **Table 1** Our results for different graph classes and numbers, k , of robots.

Setting	Result
General graphs, $k \in \mathbb{N}$	NP-complete
Complete graphs, $k \geq 2$	NP-complete
Bipartite graphs, $k \geq 2$	NP-complete
Star graphs (and trees), $k \geq 2$	NP-complete
Planar graphs, $k \in \mathbb{N}$	NP-complete
Path graphs, with m tasks of equal duration, $k \in \mathbb{N}$	Optimal $O(kmn)$ time Algorithm (Theorem 7)
Path graphs, $k \in \mathbb{N}$	k -approximation Algorithm (Theorem 8)

Algorithmic Results for Path Graphs

1-Robot Scheduling on Path Graphs. In this section, we provide an algorithm for finding the optimal schedule for a single robot on a path. Corollary 3 shows that the time needed to complete the fastest schedule can be computed via a closed-form expression.

1-Robot Scheduling Algorithm. Let P be a path graph of length n , let $T = (t_1, t_2, \dots, t_m)$ be a set of tasks, and let R be the single robot starting on vertex $sv = v_{i_s}$. We assume, without loss of generality, that t_j is located on v_{i_j} such that v_{i_j} is left of $v_{i_{j+1}}$, i.e. $\forall j \in [1, m-1], i_j < i_{j+1}$. Note that there may exist some task t_i located on sv without contradiction. Using this notation, the optimal schedule $\mathcal{C} = \{C\}$ is:

- $C = \{ (v_s, v_{s+1}), (v_{s+1}, v_{s+2}), \dots, (v_{i_m-1}, v_{i_m}), t_m, (v_{i_m}, v_{i_m-1}), (v_{i_m-1}, v_{i_m-2}), \dots, (v_{i_{m+1}+1}, v_{i_{m+1}}), t_{m-1}, \dots, (v_{i_1+1}, v_{i_1}), t_1 \}$ if $|i_s - i_m| \leq |i_s - i_1|$.
- $C = \{ (v_s, v_{s-1}), (v_{s-1}, v_{s-2}), \dots, (v_{i_1+1}, v_{i_1}), t_1, (v_{i_1}, v_{i_1+1}), (v_{i_1+1}, v_{i_2+2}), \dots, (v_{i_2-1}, v_{i_2}), t_2, \dots, (v_{i_m-1}, v_{i_m}), t_m \}$ if $|i_s - i_m| > |i_s - i_1|$.

► **Lemma 2.** The fastest task-completing schedule for 1-ROBOT SCHEDULING on a path graph P of length n with m tasks $T = (t_1, t_2, \dots, t_m)$ located on vertices $v_{i_1}, v_{i_2}, \dots, v_{i_m}$, and a robot R starting on vertex v_s can be constructed in $O(n)$ time.

► **Corollary 3.** *The fastest task-completing schedule for 1-ROBOT SCHEDULING on a path graph P of length n with m tasks $T = (t_1, t_2, \dots, t_m)$ located on vertices $v_{i_1}, v_{i_2}, \dots, v_{i_m}$ and a robot R starting on vertex v_s requires*

$$\min(|s - i_1|, |s - i_m|) + i_m - i_1 + \sum_{t \in T} t$$

timesteps.

2-Robot Scheduling on Path Graphs. We now move to 2-ROBOT SCHEDULING on a path. First, we provide a new algorithm generalising the above algorithm for 1-ROBOT SCHEDULING. Later, we further generalise this to k -ROBOT SCHEDULING on a path; however, it is valuable to consider 2-ROBOT SCHEDULING first, both to illuminate the main algorithmic ideas and to provide a base case for later inductive arguments. We start by providing an overview of our algorithm, which we call the *partition algorithm*.

The 2-Partition Algorithm. Let P be a path graph of length n , let $T = (t_1, t_2, \dots, t_m)$ be the set of tasks, and let R_L and R_R be the pair of robots starting on vertices $sv_L = v_{i_L}$ and $sv_R = v_{i_R}$ respectively. We call R_L the *left robot* and R_R the *right robot*, with the assumption that sv_L is left of sv_R . We denote by i_j the index of the vertex containing the task t_j , and assume that $i_j < i_{j+1}$, for every $j \in [1, m-1]$. For notation, let $C_1(P, T, sv)$ return the optimal schedule for a single robot starting at sv on the path P for completing the task set T .

We construct the schedule by partitioning the tasks into 2 sets, $T_L = (t_1, t_2, \dots, t_q)$ and $T_R = (t_{q+1}, t_{q+2}, \dots, t_m)$. We determine the value of q by finding the value which minimises $\max(|C_1(P_{1, \max(\ell, i_q)}, (t_1, t_2, \dots, t_\ell), sv_L)|, |C_1(P_{\min(i_{q+1}, v_r), m}, (t_{q+1}, t_{q+2}, \dots, t_m), sv_R)|)$. We will use $C_2(P, T, (sv_L, sv_R))$ to denote the schedule returned by this process.

► **Lemma 4.** *Given an instance of 2-ROBOT SCHEDULING on an n -length path P with a set of equal-length tasks $T = (t_1, t_2, \dots, t_m)$, and starting vertices $sv_L = v_{i_L}, sv_R = v_{i_R}$, for any schedule $C = (C_\ell, C_r)$ where the rightmost task t_R assigned to the left robot is right of the leftmost T_L assigned to the right robot, there exists some schedule $C' = (C'_\ell, C'_r)$ that takes no more time than C and does not contain any such tasks.*

► **Lemma 5.** *Given an instance of 2-ROBOT SCHEDULING on an n -length path P with a set of tasks $T = (t_1, t_2, \dots, t_m)$ where the duration of t_i is equal to the duration of t_j for every $i, j \in [1, m]$. Further, let sv_L and sv_R be the starting vertices of the robots. Then $C_2(P, T, (sv_L, sv_R))$ is a fastest set of schedules for this instance and can be found in $O(m)$ time.*

► **Theorem 6.** *Given an instance of 2-ROBOT SCHEDULING on an n -length path P , with a set of tasks $T = (t_1, t_2, \dots, t_m)$ and starting vertices sv_L and sv_R . Then $C_2(P, T, (sv_L, sv_R))$ is within a factor of 2 of the fastest set of schedules solving this instance.*

k -robots on the path. Now, we generalise the 2 robots on a path instance to an arbitrary number. To do so, we build a dynamic programming algorithm based on the same principles as the previous partition algorithm.

The k -Partition Algorithm. Let P be a path of length n , $T = \{t_1, t_2, \dots, t_m\}$ be a set of tasks, and let sv_1, sv_2, \dots, sv_k be the starting vertices of the robots R_1, R_2, \dots, R_k respectively, with the assumption that R_i starts left of R_{i+1} , for every $i \in [1, k-1]$. Further, we denote by i_t the index such that v_{i_t} contains task t , and assume that $i_{t_j} < i_{t_{j+1}}$ (i.e.

task t_j is left of t_{j+1}) for every $j \in [1, m - 1]$. We construct a $k \times m$ table S , with $S[c, \ell]$ containing the time required to complete the fastest collision-free schedule completing tasks t_1, t_2, \dots, t_ℓ with robots R_1, R_2, \dots, R_c .

First, observe that $S[1, \ell]$ can be computed, for every $\ell \in [1, m]$, in $O(m)$ time. Now, assuming the value of $S[c - 1, \ell]$ has been computed for every $\ell \in [1, m]$, the value of $S[c, r]$ is computed by finding the value r' such that $\max(|C_1(P, (t_{r'+1}, t_{r'+2}, \dots, t_r))|, S[c - 1, r'])$ is minimised, formally $S[c, r] = \min_{r' \in [1, r]} \max(|C_1(P, (t_{r'+1}, t_{r'+2}, \dots, t_r))|, S[c - 1, r'])$. Letting \mathcal{S} be an auxiliary table such that $\mathcal{S}[c, \ell]$ contains the schedule corresponding to the time given in $S[c, \ell]$, a task-completing collision-free schedule for the k -ROBOT SCHEDULING instance is given in $S[k, m]$.

Let $S_k(P, T, (sv_1, sv_2, \dots, sv_k))$ return the schedule determined by this table. Note that for $S_2(P, T, (sv_1, sv_2))$, this becomes equivalent to the 2-partition algorithm.

► **Theorem 7.** *Given an instance of k -ROBOT SCHEDULING on a path $P = (V, E)$ with equal duration tasks $T = (t_1, t_2, \dots, t_m)$ on vertices $v_{i_1}, v_{i_2}, \dots, v_{i_m}$ and k robots R_1, R_2, \dots, R_k starting at $sv_1, sv_2, \dots, sv_k = v_{j_1}, v_{j_2}, \dots, v_{j_k}$, there are no schedules taking less time than the schedule returned by $S_k(P, T, (sv_1, sv_2, \dots, sv_k))$. Further, this schedule can be found in $O(kmn)$ time.*

► **Theorem 8.** *Given an instance of k -ROBOT SCHEDULING on a path $P = (V, E)$ with tasks $T = (t_1, t_2, \dots, t_m)$ on vertices $v_{i_1}, v_{i_2}, \dots, v_{i_m}$ and k robots R_1, R_2, \dots, R_k starting at $sv_1, sv_2, \dots, sv_k = v_{j_1}, v_{j_2}, \dots, v_{j_k}$, the schedule returned by $S_k(P, T, (sv_1, sv_2, \dots, sv_k))$ is no more than a factor of k slower than the optimal. Further, this schedule can be found in $O(km^2)$ time.*

4 Conclusion

We have shown that our definition of k -ROBOT SCHEDULING is hard, even on highly constrained classes of graphs while being solvable for path graphs with equal-length tasks and approximable for tasks of any length. While these results paint a strong picture of the complexity of this problem, we are left with several open questions. The most direct is as to whether our approximation algorithm for path graphs can be improved or if an optimal algorithm can be found. We conjecture that a polynomial time algorithm exists for this setting; however, at present, no such algorithm has been found. The second natural direction is to look at the remaining classes of graphs that have not been covered by our existing results. The most obvious of these are cycles, which, while closely related to paths, can not be solved by naive application of our current tools. While it seems likely that similar optimality and approximation results can be found, these are currently open problems.

References

- 1 Duncan Adamson, Nathan Flaherty, Igor Potapov, and Paul Spirakis. Collision-free robot scheduling, 2024. [arXiv:2402.12019](https://arxiv.org/abs/2402.12019).
- 2 B. Burger, P. M. Maffettone, V. V. Gusev, C. M. Aitchison, Y. Bai, X. Wang, X. Li, B. M. Alston, B. Li, R. Clowes, et al. A mobile robotic chemist. *Nature*, 583(7815):237–241, 2020.
- 3 R. D. King. Rise of the robo scientists. *Scientific American*, 304(1):72–77, 2011.
- 4 J. Li, S. G. Ballmer, E. P. Gllis, S. Fujii, M. J. Schmidt, A. M. E. Palazzolo, J. W. Lehmann, G. F. Morehouse, and M. D. Burke. Synthesis of many different types of organic small molecules using one automated process. *Science*, 347(6227):1221–1226, 2015.
- 5 S. Liu, J. Shen, W. Tian, J. Lin, P. Li, and B. Li. Balanced task allocation and collision-free scheduling of multi-robot systems in large spacecraft structure manufacturing. *Robotics and Autonomous Systems*, 159:104289, 2023.

Brief Announcement: On the Exponential Growth of Geometric Shapes

Nada Almalki ✉ 

Department of Computer Science, University of Liverpool, UK

Siddharth Gupta ✉ 

Department of Computer Science & Information Systems, BITS Pilani Goa Campus, India

Othon Michail ✉ 

Department of Computer Science, University of Liverpool, UK

Abstract

We explore how geometric structures (or *shapes*) can be grown exponentially fast from a single node, through a sequence of centralized growth operations, and if collisions during growth are to be avoided. We identify a parameter k , representing the number of turning points within specific parts of a shape. We prove that, if edges can only be formed when generating new nodes and cannot be deleted, trees having $O(k)$ turning points on every root-to-leaf path can be grown in $O(k \log n)$ time steps and spirals with $O(\log n)$ turning points can be grown in $O(\log n)$ time steps, n being the size of the final shape. For this case, we also show that the maximum number of turning points in a root-to-leaf path of a tree is a lower bound on the number of time steps to grow the tree and that there exists a class of paths such that any path in the class with $\Omega(k)$ turning points requires $\Omega(k \log k)$ time steps to be grown. In the stronger model, where edges can be deleted and neighbors can be handed over to newly generated nodes, we obtain a universal algorithm: for any shape S it gives a process that grows S from a single node exponentially fast.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases centralized algorithm, growth process, collision, programmable matter

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.23

Related Version *Full Version*: <https://arxiv.org/abs/2307.04385> [2]

1 Introduction

A *growth operation* (also called *doubling* [3] or *expansion* [8]) applied on a node u of a geometric shape S , generates a new node in one of the points adjacent to u and possibly translates some part of the shape. In this work, we explore the following two interrelated questions: “*What are the structural properties associated with exponential growth of geometric shapes?*” and “*How can some of these properties be exploited and others avoided in order to design algorithms that can grow desired shapes exponentially fast?*”

Though our model takes inspiration from natural growth processes, it also shares features with existing theoretical models of computation and robotics. Growth is a defining property of both our model and self-assembly models. In the majority of self-assembly models, growth is through passive attachment [7, 10] on the external layer of the formed structure, and, thus, is relatively slow. Our algorithms can actively control the structure’s growth without any *a priori* limitation on where to apply the growth operations, resulting in sub-linear and often (poly)logarithmic growth in the size of the final structure. An example of a self-assembly model incorporating active molecular dynamics is the *Nubot* model [11]. A difference between our model and [11] is that our processes are only allowed to update instances through growth. As is also the case in [11], most of our algorithms use the fast process of growing a line as a sub-routine. Recently, there has been growing interest in studying the algorithmic



© Nada Almalki, Siddharth Gupta, and Othon Michail;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 23; pp. 23:1–23:6

Leibniz International Proceedings in Informatics



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

foundations of *programmable matter* systems, focusing on their ability to alter their shape through local reconfiguration [1, 4, 6, 9]. The growth processes that we study could serve as a way to deploy programmable matter fast, either in its exact initial configuration or in a rough version of it that can be then refined through other types of operations. An assumption of our model is that individual operations have *linear strength*, meaning that they have enough power to move any part of the structure. This is a common simplifying assumption in the relevant literature [5, 11, 4] and can sometimes be dropped, e.g., when more than one operation can be applied in parallel. Our model also has some relevance to von Neumann’s concept of self-replicating machines.

2 Models and Problem

We consider a two-dimensional square grid, each point of which is identified by its $x \geq 0$ and $y \geq 0$ integer coordinates, x indicating the column and y the row. A *shape* is defined as a graph $S = (V, E)$ drawn on the grid. V is a set of n nodes, where each node u occupies a distinct point (u_x, u_y) of the grid. $E \subseteq \{uv \mid u, v \in V \text{ and } u, v \text{ are adjacent}\}$ is a set of edges between pairs of adjacent nodes, where two nodes u and v are *adjacent* if their orthogonal distance on the grid is one. Our results hold for any geometry of individual nodes that does not trivially make nearby nodes intersect. A shape is *connected* if the graph that defines it is a connected graph. We restrict attention to connected shapes.

One or more growth operations applied in parallel to nodes of a shape S either cause a *collision* or yield a new shape S' . There are two types of collisions: *node collisions* and *cycle collisions*. We assume that the constructed shapes are equivalent up to translations. Let $T = (V, E)$ be a tree and $u_0 \in V$ its root. A single growth operation applied on a node $u \in V$ toward an adjacent point (x, y) , results in either generating a new node u' at point (x, y) and connecting it to u , or, if (x, y) is already occupied by a node v connected to u , generating u' between u and v , connecting it to both u and v , and translating the subtree $T(v)$ by one unit away from u along the axis parallel to uv .

Let Q be a set of operations to be applied *in parallel* to a connected shape S , each operation on a distinct pair of nodes or a node and an unoccupied point. We assume that all operations in such a set are applied *concurrently*, have the same *constant execution speed*, and their *duration* is equal to one *time step*. A *node collision* occurs if the trajectories of any two nodes meet. If S is a connected shape with at least one cycle, then a set of parallel operations Q on S either causes a *cycle collision* or its effect is essentially equivalent to the application of Q on any spanning tree of S . In particular, a *cycle collision* occurs when two parts of a cycle grow unequally. A set of operations is said to be *collision free* if it does not cause any node or cycle collisions.

A growth process σ starts from an initial shape S_0 – often a single node – and, in each time step $t \geq 1$, applies a set of parallel growth operations – possibly a single operation – on the current shape S_{t-1} to give the next shape S_t , until a final shape S is reached at a time step t_f . In this case, we say that σ *grows* S from S_0 in t_f *time steps*.

The different models and processes we consider are defined as follows.

► **Definition 1.** Let S_t^b and S_t^e denote the shapes formed by the beginning and by the end of time step t , respectively, and assume that $S_1^b = S_0$. A cycle-preserving growth process applies a collision free set of parallel growth operations Q_t to S_t^b , for all time steps $t \geq 1$. A cycle-breaking growth process additionally removes a – possibly empty – subset of the edges of S_t^b , whose removal does not disconnect the shape, before applying Q_t to it. If neighbor handover is allowed, growth of a node u generating a new node u' in direction d can hand

any neighbor w of u perpendicular to d over to u' . In the connectivity graph model, for all $t \geq 1$, $S_{t+1}^b = S_t^e$ holds. In the adjacency graph model, for all $t \geq 1$, $S_{t+1}^b = AC(S_t^e)$ holds, where $AC(S)$ is the adjacency closure of a shape S .

Intuitively, the additional assumption in the adjacency graph model is that, at the end of every time step, the graph model updates the shape by connecting all adjacent nodes that are not connected. Combining the adjacency graph model with cycle-breaking processes captures the less extreme case, in which the process can choose any spanning connected sub-shape of the adjacency closure.

We study a reachability problem between classes of shapes through growth. The definition of the problem is the same for all growth models of Definition 1.

► **Problem 1.** *Let \mathcal{I} be a class of initial shapes and \mathcal{F} a class of final shapes. We want to determine a bound τ such that for all $S_0 \in \mathcal{I}$ and all $S \in \mathcal{F}$ there is a growth process σ that grows S from S_0 in τ time steps.*

Given that our focus is on exponential growth, upper bounds must be of the form $\tau = O(\log n)$ or $\tau = (\text{poly}) \log n$. As there is a straightforward $\Omega(\log n)$ lower bound, non-trivial lower bounds should be at least $\omega(\log n)$. In all instances of the problem that we study, at least one of \mathcal{I} and \mathcal{F} is a singleton, the initial shape typically being a single node. Our upper bounds are constructive: for each instance of the problem an algorithm is presented, which for every (S_0, S) from the respective classes gives a process that grows S from S_0 in τ time steps.

3 Technical Overview

In this section, we present the main results of our work, starting with the results of the connectivity graph model and then moving on to the adjacency graph model.

Connectivity Graph Model. Starting growth from a single node, the class of shapes that can be grown in this model is limited to tree structures only. We start by focusing on efficiently growing trees. We identify a parameter k , representing the number of turning points within specific parts of a shape. A node u of a shape S is called a *turning point* if either u is a leaf or there are at least two neighbors v_1 and v_2 of u such that v_1u is perpendicular to uv_2 . The nodes between any two consecutive turning points of a path form a line segment.

Upper Bounds Trees with $O(k)$ turning points on every root-to-leaf path can be grown in $O(k \log n)$ time steps through a breadth-first search (BFS) on their line segments. Starting from the root u_0 , the algorithm proceeds in phases, growing all maximal line segments at distance i in parallel. Each line segment can be grown exponentially fast by doubling the number of nodes in every time step.

► **Theorem 2.** *Let T be any tree having $O(k)$ turns on every root-to-leaf path. BFS on line segments can grow T from a single node within $O(k \log n)$ time steps in the connectivity graph model.*

Similarly, spirals with $O(\log n)$ turning points can be grown within $O(\log n)$ time steps through a pipelined version of BFS. For further details, see [2].

Lower Bounds. We establish two lower bounds for trees and paths. For trees, we show that the maximum number of turning points in a root-to-leaf path of the tree is a lower bound on the number of time steps to grow the tree from a single node. Initially, we define a growth process σ for $S = (V, E)$ that induces a relation \rightarrow_σ on V , where $u \xrightarrow{t}_\sigma v$ iff node u generates node v at time step t . We also write $u \rightarrow_\sigma v$ to mean $u \xrightarrow{t}_\sigma v$ for some $t \geq 1$ and $u \rightsquigarrow_\sigma v$ iff $u = u_1 \rightarrow_\sigma u_2 \rightarrow_\sigma \dots \rightarrow_\sigma u_l = v$, for some $l \geq 2$. We omit σ , writing just $u \xrightarrow{t} v$, $u \rightarrow v$, or $u \rightsquigarrow v$ when the growth process is clear from context or when referring to any growth process. The relation \rightarrow_σ defines a graph $G_{\rightarrow_\sigma} = (V, E_{\rightarrow_\sigma})$. Further, we define a relation \mapsto_σ induced by \rightarrow_σ on the turning points of tree shapes. Given a tree T and a growth process σ for T , for any two turning points u, v of T we write $u \mapsto_\sigma v$ iff (i) $u \xrightarrow{t}_\sigma v$ or (ii) $u \rightsquigarrow u' \xrightarrow{t}_\sigma v$ and u, u', v are on the same line segment at the end of time step t .

► **Lemma 3.** *In the connectivity model, let $T = (V, E)$ be a tree and σ a growth process for T starting from $u_0 \in V$. For any root-to-leaf path (u_0, u_1, \dots, u_l) of T , where the u_i s are restricted to the turning points of the path, $u_0 \mapsto u_1 \mapsto \dots \mapsto u_l$ holds.*

The theorem below formalizes the lower bound for growing trees.

► **Theorem 4.** *Let $T = (V, E)$ be a tree and k any positive integer satisfying that for every root $u_0 \in V$ there is a root-to-leaf path in T containing at least k turning points. Then any growth process σ for T in the connectivity model requires at least $k - 1$ time steps. This lower bound is maximized for the maximum such k .*

For paths, we show that there exists a class of paths such that any path in the class with $\Omega(k)$ turning points requires $\Omega(k \log k)$ time steps to be grown from a single node. Let P be a path with k turning points and $(tp_1, tp_2, \dots, tp_k)$ be their order in P . Let σ be a process that grows P from a single node. Without loss of generality, we can assume that σ starts from a turning point tp_i of the path P . We prove that the sets $\{tp_{i+1}, tp_{i+2}, \dots, tp_k\}$ and $\{tp_1, tp_2, \dots, tp_{i-1}\}$ of turning points are generated in the order $(tp_{i+1}, tp_{i+2}, \dots, tp_k)$ and $(tp_{i-1}, tp_{i-2}, \dots, tp_1)$, respectively by σ . Moreover, σ respects the direction of P at every node while generating the next node from it.

Let P be an incompressible (meaning that it has no columns or rows without any turning points) spiral path between u and v with k turning points. Moreover, let u be the internal endpoint of P . The following lemma gives a lower bound on the number of time steps taken by any process that grows P from a single node starting from u .

► **Lemma 5.** *Let P be an incompressible spiral path between u and v with k turning points. Moreover, let u be the internal endpoint of P . Let σ be any process that grows P from a single node starting from u . Then, σ requires $\Omega(k \log k)$ time steps.*

Let $(tp_1 = u, tp_2, \dots, tp_k = v)$ be the order of turning points of P from u to v . We know that σ generates the turning points in the order $(tp_1 = u, tp_2, \dots, tp_k = v)$. Let GT_j be the time step when the turning point tp_j was generated by σ , for any $j \geq 2$. Let $\hat{P}(t)$ be the path constructed by σ after time step t . Further, let a and b be two vertices of P . We denote by $P[a, b]$ the path between a and b (including both a and b) of P . Moreover, we denote by $|a - b|_P$ the number of edges in $P[a, b]$. Also, we denote by $X(a, P)$ the x-coordinate of the vertex a in P . To prove the above lemma, we first prove the following lemma about the path constructed by σ .

► **Lemma 6.** *For any $j \geq 5$, the path $\hat{P}(GT_j - 1)$ grown by σ till time step $GT_j - 1$ should be the same as the subpath $P[tp_1, tp_{j-1}]$ of P between $tp_1 = u$ and tp_{j-1} .*

The lower bound follows by applying Lemma 5 on a path consisting of two interleaved spirals of equal size and observing that at least one of the two must be grown from its internal endpoint. See [2] for the full proof.

► **Theorem 7.** *Let σ be a process that grows a path from a single node. Then, there exists a path for which σ takes $\Omega(k \log k)$ time steps.*

Adjacency Graph Model. In this model, every pair of adjacent nodes is also connected in the shape. We study both cycle-preserving and cycle-breaking types of processes. For cycle-preserving processes, we cannot directly perform BFS on line segments through cycle-preserving growth due to the dependence between adjacent line segments. We give a modified BFS that overcomes this by growing adjacent line segments in different phases, and we prove that if a shape S has a spanning tree with $O(k)$ turning points on every root-to-leaf path, then the adjacency closure of S can be grown from a single node within $O(k \log n)$ time steps. For a complete description of this approach, see [2]. For cycle-breaking processes with the additional assumption that neighbors can be handed over to newly generated nodes (*neighbor handover*), our main result is an efficient universal algorithm that gives an $O(\log n)$ time steps *growth process* for any connected shape S . The algorithm achieves this by specifying an elimination order of the nodes within the shape and then inverting this order to produce the *growth process*.

► **Theorem 8.** *Given any connected shape, S with dimensions $l \times w$, the elimination algorithm grows S from a single node in $O(\log l + \log w)$ time steps.*

References

- 1 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmović, Robin Flatland, Matias Korman, Belen Palop, Irene Parada, André van Renssen, et al. Universal reconfiguration of facet-connected modular robots by pivots: the $O(1)$ musketeers. *Algorithmica*, 83(5):1316–1351, 2021.
- 2 Nada Almalki, Siddharth Gupta, and Othon Michail. On the exponential growth of geometric shapes, 2024. [arXiv:2307.04385](https://arxiv.org/abs/2307.04385).
- 3 Nada Almalki and Othon Michail. On geometric shape construction via growth operations. *Theoretical Computer Science*, 984:114324, 2024.
- 4 Abdullah Almethen, Othon Michail, and Igor Potapov. Pushing lines helps: Efficient universal centralised transformations for programmable matter. *Theoretical Computer Science*, 830-831:43–59, 2020.
- 5 Greg Aloupis, Sébastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristán, and Stefanie Wuhler. Reconfiguration of cube-style modular robots using $O(\log n)$ parallel moves. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 342–353, 2008.
- 6 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Universal shape formation for programmable matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–299, 2016.
- 7 David Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55(12):78–88, 2012.
- 8 Siddharth Gupta, Marc van Kreveld, Othon Michail, and Andreas Padalkin. Collision detection for modular robots – it is easy to cause collisions and hard to avoid them, 2023. [arXiv:2305.01015](https://arxiv.org/abs/2305.01015).
- 9 Othon Michail, George Skretas, and Paul G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences*, 102:18–39, 2019.

23:6 Brief Announcement: On the Exponential Growth of Geometric Shapes

- 10 Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13:195–224, 2014.
- 11 Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science (ITCS)*, pages 353–354, 2013.

Brief Announcement: The Dynamic Steiner Tree Problem: Definitions, Complexity, Algorithms

Stefan Balev

Université Le Havre Normandie, Univ Rouen Normandie, INSA Rouen Normandie, Normandie Univ, LITIS UR 4108, F-76600 Le Havre, France

Yoann Pigné

Université Le Havre Normandie, Univ Rouen Normandie, INSA Rouen Normandie, Normandie Univ, LITIS UR 4108, F-76600 Le Havre, France

Éric Sanlaville¹ 

Université Le Havre Normandie, Univ Rouen Normandie, INSA Rouen Normandie, Normandie Univ, LITIS UR 4108, F-76600 Le Havre, France

Mathilde Vernet

LIA, Avignon Université, Avignon, France

Abstract

This note introduces an extension of the Steiner tree problem applied to dynamic graphs. It discusses its interest, studies its complexity and proposes an algorithm tested on generated and real data.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems

Keywords and phrases Steiner Tree, Dynamic Graph, Complexity, experimental study

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.24

Funding This work was Supported by the French ANR, project ANR-22-CE48-0001 (TEMPOGRAL)

1 Introduction and contributions of this work

The goal of the classical Steiner tree problem in graphs is to maintain connectivity among selected vertices, called terminals, while minimizing the cost associated with the chosen edges or vertices in the process. The problem has practical applications in many domains, such as communication networks, social networks, and logistics networks. Although it is simple to formulate, it was proven to be NP-hard very early on. It remains NP-hard in most of its simple versions, even with unit costs on the edges, where the objective is to minimize the number of vertices needed to connect the terminal vertices. However, there exist cases for which the Steiner problem is easy, as it can be solved using polynomial algorithms. The first case is when the number of terminals is two, as it then reduces to a shortest path problem. The second case is when every vertex of the graph is a terminal, as it then reduces to a spanning tree problem.

In the application domains cited above, the underlying graph could change over time. For some years now, a growing literature has been considering the well-known classical combinatorial problems in this new setting. Recalling all papers dealing with dynamic graphs (the name may vary) is out of the scope of this short paper, but one may cite, as the most relevant for this work, [8] for paths and their extensions, [2, 6] for some considerations about connectivity issues, . . . When time is considered as a discrete variable, such a graph is basically constituted of an ordered sequence of graphs indexed by time: $G = (G_i)$, $i \in \mathcal{T}$.

¹ corresponding author



This is the way it will be considered throughout the paper. The term of dynamic graph is preferred because, as we shall see, it is not mandatory to know the graph evolution in advance to find a *Steiner set* (the set of vertices used to connect the terminals).

Connectivity in a temporal setting may have different meanings. A static Steiner tree is the less costly way to ensure connectivity between a subset of terminal nodes. We are looking for a structure that maintains connectivity whereas the graph is changing. Basically, connectivity here may be defined in two ways. In the first way, one may wish to maintain some “instantaneous” or path-based connectivity: at each time step, a path exists between each pair of terminals. Conversely, in the second way one looks for the existence of a journey, also called temporal path, from each terminal to any other terminal. This journey-based connectivity ensures that some information, or some good, will eventually be transferred. This is a one shot property: typically, after a given time, no journey may exist for a couple of terminals and the transfer is no more possible. *We claim that instantaneous connectivity, is better adapted to some situations*, like a set of mobile robots that cooperate for a given common goal, thus needing frequent communications between some distinguished nodes of the network. Note that to the best of our knowledge, no previous work considers direct extensions of Steiner trees. However, many works exist that study the existence of journeys [8]. Many works also study some journey-based extensions of spanning trees, namely spanners, see [1, 3]. The path-based extensions of spanning tree are of less interest, as it may consist either in computing the minimum spanning tree at each snapshot (without building any persistent structure), or in computing the spanning tree of the intersection of all snapshots (using only edges constantly present).

The main contributions of this paper are:

- We discuss how to extend the Steiner tree problem to dynamic graphs. Among the possible extensions, we identify the more relevant one for practical applications.
- We show that unlike its static counterpart, the dynamic Steiner problem is NP-hard even with two terminals.
- We propose an exact algorithm that computes all Steiner sets of a given size, study its complexity and test it experimentally on generated and real-world instances.

2 The Dynamic Steiner Set problem

Let us first recall the static problem definition. Let $G = (V, E)$ be a (static) graph. A non-negative weight w_e is associated to each edge $e \in E$. For a given subset of vertices $S \subset V$, called *terminals*, the objective is to find a tree of minimum weight covering all vertices of S . Note that the simpler version with unit cost is already NP-Hard. It reduces to find a Steiner set of minimal cardinality.

Let us see now how the Steiner Tree Problem can be extended to dynamic graphs. We consider here only the unit cost version, or minimal cardinality version. Let G be a undirected dynamic graph such that $G = (V, E, T) = (G_1, \dots, G_T)$ with $G_i = (V, E_i)$ and $E = \bigcup_i E_i$. The successive G_i s are called *snapshots* of G at each time step. As in the static case, one may introduce a subset S of special vertices, called terminals. The goal is still to ensure connectivity between the terminals. Note that we limit our study to the case where no travel time is associated to the edges. Hence there is no travel time associated to paths either: the (instantaneous) connectivity requirement applies to each time step.

The most straightforward way to extend the Steiner Tree problem to dynamic graphs is of course to compute, at each time step, the Steiner tree associated to S . This extension has one major drawback: at each time step, the Steiner set is different. It is not really convenient,

for instance in a communication network, to change the intermediate nodes at each time step. More importantly, we do not make any use of the knowledge of the dynamic graph as a whole, considering each G_i separately. Note also that this approach is very time consuming, as a complete Steiner tree is recomputed at each time step. Hence in the remaining of the section, we focus on computing a Steiner set that is fixed during all the lifetime of the graph.

► **Definition 1** (Fully Connected Steiner Set). *For a given vertex set $S \subset V$ of terminals, find V' with $S \subset V' \subset V$ and $E'_i \subset E_i \forall i \leq T$ such that:*

- $G'_i = (V', E'_i)$ is a connected graph $\forall i \leq T$
- $|V'|$ is minimum

We look for a subset V' of vertices containing S , such that the subgraph of G_i generated by V' is always connected, and V' has minimal cardinality (which is equivalent to minimize the total number of edges used to connect V'). But in fact the condition verified by each G'_i is stronger than we need to keep the terminals connected: the definition imposes the connectivity of *all* vertices in the Steiner set. The following definition relaxes this condition.

► **Definition 2** (Partially Connected Steiner Set). *For a given vertex set $S \subset V$ of terminals, find V' with $S \subset V' \subset V$ and $E'_i \subset E_i \forall i \leq T$ such that:*

- All vertices of S are part of the same connected component in the static graph $G'_i = (V', E'_i)$
- $|V'|$ is minimum

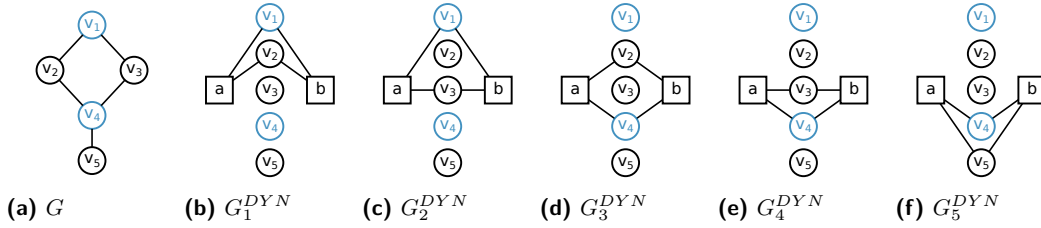
The (partially connected) Steiner set V' is the subset of nodes that keep the terminals connected for the whole lifetime of the graph, at a minimum cost while V' itself is not necessarily connected. Note that when $T = 1$, this problem is the Steiner Tree problem. However, we cannot deduce an optimal solution from the solution of static Steiner Problem at each time step.

To illustrate the difference between the two definitions, consider the simple dynamic graph with two time steps, where vertices $v_1 - v_2 - \dots - v_n$ form a path on both snapshots, and two terminals a and b . At the first time step a and b are connected to v_1 and at the second step they are both connected to v_n . If we want to keep the Steiner set fully connected, we have to take all the vertices ($k = n + 2$), but to keep the terminals connected, we only need v_1 and v_n ($k = 4$). Consider now the same setting but without the edges (v_i, v_{i+1}) . There is no fully connected Steiner set but a, b, v_1 and v_n still form a partially connected Steiner set.

We now retain definition 2 and study the complexity of the Dynamic Minimum Steiner Set (DMSS) problem which consists of finding a (partially connected) Steiner Set of minimum cardinality. Let us recall that the minimum cardinality version is NP-hard in the static case, even for bipartite or for chordal graphs [7]. On the other hand, one might consider the case of a small number of terminals. Even with arbitrary costs, the problem with two terminals is easy in the static case as the Steiner tree reduces to a single path.

► **Theorem 3.** *The DMSS problem is NP-hard even with two terminals.*

The idea of the proof is to transform Vertex Cover (VC) a well known NP-complete problem to DMSS with two terminals. Let $G = (V, E)$ be the graph of the VC instance. The dynamic graph G^{DYN} has the same vertices plus two terminals a and b . For each $(u, v) \in E$ there is a time step where the terminals are connected to u and v . A vertex cover in G corresponds to a Steiner set in G^{DYN} . An example is given in Fig. 1.



■ **Figure 1** Transforming a VC instance G with 5 edges to a DMSS instance with 5 time steps.

3 Algorithmic issues and experiments

This section focuses on efficiently solving the DMSS problem. We only consider the problem of finding a Steiner set of a given cardinality. Throughout this section, s is the number of terminals, k is the cardinality of the Steiner sets, and $k' = k - s$ is the number of vertices to add to the terminal vertices.

3.1 Basic ideas

For a given time step i , there might be a very large number of vertex sets of cardinality k' that allow the terminals to be connected. These sets will be called *candidates* for time step i . An obvious upper bound of this number is the number of vertex subsets of cardinality k' among $V - S$, that is $\binom{n-s}{k'}$. The dynamic Steiner sets we are looking for belong to this set but, hopefully, are much less numerous.

A Steiner set of cardinality k must by definition connect the terminals for each time step. Therefore, the algorithmic possibilities are straightforward. Let us discard the naive idea that consists in computing independently all candidate subsets for all time steps, and then computing their intersection. It is much more efficient to use an iterative process: suppose we have a set of candidates PSS for time steps $1, \dots, i$. At iteration $i + 1$, only the solutions in PSS which are also candidates for time step $i + 1$ are kept. So the core procedure of the algorithm is a search on small subgraphs generated by the candidate sets. Its complexity is $\mathcal{O}(m')$ where m' the number of edges of the subgraph is bounded by k^2 . The process starts with all candidates for $i = 1$. It ends when $i = T$ with all Steiner sets of cardinality k or when no more candidates exist. Of course, this algorithm is still a brute force algorithm as some enumerations have to be done.

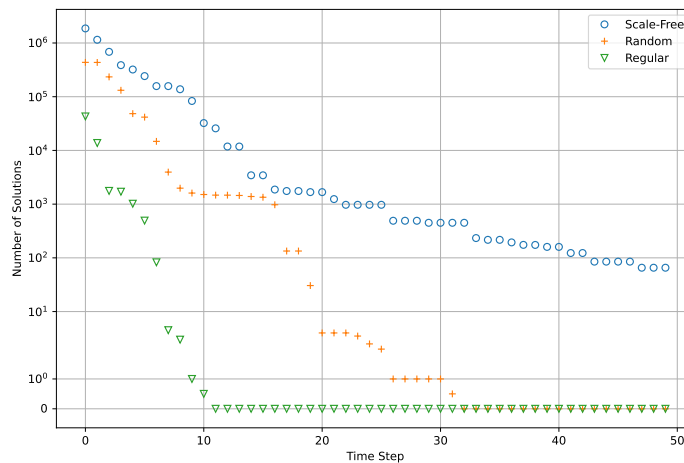
► **Theorem 4.** *There exists an algorithm that enumerates all solutions of DMSS, of complexity $\Theta(T \times (n - s)^{k'} \times \frac{k^2}{(k')!})$. It follows that DMSS is in XP class relatively to parameter k' . When k and s are fixed parameters, the complexity is $\Theta(n^{k-s})$.*

The algorithm sketched above might be used on-line: at each time step θ , all vertex sets that are Steiner sets for time interval $[0, \theta]$, are computed.

3.2 Experimental study

We performed an experimental study of our algorithm, both on randomly generated graphs and on real-world networks. The virtual machines used for this experiment have 64 GB of RAM.

We generated dynamic graphs using a method previously presented in [6] to test connected component computation. First we generate the underlying graph, then we add dynamicity to the edges using a Markovian process, see [4].



■ **Figure 2** Median number of Steiner sets with regard to the number of time steps, with $s = 3$.

The underlying graph is generated using generators from the GraphStream library². We tested three different types of graphs that present specific features. They differentiate mostly according to their clustering coefficient.

Experiments were run on randomly generated graphs up to 100 vertices, 50 time steps and up to 6 terminals. Results show the algorithm can solve these instances usually in less than an hour. The evolution of the number of solutions, hence of the computation time, heavily depends on the type of underlying graph, as shown in figure 2.

As a real-world instance, we used the CRAWDAD VT/MANIAC dataset [5], accessible through the IEEE Dataport comes from the CRAWDAD collection. This dataset encompasses routing and topology traces gathered during the Mobile Ad hoc Networks Interoperability And Cooperation (MANIAC) Challenges that took place in 2007 and 2009 in conjunction with the IEEE Globecom IEEE and PerCom conferences. These traces provide insights into the communication patterns, node mobility, and network structure characteristic of MANETs (Mobile Adhoc Networks) in real-world scenarios.

The resulting dynamic graph has 14 nodes, 74 edges (for the footprint), and a high number 1244 of timesteps. An edge remains present during less than 10% of the time horizon on average. The algorithm runs very fast on this instance (a few seconds) to provide all possible Steiner sets for all possible values of k .

References

- 1 Kyriakos Axiotis and Dimitris Fotakis. On the size and the approximability of minimum temporally connected subgraphs. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, 2016. arXiv preprint arXiv:1602.06411.
- 2 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.
- 3 Arnaud Casteigts, Joseph G Peters, and Jason Schoeters. Temporal cliques admit sparse spanners. *Journal of Computer and System Sciences*, 121:1–17, 2021.

² <http://graphstream-project.org>

24:6 Brief Announcement: The Dynamic Steiner Tree Problem

- 4 Andrea EF Clementi, Claudio Macci, Angelo Monti, Francesco Pasquale, and Riccardo Silvestri. Flooding time in edge-markovian dynamic graphs. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 213–222, 2008.
- 5 Amr Hilal, Jawwad N Chattha, Vivek Srivastava, Michael S Thompson, Allen B MacKenzie, Luiz A DaSilva, and Pallavi Saraswati. *Crowdad vt/maniac*, 2022. doi:10.15783/C7WG6T.
- 6 Mathilde Vernet, Yoann Pigne, and Eric Sanlaville. A study of connectivity on dynamic graphs: computing persistent connected components. *4OR*, 21(2):205–233, 2023.
- 7 Kevin White, Martin Farber, and William Pulleyblank. Steiner trees, connected domination and strongly chordal graphs. *Networks*, 15(1):109–124, 1985.
- 8 B Bui Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.



Brief Announcement: Crash-Tolerant Exploration of Trees by Energy Sharing Mobile Agents

Quentin Bramas  

University of Strasbourg, ICUBE, CNRS, Strasbourg, France

Toshimitsu Masuzawa  

Graduate School of Information Science and Technology, Osaka University, Japan

Sébastien Tixeuil  

Sorbonne University, CNRS, LIP6, Institut Universitaire de France, Paris, France

Abstract

We consider the problem of graph exploration by energy sharing mobile agents that are subject to crash faults. More precisely, we consider a team of two agents where at most one of them may fail unpredictably, and the considered topology is that of acyclic graphs (*i.e.* trees). We consider both the asynchronous and the synchronous settings, and we provide necessary and sufficient conditions about the energy in two settings: line-shaped graphs, and general trees.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Computing methodologies → Mobile agents

Keywords and phrases Mobile Agents, Distributed Algorithms, Energy sharing

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.25

Funding This research was supported in part by Agence Nationale de la Recherche under Grant Agreement SAPPORO (Ref. 2019-CE25-0005-1) and by JSPS KAKENHI 20KK0232.

1 Context

This paper investigates the collective exploration of a known edge-weighted graph by mobile agents originating from arbitrary nodes. The objective is to traverse every edge at least once. Each agent possesses a battery with an initial energy level (that may differ among agents). An agent's battery is depleted by x when it travels a distance of x . Also, when two agents meet, they may freely exchange remaining energy. Finally, the possibility for one of the two agents to crash, or cease functioning indefinitely and unpredictably, exists.

Energy transfer by mobile agents was previously considered by Czyzowicz et al. [3]. Agents travel and spend energy proportional to distance traversed. Some nodes have information acquired by visiting agents. Meeting agents may exchange information and energy. They consider communication problems where information held by some nodes must be communicated to other nodes or agents. They deal with data delivery and convergecast problems for a centralized scheduler with full knowledge of the instance. With energy exchange, both problems have linear-time solutions on trees. For general undirected and directed graphs, these problems are NP-complete. Then, Czyzowicz et al. [2] consider the gossiping problem in tree networks. In an edge-weighted tree network, agents spend energy while traveling and collect copies of data packets from visited nodes. They deposit copies of possessed data packets and collect copies of data packets present at the node. Czyzowicz et al. [2] prove that gossiping can be solved in $O(k^2n^2)$ time for an n -node tree with k agents.

Most related to our paper are the works by Czyzowicz et al. [4], Sun et al. [5], and Bramas et al. [1]. On the one hand, Czyzowicz et al. [4] study the collective exploration of a known n -node edge-weighted graph by k mobile agents with limited energy and energy transfer capability. The goal is for every edge to be traversed by at least one agent. For an n -node path, they give an $O(n+k)$ time algorithm to find an exploration strategy or report that



© Quentin Bramas, Toshimitsu Masuzawa, and Sébastien Tixeuil;
licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 25; pp. 25:1–25:5

Leibniz International Proceedings in Informatics



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

none exists. For an n -node tree with ℓ leaves, they provide an $O(n + \ell k^2)$ algorithm to find an exploration strategy if one exists. For general graphs, deciding if exploration is possible by energy-sharing agents is NP-hard, even for 3-regular graphs. However, it's always possible to find an exploration strategy if the total energy of agents is at least twice the total weight of edges; this is asymptotically optimal. Next, Sun et al. [5] examines circulating graph exploration by energy-sharing agents on an arbitrary graph. They present the necessary and sufficient energy condition for exploration and an algorithm to find an exploration strategy if one exists. The exploration requires each node to have the same number of agents before and after. Finally, Bramas et al. [1] considered the problem of exploring every weighted edge of a given ring-shaped graph using a team of two mobile energy-sharing agents. They introduce the possibility for one of the two agents to fail unpredictably and cease functioning permanently (i.e., crashing). In this context, Bramas et al. [1] considered two scenarios: asynchronous (where no limit on the relative speed of the agents is known, so one agent cannot wait at a meeting point for another agent for a bounded amount of time and infer that the other agent has crashed, as it may simply be arbitrarily slow), and synchronous (where the two agents have synchronized clocks and move at precisely the same speed).

2 Model

Our model is similar to that proposed by Bramas et al. [1].

We are given a weighted graph $G = (V, E)$ where V is a set of n nodes, E is a set of m edges, and each edge $e_i \in E$ is assigned a positive integer $w_i \in \mathbb{N}^+$, denoting its weight (or length). We have k mobile agents (or agents for short) r_0, r_1, \dots, r_{k-1} respectively placed at some of the nodes s_0, s_1, \dots, s_{k-1} of the graph. We allow more than one agent to be located in the same place. Each agent r_i initially possesses a specific amount en_i of energy for its moves. An agent has the ability to travel along the edges of graph G in any direction. It can pause its movement if necessary and can change its direction either at a node or while traveling along an edge. The energy consumed by a moving agent is equal to the distance x it moved. An agent can move only if its energy is greater than zero. Now, the distance between two agents (that is, the minimum sum of the weights for all the paths connecting them) is the smallest amount of energy needed for them to meet at some point.

In our setting, agents can share energy with each other. When two agents, r_i and r_j , meet at a vertex or edge, r_i can take some energy from r_j . If their energy levels at meeting time meeting are en'_i and en'_j , then r_i can take an amount of energy $0 < en \leq en'_j$ from r_j . After the transfer, their energy levels are $en'_i + en$ and $en'_j - en$, respectively.

Each agent adheres to a pre-established trajectory until encountering another agent. At this point, the agent determines if it acquires energy, and calculates its ensuing trajectory. The definition of a trajectory depends on the synchrony model:

- **In the synchronous model**, a trajectory is a sequence of pairs $((u_0, t_0), (u_1, t_1), \dots)$, where u_i is a node, and t_i denotes the time at which the agent should reach u_i . For each $i \geq 0$, $t_i < t_{i+1}$, and u_{i+1} is either equal to u_i (i.e., the agent waits at u_i between t_i and t_{i+1}), or is adjacent to u_i (i.e., the agent leaves u_i at time t_i and arrives at u_{i+1} at time t_{i+1}). For simplicity, we assume in our algorithm that the moving speed is always one (it takes time d to travel distance d , so if $u_i \neq u_{i+1}$ and the weight of edge (u_i, u_{i+1}) is w , then $t_{i+1} - t_i = w$).
- **In the asynchronous model**, a trajectory is just a sequence of nodes (u_0, u_1, u_2, \dots) , u_{i+1} being adjacent to u_i for each $i \geq 0$, and the times at which it reaches the nodes are determined by an adversary.

In other words, in the synchronous model, the agent controls its speed and its waiting time at nodes, while an adversary decides them in the asynchronous model.

The computation of the trajectory and the decision to exchange energy is based on a *localized algorithm* (that is, an algorithm executed by the agent). In a given execution, the configuration at time t is denoted by C_t .

Localized algorithm. A *localized algorithm* f_i executed by an agent r_i at time t takes as input the *pasts* of r_i and its collocated agents, and returns (i) its ensuing trajectory $traj_i$ and (ii) the amount of energy $take_{i,j}$ taken from each collocated agent r_j . The past $Past_i(t)$ of r_i at time t corresponds to the path already traversed by r_i union the past of all the previously met agents. More formally:

$$Past_i(t) = \{path_i(t)\} \cup \{Past_j(t') \mid r_i \text{ met } r_j \text{ at time } t' \leq t\}$$

A set of localized algorithms is *valid* for a given initial configuration c if, for any execution starting from c , agents that are ordered to move have enough energy to do so and when an agent r_i takes energy from an agent r_j at time t , then r_j does not take energy from r_i at t .

In this paper, we consider the possibility of agent crashes. At any point in the execution, an agent r_i may crash and stop operating forever. However, if r_i has remaining energy $en'_i > 0$, then other agents meeting r_i may take energy from r_i . Now, a set of localized algorithms is t -crash-tolerant if it is valid even in executions where at most t agents crash.

We are interested in solving the problem of t -crash-tolerant collaborative exploration:

t -crash-tolerant collaborative exploration. Given a weighted graph $G = (V, E)$ and k mobile agents r_0, r_1, \dots, r_{k-1} together with their respective initial energies $en_0, en_1, \dots, en_{k-1}$ and positions s_0, s_1, \dots, s_{k-1} in the graph, find a valid set of localized algorithms that explore (or cover) all edges of the graph despite the unexpected crashes of at most $t < k$ agents.

This paper focuses on the 1-crash-tolerant collaborative exploration of *trees* by *two* agents.

3 Our Results

We consider the problem of graph exploration by energy-sharing mobile agents that are subject to crash faults. More precisely, we consider a team of two agents where at most one of them may fail unpredictably, and the considered topology is that of acyclic graphs (*i.e.* trees). Similarly to Bramas et al. [1] who studied the case of ring-shaped networks, we consider both the asynchronous and the synchronous settings, and we provide necessary and sufficient conditions for the initial amounts of energy in two settings: lines and trees. In the following, en_0 and en_1 denote the initial energy of the first and second agents, respectively.

Lines. In the case of the line, x (resp. y) denotes the distance of the first (resp. second) agent to the left border of the line, assuming $x \leq y$ and $x \leq \ell - y$, while ℓ denotes the weight of the line. In the asynchronous case, a necessary and sufficient condition is:

$$\begin{aligned} & (en_0 \geq x + y) \wedge (en_1 \geq y) \wedge (en_0 + en_1 \geq 2\ell + x + y) \\ \vee & (en_0 \geq \ell - x) \wedge (en_1 \geq 2\ell - (x + y)) \wedge (en_0 + en_1 \geq 4\ell - (x + y)) \\ \vee & (en_0 \geq \ell + x) \wedge (en_1 \geq 2\ell - y) \\ \vee & (en_0 \geq y - x) \wedge (en_1 \geq y - x) \wedge (en_0 + en_1 \geq \min(3\ell + y - x, 2\ell - x + 3y)) \end{aligned}$$

In the synchronous case, a necessary and sufficient condition is:

$$\begin{aligned} & (en_0 \geq x + y) \wedge (en_1 \geq y) \wedge (en_0 + en_1 \geq \max(\ell + x + y, 2\ell + x - y)) \\ \vee & (en_0 \geq \ell - x) \wedge (en_1 \geq 2\ell - (x + y)) \wedge (en_0 + en_1 \geq 3\ell - x - y) \\ \vee & (en_0 \geq \ell + x) \wedge (en_1 \geq 2\ell - y) \\ \vee & (en_0 \geq y - x) \wedge (en_1 \geq y - x) \wedge (en_0 + en_1 \geq 2\ell - x + y) \end{aligned}$$

Trees. In the case of a weighted tree T , d denotes the diameter of the tree, x the initial distance between the two agents, and W its total weight. In the asynchronous case, a sufficient condition is:

$$(en_0 \geq x) \wedge (en_1 \geq x) \wedge (en_0 + en_1 \geq 2W + 2d\lceil \log_{3/2} W \rceil + x + 2) \quad (1)$$

We provide a lower bound on the total energy for unweighted star graphs: $en_0 + en_1$ cannot be in $2W + 2\log(o(W))$ (notice $W = |E|$).

The main ingredients for our positive result are as follows.

First, we construct a family of k connected non-empty subtrees of T named T_1, T_2, \dots, T_k , where $T_i = (V_i, E_i)$ and $(E_i)_{1 \leq i \leq k}$ forms a partition of E . At the beginning, the agents meet to share energy. Then the agents repeat a procedure $explore(T_i)$ for all $i \in \{1, \dots, k\}$ from 1 to k . The procedure assumes that the agents are initially at the same location (possibly on an edge), and ensure that after the execution the agents are at the same location (not necessarily the same as the initial one) if $i < k$ (when $i = k$ the agents can terminate anywhere on completion of the exploration).

Agent r_0 (resp. r_1) executing $explore(T_i)$ first moves to the closest node v_i of T_i , executes *EulerianExplore*(T_i) (resp. *ReverseEulerianExplore*(T_i)), and moves back to its initial location, until it meets the other agent, and T_i is explored. If the agents meet before ending this sequence of moves and E_i is explored, then the procedure terminates. This occurs during the exploration of the Eulerian tour from v_i , or when one of the agents r comes back from v_i to its initial location after completing its Eulerian tour while the other has not started it (it is still moving towards v_i from the location where it started executing $explore(T_i)$).

Since the length of the Eulerian tour is $2w(T_i)$ (where $w(T_i)$ denotes the weight of T_i) and the distance to v_i from their initial location is d in the worst case, each agent must have, at the beginning of the procedure, the energy of at least $2d + 2w(T_i)$ if $i < k$ (to terminate even when the other agent remains at the initial location), at least $d + 2w(T_i)$ if $i = k$. When the procedure terminates, the total energy consumed during the procedure is at most $2d + 2w(T_i)$ (because every edge traversed in the procedure is traversed exactly twice if $i < k$, and at most twice if $i = k$).

Consequently, to complete all $explore(T_i)$, for every i , sequentially, our algorithm requires that the total remaining energy EN_i at the beginning of the procedure $explore(T_i)$ is as follows, where x is the initial distance between the agents:

- $EN_k \geq 2d + 4w(T_k)$
- $EN_i \geq \max(2d + 2w(E_i) + EN_{i+1}, 4d + 4w(T_i))$ ($2 \leq i \leq k - 1$)
- $EN_1 \geq x + \max(2d + 2w(T_1) + EN_2, 4d + 4w(T_1))$ where x is the initial weighted distance between the agents.

Moreover, the total energy consumption for exploring T is at most $x + \sum_{i=1..k} (2d + 2w(T_i)) = 2W + 2kd + x$.

We now have to construct the partition T_1, T_2, \dots, T_k of T so that k should be small to reduce the number of calls to $explore()$, but each $w(T_i)$ should not be too large to avoid increasing the energy required at the beginning of $explore(T_i)$. A good partition could be to have $w(T_k) = 1$ and $w(T_i) = 2w(T_{i+1})$, which results in $k = \lceil \log W \rceil$. In general trees, such a partition does not exist, but we can obtain a similar result using the centroid-based partition recursively, which guarantees $W_i/3 \leq w(T_i) \leq W_i/2$ (W_i is the total weight of the remaining part of the tree).

Let $T = (V, E)$ be a weighted tree with total weight W . The centroid of T is defined as follows. In the following, for a tree T and a node u of T , T can be regarded as a rooted tree, denoted by T^u , rooted at u . For the root u and its neighbor v , let T_v^u be the subtree of T^u rooted at v .

1. When there exists an edge $(u, v) \in E$ satisfying $w(T_v^u) < W/2$ and $w(T_u^v) < W/2$, the centroid of T is the point p on edge (u, v) such that $w(T_v^u) + w(v, p) = w(T_u^v) + w(u, p) = W/2$. We call p the *edge centroid*.
2. When there exists a node $u \in V$ satisfying $w(T_v^u) + w(u, v) \leq W/2$ for each neighbor v of u , the centroid of T is node u . We call u the *node centroid*.

By splitting the tree recursively at the centroid point, we can construct a partition T_1, \dots, T_k with $k = \lceil \log_{3/2} W \rceil$ to obtain the sufficient condition (1).

In the synchronous case, a sufficient condition is:

$$(en_0 \geq x) \wedge (en_1 \geq x) \wedge (en_0 + en_1 \geq 2W + d + x)$$

On the other hand we show that there exists an infinite family of trees such that the required total energy is at least $2W + \frac{d}{2} - 3$.

4 Conclusion

We characterized the solvability of exploration with two crash-prone energy-sharing mobile agents in the case of tree topologies, both in the synchronous and in the asynchronous settings. Obvious open questions include further closing the gap between necessary and sufficient conditions for the initial amounts of energies in the case of trees, solving the problem with more than two agents, and considering general graphs.

Also, our model for energy transfer is very simple (all energy can be transferred instantaneously between two agents, at no cost). It would be interesting to study non-linear battery models (where the capacity decreases faster if more instantaneous current is drawn, and the capacity increases less if faster charge is executed) in this context.

References

- 1 Quentin Bramas, Toshimitsu Masuzawa, and Sébastien Tixeuil. Brief announcement: Crash-tolerant exploration by energy sharing mobile agents. In Shlomi Dolev and Baruch Schieber, editors, *Stabilization, Safety, and Security of Distributed Systems - 25th International Symposium, SSS 2023, Jersey City, NJ, USA, October 2-4, 2023, Proceedings*, volume 14310 of *Lecture Notes in Computer Science*, pages 380–384. Springer, 2023. doi:10.1007/978-3-031-44274-2_28.
- 2 Jurek Czyzowicz, Dariusz Dereniowski, Robert Ostrowski, and Wojciech Rytter. Gossiping by energy-constrained mobile agents in tree networks. *Theor. Comput. Sci.*, 861:45–65, 2021. doi:10.1016/j.tcs.2021.02.009.
- 3 Jurek Czyzowicz, Krzysztof Diks, Jean Moussi, and Wojciech Rytter. Communication problems for mobile agents exchanging energy. In Jukka Suomela, editor, *Structural Information and Communication Complexity - 23rd International Colloquium, SIROCCO 2016, Helsinki, Finland, July 19-21, 2016, Revised Selected Papers*, volume 9988 of *Lecture Notes in Computer Science*, pages 275–288, 2016. doi:10.1007/978-3-319-48314-6_18.
- 4 Jurek Czyzowicz, Stefan Dobrev, Ryan Killick, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, Denis Pankratov, and Sunil M. Shende. Graph exploration by energy-sharing mobile agents. In Tomasz Jurdzinski and Stefan Schmid, editors, *Structural Information and Communication Complexity - 28th International Colloquium, SIROCCO 2021, Wroclaw, Poland, June 28 - July 1, 2021, Proceedings*, volume 12810 of *Lecture Notes in Computer Science*, pages 185–203. Springer, 2021. doi:10.1007/978-3-030-79527-6_11.
- 5 X. Sun, N. Kitamura, T. Izumi, and T. Masuzawa. Circulating exploration of an arbitrary graph by energy-sharing agents. In *Proc. of the 2023 IEICE General Conference*, pages 1–2, 2023.

Brief Announcement: Collision Detection for Modular Robots – It Is Easy to Cause Collisions and Hard to Avoid Them

Siddharth Gupta ✉ 🏠 

BITS Pilani, Goa Campus, India

Marc van Kreveld ✉ 

Utrecht University, The Netherlands

Othon Michail ✉ 

University of Liverpool, United Kingdom

Andreas Padalkin ✉ 

Paderborn University, Germany

Abstract

We consider geometric collision-detection problems for modular reconfigurable robots. Assuming the nodes (modules) are connected squares on a grid, we investigate the complexity of deciding whether collisions may occur, or can be avoided, if a set of expansion and contraction operations is executed. We study both discrete- and continuous-time models, and allow operations to be coupled into a single parallel group. Our algorithms to decide if a collision may occur run in $O(n^2 \log^2 n)$ time, $O(n^2)$ time, or $O(n \log^2 n)$ time, depending on the presence and type of coupled operations, in a continuous-time model for a modular robot with n nodes. To decide if collisions can be avoided, we show that a very restricted version is already NP-complete in the discrete-time model, while the same problem is polynomial in the continuous-time model. A less restricted version is NP-hard in the continuous-time model.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases Modular robots, Collision detection, Computational Geometry, Complexity

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.26

Related Version *Full Version*: <https://arxiv.org/abs/2305.01015> [6]

Funding *Andreas Padalkin*: This author was supported by the DFG Project SCHE 1592/10-1.

Acknowledgements The authors thank all participants of the Bertinoro Workshop on Distributed Geometric Algorithms, in particular Peyman Afshani for suggesting the $O(n \log^2 n)$ time solution for detecting collisions when there are no couplings. We thank Irina Kostitsyna and Christian Scheideler for the organization, and the latter also for proposing the collision detection problem. Finally, we thank Jesper Nederlof for some useful observations.

1 Introduction

Modular reconfigurable robotics and the related concept of programmable matter concern systems composed of interconnected elementary entities, called modules. The collection of modules can coordinate its limited communication, computation, sensing, and local actuation to accomplish nontrivial global tasks. Local actuation of modules is enabled through a set of one or more mechanical operations that they can perform. An operation typically involves the module that applies it as well as modules in its local neighborhood. Examples of such operations are pushing, pulling, expanding, contracting, doubling, and rotating. Apart from



© Siddharth Gupta, Marc van Kreveld, Othon Michail, and Andreas Padalkin; licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 26; pp. 26:1–26:5

Leibniz International Proceedings in Informatics



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

their induced local changes, these operations are often capable of causing a more global effect on the robotic structure within a limited period of time. An example is when a large part of the structure moves due to the simultaneous application of one or more local operations.

The ability of local operations to globally affect the robotic structure is a double-edged sword. On one hand, it is a convenient form of parallelism, where global structural changes can happen faster. On the other hand, if not properly orchestrated, it could cause small violations of the structure or even complete structural failure, such as uneven cycle growth, global connectivity breaking, and self-intersection of the structure. We, hereafter, shall call all structural violations and failures *collisions*. Operations that – when applied on individual modules – can globally affect the structure, are sometimes called *linear-strength operations*.

The positive effect of such operations has been studied from a theoretical point of view in a number of papers, for different underlying models and types of operations. In the crystalline model, square modules can expand and contract by extending and retracting their faces. In [2], Aloupis et al. gave a universal centralized reconfiguration algorithm that, for any pair of connected shapes S_I, S_F of the same number of modules n , can transform S_I into S_F within $O(\log n)$ parallel time steps by performing $\Theta(n \log n)$ individual operations.

In [8], Woods et al. proposed the nubot model, motivated by the programmable self-assembly of molecules, such as DNA strands. The model allows insertion, deletion, and rotation of modules. Their main result is a distributed, asynchronous algorithm which, starting from a singleton, can grow any connected 2D shape and pattern of size n , within a polylogarithmic (in n) number of parallel time steps in expectation.

Almalki and Michail [1], building on the insertion operations of [8] and the growth processes on graphs by Mertzios et al. [7], investigated what families of shapes can be grown in time polylogarithmic in their size by using only growth operations. They gave centralized algorithms for growing a shape S_F from a shape S_I (possibly a singleton), which yield polylogarithmic parallel time-step schedules for large classes of shapes.

The amoebot model of Derakhshandeh et al. [4] –and its recent canonical extension [3]– is another model in which the main operations considered are expansions and contractions of modules. Shape formation algorithms in this model are usually designed in a way that operations are parallel but each is affecting only a local region around it and not larger parts of the shape. Recently, Feldmann et al. [5] have proposed to add linear-strength operations to the model, but they have left the details of such an extension for future work.

It is evident that most studies have restricted attention to those operations that are safe to perform in parallel. These are either linear-strength operations that cannot collide or operations that affect only the local region around them. In this paper, we explicitly pose the algorithmic question of determining when a set of operations may cause a collision and when a collision can be avoided. In particular, given a shape and a set of linear-strength operations on that shape we aim to give centralized algorithms that can compute a schedule of these (sets of) operations that would (i) cause a collision or (ii) avoid collisions. The former subquestion is motivated by asynchronous distributed algorithms, in which any of the possible interleavings of operations might be the one that the modules will actually realize; the latter by the need to design efficient reconfiguration algorithms that avoid collisions, instead of having collision-avoidance built into the model. To the best of our knowledge, the present is the first study to explicitly consider these types of questions.

2 Model

We assume a 2-dimensional square grid where each cell has integer coordinates (x, y) . Nodes (modules) occupy cells, defining a set of occupied integer points such that no two nodes occupy the same cell. We represent every node $u = (u_x, u_y)$ as a square of size equal to and

perfectly aligned with cell (u_x, u_y) of the grid. A *shape* $S = (V, E)$ is a configuration of nodes V together with their connectivity, represented by E . Only orthogonally adjacent nodes can be connected, but adjacent nodes are not necessarily connected. We use n to denote $|V|$ and restrict our attention to connected shapes, throughout.

Operations and collisions. In general, applying one or more *operations* to a shape S either causes a *collision* or yields a new shape S' . Collisions come in two types: *node collisions* and *cycle collisions*. Given that all collisions here will be “self-collisions” of a connected shape, we can assume without loss of generality (abbreviated “w.l.o.g.” throughout) that there is an *anchor* node $u_0 \in V$ that is stationary and other nodes move relative to it. We begin with the simpler case where the shape is a tree $T = (V, E)$, where cycle collisions do not exist, and then generalize to any connected shape S .

We start by defining single *expansion* and *contraction* operations¹. An *expansion* operation is applied to a pair of adjacent integer points uv , where either (i) $u \in V$ and $v \notin V$, or (ii) $u, v \in V$ and $uv \in E$ holds. The remaining case where $u, v \in V$ but $uv \notin E$ immediately gives a collision. In case (i), the expansion generates a node at the empty cell v connected to u . In case (ii), assume w.l.o.g. that u is closer to u_0 in T than v . Let $T(v)$ denote the subtree of T rooted at v . Then, the expansion generates a node between u and v , connected to both, which translates $T(v)$ by one unit away from u along the axis parallel to uv . In both cases, the new node starts as a unit-length segment that widens into a unit square. A *contraction* operation is applied to a pair of nodes $uv \in E$, v being the furthest from the anchor. It merges v with u by translating $T(v)$ by one unit toward u while v narrows to a unit-length segment. In both types of operations, if after $T(v)$'s translation two nodes occupy the same cell then a collision has occurred. We call this type of collision a *node collision* and more generally define it as the non-empty intersection of the areas of any two nodes at any point in time. Otherwise, a new tree T' has been obtained.

We assume that no node is ever adjacent to more than one operation.

Coupling. Let Q be a set of operations to be applied *in parallel* to a connected shape S , each operation on a distinct pair of nodes or a node and an unoccupied cell. We call such a set a *coupling*, and the operations it contains are *coupled* or *parallel*. We assume that all operations in Q are applied *concurrently*, have the same *constant execution speed*, and their *duration* is equal to one unit of time.

Let $T = (V, E)$ be a tree and $u_0 \in V$ its anchor. We set u_0 to be the root of T . We want to determine the displacement of every $v \in V \setminus \{u_0\}$ due to the parallel application of the operations in Q . As u_0 is stationary and each operation translates a subtree, only the operations on the unique u_0v path contribute to v 's displacement. In particular, any such operation contributes one of the unit vectors $\langle -1, 0 \rangle, \langle 0, -1 \rangle, \langle +1, 0 \rangle, \langle 0, +1 \rangle$ to the motion vector \vec{v} of v . Moreover, for any node $u \in V$ that expands toward an empty cell, we add a new node v with a corresponding unit motion vector \vec{v} . We can use the set of motion vectors to determine whether the trajectories of any two nodes will collide at any point.

Now, let S be any connected shape with at least one cycle and any node u_0 be its anchor. Then, a set of operations Q on S either causes a *cycle collision* or its effect is essentially equivalent to the application of Q on any spanning tree of S rooted at u_0 . Let u, v be any

¹ We believe that our definitions and techniques can be extended to alternative versions of expansion and contraction – including the case where the operations can be reversed – and to different geometries such as a triangular grid.

two nodes on a cycle. If p_1 and p_2 are the two uv paths of the cycle, then $\vec{v}_{p_1} = \vec{v}_{p_2}$ must hold: the displacement vectors of v along the paths p_1 and p_2 are equal. Otherwise, we cannot maintain all nodes or edges of the cycle. Such a violation is called a *cycle collision*. We call a set of operations that does not cause any node or cycle collisions *collision free*.

Discrete and continuous time. We consider two different models for the scheduling of the operations. In the *discrete-time model*, each operation or coupling starts at a different integer time and takes one unique unit of time. In other words, no two operations are active at the same time unless they are coupled. In the *continuous-time model*, we do not make the integer starting-time assumption. Operations can start at any time and their active times can overlap. Coupled operations start and finish at the same time. Our assumption that each operation takes one unit of time to complete and has constant execution speed holds for both timing models. In the discrete-time model, only the order of the operations (individual or coupled) matters for having collisions or not. In the continuous-time model, the precise starting times of the operations matter.

Problem definitions. We now define the problems considered. Given a shape S and an assignment of operations on S that involve any node at most once, a *coupling partition of operations on S* is a collection of sets $\{Q_1, Q_2, \dots, Q_k\}$, where each Q_i (possibly a singleton) denotes a subset of the operations that should be performed in parallel.

COLLIDING SCHEDULE. Given a shape $S = (V, E)$ from a given family of shapes and a coupling partition of operations $\{Q_1, Q_2, \dots, Q_k\}$ on S , decide if a starting time $t_0(Q_i) \in \mathbb{R}$ for each coupled set Q_i exists such that the application of the operations according to these starting times causes a collision.

COLLISION-FREE SCHEDULE. Given a shape $S = (V, E)$ from a given family of shapes and a coupling partition of operations $\{Q_1, Q_2, \dots, Q_k\}$ on S , decide if a starting time $t_0(Q_i) \in \mathbb{R}$ for each coupled set Q_i exists such that the application of the operations according to these starting times is collision free.

The discrete special cases of these problems, DISCRETE COLLIDING SCHEDULE and DISCRETE COLLISION-FREE SCHEDULE, respectively, are obtained by requiring all $t_0(Q_i)$'s to be unique integers.

3 Algorithms for Colliding Schedule

In this section, we present algorithms to decide whether a connected shape can have collisions for some schedule of operations. We first consider the continuous model followed by the discrete model. We distinguish the cases based on the type of coupling.

We assume that the topology of S is that of a tree. We refer the readers to [6] for details regarding general graphs. In the case of continuous model, we get the following results.

► **Theorem 1.** *Let S be a shape consisting of n unit square nodes with operations defined on the edges between adjacent nodes, and let the adjacency structure of S be a single tree. Then we can solve COLLIDING SCHEDULE*

- *in $O(n^2 \log^2 n)$ time if couplings exist;*
- *in $O(n^2)$ time if each coupling has constant size, or is horizontal-only or vertical-only;*
- *in $O(n \log^2 n)$ time if the operations are not coupled.*

We can also solve DISCRETE COLLIDING SCHEDULE in polynomial time. The algorithm without coupling is still correct, but with coupling we need a different approach. Thus, we get the following results.

- **Theorem 2.** *Let S be a shape consisting of n unit square nodes with operations defined on the edges between adjacent nodes, and let the adjacency structure of S be a single tree. Then we can solve DISCRETE COLLIDING SCHEDULE*
- *in $O(n^{17/3})$ time if couplings exist;*
 - *in $O(n^5)$ time if each coupling has constant size, or is horizontal-only or vertical-only;*
 - *in $O(n \log^2 n)$ time if the operations are not coupled.*

4 Continuous and Discrete Collision-free Schedule

So far we considered detecting whether collisions might occur for an input instance. In this section, we consider the problem of deciding if all operations can be performed without any collisions, for a suitable choice of operation order or starting times. We show that, even if there are only expansions that are w.l.o.g. horizontal and couplings have size $O(1)$, in the discrete-time model the problem is NP-complete. Interestingly, the same problem is solvable in polynomial time in the continuous-time model. When we add vertical expansions, the problem is NP-hard in the continuous-time model.

- **Theorem 3.** *DISCRETE COLLISION-FREE SCHEDULE is NP-complete even if all operations are horizontal expansions and all couplings have size $O(1)$.*
- **Theorem 4.** *COLLISION-FREE SCHEDULE is solvable in linear time if all operations are horizontal.*
- **Theorem 5.** *COLLISION-FREE SCHEDULE is NP-hard.*

References

- 1 Nada Almalki and Othon Michail. On geometric shape construction via growth operations. *Theor. Comput. Sci.*, 984:114324, 2024.
- 2 Greg Aloupis, Sébastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristán Adinolfi, and Stefanie Wuhler. Reconfiguration of cube-style modular robots using $O(\log n)$ parallel moves. In *ISAAC*, volume 5369 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2008.
- 3 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The canonical amoebot model: algorithms and concurrency control. *Distributed Comput.*, 36(2):159–192, 2023.
- 4 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: amoebot - a new model for programmable matter. In *SPAA*, pages 220–222. ACM, 2014.
- 5 Michael Feldmann, Andreas Padalkin, Christian Scheideler, and Shlomi Dolev. Coordinating amoebots via reconfigurable circuits. *J. Comput. Biol.*, 29(4):317–343, 2022.
- 6 Siddharth Gupta, Marc J. van Kreveld, Othon Michail, and Andreas Padalkin. Collision detection for modular robots - it is easy to cause collisions and hard to avoid them. *CoRR*, abs/2305.01015, 2023.
- 7 George B. Mertzios, Othon Michail, George Skretas, Paul G. Spirakis, and Michail Theofilatos. The complexity of growing a graph. In *ALGOSENSORS*, volume 13707 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2022.
- 8 Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *ITCS*, pages 353–354. ACM, 2013.

Brief Announcement: On the Existence of δ -Temporal Cliques in Random Simple Temporal Graphs

George B. Mertzios ✉ 

Department of Computer Science, Durham University, UK

Sotiris Nikolettseas ✉ 

Computer Engineering and Informatics Department, University of Patras, Greece

Christoforos Raptopoulos ✉ 

Department of Mathematics, University of Patras, Greece

Paul G. Spirakis ✉ 

Department of Computer Science, University of Liverpool, UK

Abstract

We consider random simple temporal graphs in which every edge of the complete graph K_n appears once within the time interval $[0, 1]$ independently and uniformly at random. Our main result is a sharp threshold on the size of any maximum δ -clique (namely a clique with edges appearing at most δ apart within $[0, 1]$) in random instances of this model, for any constant δ . In particular, using the probabilistic method, we prove that the size of a maximum δ -clique is approximately $\frac{2 \log n}{\log \frac{1}{\delta}}$ with high probability (whp). What seems surprising is that, even though the random simple temporal graph contains $\Theta(n^2)$ overlapping δ -windows, which (when viewed separately) correspond to different random instances of the Erdős-Rényi random graphs model, the size of the maximum δ -clique in the former model and the maximum clique size of the latter are approximately the same. Furthermore, we show that the minimum interval containing a δ -clique is $\delta - o(\delta)$ whp. We use this result to show that any polynomial time algorithm for δ -TEMPORAL CLIQUE is unlikely to have very large probability of success.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Mathematics of computing \rightarrow Discrete mathematics

Keywords and phrases Simple random temporal graph, δ -temporal clique, probabilistic method

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.27

Funding *George B. Mertzios*: Supported by the EPSRC grant EP/P020372/1.

Christoforos Raptopoulos: Supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the “2nd Call for H.F.R.I. Research Projects to support Post-Doctoral Researchers” (Project Number: 704).

Paul G. Spirakis: Supported by the EPSRC grant EP/P02002X/1.

1 Introduction

Dynamic network analysis, i.e. analysis of networks that change over time, is currently one of the most active topics of research in network science and theory. Many modern real-life networks are dynamic in nature, in the sense that the network structure undergoes discrete changes over time [15, 19, 21]. Here we deal with the discrete-time dynamicity of the network links (edges) over a fixed set of nodes (vertices), according to which edges appear in discrete times and are absent otherwise. This concept of dynamic network evolution is given by *temporal graphs* [12, 16], which are also known by other names such as *evolving graphs* [3, 8], or *time-varying graphs*.



© George B. Mertzios, Sotiris Nikolettseas, Christoforos Raptopoulos, and Paul G. Spirakis; licensed under Creative Commons License CC-BY 4.0

3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024).

Editors: Arnaud Casteigts and Fabian Kuhn; Article No. 27; pp. 27:1–27:5

Leibniz International Proceedings in Informatics



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

► **Definition 1** (Temporal Graph). *A temporal graph is a pair $\mathcal{G} = (G, \lambda)$, where $G = (V, E)$ is an underlying (static) graph and $\lambda : E \rightarrow 2^{\mathbb{N}}$ is a time-labeling function which assigns to every edge of G a discrete-time label. Whenever $|\lambda(e)| \leq 1$ for every $e \in E$, \mathcal{G} is called a simple temporal graph.*

Our focus is on *simple* temporal graphs (in which edges appear only once), as, due to their conceptual simplicity, they offer a fundamental model for temporal graphs and they prove to be good prototypes for studying temporal computational problems. More specifically, we consider simple temporal graphs whose edge labels are chosen *uniformly at random* from a very large set of possible labels (e.g. the label of each edge is chosen uniformly at random within $[1, N]$ where $N \rightarrow \infty$). This can be equivalently modeled by choosing the time labels uniformly at random as real numbers in the interval $[0, 1]$, which leads to the following definition.

► **Definition 2** (Random Simple Temporal Graph). *A random simple temporal graph is a pair $\mathcal{G} = (G, \lambda)$, where $G = (V, E)$ is an underlying (static) graph and $\{\lambda(e) : e \in E\}$ is a set of independent random variables uniformly distributed within $[0, 1]$.*

Note that, in Definition 2, the probability that two edges have equal labels is zero. For every $v \in V$ and every time slot t , we denote the *appearance of vertex v at time t* by the pair (v, t) . For $Q \subseteq V$, the *restricted temporal graph* $(G, \lambda)|_Q$ is the temporal graph $(G[Q], \{\lambda(e) : e \in E(G[Q])\})$.

In the seminal paper of Casteigts, Raskin, Renken, and Zamaraev [5], the authors consider a related (essentially equivalent to ours) model of random simple temporal graphs based on random permutation of edges. They provide a thorough study of the temporal connectivity of such graphs and they provide sharp thresholds for temporal reachability. Their work motivated our research in this paper.

In many applications of temporal graphs, information can naturally only move along edges in a way that respects the ordering of their timestamps (i.e. time labels). That is, information can only flow along sequences of edges whose time labels are increasing (or non-decreasing). Motivated by this fact, most studies on temporal graphs have focused on “path-related” problems, such as e.g. temporal analogues of distance, diameter, reachability, exploration, and centrality [1, 4–7, 10, 13, 14, 16, 20, 24]. In these problems, the most fundamental notion is that of a *temporal path* from a vertex u to a vertex v , which is a path from u to v such that the time labels of the edges are increasing (or at least non-decreasing) in the direction from u to v . To complement this direction, several attempts have been recently made to define meaningful “non-path” temporal graph problems which appropriately model specific applications. Some examples include temporal cliques, cluster editing, temporal vertex cover, temporal graph coloring, temporally transitive orientations of temporal graphs [2, 9, 11, 17, 18, 22, 23].

What is common to most of the path-related problems is that their extension from static to temporal graphs often follows easily and quite naturally from their static counterparts. For example, requiring a graph to be (temporally) connected results in requiring the existence of a (temporal) path among each pair of vertices. In the case of non-path related problems, the exact definition and its application is not so straightforward. For example, defining cliques in a temporal graph as the set of vertices that interact at least once in the lifetime of the graph would be a bit counter intuitive, as two vertices may just interact at the first time step and never again. To help with this problem, Viard et al. [22] introduced the idea of the *sliding time window* of some size δ , where they define a temporal clique as a set of vertices where in all δ consecutive time steps each pair of vertices interacts at least once. There

is a natural motivation for this problem, namely to be able to find the contact patterns among high-school students. Following the idea of Viard et al. [22], many other problems on temporal graph were defined using sliding time windows. For an overview of recent works on sliding windows in temporal graphs, see [15].

In the next definition we introduce the notion of a δ -temporal clique in a random simple temporal graph, and the corresponding maximization problem.

► **Definition 3** (δ -TEMPORAL CLIQUE). *Let (G, λ) be a random simple temporal graph with n vertices, let $\delta \in [0, 1]$, and let $Q \subseteq V$ be a subset of vertices such that $G[Q]$ is a clique. The restricted temporal graph $(G, \lambda)|_Q$ is a δ -temporal clique, if $|\lambda(e) - \lambda(e')| \leq \delta$, for every two edges e, e' which have both their endpoints in Q .*

δ -TEMPORAL CLIQUE

Input: A simple temporal graph (G, λ) .

Output: A δ -temporal clique Q of (G, λ) with maximum cardinality $|Q|$.

Our contribution. In this work, we consider simple random temporal graphs where the underlying (static) graph is the complete graph on n vertices, and we provide a sharp threshold on the size of maximum δ -cliques in random instances of this model, for any constant δ . In particular, using the probabilistic method, we prove that the size of a maximum δ -clique is approximately $\frac{2 \log n}{\log \frac{1}{\delta}}$ whp (Theorem 4). What seems surprising is that, even though the random simple temporal graph contains $\Theta(n^2)$ overlapping δ -windows, which (when viewed separately) correspond to different random instances of the Erdős-Rényi model $\mathcal{G}_{n, \delta}$ (in which edges appear independently with probability δ), the size of the maximum δ -clique and the maximum clique size of the latter are approximately the same. Furthermore, we show that the minimum interval containing a δ -clique is $\delta - o(\delta)$ whp (Theorem 5). We use this result to show that any polynomial time algorithm for δ -TEMPORAL CLIQUE is unlikely to have very large probability of success (Theorem 7). Finally, we discuss some open problems related to the average case hardness of δ -TEMPORAL CLIQUE in the general case.

2 Existence of δ -Temporal Clique

We employ the first and second moment probabilistic methods to show the following threshold property.

► **Theorem 4.** *Let (K_n, λ) be a random simple temporal graph where the underlying graph is the complete graph with n vertices, and let $\delta \in (0, 1)$ be a constant. Define $k_0 \stackrel{\text{def}}{=} \frac{2 \log n}{\log \frac{1}{\delta}}$. As $n \rightarrow \infty$ we have the following:*

(i) *With high probability, (K_n, λ) has no δ -temporal clique of size $(1 + o(1))k_0$.*

(ii) *With high probability, (K_n, λ) contains a δ -temporal clique of size $(1 - o(1))k_0$.*

For the proof of the above theorem, we first give an exact formula for the probability that a graph H appears as a subgraph within a δ -window, and then we show that the expected number $\mathbb{E}[X^{(k)}]$ of δ -cliques of size at most k_0 goes to ∞ (while the expected number of δ -cliques of larger size goes to 0), and also that $\frac{\mathbb{E}[(X^{(k)})^2]}{\mathbb{E}^2[X^{(k)}]}$ goes to 1 for $k \leq (1 - \epsilon)k_0$, as $n \rightarrow \infty$. Furthermore, our main theorem implies the following:

► **Theorem 5.** *Let (K_n, λ) be a random simple temporal graph where the underlying graph is the complete graph with n vertices, and let $\delta \in (0, 1)$ be a constant. Let also $k_0 = \frac{2 \log n}{\log \frac{1}{\delta}}$ and let Q be any δ -temporal clique of size at least $(1 - o(1))k_0$. Define the interval $\Delta(Q) \stackrel{\text{def}}{=} [\min(\lambda(e) : e \in Q), \max(\lambda(e) : e \in Q)]$. Then $|\Delta(Q)| = \delta - o(\delta)$ whp.*

3 Average case hardness implications and open problems

The threshold given in Theorem 4 on the size of the maximum δ -clique reveals an interesting connection between simple random temporal graphs (K_n, λ) and Erdős-Rényi random graphs $G_{n,\delta}$. On one hand, notice that, if we only consider edges with labels within a given δ -window, then the corresponding graph is an instance of $\mathcal{G}_{n,\delta}$, which has maximum clique size asymptotically equal to $k_0 \stackrel{\text{def}}{=} \frac{2 \log n}{\log \frac{1}{\delta}}$ whp. On the other hand, the random simple temporal graph contains $\Theta(n^2)$ different instances of $\mathcal{G}_{n,\delta}$, but the size of a maximum δ -clique size is asymptotically the same. One explanation why this happens is that the different instances of $\mathcal{G}_{n,\delta}$ contained in the random simple temporal graph are highly dependent, even if these correspond to disjoint δ -windows (indeed, edges with labels appearing in one window do not appear in the other and vice versa).

It is therefore interesting to ask whether we can use the above connection algorithmically. One direction is clearly easier than the other: If there is a polynomial time algorithm $\mathcal{A}_{ER}(\delta)$ that can find a clique of size $q = \Theta(k_0)$ in a random instance of $\mathcal{G}_{n,\delta}$ whp, then we can use this algorithm to find an asymptotically equally large δ -clique in a random instance of (K_n, λ) with the same probability of success. We note that, finding a clique of size asymptotically close to k_0 in $G_{n,\delta}$ is believed to be hard in the average case and there is no known algorithm for this problem that runs in polynomial time in n .

For the other direction, we conjecture that the following reduction may be possible:

► **Conjecture 6.** *Suppose that, for any $\delta \in [0, 1]$ there is a polynomial time algorithm $\mathcal{A}_{SRT}(\delta)$ that finds an $(1 - o(1))$ -approximation of a maximum δ -clique in a random instance of (K_n, λ) whp. Then $\mathcal{A}_{SRT}(\delta)$ can be used to design a polynomial time algorithm that finds an $(1 - o(1))$ -approximation of a maximum in $\mathcal{G}_{n,\delta}$ whp.*

It is clear that the probability of success of $\mathcal{A}_{SRT}(\delta)$ in the above Conjecture cannot be equal to 1 unless $P = NP$. In the following Theorem we also prove that the probability of success is unlikely to be too large.

► **Theorem 7.** *Suppose that, for any constant $\delta \in (0, 1)$, the probability of success of algorithm $\mathcal{A}_{SRT}(\delta)$ is $1 - \exp(-\omega(n^2))$. Then $\mathcal{A}_{SRT}(\delta/2)$ can be used to find a clique of size $(1 - o(1))k_0$ in $\mathcal{G}_{n,\delta}$ whp.*

References

- 1 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. Ephemeral networks with random availability of links: The case of fast networks. *J. Par. and Distr. Comp.*, 87:109–120, 2016.
- 2 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *J. Comp. Sys. Sci.*, 107:108–123, 2020.
- 3 A. Anagnostopoulos, J. Lacki, S. Lattanzi, S. Leonardi, and M. Mahdian. Community detection on evolving graphs. In *Proceedings of the 30th NIPS*, pages 3522–3530, 2016.
- 4 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021.
- 5 Arnaud Casteigts, Michael Raskin, Malte Renken, and Viktor Zamaraev. Sharp thresholds in random simple temporal graphs. In *Proceedings of the 62nd FOCS*, pages 319–326, 2021.
- 6 Jessica A. Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *J. Comp. Sys. Sci.*, 119:60–77, 2021.
- 7 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *J. Comp. Sys. Sci.*, 119:1–18, 2021.

- 8 A. Ferreira. Building a reference combinatorial model for MANETs. *IEEE Network*, 18(5):24–29, 2004.
- 9 Thekla Hamm, Nina Klobas, George B. Mertzios, and Paul G. Spirakis. The complexity of temporal vertex cover in small-degree graphs. In *Proceedings of the 36th AAAI*, pages 10193–10201, 2022.
- 10 Klaus Heeger, Danny Hermelin, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Dvir Shabtay. Equitable scheduling on a single machine. In *Proceedings of the 35th AAAI*, pages 11818–11825, 2021.
- 11 Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the bron-kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Soc. Netw. Analysis and Mining*, 7(1):35:1–35:16, 2017.
- 12 D. Kempe, J. M. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the 32nd (STOC)*, pages 504–513, 2000.
- 13 Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: temporally disjoint paths. *Auton. Agents Multi-Agent Syst.*, 37(1), 2023.
- 14 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. The complexity of computing optimum labelings for temporal connectivity. In *Proceedings of the 47th MFCS*, pages 62:1–62:15, 2022.
- 15 Nina Klobas and George B. Mertzios Paul G. Spirakis. Sliding into the future: Investigating sliding windows in temporal graphs. In *Proceedings of the 48th MFCS*, pages 5:1–5:12, 2023.
- 16 George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019.
- 17 George B. Mertzios, Hendrik Molter, Malte Renken, Paul G. Spirakis, and Philipp Zschoche. The complexity of transitively orienting temporal graphs. In *Proceedings of the 46th MFCS*, pages 75:1–75:18, 2021.
- 18 George B. Mertzios, Hendrik Molter, and Viktor Zamaraev. Sliding window temporal graph coloring. *J. Comp. Sys. Sci.*, 120:97–115, 2021.
- 19 O. Michail and P.G. Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2):72–72, January 2018.
- 20 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theor. Comp. Sci.*, 634:1–23, 2016.
- 21 N. Santoro. Computing in time-varying networks. In *Proceedings of the 13th SSS*, page 4, 2011.
- 22 Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theor. Comp. Sci.*, 609:245–252, 2016.
- 23 Feng Yu, Amotz Bar-Noy, Prithwish Basu, and Ram Ramanathan. Algorithms for channel assignment in mobile wireless networks using temporal coloring. In *Proceedings of the 16th MSWiM*, pages 49–58, 2013.
- 24 Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *J. Comp. Sys. Sci.*, 107:72–92, 2020.

