

Algorithmic Methods for Coordinating Swarms of Simple Robots

Von der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation
(kumulative Arbeit)

von
Arne Schmidt
geboren am 26.07.1991
in Gifhorn

| | |
|-------------------------|--------------------------------|
| Eingereicht am: | 28.05.20 |
| Disputation am: | 13.07.20 |
| 1. Referentin/Referent: | Prof. Dr. Sándor Fekete |
| 2. Referentin/Referent: | Prof. Dr. Aaron Becker |
| 3. Referentin/Referent: | Prof. Dr. Christian Scheideler |

2020

Abstract

Reconfiguration and assembly of robots, and manipulation of particles by robots are prominent problems in robotics. Additionally, the concept of online algorithms plays a big role in robotics, because robots always have to deal with events that are unknown in advance. Any wrong handling could lead to catastrophic failure. In this cumulative thesis we present results for these problems when robots are simple, i.e., they have limited computational power, memory size, and energy supply, or may only perform a handful of physical operations. In particular, we consider four categories of problems (A), (B), (C), and (D) that are defined below. Results in categories (A)—(D) are published in papers that appeared in the following proceedings and journals: *Algorithmica*, *International Conference on Robotics and Automation* (ICRA), *Robotics and Automation Letters* (RA-L), *Workshop on Algorithmic Foundations of Robotics* (WAFR), *Workshop on Approximation and Online Algorithms* (WAOA).

For category (A), we consider the assembly problem of robots that can only be moved through global control, i.e., on actuation, every robot moves in the same direction. The goal is to construct a workspace with obstacles (also called a maze) in which all robots are assembled correctly. We consider two variants: In the first variant, only one robot is allowed to be added to a subassembly at a time, whereas in the second variant, several subassemblies may be combined at the same time. We provide several hardness results in the first variant and efficient algorithms for a large class of shapes in both variants.

Category (B) deals with reconfiguring robots through global force in a convex workspace without obstacles. Using static friction at boundary walls allows arbitrary reconfiguration of two robots if the friction is sufficiently large.

Category (C) includes the problem of manipulating an environment by robots. In our model, robots acting as finite automata move on an infinite grid, whose cells may be occupied by particles, and perform physical operations such as picking up and removing particles from the grid. We give an overview that a single robot can perform various operations like bounding, copying, reflecting, and rotating on a given arrangement. However, the arrangement may be disconnected during the transformation, which is problematic in outer space, where parts would simply float away. We further show that two robots are sufficient to perform any of those operation while preserving connectivity.

Finally, category (D) considers the problem of handling events that are unknown in advance. In a system with multiple robots it is often sufficient that only one of them provides a good solution after the input sequence. We demonstrate this for the bin packing problem and provide better results than previous work for a small number of parallel algorithms.

Kurzfassung

Das Rekonfigurieren und Bilden von Formen von Robotern, sowie die Manipulation von Partikeln durch Roboter sind prominente Probleme in der Robotik. Auch das Behandeln von Ereignissen, die im Vorhinein unbekannt sind, ist essentiell, da falsches Handeln schwerwiegende Fehler verursachen kann. In dieser kumulativen Arbeit betrachten wir Roboter mit starken Einschränkungen in Rechenleistung, Arbeitsspeicher, Energieversorgung oder physikalischen Operationen. Die Ergebnisse für diese Probleme, welche in Kategorien (A) bis (D) unterteilt werden, sind in den folgenden Konferenzen und Journals erschienen: *Algorithmica*, *International Conference on Robotics and Automation (ICRA)*, *Robotics and Automation Letters (RA-L)*, *Workshop on Algorithmic Foundations of Robotics (WAFR)*, *Workshop on Approximation and Online Algorithms (WAOA)*.

In Kategorie (A) betrachten wir das Bilden einer Form aus Robotern, wobei die Roboter nur durch eine globale Steuerung bewegt werden können, das heißt, bei einem Signal bewegen sich alle Roboter in die gleiche Richtung. Das Ziel ist, eine Umgebung mit Hindernissen zu erstellen, sodass darin eine vorgegebene Form konstruiert werden kann. Wir unterscheiden zwei Varianten: In der ersten Variante darf nur ein Roboter zu einer bestehenden Konstruktion hinzugefügt werden, während bei der zweiten Variante mehrere Teilkonstruktionen miteinander verknüpft werden dürfen. Wir zeigen komplexitätstheoretische Ergebnisse mehrerer Teilprobleme für die erste Variante und präsentieren effiziente Algorithmen für spezielle Klassen von Formen für beide Varianten.

Kategorie (B) befasst sich mit dem Problem der Rekonfigurierung, wobei Roboter in einer konvexen Form wieder nur durch globale Steuerung bewegt werden können. Durch das Ausnutzen von Reibungskräften an den Wänden können wir zwei Roboter beliebig rekonfigurieren, sofern die Reibungskraft genügend groß ist.

In Kategorie (C) betrachten wir das Problem wie Roboter eine gegebene Ansammlung von Partikeln umstrukturieren kann. Dabei agieren die Roboter wie endliche Automaten, bewegen sich auf einem Gitter, dessen Zellen teilweise durch Partikel belegt ist, und können Partikel auf das Gitter platzieren oder Partikel entfernen. Wir zeigen, dass ein einzelner Roboter verschiedene Operationen wie dem Begrenzen, Kopieren, Rotieren und Skalieren durchführen kann, solange die Zwischenstrukturen unzusammenhängend sein dürfen. Allerdings ist Zusammenhang eine wichtige Eigenschaft, wenn beispielsweise Strukturen im Weltraum konstruiert werden sollen, die sich ohne Zusammenhang auseinander bewegen würden. Sobald zwei Roboter involviert sind, kann der Zusammenhang während der Durchführung dieser Operationen gewährleistet werden.

Als letztes betrachten wir in Kategorie (D) das Problem, dass Roboter unbekannte Ereignisse bearbeiten müssen. Da es oftmals reicht, eine einzige gute Lösung zu erhalten, bietet es sich an, mehrere Roboter zu nutzen, die die gleichen Ereignisse parallel bearbeiten. Wir betrachten diese Herangehensweise am Beispiel des Bin-Packing-Problems.

Acknowledgements

This thesis was developed in the *Institute of Operating Systems and Computer Networks* within the *Algorithm Group* at Technische Universität Braunschweig under the supervision of Sándor Fekete. I want to thank

- Sándor Fekete for the opportunity to join his group, and to work on this thesis. I enjoy that we can always come to him, whenever something is on our mind no matter if it is business or private.
- My former and current colleagues Andreas, Anna, ChRistian, ChriStian, Dominik, Isabel, Linda, Phillip, Sándor, and Ute at TU Braunschweig for interesting discussions and chats, and a nice atmosphere. I also really enjoyed the game nights, karting, and bowling (no, I will not give the trophy back to you, ChRistian!).
- Ute Marchot for being such a friendly and supportive person. In particular, I want to thank Ute for her exceptional administrative work.
- Linda Kleist for the *perfect* laughs, for the “Bäm”, and for reviewing this thesis.
- Christian Rieck for fun times in research and teaching, and for reviewing this thesis.
- Aaron Becker for being such a nice person, full of ideas how to produce practical work based on our theoretical results. It is always fun to work with him.
- All my coauthors Amira Abdel-Rahman, Victor Baez, Aaron Becker, Daniel Biediger, Kenny Cheung, Erik Demaine, Sándor Fekete, Stephan Friedrichs, Neil Gershenfeld, Robert Gmyr, Jonas Grosse-Holz, Andreas Haas, Winfried Hellmann, Michael Hemmer, Michael Hoffmann, Sven von Höveling, Li Huang, Sabrina Hugo, Ben Janett, Phillip Keldenich, Linda Kleist, Irina Kostitsyna, Dominik Krupke, Sheryl Manzoor, Florian Maurer, Joe Mitchell, Eike Niehs, Melanie Papenberg, Michael Perk, Christian Rieck, Christian Scheffer, Christiane Schmidt, Julian Troegel, Mike Yannuzzi, and James Zuber, without whom research would be half as fun.
- All organizers of the Dagstuhl seminar “Algorithmic Foundations of Programmable Matter”. These seminars have a friendly atmosphere and give young researchers like me the opportunity to meet people from different areas such as Robotics, Bio-Informatics, Distributed Algorithms, and Computational Geometry. I really look forward to participating in the next seminar.
- My family, especially my parents Jürgen and Kerstin Schmidt, for all the support over all the years.

I want to conclude this section with a (shortened) quote that was featured in one of my favorite band's song "*Greatest Show on Earth*" by Nightwish; appreciating the moment that out of bazillion possibilities it is I who is here.

"The potential people who could have been here in my place but who will in fact never see the light of day outnumber the sand grains of Arabia. Certainly those unborn ghosts include greater poets than Keats, scientists greater than Newton. We know this because the set of possible people allowed by our DNA so massively exceeds the set of actual people. In the teeth of these stupefying odds it is you and I, in our ordinariness, that are here." — Richard Dawkins, *Unweaving the Rainbow: Science, Delusion and the Appetite for Wonder*.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 1.1 | Preliminaries | 3 |
| 1.2 | Outline | 4 |
| 2 | Related Work | 7 |
| 2.1 | Coordinating Swarms of Particles | 7 |
| 2.1.1 | Reconfiguration | 7 |
| 2.1.2 | Shape Formation | 8 |
| 2.1.3 | Assembly | 9 |
| 2.2 | Manipulating Programmable Matter | 10 |
| 2.2.1 | Reconfiguring Objects | 11 |
| 2.2.2 | Finite Automata as Active Particles | 11 |
| 2.2.3 | Robots and Material | 12 |
| 2.3 | Online Algorithms | 12 |
| 2.3.1 | Related Work on Online Bin Packing With Advice | 13 |
| 2.3.2 | Related Work on Parallel Online Algorithms | 13 |
| 3 | Summary | 15 |
| 3.1 | Assembly With Global Control | 15 |
| 3.1.1 | Tilt Assembly | 15 |
| 3.1.2 | Parallel Tilt Assembly | 21 |
| 3.2 | Reconfiguration With Global Control and Static Friction | 26 |
| 3.3 | Reconfiguring Passive Particles With Finite Automata | 29 |
| 3.3.1 | Without Connectivity Constraints | 29 |
| 3.3.2 | Connected Reconfiguration | 36 |
| 3.4 | Parallel Online Bin Packing | 40 |
| | Bibliography | 45 |
| | Tilt Assembly: Algorithms for Micro-factories That Build Objects with Uniform External Forces | 61 |

| | |
|---|------------|
| Efficient Parallel Self-Assembly Under Uniform Control Inputs | 85 |
| Coordinated Particle Relocation using Finite Static Friction With Boundary Walls | 93 |
| CADbots: Algorithmic Aspects of Manipulating Programmable Matter with Finite Automata (Conference) | 101 |
| CADbots: Algorithmic Aspects of Manipulating Programmable Matter with Finite Automata (Preprint for journal publication) | 117 |
| Recognition and Reconfiguration of Lattice-Based Cellular Structures by Simple Robots | 143 |
| Parallel Online Algorithms for the Bin Packing Problem (Conference) | 151 |
| Parallel Online Algorithms for the Bin Packing Problem (Preprint for journal publication) | 165 |

List of Figures

Introduction

| | | |
|-----|--|---|
| 1.1 | Practical demonstration of Tilt Assembly | 2 |
| 1.2 | Dimensions of the ISS and the Death Star | 2 |

Tilt Assembly

| | | |
|-----|--|----|
| 3.1 | Example for locally convex particles | 16 |
| 3.2 | One-way gadget for Tilt Assembly | 17 |
| 3.3 | Gadgets used in the reduction for the decision problem in 3D Tilt Assembly | 18 |
| 3.4 | Reduction example for the decision problem in 3D Tilt Assembly | 18 |
| 3.5 | Constructing a polyomino in a maze | 20 |
| 3.6 | Approximating a shape for Tilt Assembly | 21 |

Parallel Tilt Assembly

| | | |
|------|---|----|
| 3.7 | Assembling a convex polyomino | 22 |
| 3.8 | Gadget to assemble two subpolyominoes horizontally | 24 |
| 3.9 | Gadget to assemble two subpolyominoes vertically | 24 |
| 3.10 | An example to hierarchically build a polyomino with global control | 25 |
| 3.11 | Worst-case example for 2-cuts and an example that cannot be constructed using 2-cuts | 25 |

Reconfiguration with Global Control and Static Friction

| | | |
|------|--|----|
| 3.12 | Model for global control and static friction | 27 |
| 3.13 | A polygon and its Δ -configuration | 27 |

Reconfiguring passive particles without connectivity constraint

| | | |
|------|---|----|
| 3.14 | Bounding box around a polyomino | 30 |
| 3.15 | Shifting and deconstruction process during the bounding operation | 31 |
| 3.16 | Cleanup process after enclosing the polyomino | 31 |
| 3.17 | Overview of the Turing machine simulation | 32 |

| | |
|---|----|
| 3.18 Intermediate steps of copying a polyomino | 33 |
| 3.19 Intermediate steps of reflecting a polyomino | 34 |
| 3.20 Intermediate steps of rotating a polyomino | 35 |

Reconfiguring passive particles with connectivity constraint

| | |
|--|----|
| 3.21 2D, 3D, and hardware demonstration of building a bounding box | 37 |
| 3.22 2D experiments constructing a bounding box | 38 |
| 3.23 Scaling with connectivity constraints | 38 |
| 3.24 Lattice structure of light-weight material | 39 |
| 3.25 Robots moving on and inside a three-dimensional lattice | 40 |

Parallel Online Bin Packing

| | |
|--|----|
| 3.26 Competitive ratio of PH3 | 42 |
| 3.27 Comparison of PH3 to various other bounds | 43 |

1 Introduction

Reconfiguration of tiny robots in micro- or nanoscale, or manipulation of particles by simple robots comes with challenges. In the former case, small robots have a limited amount of energy supply and may lose the ability to move on their own. Instead, a global force like gravity or an electromagnetic field is often used to control the movements of all robots, i.e., on actuation, all robots move in the same direction. As shown in Figure 1.1, we can assemble particles into a desired shape with global control within a custom-designed maze. By repeating a small sequence of control moves, one particle after another is added to an existing assembly. How can we decide, whether there is a custom-designed maze in which we can construct a certain shape, and how can we perform efficient assembly? On the practical side, constructing such mazes in micro- or nanoscale is difficult because any incorrectly placed obstacle may yield a wrong assembly. When we get rid of these difficulties, we are left with a bounded workspace that contains no obstacles. This raises the question, “how can particles be reconfigured in this specific class of mazes?”

In the latter case, simple, mass-produced robots may have restrictions in memory size but are still able to perform a certain set of physical operations such as grabbing material. A plausible scenario is large-scale constructions in space, where the difficulties of expensive supply chains, scarcity of building materials, dramatic costs and consequences of even small errors, and the limitations of outside intervention in case of malfunctions pose a vast array of extreme challenges. Right now, the largest in-space construction is the International Space Station (ISS, see Figure 1.2). However, the actual construction needs specially trained astronauts and took about 20 years for its current state. This approach clearly is not scalable. While the ISS is relatively small ($109\text{m} \times 51\text{m} \times 73\text{m}$), the construction of larger structures like the Death Star at the same rate could take about 30 million years, according to [techrepublic.com](https://www.techrepublic.com/blog/geekend/how-much-and-how-long-would-it-take-nasa-to-build-a-death-star/)¹. This raises the question, “how can elementary operations like scaling be performed by a considerable amount of simple robots instead of relying on highly trained astronauts?”

Another problem that arises in space are unknown events. Robots may malfunction due to wrong handling and thus, may not be able to complete their tasks. One way

¹<https://www.techrepublic.com/blog/geekend/how-much-and-how-long-would-it-take-nasa-to-build-a-death-star/>

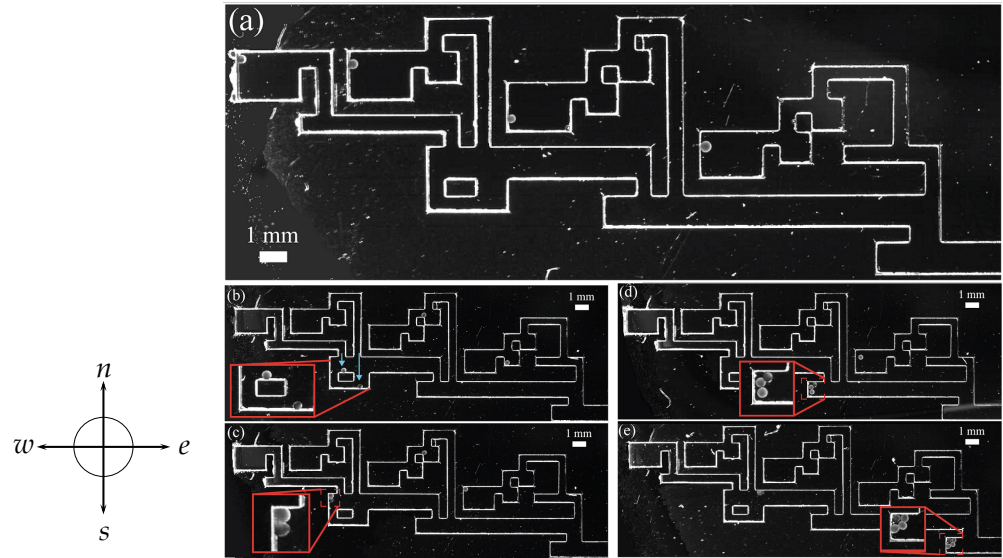


Figure 1.1: A practical demonstration of assembling magnetized alginate (i.e., a gel made by combining a powder derived from seaweed with water) particles [124]. Obstacles appear as white lines and block the motion of particles. (a) Particles in initial positions. (b) Situation after control moves of $\langle e, s, w, n, e, s \rangle$ (for east, south, west, north). (c) Situation after $\langle w, n \rangle$ input. (d) The next subshape is produced after applying multiple $\langle e, s, w, n \rangle$ cycles. (e) Final shape consisting of four particles.

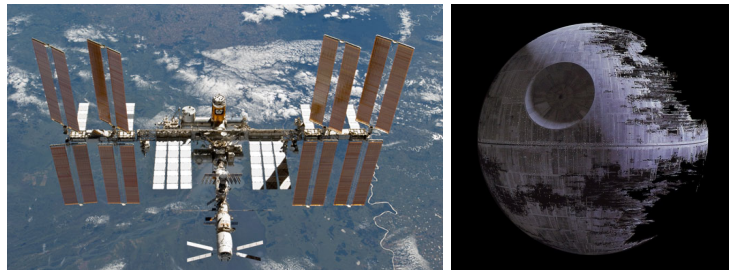


Figure 1.2: Left: The international space station [129] with dimensions $109\text{m} \times 51\text{m} \times 73\text{m}$. Construction began in 1998. Right: Death Star with a radius of approximately 160km . Picture taken from wallpapersafari.com.

of handling this problem is to use multiple robots performing the same task. Another benefit of this redundancy is that each robot could handle the events differently. In the end, at least one robot survives and can complete the task. Further examples where redundancy is used against extreme failure include scenarios from biology, where a large and diverse progeny increases the odds of surviving offspring; finance and insurance, where a suitable combination of investment strategies is employed to balance a portfolio against extreme losses; and engineering, where redundancy is used to protect against catastrophic failure, either on individual components (such as parts in a machine) or on whole systems (such as automata in a robot swarm or spacecraft in a group of satellites), for which it suffices that just one machine delivers a good outcome. This raises the question, “how can multiple strategies be synchronized such that there is always at least one strategy with a good outcome?” As is common for online problems, the quality of the solution is analyzed with the *competitive ratio*, i.e., the cost of our solution compared to the cost of an optimal offline solution.

1.1 Preliminaries

In this thesis, we present results for four categories (A) *assembly* of polyominoes and (B) *reconfiguration* of particles using global force, (C) *manipulation* of passive particles by simple robots, and (D) *event handling* of identical machines to guarantee a better outcome.

(A) Tilt Assembly: Consider a polyomino P , i.e., a set of unit squares connected through edge-to-edge contact. Decide whether there exists a workspace, a configuration of square-shaped particles, and a sequence of moves that assemble the polyomino P from particles under the following constraints. The workspace is a grid of empty and occupied pixels. Pixels may be occupied by a fixed element called walls, or by movable particles. A move is defined by a direction d . Every particle moves as long as possible into the direction d until a wall or another particle is hit. Whenever two particles meet, i.e., they occupy adjacent pixels, they stick together. If only one particle can be added at a time step, we call this the *Tilt Assembly Problem* (A.1). If instead whole subassemblies can be combined in a single step, then we refer to this problem as the *Staged Tilt Assembly Problem* (A.2).

To allow a pipelined construction, i.e., by repeating a sequence of moves we obtain several copies of the same shape, we relax the constraint that any two particles stick together. Instead, we allow two different types of particles, namely red and blue particles. Two particles stick together when they are of different colors. With this

relaxation, particles of the same color can be kept in depots without sticking to each other.

(B) Tilt Reconfiguration: Given a set of particles lying in a convex polygon, the task is to find a sequence of moves such that each robot ends in the target position under the following constraints. The workspace does not contain fixed obstacles but particles induce static friction when lying at a boundary wall. Similar to the assembly problem, all particles are controlled by a global force. However, by switching off the actuation, all particles stop immediately. We refer to the problem of reconfiguring particles with friction along boundary walls using global force by (B.1).

(C) Manipulation with Finite Automata: Given a set of robots on a grid arrangement of empty and occupied pixels, the task is to reconfigure a polyomino defined by occupied pixels. The reconfigurations may be the outcome of transformations like a reflection, or a rotation. The robots may perform physical tasks like placing or removing particles, or moving to an adjacent particle. We further assume that all robots have constant memory, i.e., they act as finite automata.

We consider two variants (C.1) and (C.2): In the first variant, only one robot is allowed. However, the robot may move on empty pixels and may not be connected to the possibly disconnected set of particles. Because unconnected parts would float away in space, we add the constraint that intermediate configurations must be connected in the second variant.

(D) Parallel Online Algorithms: Consider k online algorithms A_1, \dots, A_k , each of them processing the same input list $I := [\sigma_1, \dots, \sigma_m]$ of m items in parallel. For some $1 \leq i \leq m$, as soon as item σ_i arrives, any online algorithm must already have processed items σ_1 to σ_{i-1} without handling these items again. We call the set $\mathcal{A} := \{A_1, \dots, A_k\}$ a *k-copy online algorithm*. The cost of a k -copy online algorithm is evaluated as the smallest cost of any algorithm $A_i \in \mathcal{A}$.

We apply this model to the bin packing problem, where the task is to pack a given sequence of items of size $[0, 1]$ into as few unit sized bins as possible. We refer to this problem as the *Parallel Online Bin Packing Problem* (D.1).

1.2 Outline

This thesis is organized as follows. Chapter 2 provides an overview of all mentioned problems and related work. We start with coordinating swarms of particles (see Section 2.1). In particular, we give an overview of reconfiguration problems in different models in

Section 2.1.1 and proceed with a subproblem: *Shape formation*. In this special case, the target configuration is a connected shape (see also Section 2.1.2). The third part of this section, i.e., Section 2.1.3, is dedicated to assembly problems.

In Section 2.2, we list previous work on how robots can manipulate a given arrangement of particles or reconfiguring objects. While Section 2.2.1 gives an overview of practical oriented work, Section 2.2.2 focuses on more theoretical results and sketches possibilities and impossibilities using simple robots acting as finite automata. A short survey of the current state of materials and robots used for ultralight construction is given in Section 2.2.3. The last section gives an overview on various online algorithms for bin packing (see Section 2.3), online algorithms with advice (see Section 2.3.1), and parallel online algorithms (see Section 2.3.2).

Chapter 3 then highlights results from the following conference paper and journal articles from the past years.

- (A.1) The journal article “*Tilt Assembly: Algorithms for Micro-factories That Build Objects with Uniform External Forces*” [28] appeared 2020 in the *Algorithmica* special issue: *Algorithms and Computation* (see Section 3.1.1).
- (A.2) The journal article “*Efficient Parallel Self-Assembly Under Uniform Control Inputs*” [146] appeared 2018 in *Robotics and Automation Letters* (see Section 3.1.2).
- (B.1) The journal article “*Coordinated Particle Relocation using Finite Static Friction With Boundary Walls*” [145] appeared 2020 in *Robotics and Automation Letters* (see Section 3.2).
- (C.1) The conference article “*CADbots: Algorithmic Aspects of Manipulating Programmable Matter with Finite Automata*” [75] appears 2020 in *International Workshop on Algorithmic Foundations of Robotics* (see Section 3.3.1). Also attached to this thesis is a preprint version submitted for journal publication.
- (C.2) The conference paper “*Recognition and Reconfiguration of Lattice-Based Cellular Structures by Simple Robots*” [131] appeared 2020 in *International Conference on Robotics and Automation* (see Section 3.3.2).
- (D.1) The conference paper “*Parallel Online Algorithms for the Bin Packing Problem*” [76] appeared 2020 in *Workshop on Approximation and Online Algorithms* (see Section 3.4). Also attached to this thesis is a preprint version submitted for journal publication.

2 Related Work

In this section, we give an overview of existing work related to the problems considered in this thesis. We start with coordinating a swarm of particles in different models and problem settings, i.e., work related to categories (A) and (B). After this, we consider work related to category (C), i.e., reconfiguration problems regarding simple robots and passive particles. Finally, we survey related work on parallel online algorithms for category (D).

2.1 Coordinating Swarms of Particles

We split this section into three subsections. We begin with related work for reconfiguration problems and proceed with subproblems, namely shape formation and assembly.

2.1.1 Reconfiguration

Coordinating movable objects has a long history. In 1979, Reif [138] showed that the *movers problem* can efficiently be solved in two and three dimensions. Given a set R of polyhedral obstacles and a shape S , the movers problem asks whether S can be moved from an initial position to a target position by only allowing translations and rotations. Schwartz and Sharir [147] showed in 1983 how multiple circular objects can be moved collision-free through a region containing obstacles. Their algorithm is polynomial in the complexity of the region but exponential in the number of movable objects. Rubenstein et al. [144] present algorithms for robots that are able to perform local operations to avoid collisions among themselves. They evaluated their algorithms with a swarm of over a thousand kilobots [142]. A survey from 2017 about robotics and algorithmic motion planning can be found in [98, 99]. Demaine et al. [61] showed how a set of individual movable robots in free space can be reconfigured with constant stretch, i.e., the makespan is only a constant larger than an optimal makespan. However, if the robots are too densely packed, no constant stretch is possible. See also [27] for a visual presentation.

With robots getting smaller, using an external force for moving the robots becomes inevitable at some scale because the energy capacity decreases faster than the energy demand. On actuation, all non-fixed robots/particles perform a movement into the same direction until they hit an obstacle or another particle. Thus, making use of obstacles

allows to reconfigure a set of particles. Deciding whether a given initial configuration of particles in a given environment can be transformed into a desired target configuration is NP-hard [22]. This even holds in a grid-like setting. Finding an optimal control sequence is shown to be PSPACE-complete by Becker et al. [24]. However, if designing the obstacles is allowed, arbitrary permutations of particles arranged as a rectangle is possible [22]. Moreover, even complex computations become possible: If we allow additional particles of double size (i.e., two adjacent fields), full computational complexity is achieved, see Shad et al. [149]. Further related work includes gathering a particle swarm at a single position within polyominoes [26, 122] or Jordan regions [128]. For the case in which human controllers have to move objects by such a swarm, Becker et al. [25] study different control options. The results are used by Shahrokhi and Becker [151] to investigate an automatic controller.

Considering this approach of navigation by a global external force gives rise to a number of problems, including navigation of one particle from a start to a goal position [117], particle computation [23, 24], or emptying a polygon [9]. Although the kilobots have individual actuation and energy supply, they often make use of an external light source for navigation [142]; as a consequence, directing a swarm of kilobots by switching on a light beacon works just like activating an external force.

Recent work by Zhang et al. [174, 175] shows that there exists a workspace a constant factor larger than the number of agents that enables complete rearrangement for an initial rectangular configuration of agents. Shahrokhi et al. [152, 153] considered reconfiguration problems of particles using friction at the boundary walls. However, they assume walls have infinite friction, i.e., a particle lying at a wall will not move when there is a resulting force parallel to the wall. Schmidt et al. [145] provide reconfiguration strategies in convex workspaces using finite static friction.

2.1.2 Shape Formation

When robots have to reconfigure into a connected set with possibly connected intermediate configurations, then we call the reconfiguration *shape formation*. There are a number of different, related basic models. Arbuckle and Requicha [15] show how a self-organized swarm of robots can construct a certain shape. In case of robot failures or external disturbance, the swarm is also able to repair the shape.

In the *crystalline model* [163], robots are cubes with expandable and contractible arms. With these arms, the robots can push themselves away or pull themselves to other robots. Aloupis et al. [11] show that $O(\log n)$ parallel steps suffice to transform an arbitrary shape into a target shape. In a subsequent paper, Aloupis et al [10] consider the model under physical constraint like maximum velocity and strength of a single robot.

Walter et al. [164] consider a model of hexagonal robots that can move on a hexagonal grid. They provide algorithms that are (asymptotically) optimal in the number of moves. Similar to this model, Derakhshandeh et al. [63] introduced a fundamental concept for studying algorithmic approaches for extremely simple robots, called the *Amoebot model*. In this model, decentralized active particles move on hexagonal cells by expanding (and thus occupying two cells) and contracting. With only a few rules how the particles have to move, the particles can form shapes like lines, triangles or hexagons [65, 67]. By first performing a leader election, further operations are possible [57, 68]. An algorithm for universal shape formation was presented by Di Luna et al. [70] and by Derakhshandeh et al. [66]. The problem of coating an arbitrarily shaped object by particles was solved by Derakhshandeh et al. [67] and analyzed in [64].

When robots are able to duplicate themselves, shapes like lines and squares can be constructed in polylogarithmic time [171]. Further work regarding the *Nubot model* can be found in [46, 47]. In the *Claytronics Project* [91, 92], micro-robots called Claytronic atoms (or in short: Catoms) stick together and can move on the perimeter of each other. Various work shows practical and theoretical results using Catoms [35, 36, 130, 136, 159, 161]. Butler et al. [41] introduced a generic model to reconfigure cellular automata. This work is then extended by Hurtado et al. [103]. They show that any reconfiguration can be done in $O(n)$ steps.

Other related work includes shape formation in a number of different models, such as agents walking on DNA-based shapes [139, 160, 169], or variants of population protocols [127]. Further work includes approaches like self-folding matter [78, 100, 115] or self-disassembling magnetic robot pebbles [87].

2.1.3 Assembly

When particles start with a disconnected configuration, stick together on contact, and the goal is a connected shape, we call this *assembly*. In *DNA tile self-assembly*, the particles are equipped with sophisticated bonds that ensure only a predesigned shape is produced when mixing together a set of tiles, see [170]. This *abstract Tile Assembly Model* (aTAM) uses Wang Tiles [165], which have a square shape and glue types on every side. Tiles with equal glues on a square side can stick together to form larger arrangements of tiles, provided their bonding *strength* exceeds the threshold value τ , called the *temperature*. For early examples of related work, see Rothmund and Winfree [141] and Adleman et al. [4] for the running time and program size for self-assembling squares. Apart from the aTAM, there are various other models like the *Two-Handed Tile Assembly Model* (2HAM) [42] and the *Hierarchical Tile Assembly Model* [45], in which we have no single seed but pairs of subassemblies that can attach to each other. Furthermore, the *Staged Tile Assembly*

Model [43, 44, 59, 62, 148] allows greater efficiency by assembling polyominoes in multiple bins which are gradually combined with the content of other bins.

The aTAM and its extensions (e.g. [3, 5, 155]) are used for further research in tile-based self-assembly. Another generalization was defined by Fekete et al. in [77], where they use polyominoes, which consist of any number of unit square tiles, instead of only single square tiles. Furthermore, Demaine et al. [60] show that if polygonal tile shapes are allowed, any square-shape based assembly system can be simulated by a polygonal tile assembly system consisting of a single tile. A survey of DNA self-assembly can be found in [134].

In DNA self-assembly, the assembly process is non-deterministic and works by diffusion. When particles cannot move on their own but can be controlled by a global force (e.g., gravity or magnetic fields) as seen in Section 2.1.1, we consider *Tilt Assembly*. Recent practical work by Manzoor et al. [124] shows, it is possible to use “sticky” particles that can be forced to bond: the overall assembly is achieved by adding particles one at a time, attaching them to the existing sub-assembly. Becker et al. [28] show that it is possible to decide in polynomial time if a hole-free polyomino can be constructed and provide several theoretical results for two-dimensional and three-dimensional shapes. Schmidt et al. [146] provide algorithms for assembling subassemblies instead adding single tiles to a seed assembly. Balanza et al. [17, 18] give complexity results for combining single tiles or subassemblies.

This approach of externally movable tiles has been considered in practice at the microscale level using biological cells and an MRI, see [29, 112, 113]. Becker et al. [30] consider this for the assembly of a magnetic *Gauß gun*, which can be used for applying strong local forces by very weak triggers, allowing applications such as micro-surgery.

2.2 Manipulating Programmable Matter

In this section we consider active particles (e.g., robots) that reconfigure, shape, assemble, or disassemble passive particles (e.g., building material, or polygonal objects). In general, a discrete grid is considered on which the robots can walk and move particles. When the grid is the square grid, connected shapes of passive particle are also called polyominoes. The first appearance of polyominoes is in the work from Golomb [93]. Future work then considered different problems regarding polyominoes: Packing rectangles with polyominoes [96, 114, 137, 157], or tiling the plane [21, 94]. Work for three-dimensional tiling can be found in [172]. Tiling the plane or a given polyominoe with shapes is important in our setting because robots could simply construct a scaffold and tile this with simple shapes.

2.2.1 Reconfiguring Objects

On the practical side, recent years have seen significant progress towards realizing systems with simple robots moving on programmable matter. For example, it has been shown that nanomachines have the ability to act like the head of a finite automaton on an input tape [139], and to walk on a one- or two-dimensional surface [120, 132, 169]. Thubagere et al. [154, 160, 166] show that robots made from DNA can simultaneously sort molecular cargoes. Rubenstein et al. [143] consider a swarm of simple robots moving an object to a desired destination without knowing its shape and weight. Becker et al. [31] extend this work by using swarms of kilobots for moving objects. Hoffmann [102] proves that it is NP-hard to decide if a robot can push its way through an area filled with movable blocks. Akella et al. [6] consider the problem of reconfiguring an object on a conveyor belt with a simple robot, and Lynch et al. [121] use a mobile robot with a flat pusher plate as the gripper to manipulate objects. Sliding a component using an active tilting tray has a rich history, especially on sensorless part orientation, see [74, 123]. Similar work also applies to using sliding-jaw grippers with low-friction contact surfaces to localize parts without sensing [90]. Werfel and Nagpal [167, 168] show how multiple robots can move tiles to a partial assembly to construct a desired shape in 2D and 3D. For a connection to other approaches to agents moving tiles see [58, 158].

2.2.2 Finite Automata as Active Particles

When dealing with micro- and nano-robotics, memory may be limited. Therefore, considering robots with constant memory size is crucial. On the algorithmic side, there has been a considerable amount of work dealing with robots or agents on graphs. Blum and Kozen [33] showed that two finite automata can jointly search any unknown maze. Other work has focused on exploring general graphs (e.g., [79, 84, 133]), as a distributed or collaborative problem using multiple agents (e.g. [32, 39, 56, 82]) or with space limitations (e.g. [72, 83–86]). Recent work by Disser et al [73] shows that for agents with constant memory $\Theta(n \log n)$ pebbles are always sufficient and sometimes necessary to explore a graph.

Another problem related to robots on graphs is the rendezvous search problem [12], where two robots are searching for each other in a known or unknown environment. Anderson and Weber [13] presented an optimal strategy for discrete locations. The deterministic rendezvous problem as a variant of the previously described problem was also a subject of research [69, 125, 135, 156]. Considering more than two robots, we face the gathering problem, which was investigated on graphs [54, 71, 111] or, related to more realistic robotic scenarios, in the plane [49, 53, 80]. Further work focusing on

a setting of robots on graphs includes network exploration [55], intruder caption and graph searching [34, 81], and black hole search [126]. Models based on automate and movable objects have also been studied in the context of one-dimensional arrays, e.g., pebble automata [150].

Gmyr et al. [89] introduced a model with two types of particles: active robots acting like a deterministic finite automaton moving on passive tile particles. They present algorithms for shape formation [89] and shape recognition [88] using this *Hybrid model*. Using the same model with some modifications, Fekete et al. [75] introduced more complex geometric algorithms for copying, reflecting, rotating and scaling a given polyomino as well as an algorithm for constructing a bounding box surrounding a polyomino. One modification with a big impact is that the robot and the particles may now be disconnected during the transformation. As shown in [131], we are able to adapt all our presented algorithms to maintain connectivity during the runtime, provided that we are allowed to use at least two robots or one robot and a special marker. Practical motivation for this work arises both at very small and very large dimensions. See also the overview in [23] for a discussion of work on particle computation and programmable matter, and [1, 131] for a description of possible applications for building large-scale structures in space.

2.2.3 Robots and Material

On the practical side *ultra-light material* has been developed, as described by Cheung and Gershenfeld [48], and Gregg et al. [95]. Modular two-dimensional elements mechanically link in 3D to form reversibly assembled composite lattices. This process is not limited by scale, and it enables disassembly and reconfiguration. As shown by Cramer et al. [51] and Jenett et al. [105], large but light-weight structures can be built from these components. Jenett et al. have developed autonomous robots that move on the surface [104, 107] or within the cellular structure [106]. With the help of these robots, individual cells can be attached to an existing assembly, or moved to a different location. An approach for global optimization of a corresponding motion plan has been described by Costa et al. [50], while the design of hierarchical structures was addressed by Jenett et al. [108].

2.3 Online Algorithms

Online algorithms have a rich history. However, we will only give an overview of online bin packing and online algorithms with advice in this section. For an overview for general online algorithms, e.g., see the survey of Albers [7].

Regarding bin packing, there is a wide range of classical online algorithms. The NEXT FIT algorithm [52] achieves a competitive ratio of 2, whereas ALMOST ANY

FIT algorithms [109] like FIRST FIT or BEST FIT achieve competitive ratios of 1.7. An important online bin packing algorithm is HARMONIC_M, introduced by Lee and Lee [116], which achieves a competitive ratio of less than 1.692 for $M \rightarrow \infty$. In this algorithm, M is an input parameter denoting how many different item classes should be considered. Based on HARMONIC_M, SON OF HARMONIC by Heydrich and van Stee [101] achieves a competitive ratio of 1.5816. The currently best known algorithm is ADVANCED HARMONIC, which achieves a competitive ratio of 1.57829 [19].

For lower bounds, Yao [173] established a value of $3/2$ that was later improved to 1.536, independently by Brown [40] and by Liang [118]. Using a generalization of their methods, van Vliet [162] proved a lower bound of 1.5401. Balogh et al. [20] improved the lower bound to 1.54278.

2.3.1 Related Work on Online Bin Packing With Advice

In the context of online algorithms with advice, Boyar et al. [38] showed that an online algorithm with $n \lceil \log(OPT(I)) \rceil$ bits of advice is sufficient and that at least $(n - 2OPT(I)) \cdot \log(OPT(I))$ bits of advice are necessary to achieve optimality. Here, n denotes the length of the input sequence I , and $OPT(I)$ denotes the cost of an optimal offline solution. In the same paper, they presented an online bin packing algorithm, namely RESERVECRITICAL, with $O(\log(n)) + o(\log(n))$ bits of advice that is 1.5-competitive and an algorithm with $2n + o(n)$ bits of advice that is $\frac{4}{3}$ -competitive. Zhao and Shen [176] developed an algorithm using $3n + o(n)$ bits of advice achieving a competitive ratio of $\frac{5}{4}OPT + 2$. Renault et al. [140] developed an $(1 + \varepsilon)$ -competitive algorithm using $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ bits of advice per request.

Based on RESERVECRITICAL, Angelopoulos et al. [14] developed the algorithm RED-BLUE with constant advice that is 1.5-competitive. Their second algorithm achieves a competitive ratio of $1.47012 + \varepsilon$ with finite advice that is exponentially dependent on ε . However, to beat the competitive ratio of 1.5 an enormous amount of advice is needed, which makes the algorithm impractical.

In terms of lower bounds, Boyar et al. [38] proved that no competitive ratio better than $9/8$ can be reached by any algorithm that uses sub-linear advice. Angelopoulos et al. [14] improved this bound to $7/6$.

2.3.2 Related Work on Parallel Online Algorithms

Parallel algorithms have already been considered in the field of online algorithms with advice. Boyar et al. [37] presented an algorithm for the online list update problem, making use of 2 bits of advice to choose one out of three algorithms. This algorithm

achieves a competitive ratio of $5/3$, beating the lower bound for conventional online algorithms of 2. A practical application of this algorithm was shown by Kamali and Ortiz [110], who applied it in the Burrows-Wheeler transform compression. More work on parallel online algorithms include parallel scheduling [8], finding independent sets [97] and the “multiple-cow” version of the linear search problem [119].

While online algorithms with advice mostly focus on the amount of advice to allow classification of online algorithms and problems, k -copy online algorithms focus on small finite values for k and thus small finite amounts of advice, with more emphasis on practical application. The perspective on different algorithms running in parallel instead of abstract arbitrary information facilitates finer optimization in some cases.

Also, when considering online algorithms with advice, the number of algorithm can only be doubled by increasing the amount of advice by one bit. The perspective of k -copy algorithms allows arbitrary $k \in \mathbb{N}$ for the number of algorithms.

3 Summary

This section highlights the results from our papers attached to this thesis and provides a set of open problems for each category. In addition to the published conference version for (C.1) and (D.1), we also attach a version submitted for journal publication.

3.1 Assembly With Global Control

This first section is dedicated to category (A): Tilt Assembly. We start with providing results for (A.1), i.e., when only one particle at a time can be added. In the second subsection we show results for (A.2), i.e., when multiple subpolyominoes can be combined in a single step.

3.1.1 Tilt Assembly

In our paper “*Tilt Assembly: Algorithms for Micro-factories That Build Objects with Uniform External Forces*” [28], we consider the problem (A.1), i.e., assembling a polyomino from N particles through global control with the restriction that only one particle at a time can be added to the seed assembly. We assume that the initial seed assembly is a single particle. Furthermore, we consider a relaxed model, in which the current assembly P is fixed in space and particles are shot at P . More precisely, we consider the following model: Given a direction $d \in \{n, e, s, w\}$ (for *north*, *east*, *west*, and *south*) and $\ell \in \mathbb{Z}$, a *construction step* is defined by (d, ℓ) . In a construction step, a particle t arrives from coordinates (ℓ, ∞) (i.e., from the north), (∞, ℓ) (i.e., from the east), $(\ell, -\infty)$ (i.e., from the south), or $(-\infty, \ell)$ (i.e., from the west) and moves in direction d until t is adjacent to a particle of P . A polyomino P is constructible if and only if there is a sequence $\sigma = ((d_1, \ell_1), (d_2, \ell_2), \dots, (d_{N-1}, \ell_{N-1}))$, such that the resulting polyomino P' , induced by successively adding tiles with σ , is equal to P .

Clearly, the relaxed model can be converted back to the original model. Given a construction sequence σ for a polyomino P , we can show how to generate a maze from σ , in which we can produce several copies of P , with each copy assembled from N particles in amortized constant time.

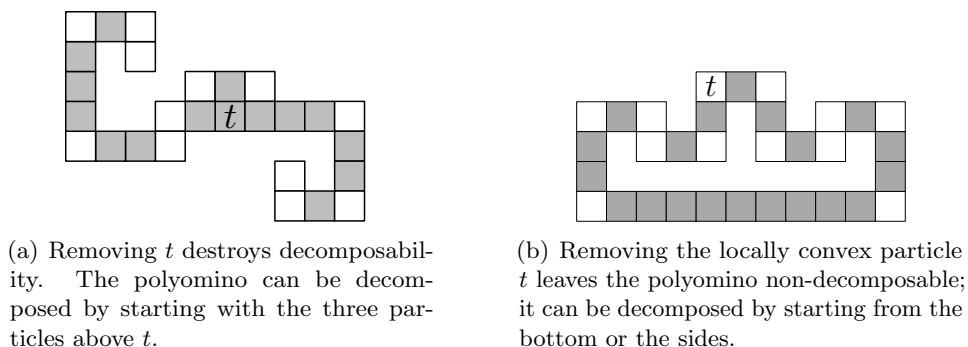


Figure 3.1: Two polyominoes and their locally convex particles (white). (a) Removing non-locally convex particles may destroy decomposability. (b) With non-simple polygons we may not be able to remove locally convex particles.

To decide if a polyomino P is constructible, we look at the reverse. P is decomposable if and only if there is a particle t such that $P \setminus t$ is decomposable and such that we can remove t from P along a vertical or horizontal line without hitting or sliding along another particle. Reversing this *decomposition sequence* yields a construction sequence.

Now, the goal is to identify particles in P that we can remove without destroying constructibility. Figure 3.1 shows that removing certain particles may destroy decomposability. A class of particles preserving decomposability after their removal for *simple* polyominoes (i.e., polyominoes that do not contain holes) is the set of locally convex particles. We call a particle t *locally convex* if we can place a 2×2 square S such that S only contains t . With that, we can remove one locally convex particle after another yielding a decomposition sequence. By reversing this sequence we obtain a construction sequence. If at some point no locally convex particle can be removed, then P is not constructible. By using balanced trees of particles for every x - and y - coordinate, we achieve a runtime of $O(N \log N)$.

Theorem 1. *The decision problem, whether a simple polyomino with N pixels can be constructed, can be solved in time $O(N \log N)$.*

For *non-simple* polyominoes (i.e., polyominoes with holes) the hardness of the decision problem is still open. One problem is that the removal of a locally convex particle can destroy constructibility (see Figure 3.1(b)).

This leads us to the following question. What can we do for polyominoes that are not constructible in this model or for which we do not know if they are constructible? We also consider the maximization problem: For a given polyomino P , what is the largest

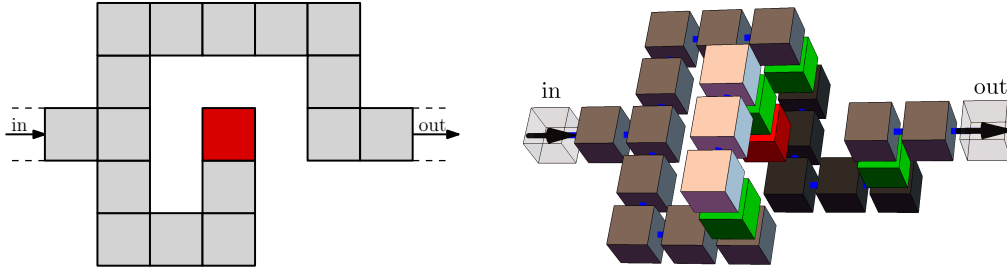


Figure 3.2: (Left) This polyomino can only be constructed by starting at “in” and ending at “out”. (Right) Generalization to three dimensions. If we start on the right side, then we cannot insert the red cube because it is blocked from all six directions. With these gadgets we can enforce a seed particle.

$P' \subseteq P$ that is constructible? By a reduction from the independent set problem, we see that it is hard to even approximate the polyomino P' better than $\Omega(N^{\frac{1}{3}})$. However, an $O(\sqrt{N})$ approximation is possible with the following strategy, if P is simple. Consider an optimal solution P^* and a smallest enclosing box B containing P^* . Then, there are two pixels on the boundary, such that the shortest path S between both pixels has length at least $\sqrt{|P^*|}$. To find a constructible path that has length at least $|S|$, we check for any two pixel p_1 and p_2 in P if a shortest path from p_1 to p_2 within P is constructible. Clearly, taking the longest constructible path has length at least $|S|$ and is thus a \sqrt{N} approximation.

What about 3D?

In 3D, we can easily prove the decision problem to be NP-hard by a reduction from 3SAT. By using a one-way gadget as shown in Figure 3.2, we can enforce a construction sequence to begin with this gadget. Then a layer is used to choose a true/false assignment of variables (see Figure 3.3(a)). Beneath this layer we use clause gadgets (see Figure 3.3(b)). Only if at least one of the three parts is kept free from above, then we can construct this gadget completely. Therefore, we can construct this polycube if only if the 3SAT formula is satisfiable. An example of the resulting polycube can be seen in Figure 3.4.

Theorem 2. *Deciding whether a polycube is constructible is NP-hard.*

Similar to the two-dimensional case, approximating polycubes is NP-hard within a certain factor. However, no approximation algorithm is known so far. The difficulties of 3D constructions is further highlighted by the fact that even deciding if there exists

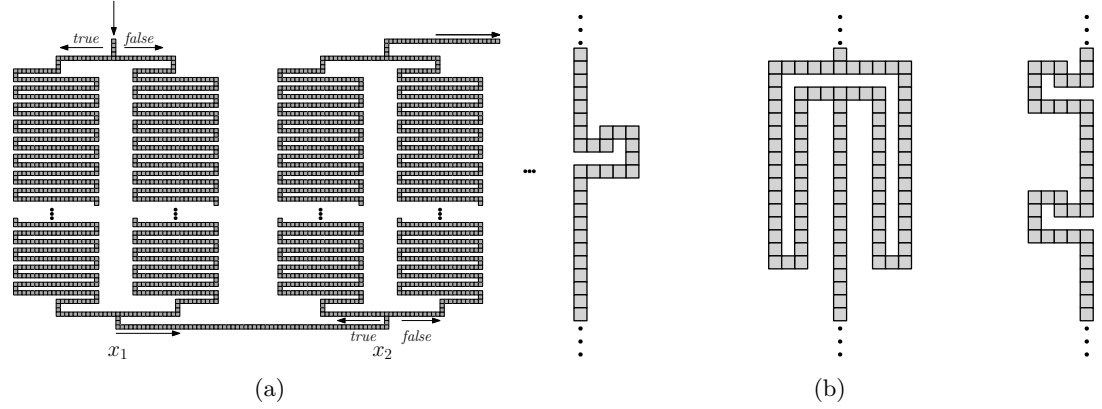


Figure 3.3: Top-view on the polycube. (Left) In the beginning we have to block the access from the top for either the *true* or *false* part of the variable. The variable is assigned the blocked value. (Right) Three gadgets for a clause. Only two of them can be built if the particles are only able to come from the east, south, and west.

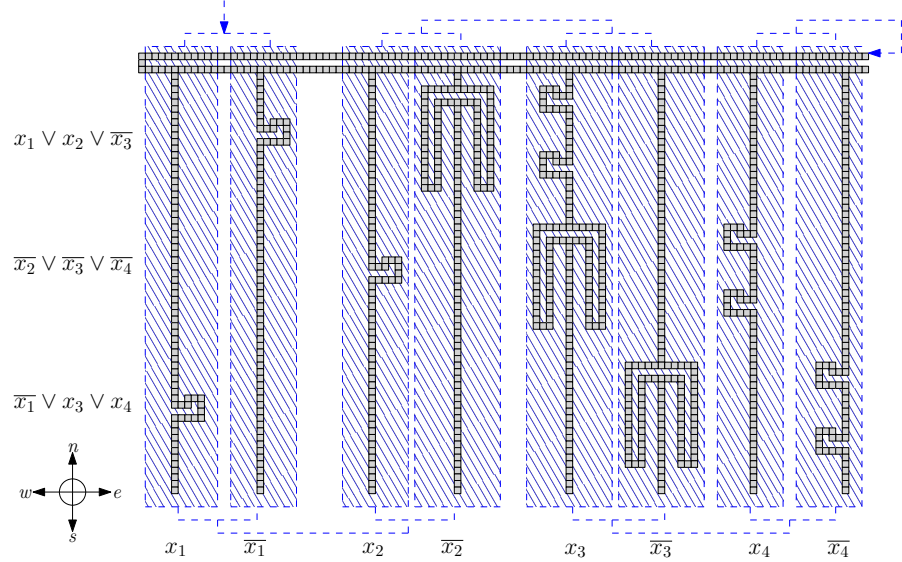


Figure 3.4: Top-view on the polycube. We start building the top layer (crosshatched area as in Figure 3.3(a)) and have to block either the *true* or the *false* part of each variable from above. The blocked parts have to be built with insertions only from east, west, and south. For each clause, we use the gadget from Figure 3.3(b). All other parts can simply be inserted from above in the end.

a constructible path between two particles is NP-hard. Again, a proof is based on a reduction from 3SAT.

Open Problems

Using our relaxed model for Tilt Assembly, we have shown that we can decide constructibility in polynomial time for simple polyominoes. However, when using the original model, i.e., the seed assembly moves within a maze together with all other particles, the class of constructible shapes increases. Figure 3.5 shows a polyomino that is not constructible in our relaxed model but constructible in the original model using a sophisticated maze construction. This also opens the question whether scaling is useful. By scaling up a given polyomino P , the space between two particles of P increases and we may be able to transport a new particle within the shape using a small maze gadget.

Open Problem 1. *In the Tilt Assembly model, classify constructible shapes. Furthermore, is it true that any shape can be constructed if scaling is allowed?*

When considering non-simple polyominoes, multiple questions remain open. The most interesting question is the constructibility of these shapes.

Open Problem 2. *What is the complexity of deciding whether a non-simple polyomino is constructible? Is there an approximation algorithm for the optimization problem?*

In our model, we considered full tilts only, i.e., particles move as long as they are not blocked by an obstacle or by another particle. What happens if we consider single step tilts? One conjecture is that any shape can be built if we are allowed to scale the shape by factor three. The idea is to construct a scaffold and keep tunnels free to still reach positions within the shape.

Open Problem 3. *How do the complexities of problems change when we consider tilt assembly with single step tilts?*

Another problem is that we only can guarantee a approximation factor of $O(\sqrt{N})$ when searching for a polyomino $P' \subseteq P$. However, when adding/removing only a few pixels to/from P , the resulting polyomino is easily constructible. For an example see Figure 3.6. When only adding pixels is allowed, this problem gets closely related to 3D printing where auxiliary material is used to print areas that are not reachable otherwise, or to stabilize weak spots.

Open Problem 4. *Let P be a polyomino. What is the complexity to find a constructible polyomino P' that best approximates P ? What is the complexity in three dimensions?*

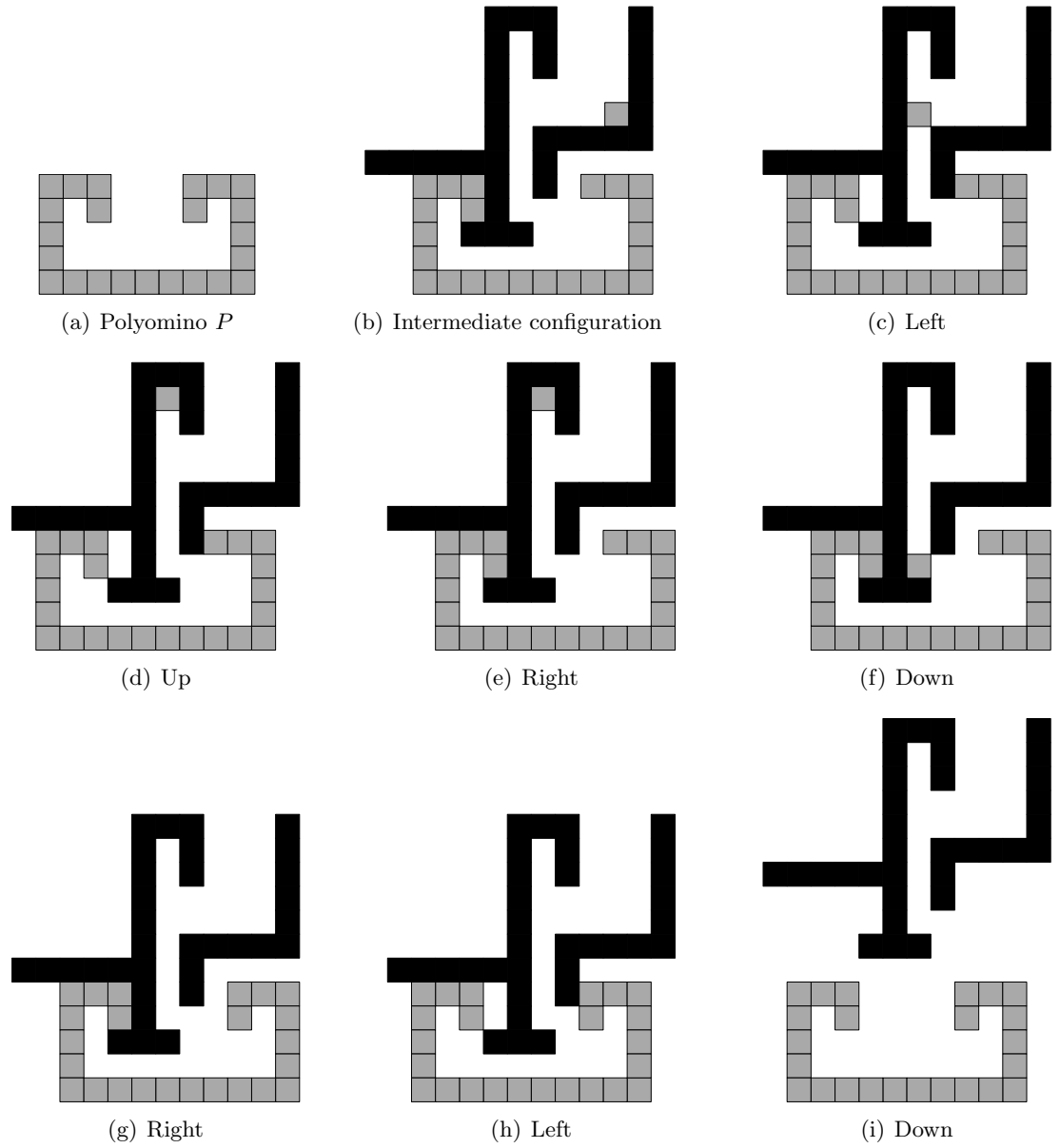


Figure 3.5: (a): A polyomino P that is not constructible in our relaxed model. (b): An intermediate configuration. (c)-(g): Steps for adding a particle to the shape at a position that is unreachable in our relaxed model. (h)-(i) Releasing P .

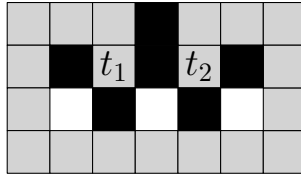


Figure 3.6: A polyomino P (gray) we cannot construct. When adding any of the black particles to P or removing t_1 or t_2 from P , the resulting shape becomes constructible.

3.1.2 Parallel Tilt Assembly

For problem (A.2), where subpolyominoes can be combined with each other, challenges arise because all particles must stay in the same labyrinth all the time: How can we synchronize the global movements such that all subpolyominoes are built and fused in a correct way, and how many subpolyominoes can we combine at the same time?

In our second paper of this thesis, “*Efficient Parallel Self-Assembly Under Uniform Control Inputs*” [146], we show that orthogonal convex polyominoes can be constructed using $O(1)$ steps (see Figure 3.7). A polyomino P is called *orthogonal convex* if the intersection of any vertical or horizontal line with the polyomino P is a connected set. Furthermore, we provide construction sequences for a large class of polyominoes, where, apart from orthogonal convex subpolyominoes, subpolyominoes can only be joined pairwise at each step. The class of polyominoes includes simple polyominoes as well as non-simple polyominoes, whose holes are convex.

Similar to the previous section, we consider a deconstruction sequence instead of constructing the shape bottom-up. We start with the definition of cuts. Consider some polyomino P and some line ℓ cutting P between particles. We say ℓ is a *valid p -cut* if ℓ cuts P into p subpolyominoes that can be pulled apart into opposite directions without blocking each other. A valid p -cut is *straight* if any intersection of the cut with P has no turns. A valid p -cut in a polyomino P *preserves deconstructibility* if P and all induced subpolyominoes P_1, \dots, P_p are deconstructible by cuts. Using only valid 2-cuts for polyominoes, we can show that these cuts preserve deconstructibility. Another positive aspect of valid 2-cuts is that only 2-cuts starting at locally reflex particles have to be considered. A particle t is called *locally reflex* if and only if there is a 2×2 square containing t , two neighbors of t , and an empty pixel.

Theorem 3. *For simple polyominoes with N pixels, a valid straight 2-cut can be found in time $O(N + r^2 \log r)$, where r is the number of locally reflex particles. For polyominoes with convex holes, the time increases to $O(N + r^3 \log r)$.*

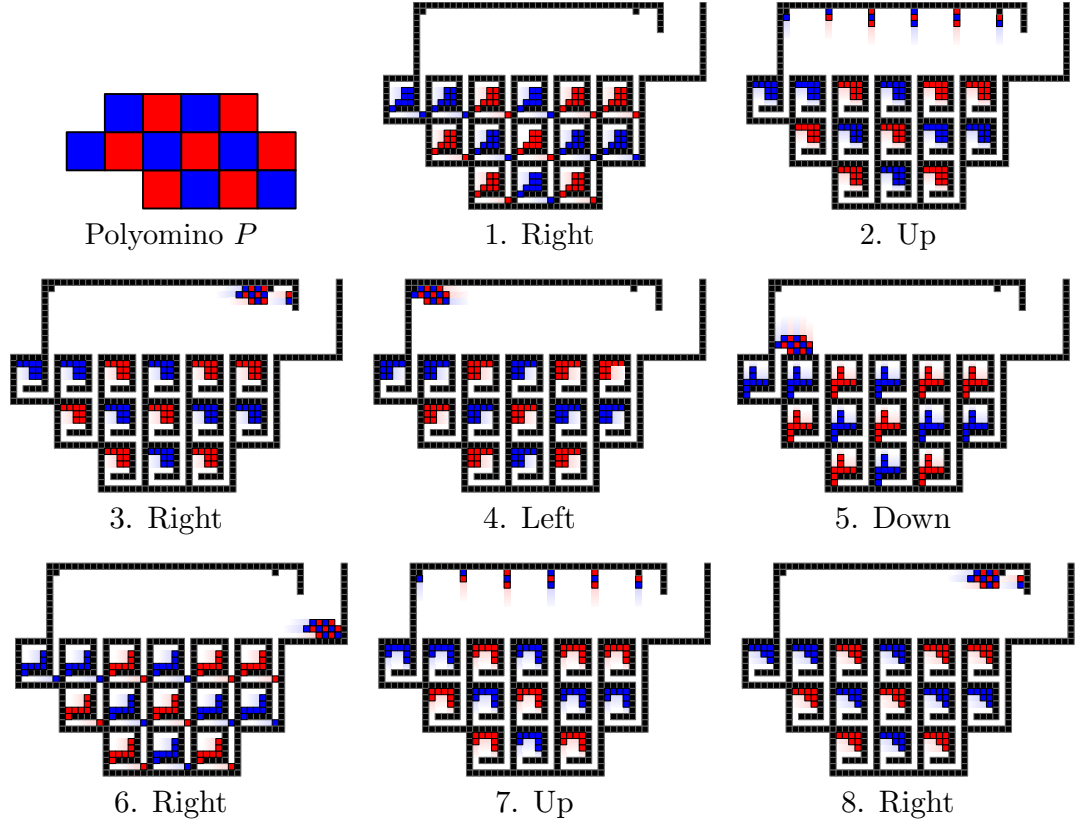


Figure 3.7: Convex polyominoes can be assembled in six movement steps. A copy of the polyomino P is released every five steps after the first copy. A video showing an animation can be found at https://youtu.be/_R_pu00smPs.

Recursively using valid 2-cuts on non-convex polyominoes yields a deconstruction tree, whose leaves are orthogonal convex polyominoes. By using cuts through locally reflex corners, we can bound the depth of this tree by $O(r)$ for simple polyominoes ($O(r^2)$ for polyominoes with convex holes). From this decomposition tree we can easily construct a maze and a move sequence that build the polyomino. We reverse the decomposition tree and obtain a construction tree. Using the construction shown in Figure 3.7, we can construct the convex polyominoes in the leaves in parallel with a constant number of commands, i.e., $\langle r, u, r, l, d \rangle$. Note that repeating this sequence produces a convex shape in each gadget for each repetition.

Now, consider two nodes in the construction tree that have a common parent. We can use the gadgets shown in Figure 3.8 and 3.9 to easily combine two subpolyominoes induced by a vertical 2-cut. When turning these gadgets by 90 degrees we obtain gadgets that combine subpolyominoes induced by horizontal 2-cuts. If we use a rotated gadget, then we extend our tilt sequence to $\langle r, u, d, u, r, l, d \rangle$. Note that this sequence still correctly assembles all convex polyominoes and subpolyominoes (comparing with Figure 3.7, we would insert a d and u tilt between steps 2 and 3, and between 7 and 8). All these gadgets can be used in parallel and can also be used with the same sequence of commands (see Figure 3.10 for an example). Whenever we repeat the sequence $\langle r, u, d, u, r, l, d \rangle$, we move one level up in the construction tree. Thus, $O(r)$ ($O(r^2)$, resp.) commands suffice to produce the polyomino. This is sublinear in the number N of particles of many polyominoes, namely for polyominoes with $N \in \omega(r)$ ($N \in \omega(r^2)$, resp.). However, for simple polyominoes there are examples for which we cannot perform better than $O(N)$ if we only use 2-cuts for non-convex polyominoes (see Figure 3.11).

A further positive result is that monotone shapes can be constructed in time logarithmic in the number of locally reflex particles. A polyomino P is *x-monotone* if and only if any intersection of a vertical line with P results in a connected set. It comes immediately clear that cutting with any vertical line yields a valid 2-cut. By recursively choosing the cut such that the number of locally reflex particles is halved, we obtain a decomposition tree with height $O(\log r)$, whose leaves are again orthogonal convex polyominoes.

Open Problems

Instead of only relying on straight 2-cuts, we can also use valid non-straight 2-cuts to decompose polyominoes. However, it is currently not known whether using non-straight 2-cuts increases the class of constructible shapes using 2-cuts.

Open Problem 5. Let \mathcal{P}_2^- (\mathcal{P}_2^\sim , resp.) be the class of polyominoes constructible by valid straight 2-cuts (valid non-straight 2-cuts, resp.). Is it true that

$$\mathcal{P}_2^- \neq \mathcal{P}_2^\sim?$$

Another problem occurs when considering p -cuts for $p > 2$ for non-convex polyominoes. As can be seen in Figure 3.11 right, there are polyominoes that cannot be constructed by only using 2-cuts. However, combining more than two subpolyominoes seems harder and requires sophisticated maze gadgets.

Open Problem 6. How can we make use of p -cuts to efficiently assembly polyominoes for any $p \geq 2$? Are there shapes we cannot construct in this model?

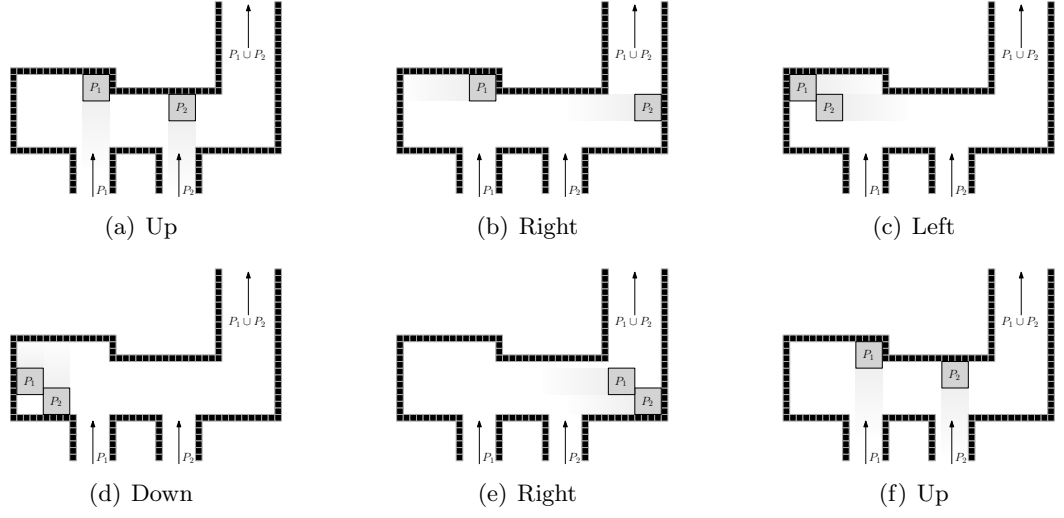


Figure 3.8: Assembling two subpolyominoes P_1 and P_2 , where the topmost particle of P_1 lies above the topmost particle of P_2 . This is the same movement sequence as for constructing convex polyominoes.

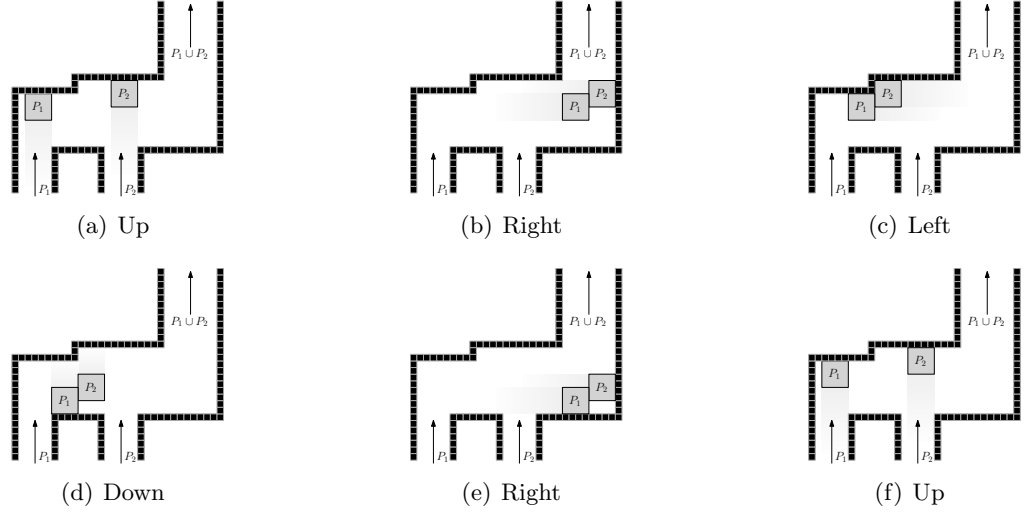


Figure 3.9: Assembling two subpolyominoes P_1 and P_2 , where the topmost particle of P_2 lies above the topmost particle of P_1 . This is the same movement sequence as for constructing convex polyominoes.

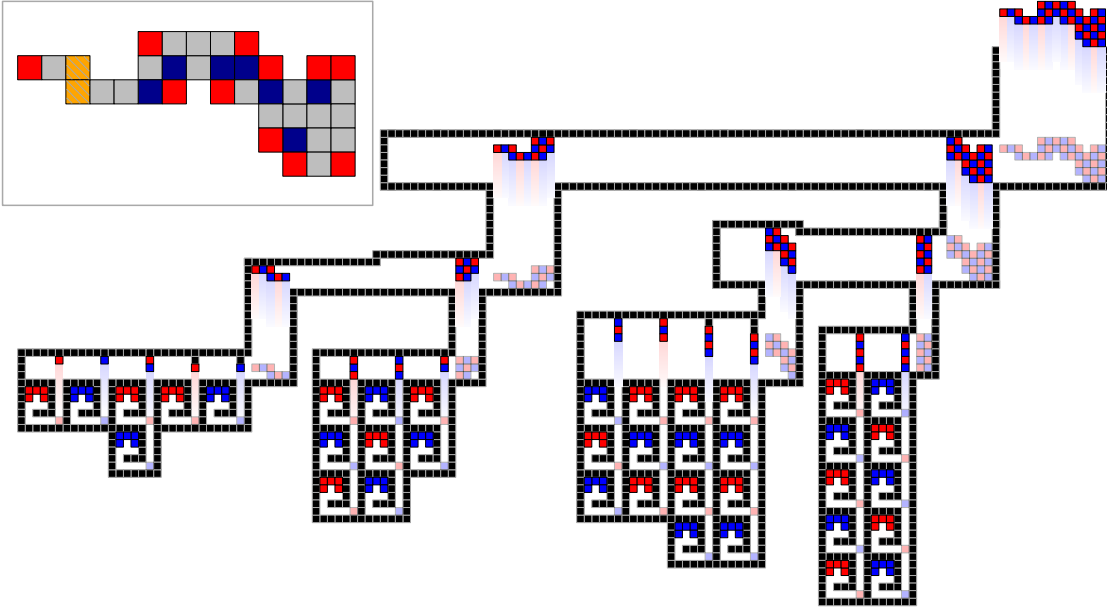


Figure 3.10: A complete example constructing the polyomino in the top-left box. State shown is after an up-movement and its previous state in translucent colors. Top-left box: A polyomino P with locally convex particles (red), locally reflex particles (blue), and particles that are both locally convex and locally reflex (orange, striped).

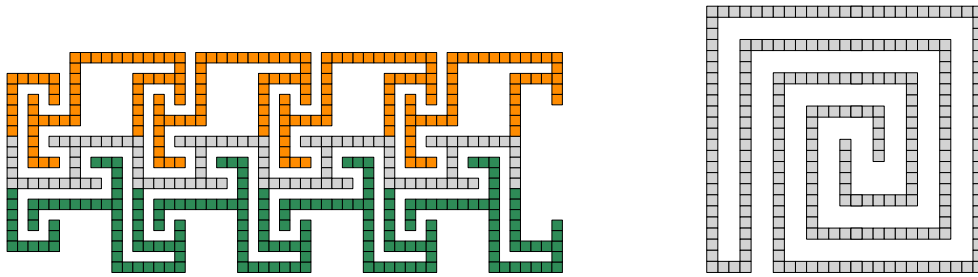


Figure 3.11: Left: A polyomino needing $\Omega(N)$ steps to build because we cannot separate the green nor the orange part efficiently from the grey part. Right: Polyomino which is not 2-cuttable. Any cut splits the polyomino either in two subpolyominoes which cannot be pulled apart or into more than two subpolyominoes.

3.2 Reconfiguration With Global Control and Static Friction

This section considers problem (B.1) and highlights results from our paper “*Coordinated Particle Relocation using Finite Static Friction With Boundary Walls*” [145]. See also [16] for an animation of our results. Given a start configuration C and a target configuration C' , we are interested in a sequence of moves that transforms C into C' .

Previous work [22] use obstacles placed at specific locations as seen in the previous section. However, this can be difficult and intricate in practice. Therefore, we are interested in the following question: How can we reconfigure particles when no obstacles are present and when the workspace is convex? We consider a model of global control in junction with finite static friction along the boundary. For infinite friction, Shahrokhi et al. [152, 153] show that any configuration can be transformed into any other by a minimum number of steps.

In the following, we denote by θ the *angle of friction* and by $\mu := \tan \theta$ the *coefficient of friction*. For a particle r at time t , let $N(\text{boundary}_{x_r(t)})$ be the normal to the boundary at position $x_r(t)$. If it is clear from the context we also refer to r as the position of the particle. Let $u(t)$ denote a move command at time t . For a particle r lying on the boundary, let $\alpha = \arccos(u(t) \cdot N(\text{boundary}_{x_r(t)}))$. The velocity $\dot{x}_r(t)$ of particle r at time t is defined as follows:

$$\dot{x}_r(t) = \begin{cases} 0, & \text{if } x_r(t) \in \text{boundary} \text{ and } \alpha \leq \theta, \\ c_k \|u(t)\| \cdot \sin \alpha, & \text{if } x_r(t) \in \text{boundary} \text{ and } \frac{\pi}{2} \geq \alpha > \theta, \\ \|u(t)\|, & \text{otherwise} \end{cases}$$

Here, $c_k < 1$ is some coefficient depending on the kinetic friction. See Figure 3.12 for an illustration. In our paper, we only consider the first and the third case, i.e., each particle moves at full speed or does not move at all.

It is straightforward to see that this model is less powerful than with infinite friction. This is shown by the fact that in a square with three particles there are extreme configurations we cannot reach: Let r_1, r_2 and r_3 be three particles in a unit square and let $\tilde{r}_1 = (0, 0)$, $\tilde{r}_2 = (0, 1)$ and $\tilde{r}_3 = (1, \frac{1}{2})$ be their target positions. By a simple case analysis, we can show that none of the three particles can be the last particle that reaches its target position. Therefore, we firstly concentrate on two particles. To reconfigure two particles r_1 and r_2 , we proceed in two steps: As a first step, reconfigure the two particles such that their relative position matches the relative position of their goals, i.e., $r_1 - r_2 = \tilde{r}_1 - \tilde{r}_2$. Then, translate both particles to their goal positions.

A tool used for the analysis is the Δ -configuration Δ_P denoting the set of all possible relative positions of two particles in a polygon P . More formally, $\Delta_P := \{p_1 - p_2 \mid$

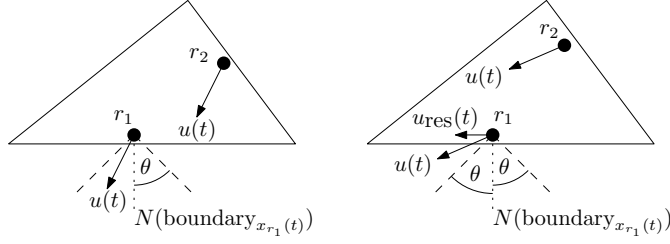


Figure 3.12: Left: An input force command $u(t)$ within the cone $\pm\theta$ about the normal to the boundary results in no motion of r_1 . Right: An input force command $u(t)$ outside the cone results in a motion of both particles. Observe that r_1 slides along the boundary with a resulting force $u_{\text{res}}(t)$.

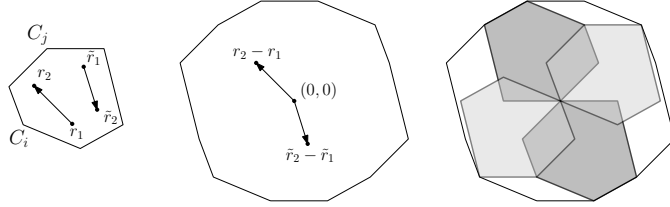


Figure 3.13: Left: A six-sided polygon P with start positions r_1 and r_2 for two particles and their goal positions \tilde{r}_1 and \tilde{r}_2 . Middle: The Δ configuration of the polygon and the positions of the start and end configuration. Right: Lightgray (darkgray) area corresponds to the C_i -area (C_j -area, resp.).

$p_1, p_2 \in P\}$. By defining strategies how to move the particles, we cover parts of Δ_P . If the covered area in Δ_P over all strategies is exactly Δ_P , then we can reconfigure two particles to an arbitrary configuration. In particular, the strategies we developed for a polygon with n corners C_0, \dots, C_{n-1} cover parts of the C_i -area for some $0 \leq i \leq n-1$. The C_i -area is the union of P and $-P$ (i.e., P rotated by $\frac{\pi}{2}$) in Δ_P both having C_i centered at the origin. For examples see Figure 3.13.

In the following, we describe a strategy to cover a C_i -area. One approach is to fix one particle r_1 in C_i and use zig-zagging movements to translate the second particle r_2 . Provided that θ is sufficiently large, we can place r_2 anywhere in P without moving r_1 . The second approach is to place r_2 in a corner C_j and r_1 in C_i . From this configuration, we try to move r_2 anywhere in P while keeping r_1 in C_i . This, however, does not work for any corner C_j because some movements for the particle in C_j will always move the particle in C_i no matter how large θ is. On the positive side, for any corner C_i there is at least one corner C_j from which we can move r_2 anywhere in P while keeping r_1

in C_i provided that there is sufficient friction. For a corner C_i , the set of all such C_j is denoted by P_i . With this strategy, we can provide a sufficient large angle of friction with which we can cover the whole Δ -configuration, i.e., we can reach any configuration. The following theorem shows that θ only depends on the geometry of the polygon.

Theorem 4. *Let P be a convex polygon with vertices C_0, \dots, C_{n-1} , angles $\gamma_0, \dots, \gamma_{n-1}$, and exterior angles $\delta_0, \dots, \delta_{n-1}$. Also define $P_{i,j}^+ := \{C_i, C_{i+1}, \dots, C_{j-1}, C_j\}$ and $P_{i,j}^- := \{C_i, C_{i-1}, \dots, C_{j+1}, C_j\}$. If $\theta > \max_{0 \leq i < n} \left(\min_{j \in P_i} \left(\frac{\gamma_i}{2}, \max \left(\frac{\gamma_j}{2}, \eta_{i,j}^+ - \frac{\pi}{2}, \eta_{i,j}^- - \frac{\pi}{2} \right) \right) \right)$, where $\eta_{i,j}^+ := \sum_{C_k \in P_{i+1,j-1}^+} \delta_k$ and $\eta_{i,j}^- := \sum_{C_k \in P_{i-1,j+1}^-} \delta_k$, then every configuration of two particles can be reached.*

Theorem 4 directly implies that for regular polygons a coefficient of friction of $\mu > \cot(\frac{\pi}{n})$ is sufficient for reconfiguration, and for triangles with angles $\alpha \leq \beta \leq \gamma$ an angle of friction $\theta > \frac{\pi}{2} - \beta$ suffices to guarantee any reconfiguration. But, what can we achieve with more than two particles? As already noted, for three particles there are configurations we cannot reach. However, if $\theta > \frac{\pi}{4}$,¹ then we can reach all configurations, where one particle lies within the rectangle spanned by the other two particles. Another positive result is that we can arbitrarily permute a line of particles in a square provided that $\theta > \frac{\pi}{4}$.¹

Open Problems

In our paper, we only provided angles of friction that are sufficient. Therefore, an interesting open problem is to find lower bounds for θ until which we are not able to reach all configurations.

Open Problem 7. *Given a convex polygon P and two particles, what is the minimum required angle of friction to reach all configurations?*

Furthermore, we only considered special cases for many particles. Extending the work to more classes of configurations seems natural.

Open Problem 8. *Classify configurations of n particles that can or cannot be reached.*

Another extension is to consider dynamic friction. Depending on the shape of the polygon, even small friction values seem to suffice to reach more configuration than with static friction only.

Open Problem 9. *How can we use dynamic friction to reach more configurations?*

¹In the paper we state $\theta > \frac{\pi}{2}$, which corresponds to infinite friction. This is typographical error.

3.3 Reconfiguring Passive Particles With Finite Automata

In this section, we provide results for category (C), where the task is to reconfigure a given arrangement of particles into a target configuration through simple robots. We start without enforcing connectivity of intermediate configurations, i.e., (C.1) and then show for (C.2) how multiple robots can be used to maintain connectivity.

In particular, we consider the following model. Let $G = (\mathbb{Z}, E)$ be the (infinite) grid with edges between all pairs of nodes that are within unit distance. Each node of G is called a *pixel*, and each pixel may be *occupied* by a particle, or is *empty*. Two pixels p_1 and p_2 build a *diagonal pair* if $|x_1 - x_2| = |y_1 - y_2| = 1$. Consider the maximal connected set of particles P . If for each diagonal pair $(p_1, p_2) \in P \times P$ there is a particle p that is adjacent to p_1 and p_2 , then we call P a *polyomino*. If there is no such particle p , then (p_1, p_2) is called a *forbidden pair*. By w (h) we denote the width (height) of P . The boundary ∂P of a polyomino P is the set of all particles that are adjacent to an empty pixel or build a diagonal pair with an empty pixel.

Robots move on pixels and act as deterministic finite automata. More precisely, robots perform look-compute-move cycles, i.e., the robot reads the states of the underlying and adjacent pixels, computes the next step, and performs one of three actions. (1) The robot moves to an adjacent pixel, (2) places a particle, i.e., changes the state of the pixel to occupied, or (3) removes a particle, i.e., changes the state of the pixel to empty. To allow communication between multiple robots, the look-phase of each robot can be extended to scan the states of robots in the neighborhood, i.e., the set of all pixels within unit distance.

3.3.1 Without Connectivity Constraints

In our paper “*CADbots: Algorithmic Aspects of Manipulating Programmable Matter with Finite Automata*” [75], we show that one single robot can perform operations such as counting the number of particles or corners, or bounding, scaling, rotating, reflecting and copying a given shape. In particular, the robot is able to perform any operation that can be described by a Turing machine. However, we do not guarantee that intermediate configurations are connected.

To bound a given polyomino P , the robot seeks for a suitable starting position (e.g., local minimum in the vertical direction) and starts constructing a rectangular bounding box in a zig-zag fashion (see Figure 3.14). With this zig-zag shape the robot can always distinguish between particles of P and the bounding box because P does not contain a forbidden pair.

The construction proceeds in three phases. (I) Finding a leftmost local y -minimal particle, (II) the bounding box construction, (III) a clean-up phase. In Phase (II), we begin two positions beneath the starting position to wrap a zig-zag line around P , i.e., whenever we encounter a convex corner of P we construct a corner of the bounding box; otherwise we try to build a zig-zag line straight ahead. At some point, we cannot carry on building this line because there is a particle of P blocking further construction. In this case, we shift the current line as shown in Figure 3.15 left. If this is not possible, we distinguish two cases. (1) If the line we shift is the first line of the bounding box, then there is a particle of P to the left of our starting position. We remove the line and move left until this particle of P is reached. From there, we restart with Phase (I). (2) Otherwise, we deconstruct the current line until shifting is possible again or we proceed constructing the previous line (see Figure 3.15 right).

When the robot finds another particle of the bounding box, Phase (III), the clean-up phase is started. The robot removes tiles beginning with the starting position, and pushes the last line until we obtain a rectangular box (see Figure 3.16).

After constructing the bounding box all other operation are straightforward. To count the number of particles of the polyomino P , we scan the bounding box column-wise. Every time we encounter a particle of P , we move it aside to mark this pixel as counted, and increase a counter we place outside the bounding box. Note that this procedure is not only restricted to counting particles. We can count any feature as long as we can identify this feature with a local lookup, e.g., when counting corners of the polyomino.

Being able to count brings us to the next operation: simulating Turing machines. See Figure 3.17 for an overview of the construction. The idea is to code the height and width of the bounding box in binary on a one dimensional tape ('1' if a pixel is occupied, '0' if a pixel is empty) and then write the polyomino onto the same tape. By using particles

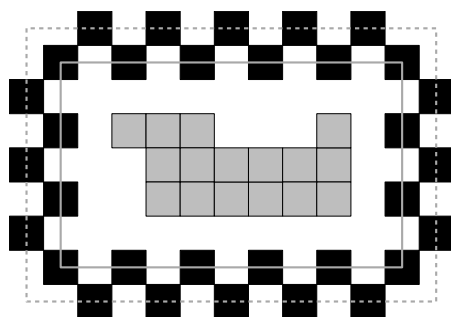


Figure 3.14: A polyomino P (gray) surrounded by a bounding box of particles (black) on the inner lane (solid line) and particles on the outer lane (dashed line).



Figure 3.15: Left: Further construction is not possible. Therefore, we shift the line upwards (see light gray particles). Right: A further shift produces a conflict with the polyomino. Thus, we remove particles (diagonal stripes) and proceed with the shift (light gray) when there is no more conflict.

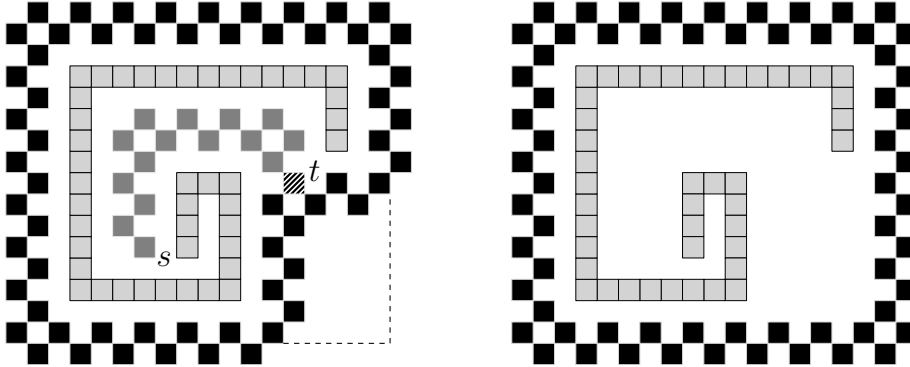


Figure 3.16: Left: During Phase (II), starting in s , we eventually reach a particle t (diagonal stripes) at which the bounding box is split into three directions. In the clean-up phase (Phase (III)), the part between s and t (dark gray particles) gets removed, followed by straightening the bounding box along the dashed line. Right: The final bounding box.

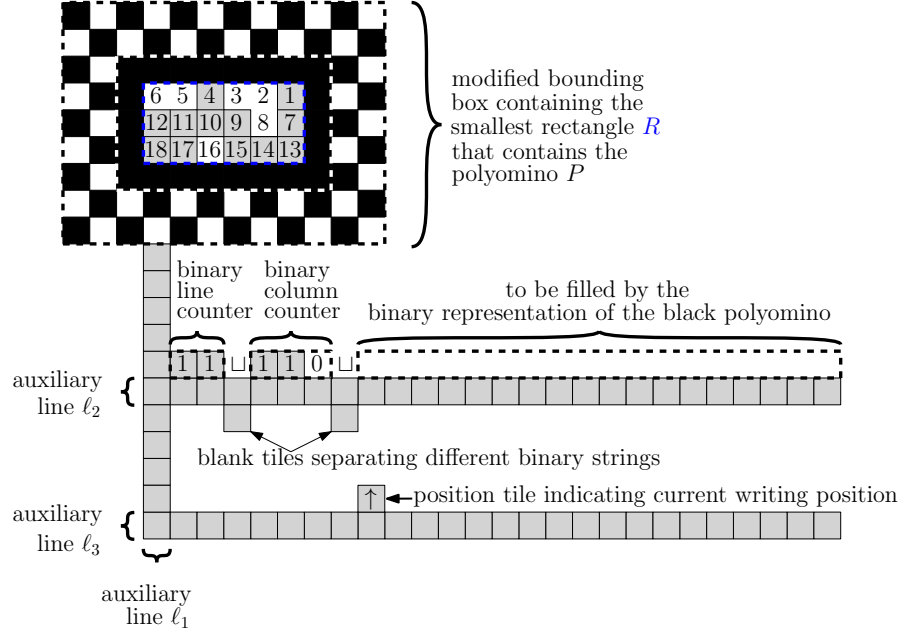


Figure 3.17: Overview of the Turing machine simulation. After writing the number of lines and columns of the polyomino onto the tape (ℓ_2), we can write down P . Particles below the tape denote separators \sqcup . Using another line ℓ_3 , we can mark the current writing position.

at special places, we can denote a separator (' \sqcup ') to split the strings for width, height, and the polyomino. Further particles mark the left and right border of the tape. After applying the operation of the Turing machine, we can transform the resulting string back to obtain a two dimensional shape again.

Theorem 5. *Let P_1 and P_2 be two polyominoes with $|P_1| = |P_2| = N$. There is a strategy transforming P_1 into P_2 if there is a Turing machine transforming $S(P_1)$ into $S(P_2)$. The robot needs $\mathcal{O}(\partial P_1 + \partial P_2 + S_{TM})$ auxiliary particles, $\mathcal{O}(N^4 + T_{TM})$ steps, and $\Theta(N^2 + S_{TM})$ of additional space, where T_{TM} and S_{TM} are the number of steps and additional space needed by the Turing machine.*

Although we can perform all further operation with the help of this reduction to a Turing machine, a lot of extra space and particles may be needed. In the following, we describe how other operations can be performed in a more efficient way.

Copying

To copy a polyomino P , we again scan column-wise through the bounding box and copy particles one at a time. To guarantee that all particles in the copy have the identical distances to all other particles like in the original polyomino, we use *bridge particles* as a special marker (see Figure 3.18). While we scan a column, we add a bridge particle to the copy for each empty pixel. When a column has been copied, we remove all bridge particles and carry on with the next column.

Theorem 6. *Copying a polyomino P column-wise can be done within $\mathcal{O}(wh^2)$ unit steps using $\mathcal{O}(N)$ auxiliary particles and $\mathcal{O}(wh)$ additional space in $\mathcal{O}(h)$ extra rows and columns.*

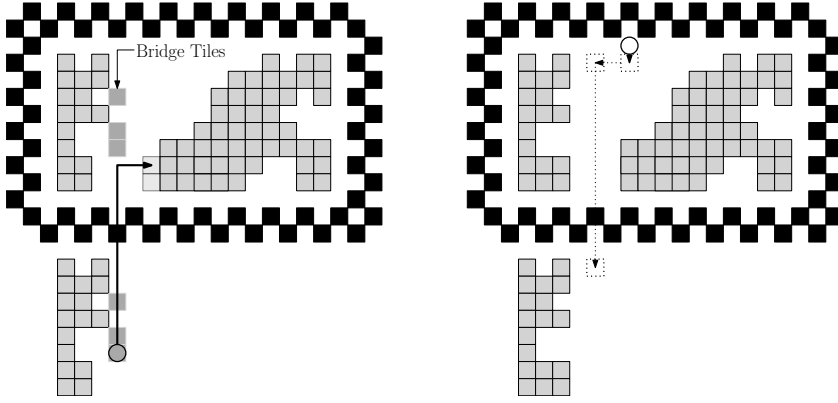


Figure 3.18: Left: Intermediate step while copying the third column of a polyomino (gray particles) with bridges (dark gray particles). The robot (\bigcirc) moves to next pixel along the black line. Right: When the column is copied the bridges get removed and the robot proceeds with the next column. In this case the robot finds an empty pixel, places a bridge particle two steps to the left and a bridge particle below the bounding box.

Reflecting

A reflection is done in a similar way. For a horizontal reflection, we proceed row-wise from bottom to top and transfer each row in the reversed order below the bounding box. Again, by using bridge particles, we also guarantee correct distances between particles. See Figure 3.19 for an illustration of intermediate steps.

Theorem 7. *Reflecting a polyomino P horizontally can be done in $\mathcal{O}(w^2h)$ unit steps, using $\mathcal{O}(w)$ of additional space and $\mathcal{O}(w)$ auxiliary particles. A vertical reflection can be performed analogously.*

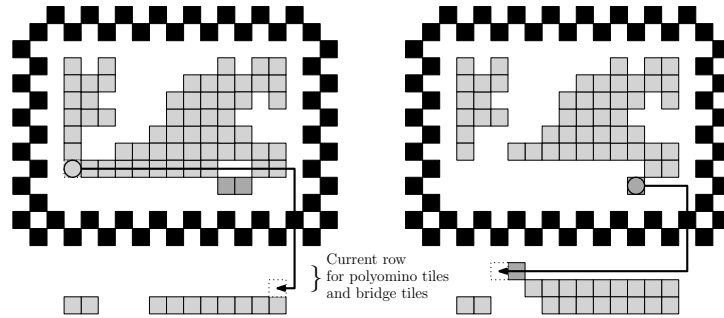


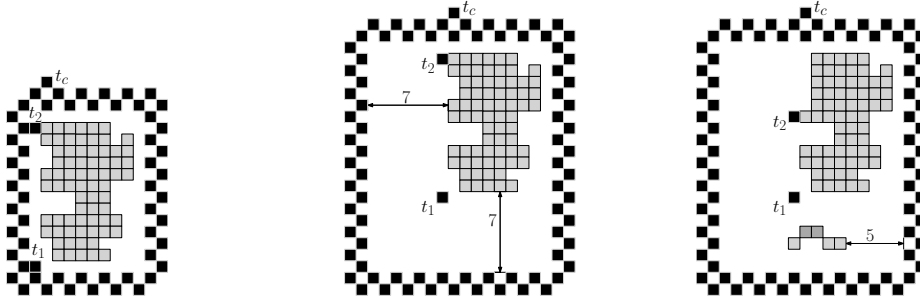
Figure 3.19: Left: Beginning of reflecting the second row. Because the current pixel is occupied, the robot moves the particle to the right below the bounding box (particle with dotted border). Note that the extra space between P and the bounding box is used by bridge particles (dark gray) to denote empty pixels above. Right: Intermediate step of reflecting the second row. The next position is empty, thus, the robot will move the corresponding bridge particle next.

Rotating

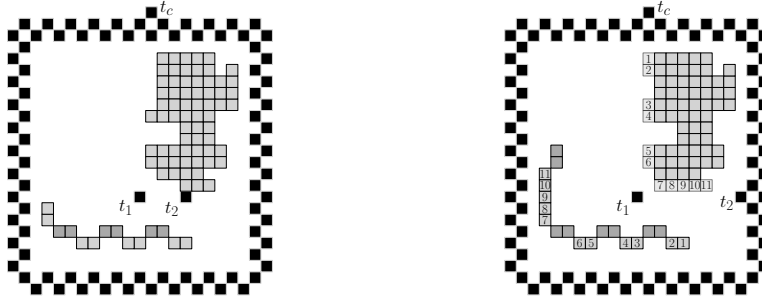
Rotating a shape poses a challenge because the dimensions of a polyomino with width w and height h changes from $w \times h$ to $h \times w$. A straightforward approach by moving the rows bottom-up to the right of the bounding box is therefore inefficient because too much space is needed. By first reflecting the shape across the line $y = x$ and then reflecting horizontally or vertically (depending on a left or right rotation), we obtain a more efficient strategy. We already know how to perform the latter reflection. The former reflection can be achieved by the following strategy.

We place particles to mark the current column and the current row we want to reflect next (see t_c and t_1 in Figure 3.20(a)). Another particle marks the pixel whose state we transfer next (see t_2 in Figure 3.20(a)). Once more, bridge particles are used to denote empty pixels. Figure 3.20 shows intermediate steps how to reflect across the line $y = x$.

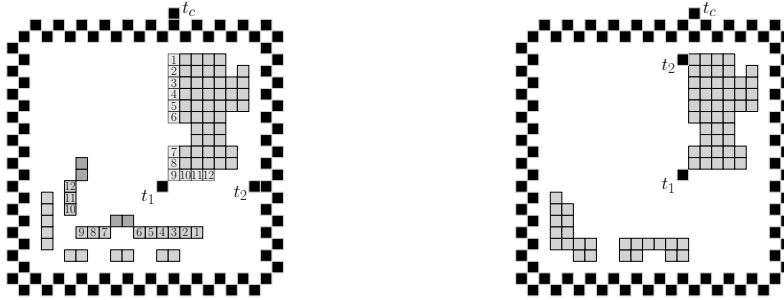
Theorem 8. *There is a strategy to rotate a polyomino P by $\pm \frac{\pi}{2}$ within $\mathcal{O}((w+h)wh)$ unit steps, using $\mathcal{O}(w+h+|w-h|h)$ of additional space in $\mathcal{O}(|w-h|+1)$ extra rows and columns and $\mathcal{O}(w+h)$ auxiliary particles.*



(a) Left: Intermediate construction before expanding the bounding box. Markers t_1 , t_2 and t_c are already placed. Middle: Construction after expanding the bounding box. Right: Intermediate step while reflecting the first column over the $x = y$ line. Note that we start the column with distance five to the bounding box. This is needed to build the row upwards without hitting P if w is close to h .



(b) Left: Intermediate step while building the first reflected row. Note that the left side of the bounding box has been extended to make more space. Right: Intermediate step after reflecting the first column and row (original column and row are marked in light gray). Numbers show the matching between original position and new position of particles.



(c) Left: First row and column have been moved down and left to make space for the next row and column. Right: Merging the first two columns and rows.

Figure 3.20: Intermediate steps of reflecting a polyomino across the line $y = x$.

Scaling

Scaling a shape by a constant factor c is straightforward when the bounding box is already present. First, scale each column, then, scale each row. To scale a column, we proceed from left to right, shift any column to the right of the current column c units further to the right, and fill up the gap with copies of the current column. Scaling a row is done analogously.

Theorem 9. *Given a constant c , the robot can scale the polyomino by c within $\mathcal{O}((w^2 + h^2)c^2N)$ unit steps using $\mathcal{O}(c^2wh)$ of additional space in $\mathcal{O}(c(w + h))$ additional rows and columns, using $\mathcal{O}(c^2N)$ auxiliary particles.*

3.3.2 Connected Reconfiguration

When connection of intermediate arrangements is required, e.g., when working in space where disconnected parts could simply float away, new challenges arise. One particular problem is that one robot cannot even search a maze. However, two robots are sufficient [33]. Also, while constructing a bounding box, the robot must always be able to distinguish between the bounding box and the polyomino while both of them are connected through a particle. Furthermore, the bounding box may not build as a zig-zag line as shown in the previous section because this structure is not connected. All these issues are addressed in our paper “*Recognition and Reconfiguration of Lattice-Based Cellular Structures by Simple Robots*” [131]. In the following, we show how two robots are able to construct a bounding box and to scale the polyomino by factor three. Being able to do this, we can then adapt all algorithms from the previous section guaranteeing connectivity.

Firstly, consider the construction of the bounding box. Similar to the approach of the previous section, we first search for a suitable starting position. From there, the second robot keeps the polyomino P and the bounding box connected, while the first robot constructs the bounding box (see Figure 3.21). The robot now builds a simple line around P . The immediate consequence is that the robot is not able to distinguish between bounding box and P by a local lookup. In the following, we describe how to decide whether a particle t belongs to P or the bounding box. We traverse the boundary of the structure t belongs to until we reach the second robot. If the second robot is met from above, then t belongs to P , otherwise, t must belong to the bounding box. By moving below the second robot and following the bounding box until the end, we can find back to the particle t . Afterwards we proceed with the bounding box construction similar to the previous section.

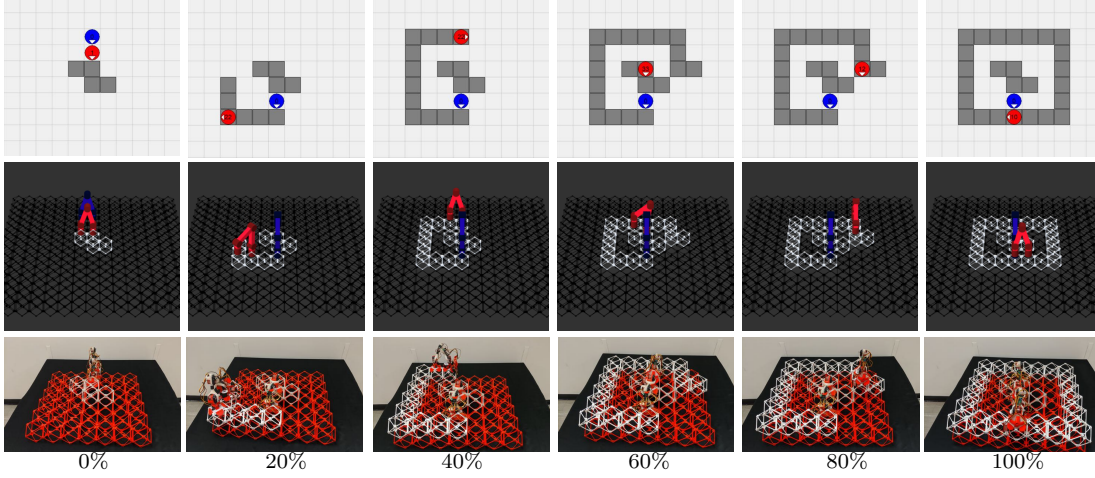


Figure 3.21: Snapshots from building a bounding box for a z-shaped polyomino using a 2D simulator, a 3D simulator, and staged hardware robots, synchronized so all are shown at steps $\{0, 24, 48, 72, 96, 120\}$.

Experiments with a 2D simulator show that most of the time is spent moving, and shifting lines of particles (see Figure 3.22). The time spent for the initial search and deleting particles is only a tiny percentage of the overall time. However, the tested shapes were nicely shaped such that these two parts were not needed often. Shapes tested are various sizes of squares, L-shapes, u-shapes, c-shapes, \sqcap -shapes, and n-shapes.

To scale a polyomino after constructing the bounding box, we fill up the rightmost column within the bounding box (except the topmost pixel), and remove the bottommost particle of the right side of the bounding box and the particle two position above. Using this *column marker* and *row marker*, the robot is always able to find the next particle that needs to be scaled next. Figure 3.23(a) shows the configurations after this *preparation phase*. To scale a pixel, the robot searches for the column marker, moves to and updates the row marker, moves two steps to the left, and stores the state of the pixel. Afterwards it moves to the position where the scaled pixel has to be built. If the pixel was occupied, then the robot constructs a 3×3 square, else, the robot constructs a 3×3 square with the middle particle missing. By updating the markers appropriately, the robot eventually scales the polyomino by factor three (see Figures 3.23(b) and 3.23(c)). After scaling, the robot can simply remove the area containing the old bounding box, and, if necessary, also remove scaled pixels that represent empty pixels.

An immediate consequence of being able to scale is that we can now perform operations from the previous section while maintaining connectivity of intermediate configurations.

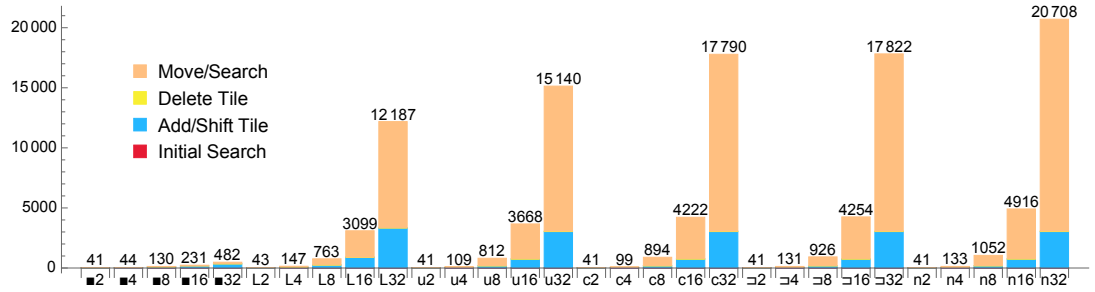


Figure 3.22: Steps required for six canonical shapes. All shapes fill an $n \times n$ bounding box, with $n \in \{2, 4, 8, 16, 32\}$. Shapes include filled squares, and five shapes composed of single width polyomino lines: L-shapes, u-shapes, c-shapes, 2-shapes, and n-shapes. The time requirements increase from left to right.

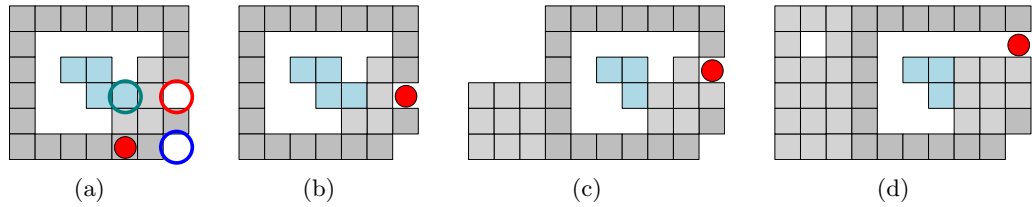


Figure 3.23: (a) Configuration after the preparation phase showing the column marker (blue circle), the row marker (red circle), and the next tile to scale (teal circle). (b)-(d) Cases that appear during the scaling: (b) Scaling an occupied vertex; (c) scaling an empty vertex; (d) reaching the end of a column. See also [2] for an animation.

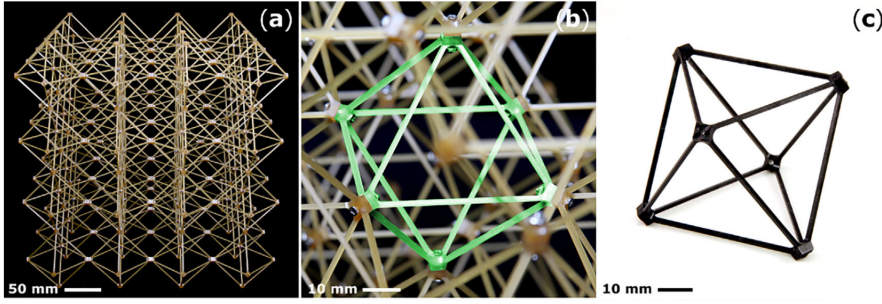


Figure 3.24: (a) An assembled cuboctahedral lattice specimen, made from (b) Ultem 2200 (20% glass fiber reinforced polyetherimide) octahedral unit cells (highlighted), termed voxels. (c) A single monolithic RTP 2187 (40% carbon fiber reinforced polyetherimide) injection molded voxel. (See [95].)

Consider any configuration of particles. By using the 3×3 squares as above, we observe that the scaled version of this configuration is connected. Thus, the robot can effectively move over empty pixels without being connected to the polyomino. With this idea, we obtain the following theorem.

Theorem 10. *If there is an algorithm \mathcal{A} for some problem Π with runtime $\mathcal{T}(\mathcal{A})$, such that the robot moves within a $w' \times h'$ rectangle, then there is an algorithm \mathcal{A}' for Π with runtime $O(wh \cdot (c^2 + cw + ch) + \max((w' - w)h', (h' - h)w') + c \cdot \mathcal{T}(\mathcal{A}))$ guaranteeing connectivity of intermediate constructions during execution.*

Open Problems

An interesting step for future work is to consider three-dimensional shapes. New light-weight material as shown in Figure 3.24 can be used to build structures by robots that can walk on the boundary of the shape or even walk through voxels as shown in Figure 3.25.

Open Problem 10. *What can be achieved in a three-dimensional setting?*

So far, we assumed that the robots can get material at the current position. When robots have to get material from a fixed pixel, i.e., a depot, it is not clear how the robots can find a depot and how they get back to their construction site without scanning the whole polyomino.

Open Problem 11. *How can we adapt our algorithms, when robots must get new material from fixed depots?*

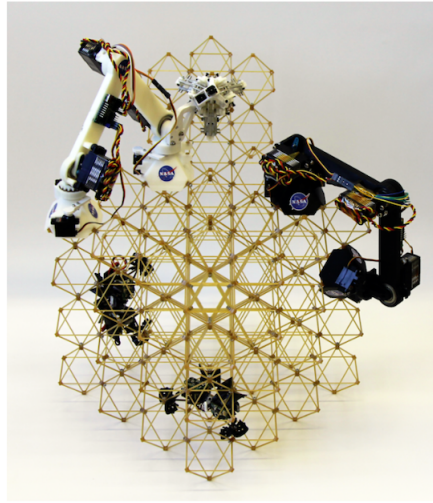


Figure 3.25: Modular reconfigurable 3D lattice structure and mobile robots, showing the small size of the robots relative to the structure that they work on, and the parallel use of multiple robots. (See [50].)

Another direction includes the use of multiple robots. Robots could work in parallel on distinct areas to speed up the construction, or they can help each other by transporting material from depots to the construction site. Then, we could classify robots into constructors and suppliers.

Open Problem 12. *How can we manage multiple robots to increase time efficiency?*

3.4 Parallel Online Bin Packing

In this final section, we highlight results from our paper “*Parallel Online Algorithms for the Bin Packing Problem*” [76] for category (D): Parallel online algorithms. Firstly, we recall the model definition.

Consider k online algorithms A_1, \dots, A_k , each of them processing the same input list I in parallel. The set $\mathcal{A} := \{A_1, \dots, A_k\}$ is called a *k-copy online algorithm*. Let $A(I)$ denote the cost of algorithm A on input list I , and $\text{OPT}(I)$ the cost of an optimal offline algorithm for the same input list. The absolute competitive factor $R_{\mathcal{A}}$ is then defined as

$$R_{\mathcal{A}} := \limsup_I \left\{ \frac{\min_{A \in \mathcal{A}} A(I)}{\text{OPT}(I)} \right\},$$

and the asymptotic competitive ratio $R_{\mathcal{A}}^{\infty}$ as

$$R_{\mathcal{A}}^{\infty} = \lim_{n \rightarrow \infty} \limsup_I \left\{ \frac{\min_{A \in \mathcal{A}} A(I)}{\text{OPT}(I)} \mid \text{OPT}(I) = n \right\}.$$

We apply this model to the bin packing problem, where the task is to pack a given sequence of items of size $[0, 1]$ into as few as possible unit sized bins. Items of size $[0, 1/3]$ are called *small*, items of size $(1/3, 1/2]$ are called *medium*, items of size $(1/2, 2/3]$ are called *large*, and all other items are called *extra large*. We denote by I_S , I_M , I_L , and I_{XL} the set of small, medium, large, and extra large items. For a set of items I' , let $\text{size}(I') := \sum_{a \in I'} a$.

Our goal is to achieve an asymptotic competitive ratio near $3/2$ for a small number of algorithm. To achieve an asymptotic competitive factor of $3/2$, any algorithm can pack each extra large items into one bin. Medium items can be packed pairwise into separate bins. All these bins will have a packing density of at least $2/3$. Large and small items must be handled differently. More precisely, because no two large items fit into one bin and each large item has size less than $2/3$, we have to pack small items together with large items. However, we do not know in advance how many large items will arrive. Therefore, it may be a good idea to fill bins only partially with small items and add large items when they arrive. But, how many bins have to reserved this way? Clearly, if we reserve too many bins and no large item arrives, we obtain a bad competitive ratio. Conversely, if we do not reserve any large bin, and many large items arrive, we obtain a bad competitive ratio again.

In our paper, we present a parameterized algorithm called Predictive Harmonic 3 (PH3) that gets as input a parameter $r_L \in [0, 1]$ representing the quotient of the number of large items to the size of all small items. PH3 packs extra large and medium items as described above. For small items, PH3 tests whether r_L is strictly larger than the quotient of the small items in large bins to the total size of small items packed so far. If this is true, then small items are packed with Next Fit into large bins, i.e., bins that will contain a large item. Otherwise, small items are packed into small bins, i.e., bins only containing small items.

Theorem 11. *Let $r_L^* = \min \left\{ \frac{|I_L|}{6 \cdot \text{size}(I_S)}, 1 \right\}$ and $\delta = r_L - r_L^*$. PH3 achieves the asymptotic competitive ratio*

$$R_{PH3}^{\infty} \leq \begin{cases} \frac{3}{2} + \min \left\{ \frac{1}{4r_L^*}, \frac{3}{6r_L^* + 2} \right\} \cdot (-\delta) & \text{for } \delta \leq 0 \\ \frac{3}{2} + \min \left\{ \frac{3}{4r_L^*}, \frac{9}{6r_L^* + 2} \right\} \cdot \delta & \text{for } \delta \geq 0. \end{cases}$$

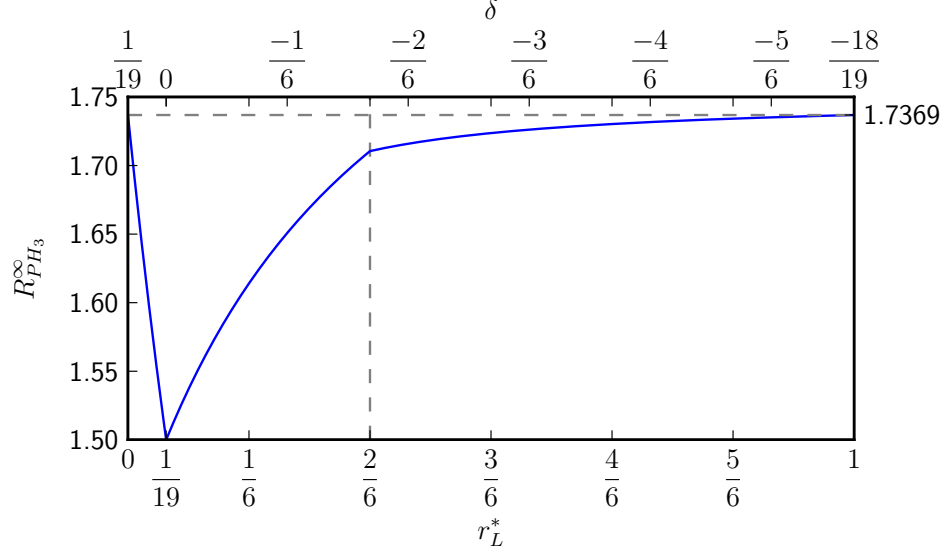


Figure 3.26: Competitive ratio of the optimal 1-copy PH3 algorithm dependent on r_L^* for a fixed $r_L = \frac{1}{19}$.

For fixed r_L we can prove that R_{PH3}^{∞} is monotonically decreasing for $r_L^* \leq r_L$, and monotonically increasing for $r_L^* \geq r_L$. Therefore, for values of $r_L^* \in [a, b]$ we can compute R_{PH3}^{∞} by taking the maximum for $r_L^* = a$ and $r_L^* = b$. By doing a binary search on r_L , we can compute the best possible competitive ratio on the interval $[a, b]$. Figure 3.26 shows a plot with the best possible choice $r_L = \frac{1}{19}$ if only one algorithm is used on the interval $[0, 1]$.

When dealing with more than one algorithm, we can distribute all algorithms such that any algorithm has a specific interval for which it is R -competitive for some fixed $R > 3/2$. Because we only have to consider the left and the right borders of the interval to compute the competitiveness, we start at $r_L^* = 0$ and compute the largest r_L for which PH3 is R -competitive at $r_L^* = 0$. Having r_L , we can compute the largest r_L^* for which we are R -competitive. From there, we repeat the procedure with the next algorithm until the $[0, 1]$ interval is completely covered. For a fixed number k of algorithms, we can do a binary search on R to compute the best values r_L for each algorithm. Figure 3.27 shows the best possible competitive factor for k algorithms. We see that with only 11 algorithms we can surpass the lower bound of classical bin packing, i.e., only one online algorithm is allowed.

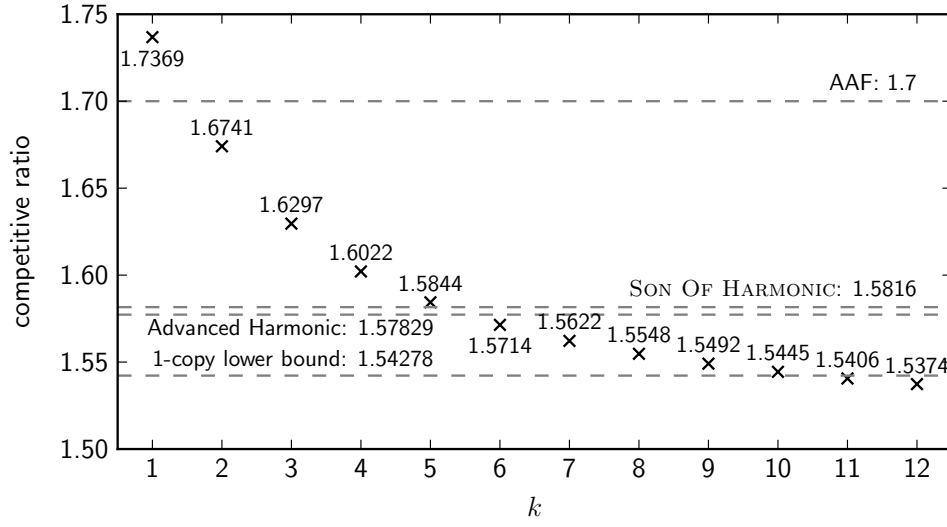


Figure 3.27: Comparison of k -copy PH3 to various upper bounds like Almost Any Fit, Son of Harmonic and Advanced Harmonic, and to the 1-copy lower bound.

An open problem in our conference version [76] asks, “how can we bound the competitive ratio from above in dependence of k ?” The full version submitted for journal publication (attached to this thesis) shows an explicit formula bounding the competitive ratio from above. By bounding how many algorithm are needed to cover both intervals $[0, 1/3]$ and $[1/3, 1]$ with a fixed competitive ratio, we obtain the following theorem.

Theorem 12. *PH3 with k algorithms has an asymptotic competitive ratio of at most*

$$\frac{3}{2} + \frac{3 \cdot 4^{1/k} - 3}{6 \cdot 4^{1/k} + 2}.$$

Open Problems

We only considered a packing density of $2/3$ in our paper and showed that we can get arbitrarily close to a competitive ratio of 1.5. Previous work [14] shows that it is possible to get below this value. However, tremendous (but constant) many algorithms are needed to even get to a competitive ratio of 1.5 and this is therefore only of theoretical interest.

Open Problem 13. *Is there a small constant K such that K algorithms suffice to guarantee an asymptotic competitive factor of ≤ 1.5 ?*

Besides the asymptotic competitive ratio, also the absolute competitive ratio is of interest. The fact that Partition is NP-hard implies a lower bound of 1.5 for the bin packing problem. This holds true, even if the number of used algorithms is polynomial in the number of arriving items.

Open Problem 14. *Can we achieve an absolute competitive ratio of $1.5 + \varepsilon$ with a small number of algorithms for bin packing?*

Related work consider online algorithms with advice, which can be translated to an k -copy algorithm. However, using our model, a more precise analysis is possible. Therefore, it seems plausible to apply the k -copy model to other problems.

Open Problem 15. *Analyze other problems with the k -copy algorithm model.*

Bibliography

- [1] A. Abdel-Rahman, A. T. Becker, D. E. Biediger, K. C. Cheung, S. P. Fekete, N. A. Gershenfeld, S. Hugo, B. Jenett, P. Keldenich, E. Niehs, C. Rieck, A. Schmidt, C. Scheffer, and M. Yannuzzi. Space ants: Constructing and reconfiguring large-scale structures with finite automata. In *Symposium on Computational Geometry (SoCG)*, 2020.
- [2] A. Abdel-Rahman, A. T. Becker, D. E. Biediger, K. C. Cheung, S. P. Fekete, N. A. Gershenfeld, S. Hugo, B. Jenett, P. Keldenich, E. Niehs, C. Rieck, A. Schmidt, C. Scheffer, and M. Yannuzzi. Space ants: Constructing and reconfiguring large-scale structures with finite automata. In *Symposium on Computational Geometry (SoCG)*, pages 73:1–73:7, 2020.
- [3] L. Adleman. Toward a mathematical theory of self-assembly (extended abstract). Technical report, 1999.
- [4] L. Adleman, Q. Cheng, A. Goel, and M.-D. Huang. Running time and program size for self-assembled squares. In *Symposium on Theory of Computing (STOC)*, pages 740–748, 2001.
- [5] G. Aggarwal, Q. Cheng, M. Goldwasser, M. Kao, P. de Espanes, and R. Schweller. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34(6):1493–1515, 2005.
- [6] S. Akella, W. H. Huang, K. M. Lynch, and M. T. Mason. Parts feeding on a conveyor with a one joint robot. *Algorithmica*, 26(3-4):313–344, 2000.
- [7] S. Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1-2):3–26, 2003.
- [8] S. Albers and M. Hellwig. Online makespan minimization with parallel schedules. *Algorithmica*, 78(2):492–520, 2017.
- [9] G. Aloupis, J. Cardinal, S. Collette, F. Hurtado, S. Langerman, and J. O’Rourke. Draining a polygon—or—rolling a ball out of a polygon. *Computational Geometry*, 47(2):316–328, 2014.

- [10] G. Aloupis, S. Collette, M. Damian, E. D. Demaine, D. El-Khechen, R. Flatland, S. Langerman, J. O'Rourke, V. Pinciu, S. Ramaswami, et al. Realistic reconfiguration of crystalline (and telecube) robots. In *Workshop on the Algorithmic Foundation of Robotics (WAFR)*, pages 433–447. 2009.
- [11] G. Aloupis, S. Collette, E. D. Demaine, S. Langerman, V. Sacristán, and S. 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.
- [12] S. Alpern. The rendezvous search problem. *SIAM Journal on Control and Optimization*, 33(3):673–683, 1995.
- [13] E. J. Anderson and R. R. Weber. The rendezvous problem on discrete locations. *Journal of Applied Probability*, 27(4):839–851, 1990.
- [14] S. Angelopoulos, C. Dürr, S. Kamali, M. P. Renault, and A. Rosén. Online bin packing with advice of small size. *Theory of Computing Systems*, 62(8):2006–2034, 2018.
- [15] D. Arbuckle and A. A. Requicha. Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations. *Autonomous Robots*, 28(2):197–211, 2010.
- [16] V. M. Baez, A. T. Becker, S. P. Fekete, and A. Schmidt. Coordinated particle relocation with global signals and local friction (media exposition). In *36th International Symposium on Computational Geometry (SoCG 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- [17] J. Balanza-Martinez, T. Gomez, D. Caballero, A. Luchsinger, A. A. Cantu, R. Reyes, M. Flores, R. Schweller, and T. Wylie. Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces. In *Symposium on Discrete Algorithms (SODA)*, pages 2625–2641, 2020.
- [18] J. Balanza-Martinez, A. Luchsinger, D. Caballero, R. Reyes, A. A. Cantu, R. Schweller, L. A. Garcia, and T. Wylie. Full tilt: Universal constructors for general shapes with uniform external forces. In *Symposium on Discrete Algorithms (SODA)*, pages 2689–2708, 2019.
- [19] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin. A new and improved algorithm for online bin packing. In *European Symposium on Algorithms (ESA 2018)*, 2018.

- [20] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin. A new lower bound for classic online bin packing. In *Workshop on Approximation and Online Algorithms (WAOA)*, pages 18–28, 2020.
- [21] D. Beauquier and M. Nivat. On translating one polyomino to tile the plane. *Discrete & Computational Geometry*, 6(4):575–592, 1991.
- [22] A. T. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin. Reconfiguring massive particle swarms with limited, global control. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS)*, pages 51–66, 2013.
- [23] A. T. Becker, E. D. Demaine, S. P. Fekete, J. Lonsford, and R. Morris-Wright. Particle computation: complexity, algorithms, and logic. *Natural Computing*, 18(1):181–201, 2019.
- [24] A. T. Becker, E. D. Demaine, S. P. Fekete, and J. McLurkin. Particle computation: Designing worlds to control robot swarms with only global signals. In *International Conference on Robotics and Automation (ICRA)*, pages 6751–6756, 2014.
- [25] A. T. Becker, C. Ertel, and J. McLurkin. Crowdsourcing swarm manipulation experiments: A massive online user study with large swarms of simple robots. In *International Conference on Robotics and Automation (ICRA)*, pages 2825–2830, 2014.
- [26] A. T. Becker, S. P. Fekete, L. Huang, P. Keldenich, L. Kleist, D. Krupke, C. Rieck, and A. Schmidt. Targeted drug delivery: Algorithmic methods for collecting a swarm of particles with uniform, external forces. In *International Conference on Robotics and Automation (ICRA)*, 2020. To appear.
- [27] A. T. Becker, S. P. Fekete, P. Keldenich, M. Konitzny, L. Lin, and C. Scheffer. Coordinated motion planning: The video (multimedia exposition). In *International Symposium on Computational Geometry (SoCG)*, 2018.
- [28] A. T. Becker, S. P. Fekete, P. Keldenich, D. Krupke, C. Rieck, C. Scheffer, and A. Schmidt. Tilt assembly: Algorithms for micro-factories that build objects with uniform external forces. *Algorithmica*, 82:165–187, 2020.
- [29] A. T. Becker, O. Felfoul, and P. E. Dupont. Simultaneously powering and controlling many actuators with a clinical MRI scanner. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2017–2023, 2014.

- [30] A. T. Becker, O. Felfoul, and P. E. Dupont. Toward tissue penetration by MRI-powered millirobots using a self-assembled Gauss gun. In *International Conference on Robotics and Automation (ICRA)*, pages 1184–1189, 2015.
- [31] A. T. Becker, G. Habibi, J. Werfel, M. Rubenstein, and J. McLurkin. Massive uniform manipulation: Controlling large populations of simple robots with a common input signal. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 520–527, 2013.
- [32] M. A. Bender and D. K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Symposium on Foundations of Computer Science (sfcs)*, pages 75–85, 1994.
- [33] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Symposium on Foundations of Computer Science*, pages 132–142, 1978.
- [34] A. Bonato and R. J. Nowakowski. *The Game of Cops and Robbers on Graphs*. AMS, 2011.
- [35] J. Bourgeois and S. C. Goldstein. Distributed intelligent mems: Progresses and perspectives. In *International Conference on ICT Innovations*, pages 15–25, 2011.
- [36] J. Bourgeois, B. Piranda, A. Naz, N. Boillot, H. Mabed, D. Dhoutaut, T. Tucci, and H. Lakhlef. Programmable matter as a cyber-physical conjugation. In *International Conference on Systems, Man, and Cybernetics (SMC)*, pages 002942–002947, 2016.
- [37] J. Boyar, S. Kamali, K. S. Larsen, and A. López-Ortiz. On the list update problem with advice. In *International Conference on Language and Automata Theory and Applications (LATA)*, pages 210–221. 2014.
- [38] J. Boyar, S. Kamali, K. S. Larsen, and A. López-Ortiz. Online bin packing with advice. *Algorithmica*, 74(1):507–527, 2016.
- [39] P. Brass, F. Cabrera-Mora, A. Gasparri, and J. Xiao. Multirobot tree and graph exploration. *Transactions on Robotics*, 27(4):707–717, 2011.
- [40] D. M. Brown. A lower bound for on-line one-dimensional bin packing algorithms. Technical report, 1979.
- [41] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *The International Journal of Robotics Research*, 23(9):919–937, 2004.

- [42] S. Cannon, E. D. Demaine, M. L. Demaine, S. Eisenstat, M. J. Patitz, R. Schweller, S. M. Summers, and A. Winslow. Two hands are better than one (up to constant factors). In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 172–184, 2013.
- [43] C. Chalk, E. Martinez, R. Schweller, L. Vega, A. Winslow, and T. Wylie. Optimal staged self-assembly of general shapes. *Algorithmica*, 80(4):1383–1409, 2018.
- [44] C. Chalk, E. Martinez, R. Schweller, L. Vega, A. Winslow, and T. Wylie. Optimal staged self-assembly of linear assemblies. *Natural Computing*, 18(3):527–548, 2019.
- [45] H.-L. Chen and D. Doty. Parallelism and time in hierarchical self-assembly. *SIAM Journal on Computing*, 46(2):661–709, 2017.
- [46] H.-L. Chen, D. Doty, D. Holden, C. Thachuk, D. Woods, and C.-T. Yang. Fast algorithmic self-assembly of simple shapes using random agitation. In *International Workshop on DNA-Based Computers*, pages 20–36, 2014.
- [47] M. Chen, D. Xin, and D. Woods. Parallel computation using active self-assembly. *Natural Computing*, 14(2):225–250, 2015.
- [48] K. C. Cheung and N. Gershenfeld. Reversibly assembled cellular composite materials. *science*, 341(6151):1219–1221, 2013.
- [49] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving the robots gathering problem. In *Automata, Languages and Programming*, pages 1181–1196, 2003.
- [50] A. Costa, A. Abdel-Rahman, B. Jenett, N. Gershenfeld, I. Kostitsyna, and K. Cheung. Algorithmic approaches to reconfigurable assembly systems. In *Aerospace Conference*, pages 1–8, 2019.
- [51] N. B. Cramer, D. W. Cellucci, O. B. Formoso, C. E. Gregg, B. E. Jenett, J. H. Kim, M. Lendraitis, S. S. Swei, G. T. Trinh, K. V. Trinh, et al. Elastic shape morphing of ultralight structures by programmable assembly. *Smart Materials and Structures*, 28(5):055006, 2019.
- [52] J. Csirik and G. J. Woeginger. On-line packing and covering problems. In *Online Algorithms: The State of the Art*, pages 147–177. 1998.
- [53] J. Czyzowicz, L. Gasieniec, and A. Pelc. Gathering few fat mobile robots in the plane. *Theoretical Computer Science*, 410(6):481 – 499, 2009.

- [54] G. D'Angelo, G. D. Stefano, and A. Navarra. *Gathering Asynchronous and Oblivious Robots on Basic Graph Topologies Under the Look-Compute-Move Model*, pages 197–222. 2013.
- [55] S. Das. Mobile agents in distributed computing: Network exploration. *Bulletin of the European Association for Theoretical Computer Science*, 109:54–69, 2013.
- [56] S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro. Map construction of unknown graphs by multiple agents. *Theoretical Computer Science*, 385(1):34 – 48, 2007.
- [57] J. J. Daymude, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Improved leader election for self-organizing programmable matter. In *International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*, pages 127–140, 2017.
- [58] E. Demaine, M. Demaine, M. Hoffmann, and J. O'Rourke. Pushing blocks is hard. *Computational Geometry*, 26(1):21–36, 2003.
- [59] E. D. Demaine, M. L. Demaine, S. P. Fekete, M. Ishaque, E. Rafalin, R. T. Schweller, and D. L. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. *Natural Computing*, 7(3):347–370, 2008.
- [60] E. D. Demaine, M. L. Demaine, S. P. Fekete, M. J. Patitz, R. T. Schweller, A. Winslow, and D. Woods. One tile to rule them all: Simulating any tile assembly system with a single universal tile. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 368–379, 2014.
- [61] E. D. Demaine, S. P. Fekete, P. Keldenich, H. Meijer, and C. Scheffer. Coordinated Motion Planning: Reconfiguring a Swarm of Labeled Robots with Bounded Stretch. In *Symposium on Computational Geometry (SoCG)*, pages 29:1–29:15, 2018.
- [62] E. D. Demaine, S. P. Fekete, C. Scheffer, and A. Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Computer Science*, 671:4–18, 2017.
- [63] Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Brief announcement: Amoebot – a new model for programmable matter. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 220–222, 2014.

- [64] Z. Derakhshandeh, R. Gmyr, A. Porter, A. W. Richa, C. Scheideler, and T. Strothmann. On the runtime of universal coating for programmable matter. In *DNA Computing and Molecular Programming*, pages 148–164, 2016.
- [65] 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 *International Conference on Nanoscale Computing and Communication (NANOCOM)*, page 21, 2015.
- [66] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal shape formation for programmable matter. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–299, 2016.
- [67] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal coating for programmable matter. *Theoretical Computer Science*, 671:56–68, 2017.
- [68] Z. Derakhshandeh, R. Gmyr, T. Strothmann, R. Bazzi, A. W. Richa, and C. Scheideler. Leader election and shape formation with self-organizing programmable matter. In *DNA Computing and Molecular Programming*, pages 117–132, 2015.
- [69] A. Dessmark, P. Fraigniaud, D. R. Kowalski, and A. Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006.
- [70] G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. Shape formation by programmable particles. *Distributed Computing*, 0:1–33, 2019.
- [71] G. Di Stefano and A. Navarra. Optimal gathering of oblivious robots in anonymous graphs. In *Structural Information and Communication Complexity*, pages 213–224, 2013.
- [72] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38 – 63, 2004.
- [73] Y. Disser, J. Hackfeld, and M. Klimm. Tight bounds for undirected graph exploration with pebbles and multiple agents. *Journal of the ACM*, 66(6):1–41, 2019.
- [74] M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *Journal on Robotics and Automation*, 4(4):369–379, 1988.

- [75] S. P. Fekete, R. Gmyr, S. Hugo, P. Keldenich, C. Scheffer, and A. Schmidt. Cadbots: Algorithmic aspects of manipulating programmable matter with finite automata. In *International Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2018.
- [76] S. P. Fekete, J. Grosse-Holz, P. Keldenich, and A. Schmidt. Parallel online algorithms for the bin packing problem. In *Workshop on Approximation and Online Algorithms (WAOA)*, pages 106–119, 2020.
- [77] S. P. Fekete, J. Hendricks, M. J. Patitz, T. A. Rogers, and R. T. Schweller. Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly. In *Symposium on Discrete Algorithms (SODA)*, pages 148–167, 2014.
- [78] S. Felton, M. Tolley, E. Demaine, D. Rus, and R. Wood. A method for building self-folding machines. *Science*, 345(6197):644–646, 2014.
- [79] R. Fleischer and G. Trippen. Exploring an unknown graph efficiently. In *Algorithms – ESA 2005*, pages 11–22, 2005.
- [80] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1):147 – 168, 2005.
- [81] F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.
- [82] P. Fraigniaud, L. Gasieniec, D. R. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006.
- [83] P. Fraigniaud and D. Ilcinkas. Digraphs exploration with little memory. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 246–257, 2004.
- [84] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph Exploration by a Finite Automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005.
- [85] L. Gasieniec, A. Pelc, T. Radzik, and X. Zhang. Tree exploration with logarithmic memory. In *Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07*, pages 585–594, 2007.
- [86] L. Gasieniec and T. Radzik. Memory efficient anonymous graph exploration. In *Graph-Theoretic Concepts in Computer Science*, pages 14–29, 2008.

- [87] K. Gilpin, A. Knaian, and D. Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *International Conference on Robotics and Automation (ICRA)*, pages 2485–2492, 2010.
- [88] R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, and C. Scheideler. Shape recognition by a finite automaton robot. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 52:1–52:15, 2018.
- [89] R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler, and T. Strothmann. Forming tile shapes with simple robots. In *International Conference on DNA Computing and Molecular Programming*, pages 122–138, 2018.
- [90] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10(2-4):201–225, 1993.
- [91] S. C. Goldstein, J. D. Campbell, and T. C. Mowry. Programmable matter. *Computer*, 38(6):99–101, 2005.
- [92] S. C. Goldstein and T. Mowry. Claytronics: A scalable basis for future robots. *Robosphere*, 2004.
- [93] S. W. Golomb. Checker boards and polyominoes. *The American Mathematical Monthly*, 61(10):675–682, 1954.
- [94] S. W. Golomb. Tiling with sets of polyominoes. *Journal of Combinatorial Theory*, 9(1):60 – 71, 1970.
- [95] C. E. Gregg, J. H. Kim, and K. C. Cheung. Ultra-light and scalable composite lattice materials. *Advanced Engineering Materials*, 20(9):1800213, 2018.
- [96] B. H. Gwee and M. H. Lim. Polyominoes tiling by a genetic algorithm. *Computational Optimization and Applications*, 6(3):273–291, Nov 1996.
- [97] M. M. Halldórsson, K. Iwama, S. Miyazaki, and S. Taketomi. Online independent sets. *Theoretical Computer Science*, 289(2):953–962, 2002.
- [98] D. Halperin, L. E. Kavraki, and K. Solovey. Robotics. In *Handbook of discrete and computational geometry*, pages 1343–1376. 2017.
- [99] D. Halperin, O. Salzman, and M. Sharir. Algorithmic motion planning. In *Handbook of Discrete and Computational Geometry*, pages 1311–1342. 2017.

- [100] E. Hawkes, B. An, N. M. Benbernou, H. Tanaka, S. Kim, E. D. Demaine, D. Rus, and R. J. Wood. Programmable matter by folding. *National Academy of Sciences*, 107(28):12441–12445, 2010.
- [101] S. Heydrich and R. van Stee. Beating the Harmonic Lower Bound for Online Bin Packing. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 41:1–41:14, 2016.
- [102] M. Hoffmann. Motion planning amidst movable square blocks: Push-* is NP-hard. In *Canadian Conference on Computational Geometry (CCCG)*, pages 205–210, 2000.
- [103] F. Hurtado, E. Molina, S. Ramaswami, and V. Sacristán. Distributed reconfiguration of 2d lattice-based modular robotic systems. *Autonomous Robots*, 38(4):383–413, 2015.
- [104] B. Jenett, A. Abdel-Rahman, K. C. Cheung, and N. Gershenfeld. Material-robot system for assembly of discrete cellular structures. *Robotics and Automation Letters*, 2019.
- [105] B. Jenett, S. Calisch, D. Cellucci, N. Cramer, N. Gershenfeld, S. Swei, and K. C. Cheung. Digital morphing wing: active wing shaping concept using composite lattice-based cellular structures. *Soft robotics*, 4(1):33–48, 2017.
- [106] B. Jenett and D. Cellucci. A mobile robot for locomotion through a 3d periodic lattice environment. In *International Conference on Robotics and Automation (ICRA)*, pages 5474–5479, 2017.
- [107] B. Jenett and K. Cheung. Bill-e: Robotic platform for locomotion and manipulation of lightweight space structures. In *Adaptive Structures Conference*, page 1876, 2017.
- [108] B. Jenett, C. Gregg, D. Cellucci, and K. Cheung. Design of multifunctional hierarchical space structures. In *Aerospace Conference*, pages 1–10, 2017.
- [109] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272 – 314, 1974.
- [110] S. Kamali and A. L. Ortiz. Better compression through better list update algorithms. In *Data Compression Conference (DCC)*, pages 372–381, 2014.
- [111] S. Kamei, A. Lamani, F. Ooshita, and S. Tixeuil. Asynchronous mobile robot gathering from symmetric configurations without global multiplicity detection. In *Structural Information and Communication Complexity*, pages 150–161, 2011.

- [112] P. S. S. Kim, A. T. Becker, Y. Ou, A. A. Julius, and M. J. Kim. Imparting magnetic dipole heterogeneity to internalized iron oxide nanoparticles for microorganism swarm control. *Journal of Nanoparticle Research*, 17(3):1–15, 2015.
- [113] P. S. S. Kim, A. T. Becker, Y. Ou, M. J. Kim, et al. Swarm control of cell-based microrobots using a single global magnetic field. In *International Conference on Ubiquitous Robotics and Ambient Intelligence (URAI)*, pages 21–26, 2013.
- [114] D. A. Klarner. Packing a rectangle with congruent n-ominoes. *Journal of Combinatorial Theory*, 7(2):107–115, 1969.
- [115] A. N. Knaian, K. C. Cheung, M. B. Lobovsky, A. J. Oines, P. Schmidt-Neilsen, and N. A. Gershenfeld. The milli-motein: A self-folding chain of programmable matter with a one centimeter module pitch. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1447–1453, 2012.
- [116] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *Journal of the ACM*, pages 562–572, 1985.
- [117] J. S. Lewis and J. M. O’Kane. Planning for provably reliable navigation using an unreliable, nearly sensorless robot. *The International Journal of Robotics Research*, 32(11):1342–1357, 2013.
- [118] F. M. Liang. A lower bound for on-line bin packing. *Information Processing Letters*, 10(2):76 – 79, 1980.
- [119] A. López-Ortiz and S. Schuierer. On-line parallel heuristics, processor scheduling and robot searching under the competitive framework. *Theoretical Computer Science*, 310(1-3):527–537, 2004.
- [120] K. Lund, A. Manzo, N. Dabby, N. Michelotti, A. Johnson-Buck, J. Nangreave, S. Taylor, R. Pei, M. Stojanovic, N. Walter, and E. Winfree. Molecular robots guided by prescriptive landscapes. *Nature*, 465(7295):206–210, 2010.
- [121] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *The International Journal of Robotics Research*, 15(6):533–556, 1996.
- [122] A. V. Mahadev, D. Krupke, J.-M. Reinhardt, S. P. Fekete, and A. T. Becker. Collecting a swarm in a grid environment using shared, global inputs. In *International Conference on Automation Science and Engineering (CASE)*, pages 1231–1236, 2016.

- [123] P. Mannam, A. V. Volkov, R. Paolini, G. Chirikjian, and M. T. Mason. Sensorless pose determination using randomized action sequences. *Entropy*, 21(2):154, 2019.
- [124] S. Manzoor, S. Sheckman, J. Lonsford, H. Kim, M. J. Kim, and A. T. Becker. Parallel self-assembly of polyominoes under uniform control inputs. *Robotics and Automation Letters*, 2(4):2040–2047, 2017.
- [125] G. D. Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, and U. Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315 – 326, 2006.
- [126] E. Markou. Identifying hostile nodes in networks using mobile agents. *Bulletin of the European Association for Theoretical Computer Science*, 108:93–129, 2012.
- [127] O. Michail and P. G. Spirakis. Simple and efficient local codes for distributed stable network construction. *Distributed Computing*, 29(3):207–237, 2016.
- [128] R. A. Moan, V. M. Baez, A. T. Becker, and J. M. O’Kane. Aggregation and localization of simple robots in curved environments. In *International Conference on Robotics and Automation (ICRA)*, 2020. To appear.
- [129] National Aeronautics and Space Administration (NASA). STS-133 Shuttle Mission Imagery, 2011.
- [130] A. Naz, B. Piranda, J. Bourgeois, and S. C. Goldstein. A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots. In *15th International Symposium on Network Computing and Applications (NCA)*, pages 254–263, 2016.
- [131] E. Niehs, A. Schmidt, C. Scheffer, D. E. Biediger, M. Yannuzzi, B. Jenett, A. Abdel-Rahman, K. C. Cheung, A. T. Becker, and S. P. Fekete. Recognition and reconfiguration of lattice-based cellular structures by simple robots. In *International Conference on Robotics and Automation (ICRA)*, 2020. To appear.
- [132] T. Omabegho, R. Sha, and N. Seeman. A bipedal DNA Brownian motor with coordinated legs. *Science*, 324(5923):67–71, 2009.
- [133] P. Panaite and A. Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33(2):281 – 295, 1999.
- [134] M. J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.

- [135] A. Pelc. Deterministic rendezvous in networks: A comprehensive survey. *Networks*, 59(3):331–347, 2012.
- [136] B. Piranda and J. Bourgeois. Geometrical study of a quasi-spherical module for building programmable matter. In *Distributed Autonomous Robotic Systems (DARS)*, pages 387–400. 2018.
- [137] M. Reid. Tiling rectangles and half strips with congruent polyominoes. *Journal of Combinatorial Theory Series A*, 80:106–123, 1997.
- [138] J. H. Reif. Complexity of the mover’s problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs)*, pages 421–427, 1979.
- [139] J. H. Reif and S. Sahu. Autonomous programmable DNA nanorobotic devices using dnazymes. *Theoretical Computer Science*, 410:1428–1439, 2009.
- [140] M. P. Renault, A. Rosén, and R. van Stee. Online algorithms with advice for bin packing and scheduling problems. *Theoretical Computer Science*, 600:155–170, 2015.
- [141] P. W. K. Rothmund and E. Winfree. The program-size complexity of self-assembled squares (extended abstract). In *ACM Symposium on Theory of Computing (STOC)*, pages 459–468, 2000.
- [142] M. Rubenstein, C. Ahler, N. Hoff, A. Cabrera, and R. Nagpal. Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, 62(7):966–975, 2014.
- [143] M. Rubenstein, A. Cabrera, J. Werfel, G. Habibi, J. McLurkin, and R. Nagpal. Collective transport of complex objects by simple robots: theory and experiments. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 47–54, 2013.
- [144] M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.
- [145] A. Schmidt, V. M. Baez, A. T. Becker, and S. P. Fekete. Coordinated particle relocation using finite static friction with boundary walls. *Robotics and Automation Letters*, 5(2):985–992, 2020.
- [146] A. Schmidt, S. Manzoor, L. Huang, A. T. Becker, and S. P. Fekete. Efficient parallel self-assembly under uniform control inputs. *Robotics and Automation Letters*, 3(4):3521–3528, 2018.

- [147] J. T. Schwartz and M. Sharir. On the piano movers' problem: III. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *The International Journal of Robotics Research*, 2(3):46–75, 1983.
- [148] R. Schweller, A. Winslow, and T. Wylie. Verification in staged tile self-assembly. *Natural Computing*, 18(1):107–117, 2019.
- [149] H. M. Shad, R. Morris-Wright, E. D. Demaine, S. P. Fekete, and A. T. Becker. Particle computation: Device fan-out and binary memory. In *International Conference on Robotics and Automation (ICRA)*, pages 5384–5389, 2015.
- [150] A. N. Shah. Pebble automata on arrays. *Computer Graphics and Image Processing*, 3(3):236–246, 1974.
- [151] S. Shahrokhi and A. T. Becker. Stochastic swarm control with global inputs. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 421–427, 2015.
- [152] S. Shahrokhi, A. Mahadev, and A. T. Becker. Algorithms for shaping a particle swarm with a shared input by exploiting non-slip wall contacts. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 4304–4311, 2017.
- [153] S. Shahrokhi, J. Shi, B. Isichei, and A. T. Becker. Exploiting nonslip wall contacts to position two particles using the same control input. *Transactions on Robotics*, 35(3):577–588, 2019.
- [154] J. Shin and N. Pierce. A synthetic DNA walker for molecular transport. *Journal of the American Chemical Society*, 126:4903–4911, 2004.
- [155] D. Soloveichik and E. Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007.
- [156] A. Ta-Shma and U. Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Transactions on Algorithms*, 10(3):12:1–12:15, 2014.
- [157] Y. Takefuji and Y. . Lee. A parallel algorithm for tiling problems. *Transactions on Neural Networks*, 1(1):143–145, March 1990.
- [158] Y. Terada and S. Murata. Automatic modular assembly system and its distributed control. *International Journal of Robotics Research*, 27(3–4):445–462, 2008.

- [159] P. Thalamy, B. Piranda, F. Lassabe, and J. Bourgeois. Scaffold-based asynchronous distributed self-reconfiguration by continuous module flow. In *International Conference on Intelligent Robots and Systems (IROS)*, 2019.
- [160] 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):eaan6558, 2017.
- [161] T. Tucci, B. Piranda, and J. Bourgeois. A distributed self-assembly planning algorithm for modular robots. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 550–558, 2018.
- [162] A. van Vliet. An improved lower bound for on-line bin packing algorithms. *Information processing letters*, 43(5):277 – 284, 1992.
- [163] S. Vassilvitskii, J. Kubica, E. Rieffel, J. Suh, and M. Yim. On the general reconfiguration problem for expanding cube style modular robots. In *International Conference on Robotics and Automation (ICRA)*, pages 801–808, 2002.
- [164] J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17(2):171–189, Aug 2004.
- [165] H. Wang. Proving theorems by pattern recognition — II. *Bell System Technical Journal*, 40(1):1–41, 1961.
- [166] Z. Wang, J. Elbaz, and I. Willner. A dynamically programmed DNA transporter. *Angewandte Chemie International Edition*, 51(48):4322–4326, 2012.
- [167] J. Werfel and R. Nagpal. Extended stigmergy in collective construction. *Intelligent Systems*, 21(2):20–28, 2006.
- [168] J. Werfel and R. Nagpal. Three-dimensional construction with mobile robots and modular blocks. *The International Journal of Robotics Research*, 27(3-4):463–479, 2008.
- [169] S. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A. Turberfield. A DNA-based molecular motor that can navigate a network of tracks. *Nature Nanotechnology*, 7(3):169–173, 2012.
- [170] E. Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

- [171] 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 *4th Conference of Innovations in Theoretical Computer Science*, pages 353–354, 2013.
- [172] H. Yamamoto, H. Ninomiya, and H. Asai. Application of neuro-based optimization to 3-d rectangular puzzles. In *International Joint Conference on Neural Networks Proceedings*, volume 2, pages 1652–1656, 1998.
- [173] A. C.-C. Yao. New algorithms for bin packing. *Journal of the ACM*, 27(2):207–227, 1980.
- [174] Y. Zhang, X. Chen, H. Qi, and D. Balkcom. Rearranging agents in a small space using global controls. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 3576–3582, 2017.
- [175] Y. Zhang, E. Whiting, and D. Balkcom. Assembling and disassembling planar structures with divisible and atomic components. *Transactions on Automation Science and Engineering*, 15(3):945–954, 2018.
- [176] X. Zhao and H. Shen. On the advice complexity of one-dimensional online bin packing. In *Frontiers in Algorithmics*, pages 320–329, 2014.



Tilt Assembly: Algorithms for Micro-factories That Build Objects with Uniform External Forces

Aaron T. Becker¹ · Sándor P. Fekete² · Phillip Keldenich² ·
Dominik Krupke² · Christian Rieck² · Christian Scheffer² ·
Arne Schmidt²

Received: 12 January 2018 / Accepted: 12 July 2018 / Published online: 3 August 2018
© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract

We present algorithmic results for the parallel assembly of many micro-scale objects in two and three dimensions from tiny particles, which has been proposed in the context of programmable matter and self-assembly for building high-yield micro-factories. The underlying model has particles moving under the influence of uniform external forces until they hit an obstacle. Particles bond when forced together with another appropriate particle. Due to the physical and geometric constraints, not all shapes can be built in this manner; this gives rise to the TILT ASSEMBLY PROBLEM (TAP) of deciding constructibility. For simply-connected polyominoes P in 2D consisting of N unit-squares (“tiles”), we prove that TAP can be decided in $O(N \log N)$ time. For the optimization variant MAXTAP (in which the objective is to construct a subshape of maximum possible size), we show *polyAPX*-hardness: unless $P=NP$, MAXTAP cannot be approximated within a factor of $\Omega(N^{\frac{1}{3}})$; for tree-shaped structures, we give an $\Omega(N^{\frac{1}{2}})$ -approximation algorithm. For the efficiency of the assembly process itself, we show that any constructible shape allows *pipelined* assembly, which produces copies of P in $O(1)$ amortized time, i.e., N copies of P in $O(N)$ time steps. These considerations can be extended to three-dimensional objects: For the class of polycubes P we prove that it is NP -hard to decide whether it is possible to construct a path between two points of P ; it is also NP -hard to decide constructibility of a polycube P . Moreover, it is *expAPX*-hard to maximize a sequentially constructible path from a given start point.

Keywords Programmable matter · Micro-factories · Tile assembly · Tilt · Approximation · Hardness

Aaron T. Becker: Work from this author was partially supported by National Science Foundation IIS-1553063 and IIS-1619278.

Extended author information available on the last page of the article

1 Introduction

In recent years, progress on flexible construction at micro- and nano-scale has given rise to a large set of challenges that deal with algorithmic aspects of programmable matter. Examples of cutting-edge application areas with a strong algorithmic flavor include self-assembling systems, in which chemical and biological substances such as DNA are designed to form predetermined shapes or carry out massively parallel computations; and swarm robotics, in which complex tasks are achieved through the local interactions of robots with severely limited individual capabilities, including micro- and nano-robots.

Moving individual particles to their appropriate attachment locations when assembling a shape is difficult because the small size of the particles limits the amount of onboard energy and computation. One successful approach to dealing with this challenge is to use molecular diffusion in combination with cleverly designed sets of possible connections: in *DNA tile self-assembly*, the particles are equipped with sophisticated bonds that ensure only a predesigned shape is produced when mixing together a set of tiles, see [31]. The resulting study of algorithmic tile self-assembly has given rise to an extremely powerful framework and produced a wide range of impressive results. However, the required properties of the building material (which must be specifically designed and finely tuned for each particular shape) in combination with the construction process (which is left to chemical reactions, so it cannot be controlled or stopped until it has run its course) make DNA self-assembly unsuitable for some applications.

An alternative method for controlling the eventual position of particles is to apply a uniform external force, causing all particles to move in a given direction until they hit an obstacle or another blocked particle. As two of us (Becker and Fekete, [4]) have shown in the past, combining this approach with custom-made obstacles (instead of custom-made particles) allows complex rearrangements of particles, even in grid-like environments with axis-parallel motion. The appeal of this approach is that it shifts the design complexity from the building material (the tiles) to the machinery (the environment). As recent practical work by Manzoor et al. [23] shows, it is possible to apply this to simple “sticky” particles that can be forced to bond, see Fig. 1: the overall assembly is achieved by adding particles one at a time, attaching them to the existing sub-assembly.

Moreover, Manzoor et al. [23] argue that it is possible to enhance the overall assembly environment to allow pipelined construction, as shown in Fig. 2: after constructing the first polyomino, each cycle of a small control sequence produces another polyomino. However, the algorithmic part of [23] is purely heuristic; providing a thorough understanding of algorithms and complexity is the content of our paper.

One critical issue of this approach is the requirement of getting particles to their destination without being blocked by or bonding to other particles. As Fig. 3 shows, this is not always possible, so there are some shapes that cannot be constructed by Tilt Assembly.

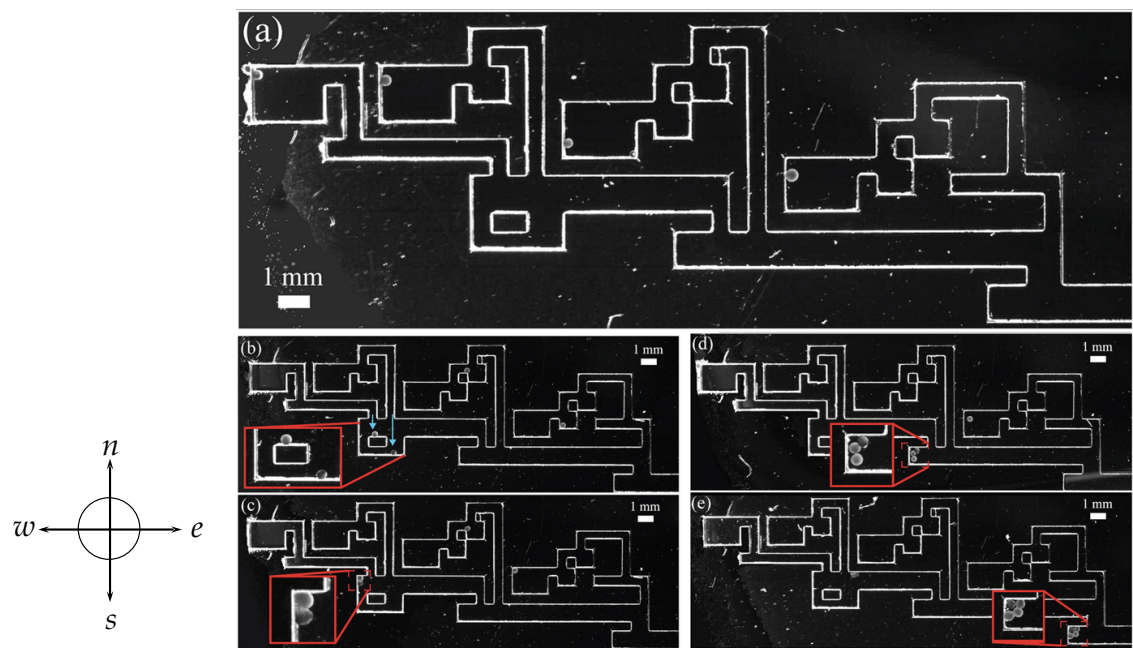


Fig. 1 A practical demonstration of Tilt Assembly based on alginate (i.e., a gel made by combining a powder derived from seaweed with water) particles on a silicon wafer with obstacles etched in photoresist [23]. The obstacles appear as white lines and block the motion of particles. **a** Alginate particles in initial positions. **b** After control moves of $\langle e, s, w, n, e, s \rangle$ (for east, south, west, north), the alginate microrobots move to the shown positions. **c** After $\langle w, n \rangle$ inputs, the system produces the first multi-microrobot polyomino. **d** The next three microrobot polyominoes are produced after applying multiple $\langle e, s, w, n \rangle$ cycles. **e** After the alginate microrobots have moved through the microfluidic factory layout, the final 4-particle polyomino is generated

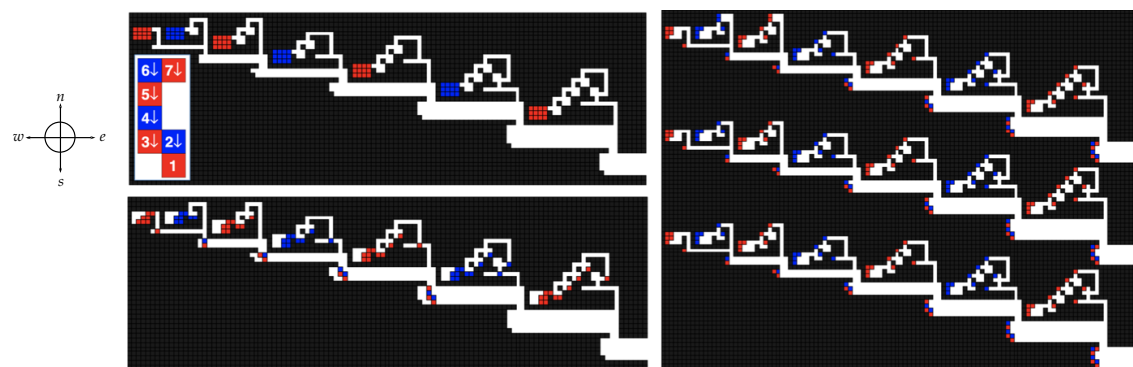


Fig. 2 (Top left) Initial setup of a seven-tile polyomino assembly; the composed shape is shown enlarged on the lower left. The bipartite decomposition into blue and red particles is shown for greater clarity, but can also be used for better control of bonds. The sequence of control moves is $\langle e, s, w, n \rangle$, i.e., a clockwise order. (Bottom left) The situation after 18 control moves. (Right) The situation after 7 full cycles, i.e., after 28 control moves; shown are three parallel “factories” (Color figure online)

This gives rise to a variety of algorithmic questions: (1) Can we decide efficiently whether a given polyomino can be constructed by Tilt Assembly? (2) Can the resulting process be pipelined to yield low amortized building time? (3) Can we compute a maximum-size subpolyomino that can be constructed? (4) What can be said about three-dimensional versions of the problem?

Fig. 3 A polyomino (black) that cannot be constructed by Tilt Assembly: the last tile cannot be attached, as it gets blocked by previously attached tiles

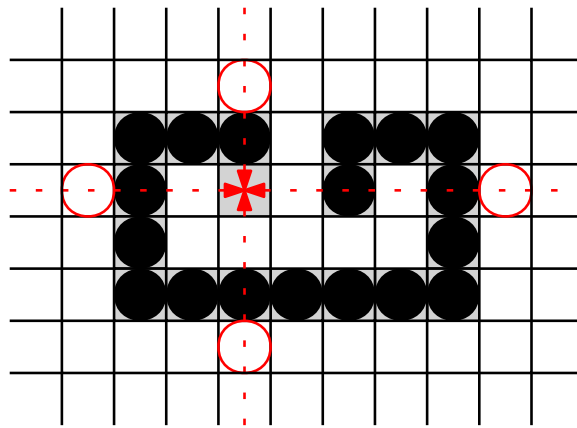


Table 1 Results for Tilt Assembly Problem (TAP) and its maximization variant (MAXTAP)

| Dimension | Polyomino | Decision | Maximization | Approximation | | Constructible path |
|-----------|-----------|-----------------------------------|-----------------------|-----------------------------|--------------------|-----------------------------------|
| 2D | Simple | $O(N \log N)$ (Sect. 3) | polyAPX-hard | $O(N^{1/3})$, (Sect. 4) | $\Omega(\sqrt{N})$ | $O(N \log N)$ (Sect. 4) |
| 3D | General | NP-complete (Sect. 5) | polyAPX-hard | $O(N^{1/3})$, (Sect. 4) | – | NP-complete (Sect. 5) |

1.1 Our Contribution

Our main contribution is a characterization of deciding constructibility and efficient construction for simply connected two-dimensional shapes: For a simple polyomino P with N pixels, we can decide in time $O(N \log N)$ whether P can be constructed; using pipelining, a constructible polyomino can be built in amortized time $O(1)$. On the other hand, we show that deciding constructibility in 3D is NP-complete. We also provide a number of additional results on approximation and the constructibility of subpaths; see Table 1 for an overview.

1.2 Related Work

Assembling polyominoes with tiles has been considered intensively in the context of *tile self-assembly*. In 1998, Erik Winfree [31] introduced the *abstract tile self-assembly model* (aTAM), in which tiles have glue types on each of the four sides and two tiles can stick together if their glue type matches and the bonding strength is sufficient. Starting with a *seed tile*, tiles will continue to attach to the existing partial assembly until they form a desired polyomino; the process stops when no further attachments are possible. For early examples of related work, see Rothmund and Winfree [24] and Adleman et al. [1] for the running time and program size for self-assembling squares. Apart from the aTAM, there are various other models like the *two-handed tile self-assembly model* (2HAM) [11] and the *hierarchical tile self-assembly model* [13], in which we have no single seed but pairs of subassemblies that can attach to each other. Furthermore, the *staged self-assembly model* [12,14,16] allows greater efficiency by assembling

polyominoes in multiple bins which are gradually combined with the content of other bins.

All this differs from the model in Tilt Assembly, in which each tile has the same glue type on all four sides, and tiles are added to the assembly one at a time by attaching them from the outside along a straight line. This approach of externally movable tiles has been considered in practice at the microscale level using biological cells and an MRI, see [7,20,21]. Becker et al. [8] consider this for the assembly of a magnetic *Gauß gun*, which can be used for applying strong local forces by very weak triggers, allowing applications such as micro-surgery.

Using an external force for moving the robots becomes inevitable at some scale because the energy capacity decreases faster than the energy demand. A consequence is that all non-fixed robots/particles perform the same movement, so all particles move in the same direction of the external force until they hit an obstacle or another particle. These obstacles allow shaping the particle swarm. Designing appropriate sets of obstacles and moves gives rise to a range of algorithmic problems. Deciding whether a given initial configuration of particles in a given environment can be transformed into a desired target configuration is *NP*-hard [4], even in a grid-like setting, whereas finding an optimal control sequence is shown to be *PSPACE*-complete by Becker et al. [5]. However, if *designing* the obstacles is allowed, the problems become more tractable [4]. Moreover, even complex computations become possible: If we allow additional particles of double size (i.e., two adjacent fields), full computational complexity is achieved, see Shad et al. [26]. Further related work includes gathering a particle swarm at a single position [22] and using swarms of very simple robots (such as Kilobots) for moving objects [9]. For the case in which human controllers have to move objects by such a swarm, Becker et al. [6] study different control options. The results are used by Shahrokhi and Becker [27] to investigate an automatic controller.

The construction of polyominoes has also applications in the field of robot swarms, e.g., *shape formation*. Werfel and Nagpal [29,30] show how multiple robots can move tiles to a partial assembly to construct a desired shape in 2D and 3D. Derakhshandeh et al. [17,18] consider the robots as building material, which have $O(1)$ memory, and provide algorithms letting the robots form or coat shapes. In a very recent paper, Demaine et al. [3,15] show that a robot swarm can be reconfigured in time $O(d)$ unit steps, where d is the maximum distance of any robot. However, this requires the robots to be well separable. Arbuckle and Requicha [2] show how a self-organized swarm of robots can construct a certain shape. In case of robot failures or external disturbance, the swarm is also able to repair the shape.

Further related work includes robots performing various tasks: Thubagere et al. [28] show that robots made from DNA can simultaneously sort molecular cargoes. Rubenstein et al. [25] consider a swarm of simple robots moving an object to a desired destination without knowing its shape and weight. Hoffmann [19] proves that it is *NP*-hard to decide if a robot can push its way through an area filled with blocks.

2 Preliminaries

Polyomino: For a set $P \subset \mathbb{Z}^2$ of N grid points in the plane, the graph G_P is the induced grid graph, in which two vertices $p_1, p_2 \in P$ are connected if they are at unit distance. Any set P with connected grid graph G_P gives rise to a *polyomino* by replacing each point $p \in P$ by a unit square centered at p , which is called a *tile*; for simplicity, we also use P to denote the polyomino when the context is clear, and refer to G_P as the dual graph of the polyomino; P is *tree-shaped* if G_P is a tree. A polyomino is called *hole-free* or *simple* if and only if the grid graph induced by $\mathbb{Z}^2 \setminus P$ is connected.

Blocking sets: For each point $p \in \mathbb{Z}^2$ we define *blocking sets* $N_p, S_p \subseteq P$ as the set of all points $q \in P$ that are above or below p and $|p_x - q_x| \leq 1$. Analogously, we define the blocking sets $E_p, W_p \subseteq P$ as the set of all points $q \in P$ that are to the right or to the left of p and $|p_y - q_y| \leq 1$.

Construction step: A *construction step* is defined by a direction $d \in \{\text{north, east, south, west}\}$ (abbreviated by n, e, s, w) from which a tile is added and a latitude/longitude l describing a column or row. The tile arrives from (l, ∞) for north, (∞, l) for east, $(l, -\infty)$ for south, and $(-\infty, l)$ for west into the corresponding direction until it reaches the first grid position that is adjacent to one occupied by an existing tile. If there is no such tile, the polyomino does not change. We note that a position p can be added to a polyomino P if and only if there is a point $q \in P$ with $\|p - q\|_1 = 1$ and one of the four blocking sets, N_p, E_p, S_p or W_p , is empty. Otherwise, if none of these sets are empty, this position is *blocked*.

Constructibility: Beginning with a seed tile at some position p , a polyomino P is *constructible* if and only if there is a sequence $\sigma = ((d_1, l_1), (d_2, l_2), \dots, (d_{N-1}, l_{N-1}))$, such that the resulting polyomino P' , induced by successively adding tiles with σ , is equal to P . We allow the constructed polyomino P' to be a translated copy of P . Reversing σ yields a *decomposition sequence*, i.e., a sequence of tiles removed from P .

3 Constructibility of Simple Polyominoes

In this section we focus on hole-free (i.e., simple) polyominoes. We show that the problem of deciding whether a given polyomino can be constructed can be solved in polynomial time. This decision problem can be defined as follows.

Definition 1 (TILT ASSEMBLY PROBLEM). Given a polyomino P , the TILT ASSEMBLY PROBLEM (TAP) asks for a sequence of tiles constructing P , if P is constructible.

3.1 A Key Lemma

A simple observation is that construction and (connectivity-preserving) decomposition are the same problem. This allows us to give a more intuitive argument, as it is easier to argue that we do not lose connectivity when removing tiles than it is to prove that we do not block future tiles.



(a) Removing t destroys decomposability. The polyomino can be decomposed by starting with the three tiles above t . **(b)** Removing the locally convex tile t leaves the polyomino non-decomposable; it can be decomposed by starting from the bottom or the sides.

Fig. 4 Two polyominoes and their locally convex tiles (white). **a** Removing not locally convex tiles may destroy decomposability. **b** With non-simple polygons we may not be able to remove locally convex tiles

Theorem 2 *A polyomino P can be constructed if and only if it can be decomposed using a sequence of tile removal steps that preserve connectivity. A construction sequence is a reversed decomposition sequence.*

Proof To prove this theorem, it suffices to consider a single step. Let P be a polyomino and t be a tile that is removed from P into some direction l , leaving a polyomino P' . Conversely, adding t to P' from direction l yields P , as there cannot be any tile that blocks t from reaching the correct position, or we would not be able to remove t from P in direction l . \square

For hole-free polyominoes we can efficiently find a construction/decomposition sequence if one exists. The key insight is that one can greedily remove *locally convex* tiles. A tile t is said to be locally convex if and only if it is locally extremal, i.e., there are two axis-parallel orthogonal directions for which there is no tile connected to t ; see Fig. 4. If a locally convex tile is *not* a cut tile, i.e., it is a tile whose removal does *not* disconnect the polyomino, its removal does not interfere with the decomposability of the remaining polyomino.

This conclusion is based on the observation that a minimal cut (i.e., a minimal set of vertices whose removal leaves a disconnected polyomino) of cardinality two in a hole-free polyomino always consists of two (possibly diagonally) adjacent tiles. Furthermore, we can always find such a removable locally convex tile in any decomposable hole-free polyomino. This allows us to devise a simple greedy algorithm.

We start by showing that if we find a non-blocked locally convex tile that is not a cut tile, we can simply remove it. It is important to focus on locally convex tiles, as the removal of not locally convex tiles can harm the decomposability: see Fig. 4a for an illustration. In non-simple polyominoes, the removal of locally convex tiles can destroy decomposability, as demonstrated in Fig. 4b.

Lemma 3 *Consider a non-blocked, non-cut, locally convex tile t in a hole-free polyomino P . The polyomino $P - t$ is decomposable if and only if P is decomposable.*

Proof The first direction is trivial: if $P - t$ is decomposable, P is decomposable as well, because we can remove the non-blocked tile t first and afterwards use the existing decomposition sequence for $P - t$. The other direction requires some case distinctions. Suppose for contradiction that P is decomposable but $P - t$ is not, i.e., t is important for the later decomposition.

Consider a valid decomposition sequence for P and the first tile t' we cannot remove if we were to remove t in the beginning. W.l.o.g., let t' be the first tile in this sequence

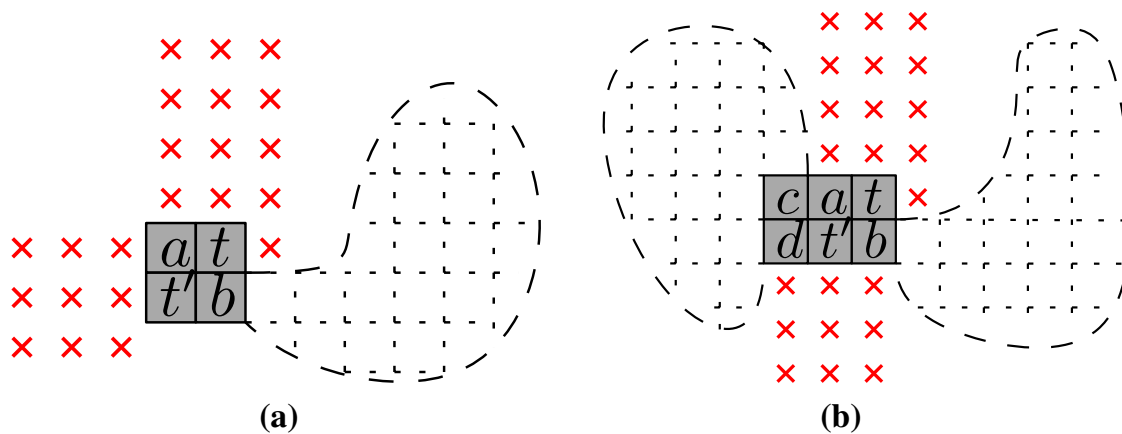


Fig. 5 The red marks indicate that no tile is at this position; the dashed outline represents the rest of the polyomino. **a** If the unblocked directions of t and t' are orthogonal, one of the two adjacent tiles (w.l.o.g. a) cannot have any further neighbors. There can also be no tiles in the upper left corner, because the polyomino cannot cross the two free directions of t and t' (red marks) and **b** if the unblocked directions of t and t' are parallel, there is only the tile c for which something can change if we remove t before t' (Color figure online)

(removing all previous tiles obviously does not destroy the decomposability). When we remove t first, we are missing a tile, hence t' cannot be blocked but has to be a cut tile in the remaining polyomino $P - t$. The presence of t preserves connectivity, i.e., $\{t, t'\}$ is a minimal cut on P . Because P has no holes, then t and t' must be diagonal neighbors, sharing the neighbors a and b . Furthermore, by definition neither of t and t' is blocked in some direction. We make a case distinction on the relation of these two directions.

The directions are orthogonal (Fig. 5a): Either a or b is a non-blocked locally convex tile, because t and t' are both non-blocked; w.l.o.g., let this be a . It is easy to see that independent of removing t or t' first, after removing a we can also remove the other one.

The directions are parallel (Fig. 5b): This case is slightly more involved. By assumption, we have a decomposition sequence beginning with t' . We show that swapping t' with our locally convex tile t in this sequence preserves feasibility.

The original sequence has to remove either a or b before it removes t , as otherwise the connection between the two is lost when t' is removed first. After either a or b is removed, t becomes a leaf and can no longer be important for connectivity. Thus, we only need to consider the sequence until either a or b is removed. The main observation is that a and b block the same tiles as t or t' , except for tile c as in Fig. 5b. However, when c is removed, c has to be a leaf, because a is still not removed and in the original decomposition sequence, t' has already been removed. Therefore, a tile $d \neq t'$ would have to be removed before c . Hence, the decomposition sequence remains feasible, concluding the proof. \square

Next we show that such a locally convex tile always exists if the polyomino is decomposable.

Lemma 4 *Let P be a decomposable polyomino. Then there exists a locally convex tile that is removable without destroying connectivity.*

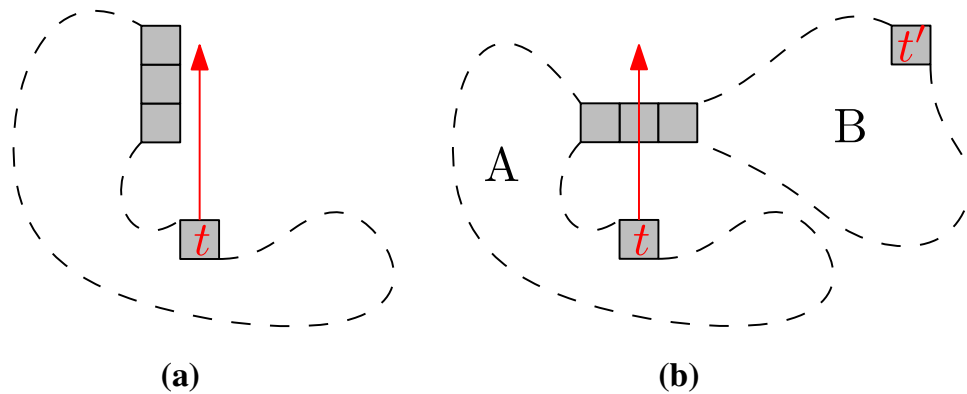


Fig. 6 Polyominoes for which no locally convex tile should be removable, showing the contradiction to t being the first blocked locally convex tile in P removed. **a** If the removal direction of t is not crossed, the last blocking tile has to be locally convex (and has to be removed before t) and **b** If the removal direction of t crosses P , then P gets split into components A and B . Component B has a locally convex tile t' that needs to be removed before t

Proof We prove this by contradiction based on two possible cases.

Assume P to be a decomposable polyomino in which no locally convex tile is removable. Because P is decomposable, there exists some feasible decomposition sequence S . Let P_{convex} denote the set of locally convex tiles of P and let $t \in P_{\text{convex}}$ be the first removed locally convex tile in the decomposition sequence S . By assumption, t cannot be removed yet, so it is either blocked or a cut tile.

t is blocked: Consider the direction in which we would remove t . If it does not cut the polyomino, the last blocking tile has to be locally convex (and would have to be removed before t), see Fig. 6a. If it cuts the polyomino, the component cut off also must have a locally convex tile and the full component has to be removed before t , see Fig. 6b. This is again a contradiction to t being the first locally convex tile to be removed in S .

t is a cut tile: $P - t$ consists of exactly two connected polyominoes, P_1 and P_2 . It is easy to see that $P_1 \cap P_{\text{convex}} \neq \emptyset$ and $P_2 \cap P_{\text{convex}} \neq \emptyset$, because every polyomino of size $n \geq 2$ has at least two locally convex tiles of which at most one ceases to be locally convex by adding t . (A polyomino of size 1 is trivial.) Before being able to remove t , either P_1 or P_2 has to be completely removed, including their locally convex tiles. This is a contradiction to t being the first locally convex tile in S to be removed. \square

3.2 An Efficient Algorithm

An iterative combination of these two lemmas proves the correctness of greedily removing locally convex tiles. As we show in the next theorem, using a search tree technique allows an efficient implementation of this greedy algorithm.

Theorem 5 *A hole-free polyomino can be checked for decomposability/constructibility in time $O(N \log N)$.*

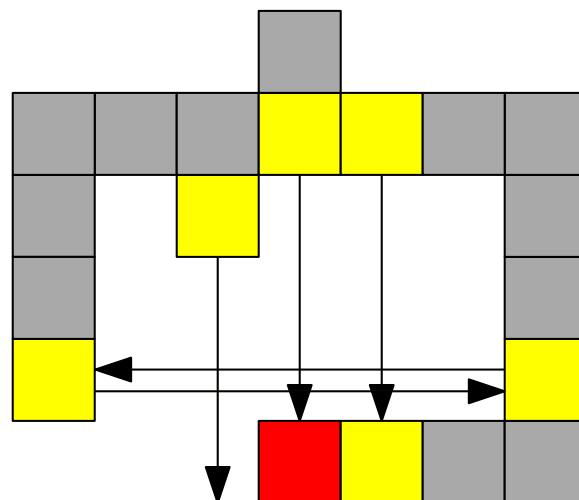
Proof Lemma 3 allows us to remove any locally convex tile, as long as it is not blocked and does not destroy connectivity. Applying the same lemma on the remaining

polyomino iteratively creates a feasible decomposition sequence. Lemma 4 proves that this is always sufficient. If and only if we can at some point no longer find a matching locally convex tile (to which we refer as *candidates*), the polyomino cannot be decomposable.

Let B be the time needed to check whether a tile t is blocked. A naïve way of doing this is to try out all tiles and check if t gets blocked, requiring time $O(N)$. With a preprocessing step, we can decrease B to $O(\log N)$ by using $O(N)$ binary search trees for searching for blocking tiles and utilizing that removing a tile can change the state of at most $O(1)$ tiles. For every vertical line x and horizontal line y going through P , we create a balanced search tree, i.e., for a total of $O(N)$ search trees. An x -search tree for a vertical line x contains tiles lying on x , sorted by their y -coordinate. Analogously define a y -search tree for a horizontal line y containing tiles lying on y sorted by their x -coordinate. We iterate over all tiles $t = (x, y)$ and insert the tile in the corresponding x - and y -search tree with a total complexity of $O(N \log N)$. Note that the memory complexity remains linear, because every tile is in exactly two search trees. To check if a tile at position (x', y') is blocked from above, we can simply search in the $(x' - 1)$ -, x' - and $(x' + 1)$ -search tree for a tile with $y > y'$. We analogously perform search queries for the other three directions, and thus have 12 queries of total cost $O(\log N)$.

We now iterate on all tiles and add all locally convex tiles that are not blocked and are not a cut tile to the set F (cost $O(N \log N)$). Note that checking whether a tile is a cut tile can be done in constant time, because it suffices to look into the local neighborhood. While F is not empty, we remove a tile from F , from the polyomino, and from its two search trees in time $O(\log N)$. Next, we check the up to 12 tiles that could have been blocked by the removed tile, see Fig. 7. Only these tiles can become unblocked or a locally convex tile. Those that are locally convex tiles, not blocked, and not a cut tile are added to F . All tiles behind those cannot become unblocked as the first tiles would still be blocking them. If one of those tiles becomes a cut tile, then we remove it from F . The cost for this is again in $O(\log N)$. This is continued until F is empty, which takes at most $O(N)$ loops each of cost $O(\log N)$. If the polyomino has been decomposed, the polyomino is decomposable/constructible by the corresponding

Fig. 7 When removing the red tile (dark gray in grayscale), only the yellow tiles (light gray in grayscale) can become unblocked or locally convex (Color figure online)



tile sequence. Otherwise, there cannot exist such a sequence. A specific start tile can be enforced by prohibiting the removal of that tile. \square

3.3 Pipelined Assembly

Given that a construction is always possible based on adding locally convex corners to a partial construction, we can argue that the heuristic idea of Manzoor et al. [23] for pipelined assembly can be formally realized for *every* constructible polyomino: We can transform the construction sequence into a spiral-shaped maze environment, as illustrated in Fig. 8. This allows it to produce D copies of P in $N + D$ cycles, implying that we only need $2N$ cycles for N copies. It suffices to use a clockwise order of four unit steps (west, north, east, south) in each cycle.

The main idea is to create a spiral in which the assemblies move from the inside to the outside. The first tile is provided by an initial south movement. After each cycle, ending with a south movement, the next seed tile of the next copy of P is added. For every direction corresponding to the direction of the next tile added by the sequence, we place a tile depot on the outside of the spiral, with a straight-line path to the location of the corresponding attachment.

Theorem 6 *Given a construction sequence $\sigma := ((d_1, l_1), \dots, (d_{N-1}, l_{N-1}))$ that constructs a polyomino P , we can construct a maze environment for pipelined tilt assembly, such that constructing D copies of P needs $O(N + D)$ unit steps. In particular, constructing one copy of P can be done in amortized time $O(1)$.*

Proof Consider the construction sequence σ , the movement sequence ζ consisting of N repetitions of the cycle (w, n, e, s) , and an injective function $m : \sigma \rightarrow \zeta$, with $m((w, \cdot)) = e$, $m((n, \cdot)) = s$, $m((e, \cdot)) = w$ and $m((s, \cdot)) = n$. We also require that $m((d_i, l_i)) = \zeta_j$ if for all $i' < i$ there is a $j' < j$ with $m((d_{i'}, l_{i'})) = \zeta_{j'}$ and j is the smallest possible. This implies that in each cycle there is at least one tile in σ mapped to one direction in this cycle.

Labyrinth construction: The main part of the labyrinth is a spiral as can be seen in Fig. 8. Consider a spiral that is making $|\zeta|$ many turns, and the innermost point q of this spiral. From q upwards, we make a lane through the spiral until we are outside the spiral. At this point we add a depot of tiles, such that after each south movement a new tile comes out of the depot (this can easily be done with bottleneck constructions as seen in Fig. 8). Then, we proceed for each turn in the spiral as follows: For the j -th turn, if $m^{-1}(\zeta_j)$ is empty we do nothing. Else if $m^{-1}(\zeta_j)$ is not empty we want to add the next tile. Let t_i be this particular tile. Then, we construct a lane in direction $-\zeta_j$, i.e., the direction from where the tile will come from, until we are outside the spiral. By shifting this line in an orthogonal direction we can enforce the tile to fly in at the correct position relating to l_i . There, we add a depot with tiles, such that the first tile comes out after $j - 1$ steps and with each further cycle a new tile comes out (this can be done by using loops in the depot, see Fig. 8). Depots, which lie on the same side of the spiral, can be shifted arbitrarily, so they do not collide. These depots can be made arbitrarily big, and

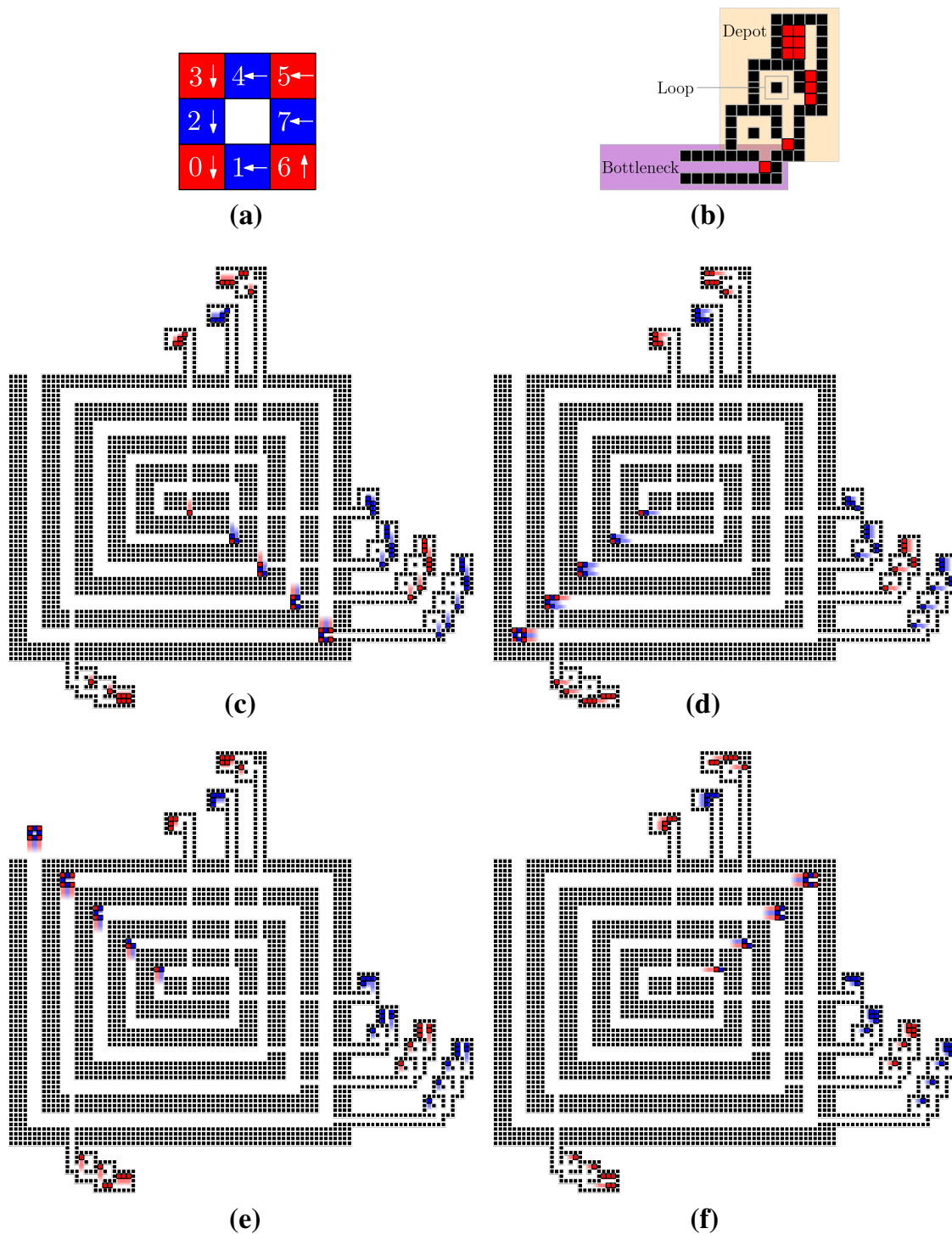


Fig. 8 **a** A polyomino P . Shown is the assembly order and the direction of attachment to the seed (tile 0). **b** A depot (orange; light gray area in grayscale) having loops to delay the tile output and a bottleneck (purple; dark gray area in grayscale) to guarantee that only one tile can move to the spiral. **c–f** A maze environment for pipelined construction of the desired polyomino P . After the fourth cycle, each further cycle produces a new copy of P . Shown states are after a sequence of down (**c**), left (**d**), up (**e**) and right (**f**) moves (Color figure online)

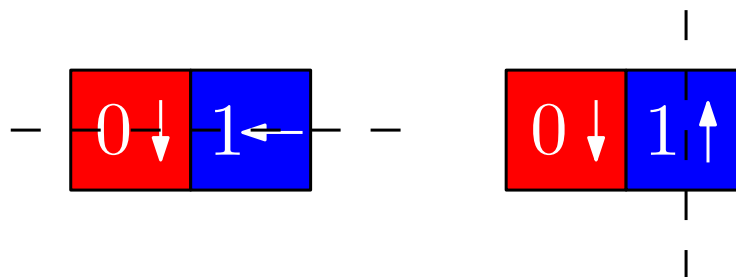


Fig. 9 Two different sequences. The red tile represents the bounding box of the current polyomino. (Left) A desired sequence. The latitude intersects the bounding box. (Right) A sequence where the latitude does not intersect the bounding box (Color figure online)

thus, we can make as many copies of P as we wish. Note that we can make the paths in the spiral big enough, such that after every turn the bounding box of the current polyomino fits through the spiral.

Correctness: We will now show that we will obtain copies of P . Consider any j -th turn in the spiral, where the i -th tile t_i is going to be added to the current polyomino. With the next step, both t_i and the polyomino move in direction ζ_j . While the polyomino does not touch the next wall in the spiral, the distance between t_i and the polyomino will not decrease. However when the polyomino hits the wall, the polyomino stops moving and t_i continues moving towards the polyomino. Wall-hitting is the same situation as in our non-parallel model: To a fixed polyomino we can add tiles from n , e , s or w . Therefore, the tile connects to the correct place. Since this is true for any tile and any copy, we conclude that every polyomino we build is a copy of P .

Time: Since the spiral has at most $4N$ unit steps (or N cycles), the first polyomino will be constructed after $4N$ unit steps. By construction, we began the second copy one cycle after beginning the first copy, the third copy one cycle after the second, and so on. This means, after each cycle, when the first polyomino is constructed, we obtain another copy of P . Therefore, for D copies we need $N + D$ cycles (or $O(N + D)$ unit steps). For $D \in \Omega(N)$ this results in an amortized constant time construction for P .

Note that this proof only considers construction sequences in the following form: If a tile t_i increases the side length of the bounding box of the current polyomino, then the tile is added from a direction with a longitude/latitude, such that the longitude/latitude intersects the bounding box (see Fig. 9). In the case there is a tile, such that the longitude/latitude does not intersect the bounding box, then we can rotate the direction by $\frac{\pi}{2}$ towards the polyomino and we will have a desired construction sequence. \square

4 Optimization Variants in 2D

For polyominoes that cannot be assembled, it is natural to look for a maximum-size subpolyomino that is constructible. This optimization variant is *polyAPX*-hard, i.e.,

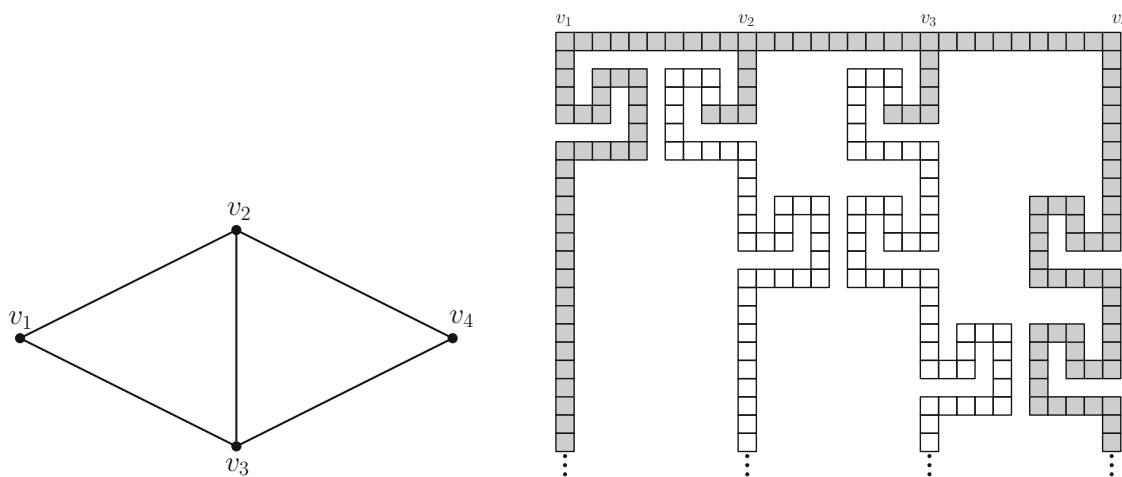


Fig. 10 Reduction from MIS to MAXTAP. (Left) A graph G with four vertices. (Right) A polyomino constructed for the reduction with a feasible, maximum solution marked in gray

we cannot hope for an approximation algorithm with an approximation factor within $\Omega(N^{\frac{1}{3}})$, unless $P = NP$.

Definition 7 (Maximum Tilt Assembly Problem). Given a polyomino P , the Maximum Tilt Assembly Problem (MAXTAP) asks for a sequence of tiles building a cardinality-maximal connected subpolyomino $P' \subseteq P$.

Theorem 8 MAXTAP is polyAPX-hard, even for tree-shaped polyominoes.

Proof We reduce MAXIMUM INDEPENDENT SET (MIS) to MAXTAP; see Fig. 10 for an illustration. Consider an instance $G = (V, E)$ of MIS, which we transform into a polyomino P_G . We construct P_G as follows. First, construct a horizontal line from which we go down to select which vertex in G will be chosen. The line must have length $10n - 9$, where $n = |V|$. Every 10th tile will represent a vertex, starting with the first tile on the line. Let t_i be such a tile representing vertex v_i . For every v_i we add a selector gadget below t_i and for every $\{v_i, v_j\} \in \delta(v_i)$ we add a reflected selector gadget below t_j , as shown in Fig. 10, each consisting of 19 tiles. Note that all gadgets for selecting vertex v_i are above the gadgets of v_j if $i < j$ and that there are at most n^2 such gadgets. After all gadgets have been constructed, we have already placed at most $19n^2 + 10n - 9 \leq 29n^2$ tiles. We continue with a vertical line with a length of $30n^2$ tiles.

Now, let α^* be an optimal solution to MIS. Then MAXTAP has a maximum polyomino of size at least $30n^2\alpha^*$ and at most $30n^2\alpha^* + 29n^2$: We take the complete vertical part of t_i for every v_i in the optimal solution of MIS. Choosing other lines block the assembly of further lines and thus, yields a smaller solution.

Now suppose we had an $N^{1-\varepsilon}$ -approximation for MAXTAP. Then we would have a solution of at least $\frac{1}{N^{1-\varepsilon}} T^*$, where T^* is the optimal solution. We know that an optimal solution has $T^* \geq 30n^2\alpha^*$ tiles and the polyomino has at most $N \leq 30n^3 + 29n^2 \leq 59n^3$ tiles. Therefore, we have at least $\frac{30n^2\alpha^*}{59^{1-\varepsilon}n^{3-3\varepsilon}}$ tiles and thus at least $\frac{1}{59^{1-\varepsilon}n^{3-3\varepsilon}}\alpha^*$ strips, because each strips is $30n^2$ tiles long. Consider some $\varepsilon \geq \frac{2}{3} + \eta$ for any $\eta > 0$, then the number of strips is $\frac{1}{59^{1/3}n^{1-3\eta}}\alpha^*$ which results in an $n^{1-\delta}$ -approximation for

MIS, contradicting the inapproximability of MIS (unless $P=NP$) shown by Berman and Schnitger [10]. \square

As a consequence of the construction, we get Corollary 9.

Corollary 9 *Unless $P = NP$, MAXTAP cannot be approximated within a factor of $\Omega(N^{\frac{1}{3}})$.*

On the positive side, we can give an $O(\sqrt{N})$ -approximation algorithm for tree-shaped polyominoes.

Theorem 10 *The longest constructible path in a tree-shaped polyomino P is a \sqrt{N} -approximation for MAXTAP, and we can find such a path in polynomial time.*

Proof Consider an optimal solution P^* and a smallest enclosing box B containing P^* . Then there must be two opposite sides of B touching at least one tile of P^* . Consider the path S between both tiles. Because (i) the area A_B of B is at least the number of tiles in P^* , (ii) $|S| \geq \sqrt{A_B}$, and (iii) a longest, constructible path in P has length at least $|S|$, we conclude that the longest constructible path is a \sqrt{N} -approximation.

To find such a path, we can search for every path between two tiles, check whether we can build this path, and take the longest, constructible path. \square

Checking constructibility for $O(N^2)$ possible paths is rather expensive. However, we can efficiently approximate the longest constructible path in a tree-shaped polyomino with the help of *sequentially* constructible paths, i.e., the initial tile is a leaf in the final path.

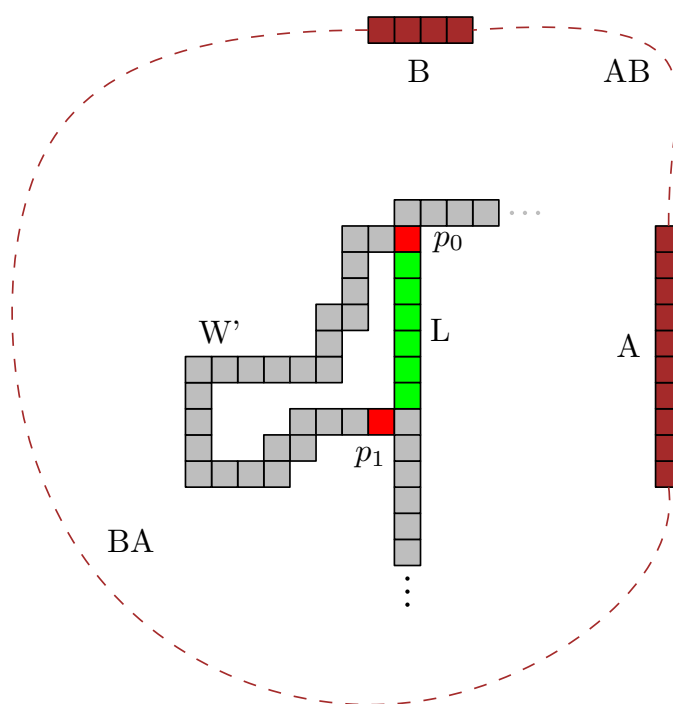
Theorem 11 *We can find a constructible path in a tree-shaped polyomino in $O(N^2 \log N)$ time that has a length of at least half the length of the longest constructible path.*

Proof We only search for paths that can be built sequentially. Clearly, the longest such path is at least half as long as the longest path that can have its initial tile anywhere. We use the same search tree technique as before to look for blocking tiles. Select a tile of the polyomino as the initial tile. Do a depth-first search and for every tile in this search, check if it can be added to the path. If it cannot be added, skip all deeper tiles, as they also cannot be added. During every step in the depth-first search, we only need to change a single tile in the search trees, doing $O(1)$ updates with $O(\log N)$ cost. As we only consider $O(N)$ vertices in the depth-first search, this results in a cost of $O(N \log N)$ for a fixed start tile. It is trivial to keep track of the longest such constructible path. Repeating this for every tile results in a running time of $O(N^2 \log N)$. \square

In tree-shaped polyominoes, finding a constructible path is easy. For simple polyominoes, additional arguments and data structures lead to a similar result.

Theorem 12 *In simple polyominoes, finding the longest of all shortest paths that are sequentially constructible takes $O(N^2 \log N)$ time.*

Fig. 11 A subpath W' and its shortcut L in green. To block L , A and B must exist. But then, either p_0 or p_1 (red tiles) will also be blocked. Therefore, also W' cannot be built (Color figure online)



Before we start with the proof of Theorem 12, we show in the next two lemmas that it is sufficient to consider shortest paths only, and that we can restrict ourselves to one specific shortest path between two tiles. Hence, we just need to test a maximum of $O(n^2)$ different paths.

Lemma 13 *In a sequentially constructible path, if there is a direct straight connection for a subpath, the subpath can be replaced by the straight connection.*

Proof Consider a sequentially constructible path W and a subpath $W' \subset W$ that has a straight line L connecting the startpoint and the endpoint of W' . W.l.o.g., L is a vertical line and we build from bottom to top. Assume that $(W \setminus W') \cup L$ is not constructible. Then at least two structures (which can be single tiles) A and B must exist, preventing us from building L . Furthermore, these structures have to be connected via a path (AB or BA , see Fig. 11). We observe that none of these connections can exist or otherwise, we cannot build W (if AB exist, we cannot build the last tile p_0 of L ; if BA exist, we cannot build the first tile p_1 of W'). Therefore, we can replace W' with L . \square

By repeating the construction of Lemma 13 we get a shortest path from tile t_1 to t_2 in the following form: Let P_1, \dots, P_k be reflex tiles on the path from t_1 to t_2 . Furthermore, for every $1 \leq i \leq k - 1$, the path from P_i to P_{i+1} is monotone. This property holds for every shortest path, or else we can use shortcuts as in Lemma 13.

Lemma 14 *If a shortest path between two tiles is sequentially constructible, then every shortest path between these two tiles is sequentially constructible.*

Proof Consider a constructible shortest path W , a maximal subpath W' that is x - y -monotone, and a bounding box B around W' . Due to the L_1 -metric, any x - y -monotone path within B is as long as W' . Suppose some path within B is not constructible. Then

we can use the same blocking argument as in Lemma 13 to prove that W' cannot be constructible as well, contradicting that W is constructible. \square

Using Lemmas 13 and 14, we are ready to prove Theorem 12.

Proof of Theorem 12 Because it suffices to check one shortest path between two tiles, we can look at the BFS tree from each tile and then proceed like we did in Theorem 11. Thus, for each tile we perform a BFS in time $O(N)$ and a DFS with blocking look-ups in time $O(N \log N)$, which results in a total time of $O(N^2 \log N)$. \square

5 Three-Dimensional Shapes

An interesting and natural generalization of TAP is to consider three-dimensional shapes, i.e., polycubes. The local considerations for simply connected two-dimensional shapes are no longer sufficient. In the following we show that deciding whether a polycube is constructible is *NP*-hard. Moreover, it is *NP*-hard to check whether there is a constructible path from a start cube s to an end cube t in a partial shape.

As a stepping stone, we start with a restricted version of the three-dimensional problem.

Theorem 15 *It is NP-hard to decide if a polycube can be built by inserting tiles only from above, north, east, south, and west.*

Proof We prove hardness by a reduction from 3SAT. A visualization for the formula $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee x_3 \vee x_4)$ can be seen in Fig. 12. It consists of two layers of interest (and some further auxiliary ones for space and forcing the seed tile by using the one-way gadget shown in Fig. 14). Due to the one-way gadget, at least part of the top layer [crosshatched area in Fig. 12, details in Fig. 13 (Left)] must be built first. Forcing a specific start tile can be done by a simple construction. For each variable we have to choose to block the left (for assigning *true*) or the right (for assigning *false*) part of the lower layer. In the end, the remaining parts of the upper layer can trivially be filled from above. The blocked parts of the lower layer then have to be built with only inserting tiles from east, south, or west. In the end, the non-blocked parts can be filled in from above. For each clause we use a part [as shown in Fig. 13 (Right)] that allows only at most two of its three subparts to be built from the limited insertion directions. We attach these subparts to the three variable values not satisfying the clause, i.e., the negated literals. This forces us to leave at least one negated literal of the clause unblocked, and thus at least one literal of the clause to be *true*. Overall, this allows us to build the blocked parts of the lower layers only if the blocking of the upper level corresponds to a satisfying assignment. If we can build the *true* and the *false* parts of a variable in the beginning, any truth assignment for the variable is possible.

It is straightforward to see that the whole construction fits into a bounding box of size $O(|C|) \times O(|V|) \times O(1)$, where C is the set of all clauses and V the set of all variables. \square

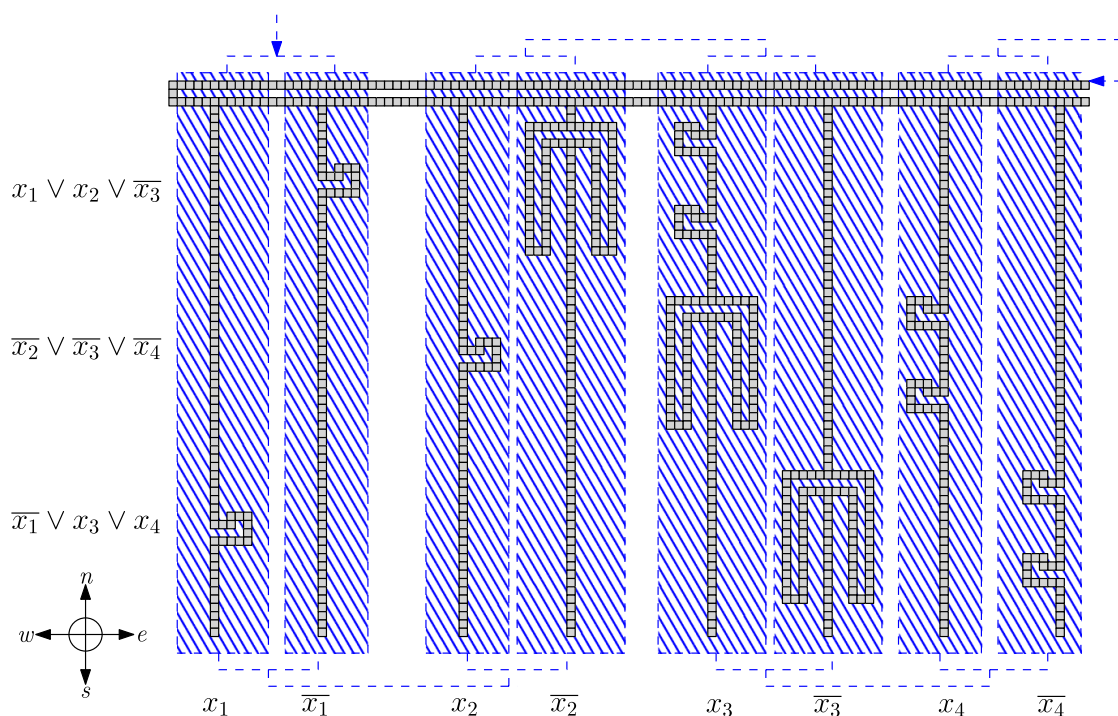


Fig. 12 Top-view on the polycube. There is a vertical part going south for the *true* and *false* assignment of each variable. We start building at the top layer (crosshatched area) and have to block either the *true* or the *false* part of each variable from above. The blocked parts have to be built with only inserting from east, west, and south. For each clause, the parts of the inverted literals are modified to allow at most two of them being built in this way. All other parts can simply be inserted from above in the end

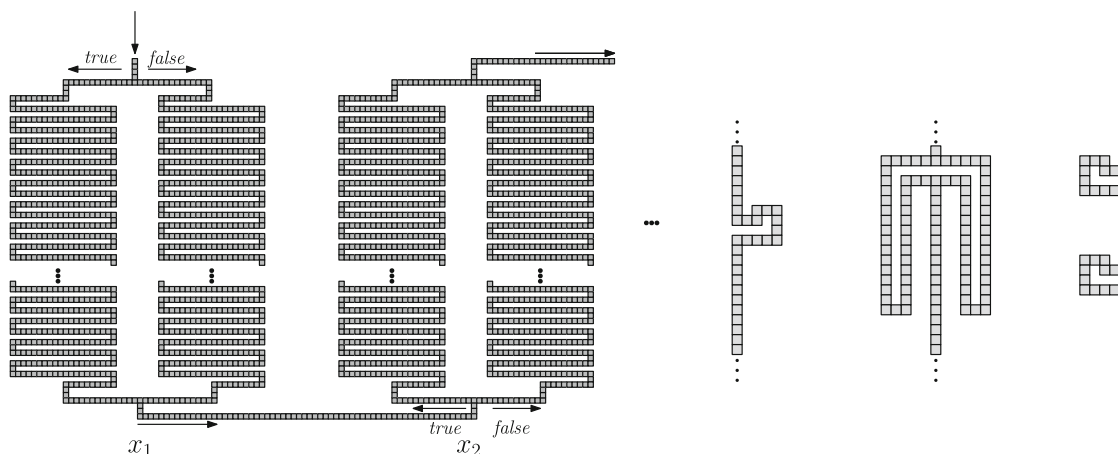


Fig. 13 Top-view on the polycube. (Left) In the beginning we have to block the access from the top for either the *true* or *false* part of the variable. The variable is assigned the blocked value. (Right) Three gadgets for a clause. Only two of them can be built if the tiles are only able to come from the east, south, and west

The construction can be extended to assemblies with arbitrary direction.

Theorem 16 *It is NP-complete to decide if a polycube can be built by inserting tiles from any direction.*

Proof We add an additional layer below the construction in Theorem 15 that has to be built first and blocks access from below. Forcing the bottom layer to be built first

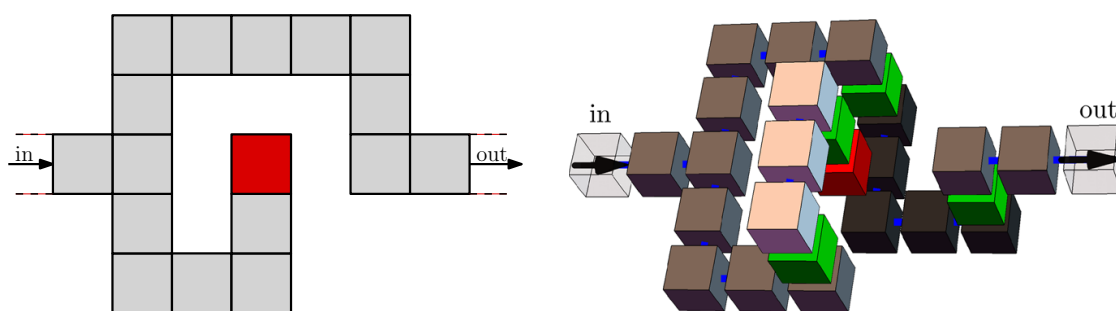


Fig. 14 (Left) This polyomino can only be constructed by starting at “in” and ending at “out”. (Right) Generalization to three dimensions. If we start on the right side, then we cannot build the red cube because it is blocked from all six directions. With these gadgets we can enforce a seed tile (Color figure online)

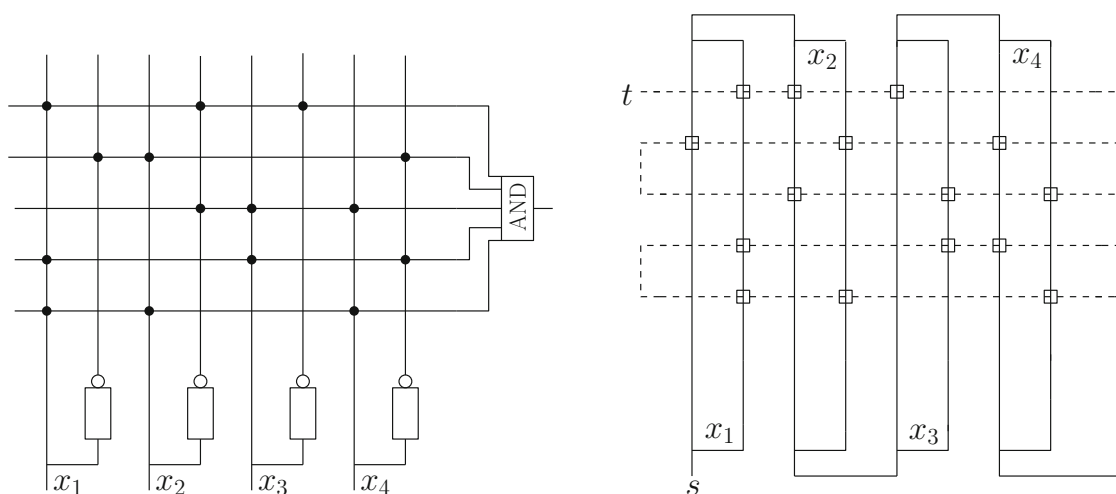


Fig. 15 (Left) Circuit representation for the SAT formula $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee x_2 \vee x_4)$. (Right) Reduction from SAT formula. Boxes represent variable boxes

can again be done with the one-way gadget shown in Fig. 14. Finally, we note that the problem of deciding whether a polycube can be built by inserting tiles from any direction is in *NP*. \square

The difficulties of construction in 3D are highlighted by the fact that even identifying constructible connections between specific positions is *NP*-hard.

Theorem 17 *It is NP-complete to decide whether a path from one tile to another can be built in a general polycube.*

Proof We prove *NP*-hardness by a reduction from SAT. For each variable we have two vertical lines, one for the *true* setting, one for the *false* setting. Each clause gets a horizontal line and is connected with a variable if it appears as literal in the clause, see Fig. 15 (Left). We transform this representation into a tour problem where, starting at a point *s*, one first has to go through either the *true* or *false* line of each variable and then through all clause lines, see Fig. 15 (Right). The clause part is only passable if the path in at least one crossing part (squares) does not cross, forcing us to satisfy at

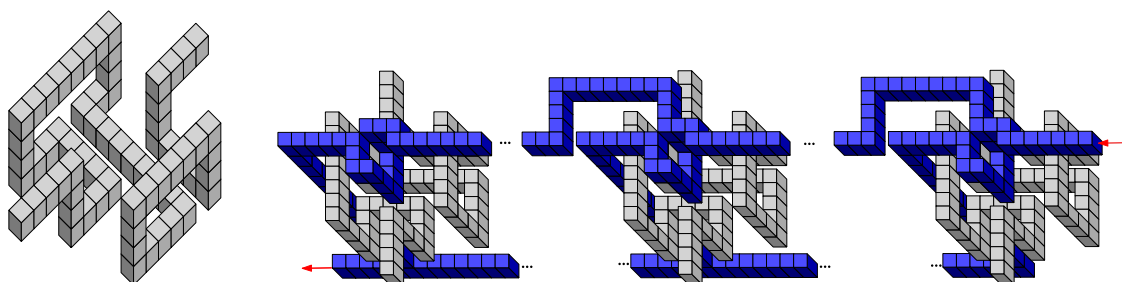


Fig. 16 (Left) Empty variable box. (Right) A clause line (blue, dark gray in grayscale) dips into a variable box. If the variable box is built, then we cannot build the dip of the clause line (Color figure online)

least one literal of a clause. As one has to go through all clauses, t is only reachable if the selected branches for the variables equal a satisfying variable assignment for the formula.

We now consider how to implement this as a polycube. The only difficult part is to allow a constructible clause path if there is a free crossing. In Fig. 16 (Left), we see a variable box that corresponds to the crossing of the variable path at the squares in Fig. 15 (Right). It blocks the core from further insertions. The clause path has to pass at least one of these variable boxes in order to reach the other side. See Fig. 15 (Right) for an example. Note that the corresponding clause parts can be built by inserting only from above and below, so there are no interferences. \square

6 Conclusion/Future Work

We have provided a number of algorithmic results for Tilt Assembly. Various unsolved challenges remain. What is the complexity of deciding TAP for non-simple polyominoes? While Lemma 4 can be applied to all polyominoes, we cannot simply remove any locally convex tile. Can we find a constructible path in a general polyomino from a given start and endpoint? This would help in finding a \sqrt{N} -approximation for non-simple polyominoes. How can we optimize the total makespan for constructing a shape? And what options exist for non-constructible shapes?

An interesting approach may be to consider *staged* assembly, as shown in Fig. 17, where a shape gets constructed by putting together subpolyominoes, instead of adding

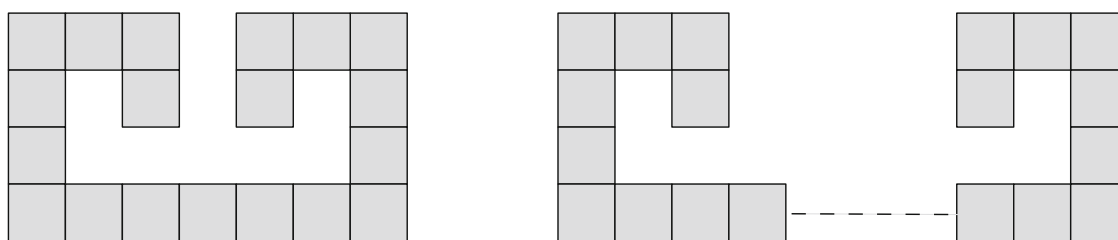


Fig. 17 (Left) A polyomino that cannot be constructed in the basic TAP model. (Right) Construction in a staged assembly model by putting together subpolyominoes

one tile at a time. This is similar to staged tile self-assembly [12,14,16]. This may also provide a path to sublinear assembly times, as a hierarchical assembly allows massive parallelization. We conjecture that a makespan of $O(\sqrt{N})$ for a polyomino with N tiles can be achieved.

All this is left to future work.








References

1. Adleman, L., Cheng, Q., Goel, A., Huang, M.-D.: Running time and program size for self-assembled squares. In: Proceedings of the ACM Symposium on Theory of Computing (STOC), pp. 740–748 (2001)
2. Arbuckle, D., Requicha, A.A.: Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations. *Auton. Robots* **28**(2), 197–211 (2010)
3. Becker, A., Fekete, S.P., Keldenich, P., Konitzny, M., Lillian, L., Scheffer, C.: Coordinated motion planning: the video. In: Proceedings of the Symposium on Computational Geometry (SoCG), pp. 74:1–74:5 (2018). Video available at <http://www.computational-geometry.org>
4. Becker, A.T., Demaine, E.D., Fekete, S.P., Habibi, G., McLurkin, J.: Reconfiguring massive particle swarms with limited, global control. In: Proceedings of the International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS), pp. 51–66 (2013)
5. Becker, A.T., Demaine, E.D., Fekete, S.P., McLurkin, J.: Particle computation: designing worlds to control robot swarms with only global signals. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 6751–6756 (2014)
6. Becker, A.T., Ertel, C., McLurkin, J.: Crowdsourcing swarm manipulation experiments: a massive online user study with large swarms of simple robots. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 2825–2830 (2014)
7. Becker, A.T., Felfoul, O., Dupont, P.E.: Simultaneously powering and controlling many actuators with a clinical MRI scanner. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2017–2023 (2014)
8. Becker, A.T., Felfoul, O., Dupont, P.E.: Toward tissue penetration by MRI-powered millirobots using a self-assembled Gauss gun. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 1184–1189 (2015)
9. Becker, A.T., Habibi, G., Werfel, J., Rubenstein, M., McLurkin, J.: Massive uniform manipulation: controlling large populations of simple robots with a common input signal. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 520–527 (2013)
10. Berman, P., Schnitger, G.: On the complexity of approximating the independent set problem. *Inf. Comput.* **96**(1), 77–94 (1992)
11. Cannon, S., Demaine, E.D., Demaine, M.L., Eisenstat, S., Patitz, M.J., Schweller, R., Summers, S.M., Winslow, A.: Two hands are better than one (up to constant factors). In: Proceedings of the International Symposium on Theoretical Aspects of Computer Science (STACS), pp. 172–184 (2013)
12. Chalk, C., Martinez, E., Schweller, R., Vega, L., Winslow, A., Wylie, T.: Optimal staged self-assembly of general shapes. In: Proceedings of the European Symposium on Algorithms (ESA), pp. 26:1–26:17 (2016)
13. Chen, H.-L., Doty, D.: Parallelism and time in hierarchical self-assembly. *SIAM J. Comput.* **46**(2), 661–709 (2017)
14. Demaine, E.D., Demaine, M.L., Fekete, S.P., Ishaque, M., Rafalin, E., Schweller, R.T., Souvaine, D.L.: Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. *Nat. Comput.* **7**(3), 347–370 (2008)
15. Demaine, E.D., Fekete, S.P., Keldenich, P., Meijer, H., Scheffer, C.: Coordinated motion planning: reconfiguring a swarm of labeled robots with bounded stretch. In: Proceedings of the Symposium on Computational Geometry (SoCG), pp. 29:1–29:15 (2018)
16. Demaine, E.D., Fekete, S.P., Scheffer, C., Schmidt, A.: New geometric algorithms for fully connected staged self-assembly. *Theor. Comput. Sci.* **671**, 4–18 (2017)

17. Derakhshandeh, Z., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: 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), p. 21 (2015)
18. Derakhshandeh, Z., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: Universal coating for programmable matter. *Theor. Comput. Sci.* **671**, 56–68 (2017)
19. Hoffmann, M.: Motion planning amidst movable square blocks: Push-* is NP-hard. In: Canadian Conference on Computational Geometry, pp. 205–210 (2000)
20. Kim, P.S.S., Becker, A.T., Ou, Y., Julius, A.A., Kim, M.J.: Imparting magnetic dipole heterogeneity to internalized iron oxide nanoparticles for microorganism swarm control. *J. Nanopart. Res.* **17**(3), 1–15 (2015)
21. Kim, P.S.S., Becker, A.T., Ou, Y., Kim, M.J., et al.: Swarm control of cell-based microrobots using a single global magnetic field. In: Proceedings of the International Conference on Ubiquitous Robotics and Ambient Intelligence (URAI), pp. 21–26 (2013)
22. Mahadev, A.V., Krupke, D., Reinhardt, J.-M., Fekete, S.P., Becker, A.T.: Collecting a swarm in a grid environment using shared, global inputs. In: Proceedings of the IEEE International Conference on Automation Science and Engineering (CASE), pp. 1231–1236 (2016)
23. Manzoor, S., Sheckman, S., Lonsford, J., Kim, H., Kim, M.J., Becker, A.T.: Parallel self-assembly of polyominoes under uniform control inputs. *IEEE Robot. Autom. Lett.* **2**(4), 2040–2047 (2017)
24. Rothmund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares (extended abstract). In: Proceedings of the ACM Symposium on Theory of Computing (STOC), pp. 459–468 (2000)
25. Rubenstein, M., Cabrera, A., Werfel, J., Habibi, G., McLurkin, J., Nagpal, R.: Collective transport of complex objects by simple robots: theory and experiments. In: Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 47–54 (2013)
26. Shad, H.M., Morris-Wright, R., Demaine, E.D., Fekete, S.P., Becker, A.T.: Particle computation: device fan-out and binary memory. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 5384–5389 (2015)
27. Shahrokhi, S., Becker, A.T.: Stochastic swarm control with global inputs. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 421–427 (2015)
28. Thubagere, A.J., Li, W., Johnson, R.F., Chen, Z., Doroudi, S., Lee, Y.L., Izatt, G., Wittman, S., Srinivas, N., Woods, D., Winfree, E., Qian, L.: A cargo-sorting DNA robot. *Science* **357**(6356), eaan6558 (2017)
29. Werfel, J., Nagpal, R.: Extended stigmergy in collective construction. *IEEE Intell. Syst.* **21**(2), 20–28 (2006)
30. Werfel, J., Nagpal, R.: Three-dimensional construction with mobile robots and modular blocks. *Int. J. Robot. Res.* **27**(3–4), 463–479 (2008)
31. Winfree, E.: Algorithmic self-assembly of DNA. Ph.D. thesis, California Institute of Technology (1998)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Aaron T. Becker¹  · Sándor P. Fekete²  · Phillip Keldenich²  ·
 Dominik Krupke²  · Christian Rieck²  · Christian Scheffer²  ·
 Arne Schmidt² 

✉ Arne Schmidt
 arne.schmidt@tu-bs.de

Aaron T. Becker
 atbecker@uh.edu

Sándor P. Fekete
 s.fekete@tu-bs.de

Phillip Keldenich
p.keldenich@tu-bs.de

Dominik Krupke
d.krupke@tu-bs.de

Christian Rieck
c.rieck@tu-bs.de

Christian Scheffer
c.scheffer@tu-bs.de

¹ Department of Electrical and Computer Engineering, University of Houston, Houston, USA

² Department of Computer Science, TU Braunschweig, Braunschweig, Germany

Efficient Parallel Self-Assembly Under Uniform Control Inputs

Arne Schmidt , Sheryl Manzoor , Li Huang , Aaron T. Becker , and Sándor P. Fekete 

Abstract—We prove that by successively combining subassemblies, we can achieve *sublinear* construction times for “staged” assembly of microscale objects from a large number of tiny particles, for vast classes of shapes; this is a significant advance in the context of programmable matter and self-assembly for building high-yield microfactories. The underlying model has particles moving under the influence of uniform external forces until they hit an obstacle; particles bond when forced together with a compatible particle. Previous work considered sequential composition of objects, resulting in construction time that is *linear* in the number N of particles, which is inefficient for large N . Our progress implies critical speedup for constructible shapes; for convex polyominoes, even a *constant* construction time is possible. We also show that our construction process can be used for pipelining, resulting in an *amortized constant* production time.

Index Terms—Computational geometry, underactuated robots, additive manufacturing.

I. INTRODUCTION

THE new field of programmable matter gives rise to a wide range of algorithmic questions of geometric flavor. One of the tasks is designing and running efficient production processes for tiny objects with given shape, without being able to individually handle the potentially huge number of particles from which it is composed, e.g., building polyominoes from their tiles without the help of tools.

In this letter we use particles that can be controlled by a uniform external force, causing all particles to move in a given direction until they hit an obstacle or another blocked particle, as shown in Fig. 1.

Recent experimental work by Manzoor *et al.* [11] showed this is practical for simple “sticky” particles, enabling assembly by

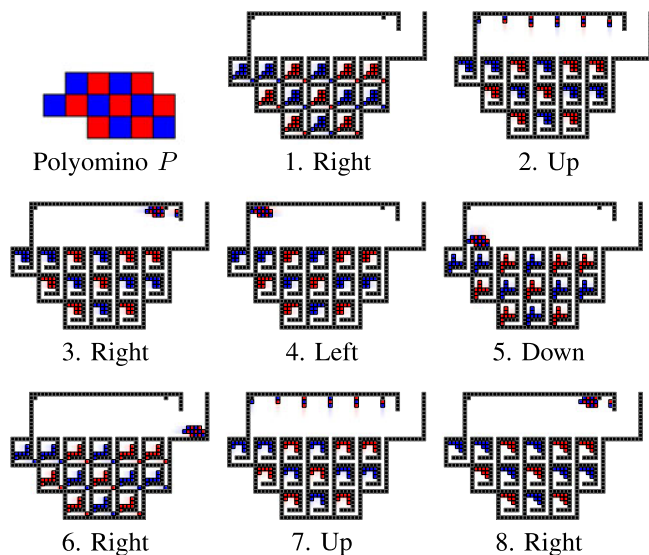


Fig. 1. Convex polyominoes can be assembled in six movement steps. A copy of the polyomino P is released every five steps after the first copy. See video attachment for animation: https://youtu.be/_R_puO0smPs.

sequentially attaching particles emanating from different depots within the workspace or supply channels from the outside to the existing subassembly, as shown in Fig. 1.

The algorithmic challenge is to design the surrounding “maze” environment and movement sequence to produce a desired shape.

A recent paper by Becker *et al.* [3] showed that the decision problem of whether a simple polyomino can be built or not is solvable in polynomial time. However, this relies on sequential construction in which one particle at a time is added, resulting in *linear* number of assembly steps, i.e., a time that grows proportional to the number N of particles, which is inefficient for large N . In this letter we provide substantial progress by developing methods that can achieve *sublinear* and in some cases even *constant* construction times. Our approaches are based on hierarchical, “staged” processes, in which we allow multi-tile subassemblies to combine at each construction step.

A. Contribution

We provide a number of contributions to achieving *sublinear* construction times for polyomino shapes consisting of N pixels (“tiles”), which is critical for the efficient assembly of large objects. Many of these results are the outcome of decomposing the shape into simpler pieces; as a consequence, we can

Manuscript received February 23, 2018; accepted June 30, 2018. Date of publication July 9, 2018; date of current version August 2, 2018. This letter was recommended for publication by Associate Editor F. van der Stappen and Editor N. Amato upon evaluation of the reviewers’ comments. The work of S. Manzoor, L. Huang, and A. T. Becker was supported by National Science Foundation under Grants IIS-1553063 and IIS-1619278. (Corresponding author: Aaron T. Becker.)

A. Schmidt and S. P. Fekete are with the Department of Computer Science, TU Braunschweig, Braunschweig 38106, Germany (e-mail: s.fekete@tu-bs.de; arne.schmidt@tu-bs.de).

S. Manzoor, L. Huang, and A. T. Becker are with the Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77004-4005 USA (e-mail: smanzoor2@uh.edu; lhuang21@uh.edu; atbecker@uh.edu).

This letter has supplemental downloadable multimedia material available at <http://ieeexplore.ieee.org>, provided by the authors. The Supplementary Materials contain a video starting with animations of parallel, “staged” assembly of micro-scale objects from tiny particles. The model has particles moving under the influence of uniform external forces until they hit an obstacle; particles bond when forced together with a compatible particle. This material is 8.58 MB in size.

Digital Object Identifier 10.1109/LRA.2018.2853758

describe the construction time in geometric parameters that may be considerably smaller than N .

- We show that we can decide if a given polyomino P can be recursively constructed from simple subpieces that are glued together along simple straight cuts (“2-cuts”) in polynomial time. The resulting production time depends on the number $r(P)$ of *locally reflex* tiles of P , which is bounded by N , but may be much smaller.
- We show that building a convex polyomino takes $O(1)$ steps.
- For a monotone polyomino P , we need $O(\log d(P))$ steps, where $d(P) \leq N$ is the number of cuts needed to decompose P into convex subpolyominoes.
- For polyominoes with convex holes, we show that $O(r)$ steps suffice to build the polyomino.
- All methods we describe can be pipelined resulting in an amortized constant construction time.

We also elaborate the running time for efficiently *computing* aspects of the decomposition, as follows. Finding cuts for a decomposition needs $O(N)$ time for monotone polyominoes. Simple polyominoes require $O(N + r^2 \log r)$ time to find a straight cut and $O(r^2 N \log N)$ time to find an arbitrary cut. Allowing convex holes increases the time to $O(N + r^3 \log r)$ and $O(r^3 N \log N)$, respectively.

For all these constructions, we show that $N \cdot (\mathcal{C}_P + \sqrt{D})$ obstacles suffice to construct D copies of an N -tile polyomino that requires \mathcal{C}_P steps to build.

B. Related Work

In recent years, the problem of assembling a polyomino has been studied intensively using various theoretical models. Winfree [14] introduced the *abstract tile self-assembly model* in which tiles with glues on their side can attach to each other if their glue type matches. Then, starting with a *seed-tile*, the tiles continuously attach to the partial assembly. If no further tile can attach, the process stops. Several years later, Cannon *et al.* [5] introduced the *2-handed tile self-assembling model* (2HAM) in which sub-assemblies can attach to each other provided that the sum of glue strengths is at least a threshold τ . Chen and Doty [7] introduced a similar model: the *hierarchical tile self-assembling model*. In 2008, Demaine *et al.* [8] introduced the *staged tile self-assembly model* which is based on the 2HAM. Here, sub-assemblies grow in various bins which can then poured together to gain new assemblies. This model was then further analyzed by Demaine *et al.* [9] and Chalk *et al.* [6]. An interesting aspect in all models is that the third dimension can be used to reach specific positions within partial assemblies. In our paper however, the challenge is to use two dimensions, i.e., an assembly can only bond to another polyomino if the bonding site is completely visible.

All these models have in common that particles, e.g., DNA-strands, self-assemble to bigger structures. In this letter, however, the particles can only move by global controls and have one glue type on all four sides. This concept has been studied in practice using biological cells controlled by magnetic fields, see [10]. In addition, see [1]. Recent work by Zhang *et al.* [15] shows there exists a workspace a constant factor larger than the number of agents that enables complete rearrangement for a rectangle of agents.

A more related paper is the work by Manzoor *et al.* [11]. They assemble polyominoes in a pipelined fashion using global control, i.e., by completing a polyomino after each small control sequence the amortized construction time of a polyomino is constant. To find a construction sequence building the polyomino only heuristics are used. Becker *et al.* [3] show that it is possible to decide in polynomial time if a hole-free polyomino can be constructed. However, both papers consider adding one tile at a time. In this letter, we allow combining partial assemblies at each step. We are also able to pipeline this process to achieve an amortized constant production time.

The complexity of controlling robots using a global control has been studied. Becker *et al.* [2] show that it is NP-hard to decide if an initial configuration of a robot swarm in a given environment can be transformed into another configuration by only using global control but becomes more tractable if it is allowed to *design* the environment. Finding an optimal control sequence is even harder. Related work for reconfiguration of robots with local movement control include work by Walter *et al.* [13], Vassilvitskii *et al.* [12], and Butler *et al.* [4].

II. PRELIMINARIES

Workspace: A workspace \mathcal{W} is a planar grid filled with unit-square particles and fixed unit square blocks (*obstacles*). Each cell of the workspace contains either a particle, an obstacle, or the cell is *free*.

Movement step: A *movement step* is one of the four directions *up*, *right*, *down*, *left*. One movement step forces every tile or assembly to move to the specified direction until the tile/assembly is blocked by an obstacle.

Polyomino: For a set $P \subset \mathbb{Z}^2$ of N grid points in the plane, the graph G_P is the induced grid graph, in which two vertices $p_1, p_2 \in P$ are connected if they are at unit distance. Any set P with connected grid graph G_P gives rise to a *polyomino* by replacing each point $p \in P$ by a unit square centered at p , which is called a *tile*; for simplicity, we also use P to denote the polyomino when the context is clear, and refer to G_P as the dual graph of the polyomino. A polyomino is called *hole-free* or *simple* if and only if the grid graph induced by $\mathbb{Z}^2 \setminus P$ is connected. A polyomino P is *column convex* (*row convex*, resp.) if the intersection of any vertical (horizontal, resp.) line and P is connected, i.e., the polyomino is *x-monotone* (*y-monotone*, resp.). Furthermore, a polyomino P is called (*orthogonal*) *convex* if P is column and row convex.

Tiles: A tile t is a unit-square of a polyomino and also represent particles in the workspace. There are two kinds of tiles: blue and red tiles. Two tiles stick together if their color differs.

Constructibility: A polyomino P is *constructible* if there exists a workspace \mathcal{W} and a sequence σ of movement steps that produce P .

Cuts: A *cut* is an orthogonal curve moving between points of \mathbb{Z}^2 . If any intersection of a cut with the polyomino P has no turn, the cut is called *straight*. A *p-cut* is a cut that splits a polyomino P into p subpolyominoes. Furthermore, a cut is called *valid* if all induced subpolyominoes can be pulled apart into opposite directions without blocking each other. A polyomino P is called (*straight*) *2-cuttable* if there is a sequence of valid (*straight*) 2-cuts that subdivide P into monotone subpolyomi-

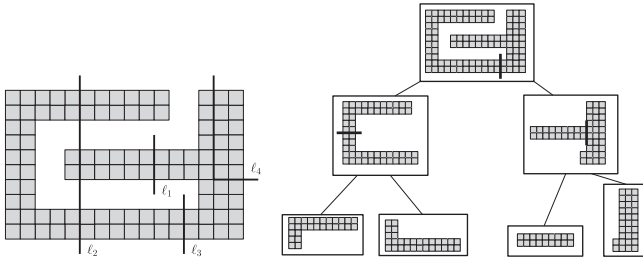


Fig. 2. Left: (Counter-)Examples for straight 2-cuts: ℓ_1 is not a 2-cut because we cannot move the left component to the right or left without getting blocked by the other component. ℓ_2 is not a 2-cut because we get more than two components. ℓ_3 is a 2-cut because we get two components which can be pulled apart. ℓ_4 is not a straight cut. Right: Decomposition tree with straight 2-cuts where the leaves are convex polyominoes.

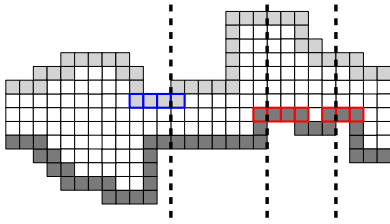


Fig. 3. A polyomino P . Light grey tiles define P_{up} , dark grey tiles define P_{1o} . Blue framed rows are minima in P_{up} , red framed rows are maxima in P_{1o} . Decomposition number $d(P) = 3$, because three vertical lines suffice and three line are necessary because every line hits a maxima/minima.

noes. If the subpolyominoes can be pulled apart in horizontal (vertical) directions, we call the cut *vertical* (*horizontal*). An example for 2-cuts can be seen in Fig. 2. In the following we only consider 2-cuts for non-convex polyominoes.

III. MONOTONE ASSEMBLIES

This section focuses on convex and monotone polyominoes.

Lemma 1: Any convex polyomino P can be assembled in six movement steps.

Proof: The idea of this proof is simple: Subdivide P into vertical lines of width one, build the lines in two steps (see Fig. 1.1 and 1.2), and connect these lines with a right and left movement (see Fig. 1.3 and 1.4). With two more movements we can flush P out of the labyrinth (see Fig. 1.5 and 1.6).

Assembling a column: To construct a column of length n , we build n containers, each below the previous. Each container releases a new tile after each left, down, right movement combination. After the right movement all n tiles move to a wall and then have the same x -coordinate. With an up movement all n tiles stick to a column once the first tile hits the top wall.

Assembling the polyomino: Assume we have built each column of the polyomino in parallel. With obstacles we can stop each column at the appropriate respective heights. A right movement combines all columns left of the column with the maximum height, and a left movement completes the assembly of P . To remove the polyomino from the assembly area we use a down, right movement. Note that the last three movements are left, down, and right, by which we start the next copy of P .

Without further precautions, a polyomino could get stuck in narrow corridors. This problem can be avoided with a simple case analysis. First, observe that the leftmost of the topmost tiles of the polyomino is blocked by an obstacle. Let t be this tile and let x_t be the corresponding x -coordinate. Also, let s be a tile stuck in a corridor having x -coordinate x_s . Only two cases can occur. (a) $x_s < x_t$: We place an additional obstacle directly where t was blocked. This forces the polyomino to stop one position earlier. (b) $x_s > x_t$: We shift every obstacle with x -coordinate higher than the corridor one unit to the right and we add an additional obstacle at the corridor end. The polyomino is then stopped by this obstacle. \square

Definition 1: Let P be an x -monotone polyomino. The decomposition number $d(P)$ is the minimum number of vertical cuts required to obtain subpolyominoes that are all convex.

The *upper envelope* $P_{up} \subset P$ consists of (1) all tiles T on the boundary that have no tiles above, and (2) tiles connecting T along the boundary. Analogously define the *lower envelope* $P_{1o} \subset P$.

We call a straight row $M = \{m_1, \dots, m_k\} \subset P_{up}$ a *minimum* of P_{up} if there are two tiles t_1 and t_2 , for which t_1 is connected to the top side of m_1 and t_2 is connected to the top side of m_k . Analogously define *maximum* for P_{1o} .

To construct an x -monotone polyomino make vertical cuts through the maxima/minima of P_{up} and P_{1o} , respectively. There are at most $d(P)$ many cuts. We now can choose a cut, such that on both subpolyominoes P' and P'' the decomposition number $d(P') \leq d(P'') \leq \frac{1}{2}d(P)$; this can be done with a median search. Repeating this procedure on each resulting subpolyomino yields a decomposition tree with depth $\log d(P)$ whose leafs are convex polyominoes.

Lemma 2: Let P be a polyomino. For each minimum and maximum M there must be a vertical cut ℓ going through M in order to decompose P into convex subpolyominoes.

Proof: Suppose we do not need such line ℓ . Let P' be a subpolyomino having a minimum M' , through which no cut is made. Consider the two tiles t_1 and t_2 as defined above. Both t_1 and t_2 must be in the same subpolyomino (because there is no cut through M'). Then, a horizontal line through t_1 and t_2 enters P' twice and therefore, P' cannot be an convex polyomino. \square

Lemma 3: Let P be an x -monotone polyomino. The decomposition number $d(P)$ and the corresponding cuts can be computed in $O(N)$ time.

Proof: Finding the minima and maxima of P_{up} and P_{1o} , respectively, can be found in $O(N)$ time by sweeping from the left boundary to the right boundary. Having the minima M_{up} and maxima M_{1o} , both in sorted order from left to right, we repeat the following procedure:

- Let $M_0 \in M_{up}$ and $M'_0 \in M_{1o}$ be the leftmost minima/maxima, resp.
- If the projection of M_0 and M'_0 to the x -axis overlaps with at least two tiles, then output a vertical line going through M_0 and M'_0 , and remove both from M_{up} and M_{1o} , resp.
- If this is not the case, output a vertical line going through the minima/maxima that ends first, and remove this minima/maxima from M_{up} or M_{1o} , resp.

This procedure costs $O(d(P))$ time. In total, this is $O(N)$ time. The correctness follows from Lemma 2. \square

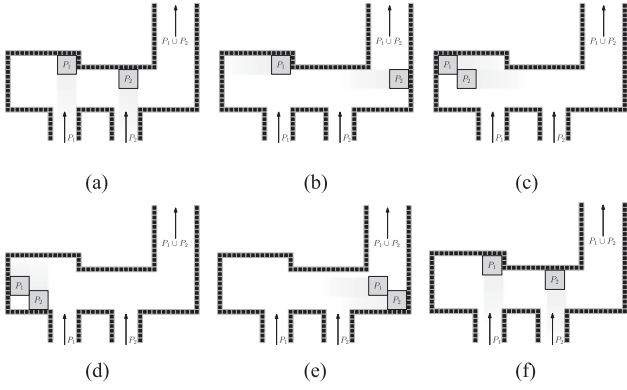


Fig. 4. Assembling two subpolyominoes P_1 and P_2 , where the topmost tile of P_1 lies above the topmost tile of P_2 . These are the same movements as seen in Fig. 1 for convex polyominoes. Thus, we can combine two subpolyominoes while constructing the next convex subpolyomino. (a) Up. (b) Right. (c) Left. (d) Down. (e) Right. (f) Up.

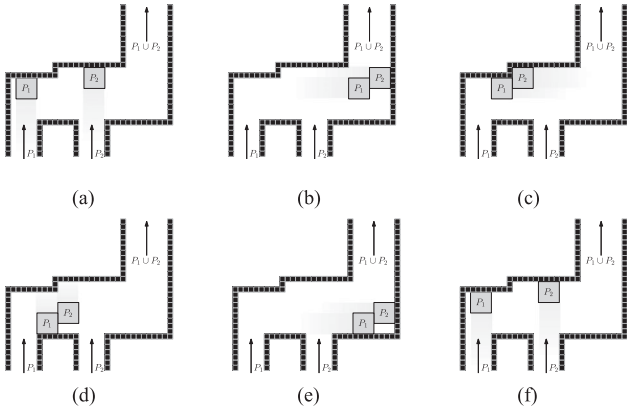


Fig. 5. Assembling two subpolyominoes P_1 and P_2 , where the topmost tile of P_2 lies above the topmost tile of P_1 . These are the same movements as seen in Fig. 1 for convex polyominoes. Thus, we can combine two subpolyominoes while constructing the next convex subpolyomino. (a) Up. (b) Right. (c) Left. (d) Down. (e) Right. (f) Up.

Theorem 1: Any x -monotone polyomino P with decomposition number $d(P) > 0$ can be assembled in $O(\lceil \log(1 + d(P)) \rceil)$ unit steps. Furthermore, this process can be pipelined yielding a construction time of amortized $O(1)$ unit steps.

Proof: As a first step we search for the vertical cuts as described above. Having this subdivision into convex subpolyominoes, we can use Lemma 1 to create all subpolyominoes in parallel. We now can use the combining gadget seen in Figs. 4 and 5 to combine two adjacent subpolyominoes in each cycle. Thus, for each cycle the number of subpolyominoes decreases by a factor of two and we have at most $\lceil \log(1 + d(P)) \rceil$ cycles to combine all subpolyominoes to obtain P .

As already described in Lemma 1, we start a new copy after every cycle. Thus, to create D copies of P we need $O(\lceil \log(1 + d(P)) \rceil + D)$ cycles. This is an amortized constant time per copy if we create $\Omega(\log d(P))$ copies. Note that $d(P)$ is in $\Omega(1)$ and $O(N)$. \square

IV. ASSEMBLING NON-MONOTONE SHAPES

In this section we show how to decide constructibility for special classes of polyominoes, namely simple polyominoes

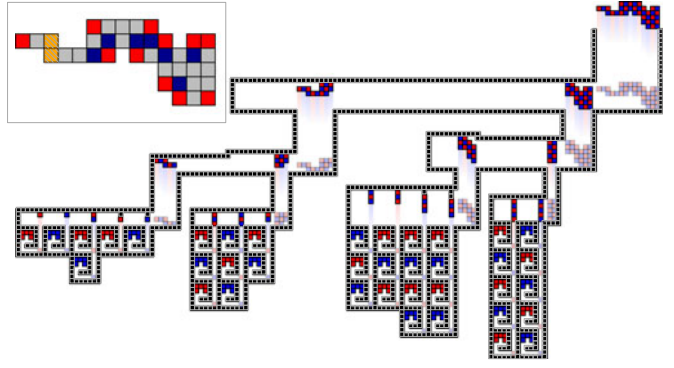


Fig. 6. A complete example constructing P with $d(P) = 3$. State shown is after an up-movement and its previous state in translucent colors. Top-left box: A polyomino P with locally convex tiles (red), locally reflex tiles (blue), and tiles that are both locally convex and locally reflex (orange, striped).

and polyominoes with convex holes. We end this section by showing how much space is needed for the workspace in which we can assemble the polyominoes.

A. Simple Polyominoes

To prove if a simple polyomino can be constructed we look at the converse process: a decomposition. As defined in the preliminary section we use 2-cuts to decompose a polyomino. If the polyomino cannot be decomposed by 2-cuts then the polyomino cannot be constructed by successively putting two subpolyominoes together. We show with the next lemma that we can greedily pick any valid straight 2-cut.

Lemma 4: Any valid straight 2-cut preserves decomposability.

Proof: Consider a straight 2-cut ℓ and a sequence $\sigma = (\ell_1, \dots, \ell_m)$ of cuts, decomposing P into single tiles. Assume ℓ is part of the cut sequence but not the first cut in σ . Then, there is a 2-cut ℓ' being made directly before ℓ in a polyomino P^* induced by cuts before ℓ' . We can now swap ℓ' and ℓ preserving their property of being 2-cuts: for ℓ we assume it is a 2-cut in P , which is also true in any subpolyomino induced by 2-cuts; the same holds for ℓ' , it is a 2-cut in P^* and thus, also in any subpolyomino induced by 2-cuts. After swapping both cuts we have the same decomposition yielding a valid decomposition of P . We can now repeat this procedure until ℓ is the first cut in P .

However, ℓ may not be in the cut sequence σ . We now show that we can use ℓ as a cut by exchanging cuts. Let ℓ_k be the last cut intersecting ℓ . This cut separates two cuts ℓ' and ℓ'' which lie on ℓ . Because ℓ is a 2-cut, also $\ell' \cup \ell''$ must be a 2-cut in the polyomino where we use cut ℓ_k . Therefore, we can first use the cut $\ell' \cup \ell''$ and then the two cuts ℓ'_k and ℓ''_k induced by the intersection of ℓ and ℓ_k . By repeating this procedure, we get ℓ as part of the cut sequence σ . \square

Definition 2: A tile t of a polyomino P is said to be *locally convex* if there exists a 2×2 square solely containing t . If the square only contains t and its two neighbors, then we call t *locally reflex*. Note that a tile can be locally convex and locally reflex at the same time (see Box in Fig. 6).

Lemma 5: Any non-convex, straight 2-cuttable polyomino P can be decomposed into convex subpolyominoes by only using straight 2-cuts cutting along a locally reflex tile.

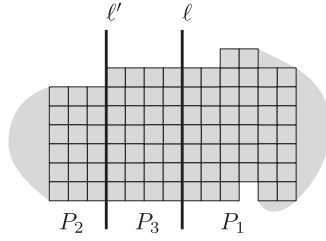


Fig. 7. The original cut ℓ and its shifted copy ℓ' , which together split the polyomino into three parts P_1 , P_2 , P_3 .

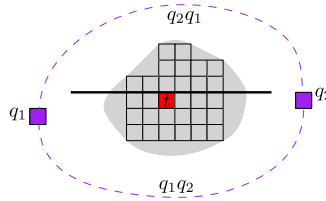


Fig. 8. A not locally convex tile t (red) in P_1 (gray area) blocked by q_1 and q_2 (purple). If the path $q_2 q_1$ exists, there is at least one blocked locally convex tile above the black bold line. If $q_1 q_2$ exists, we proceed analogously.

Proof: W.l.o.g., consider a vertical straight 2-cut ℓ that may not cut along a locally reflex tile. Then we can move a cut ℓ' to the left starting at ℓ until we reach a locally reflex tile t such that the cut goes through the corner of t that lies on the boundary of P (if we cannot reach a locally reflex tile we move ℓ' to the right). We obtain three subpolyominoes: P_1 to the right of ℓ , P_2 to the left of ℓ' , and P_3 between ℓ' and ℓ (see Fig. 7).

Assume ℓ' is not a valid 2-cut, i.e., a tile is blocked in P_2 or P_3 . (If there is a blocked tile in P_1 , then also ℓ would not be a 2-cut.) Consider the first case, where P_2 has a blocked tile t . Then, t has an y -coordinate which is at most as high as the highest tile in P_3 plus 1 and at least as high as the lowest tile in P_3 minus one (or else both blocking tiles must be in P_1 and thus, ℓ would be no 2-cut). Let $q_1 \in P_3$ to the right of t and $q_2 \in P_1 \cup P_3$ to the left of t be the two tiles blocking t . By replacing q_1 with its right neighbor we still have two tiles blocking t . Because ℓ is a 2-cut we can repeat this procedure until $q_1 \in P_1$. We can repeat the procedure for q_2 if $q_2 \in P_3$. Thus, both blocking tiles are in P_1 and ℓ cannot be a 2-cut.

For the second case the blocked tile t lies in P_3 . Then, also the right neighbor t' of t is blocked. This is also true for t'' . Therefore, we can go to the right until we reach P_1 and thus, there is a tile in P_1 which is blocked. This means, also ℓ cannot be a valid 2-cut, which is a contradiction to ℓ being a valid 2-cut.

As each cut ℓ' reduces the number of locally reflex tiles by at least one, the remaining polyominoes will be convex after a limited number of cuts. \square

Lemma 6: It is sufficient to consider locally convex tiles for checking if a cut ℓ is a valid straight 2-cut.

Proof: Assume w.l.o.g. ℓ is a vertical cut splitting the polyomino in two subpolyominoes P_1 and P_2 . W.l.o.g., consider a not locally convex tile $t \in P_1$ blocked by two tiles $q_1, q_2 \in P_2$. Because ℓ is a 2-cut and P is simple, there must be a path from q_1 to q_2 within P_2 . This path must go around P_1 either above or beneath t (see Fig. 8).

In case the path moves above t , consider a horizontal cut directly above t (see Fig. 8). This cut splits P_1 into components. In each component there are at least four locally convex tiles from which at most two became locally convex through the cut. Thus, two of these locally convex tiles were also locally convex in P_1 . It is easy to see in the figure that both locally convex tiles are also blocked by tiles on the path from q_1 to q_2 .

In the second case we proceed analogously with the difference that we use a horizontal cut directly below t . We conclude that in any case there is a locally convex tile in P_1 that is being blocked if there is a blocked, not locally convex tile. Note that the other direction may not be true. \square

Lemma 7: Checking if a 2-cut ℓ is valid can be done in $O(N + r \log r)$ time, where r is the number of locally reflex tiles.

Proof: W.l.o.g. assume ℓ to be a vertical straight cut and also assume that we are checking blue tiles only. As a first step we scan through the polyomino and search for all tiles that represent a corner, i.e., the tile is locally convex or locally reflex. Additionally, we can store the neighbor corner tiles of each corner tile (these are up to four tiles). Both steps can be done with one scan, and thus in $O(N)$ time.

Now, consider the cut ℓ splitting the polyomino into subpolyominoes P_1 and P_2 . Finding the corner tiles in P_1 and P_2 can be done in $O(r)$ time by a breadth-first search. We proceed with the following procedure for P_1 (analogously for P_2):

- 1) Get all vertical lines connecting two corner tiles in P_2 and stretch this line by one tile if a corner tile is red (this checks if a blue tile would pass a red tile).
- 2) Sort the set C_r of corner tiles in P_2 lexicographically by y -coordinate and then by x -coordinate.
- 3) Start a sweep line from bottom to top having the tiles in C_r as event points.
- 4) On each event point p do the following update:
 - If p is a start point of a vertical line but lies left of the current vertical line, remove p from C_r .
 - If p is a start point and lies to the right of the current line add the tile of the current line to C_r and jump to the new vertical line.
 - If p is an end point of the current vertical line, then jump to the nearest vertical line to the left and add the tile of this line to C_r .
 - If p is an end point but not of the current vertical line, remove p from C_r .
- 5) Repeat steps 1–4, switching left and right, to get C_l .
- 6) For each locally convex tile t in P_1 :
 - find $q_1 \in C_r$ having highest y -coordinate below t , and $q_2 \in C_r$ having lowest y -coordinate above t . (Both shall be the left-most tile in case of ties.)
 - find $q'_1 \in C_l$ having highest y -coordinate below t , and $q'_2 \in C_l$ having lowest y -coordinate above t . (Both shall be the left-most tile in case of ties.)
 - If t lies to the left of segment $q_1 q_2$ and to the right of segment $q'_1 q'_2$ return false.

This computes a left and right envelope of vertical lines in P_1 and P_2 , respectively. This allows an easy check if there is a tile on the left/right blocking a tile from P_1 in this direction (for an example, see Fig. 9).

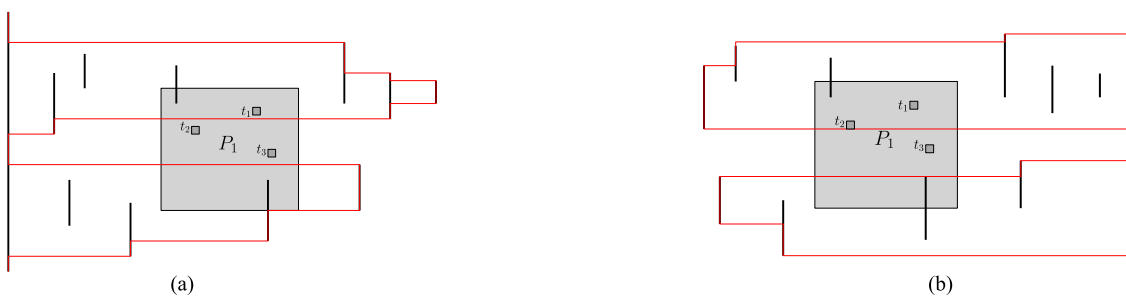


Fig. 9. Example for the data structure used in Lemma 7. We observe that t_1 is always on the wrong side of the red line and is thus blocked in both directions. Vertical lines are part of P_2 .

The runtime is in $O(r \log r)$: Step 1 needs $O(r)$ time because there are $O(r)$ corner tiles and at most two vertical lines per corner tile. Sorting a set lexicographically in two dimensions can be done in $O(r \log r)$. With a careful view on step 4, we can observe that each update of the $O(r)$ event points costs $O(\log r)$ and thus in total $O(r \log r)$ time. Step 6 can be done in $O(\log r)$ time for each locally convex tile. Therefore, we need $O(r \log r)$ time in total. \square

The next theorem is straightforward to prove.

Theorem 2: Let r be the number of locally reflex tiles. We can find a valid straight 2-cut in $O(N + r^2 \log r)$ time.

Theorem 3: A decomposition tree of valid 2-cuts for a polyomino P can be used to build a labyrinth constructing P . This labyrinth can also be used for pipelining.

Proof: Consider a cycle of the seven unit steps *right, up, down, up, right, left, down*. This is the movement sequence which was already seen for convex and monotone polyominoes but with two more movements. This cycle preserves the ability to construct monotone polyominoes in the labyrinth above. Also observe that turning the gadgets seen in Figs. 4 and 5 by 90 degrees clockwise yields gadgets that put two polyominoes on top of each other.

Transforming a decomposition tree of 2-cuts for a polyomino P can easily be done: Consider the layers of the decomposition tree, with the root being layer zero, its children being layer one, and so on. In each vertex in one layer either a horizontal or vertical cut is made. Corresponding to this cut we construct a gadget putting the two children of this vertex together. At some point only monotone subpolyominoes exist. These can be build using the methods described above.

The length of a root-leaf-path may vary. In this case we can build loops so we can put two polyominoes together at the right time. \square

Theorem 4: Any straight 2-cuttable polyomino P can be build within $O(r)$ unit steps, where r is the number of locally reflex tiles in P . D copies require $O(r + D)$ unit steps.

Proof: Doing cuts along locally reflex tiles reduces the number of locally reflex tiles by at least one. This implies a maximum depth of $O(r)$ of the decomposition tree and thus, $O(r)$ cycles to produce P . As seen before, pipelining yields a construction time of $(r + D)$ unit steps, which is an amortized constant construction time if $D \in \Omega(N)$. \square

Unfortunately, the number of locally reflex tiles r can be in $\Omega(N)$ and thus, we may need $\Omega(N)$ cuts to build the polyomino. In particular, Fig. 10 left shows an example which needs $\Omega(N)$

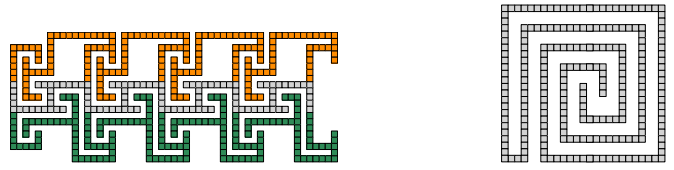


Fig. 10. Left: A polyomino needing $\Omega(N)$ steps to build as we cannot separate the green nor the orange part efficiently from the grey part. Right: Polyomino which is not 2-cuttable. Any cut splits the polyomino either in two subpolyominoes which cannot be pulled apart or into more than two subpolyominoes.

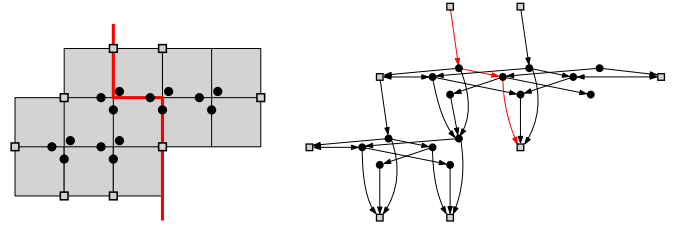


Fig. 11. A polyomino P (grey tiles) and the graph D_P (right). The vertices in V_B are shown as squares, the vertices in V_I are shown as disks. Red bold line in P is a 2-cut. Red path in the graph represents this cut.

cycles to build. Even scaling by some factor k , i.e., replacing each tile by an $k \times k$ supertile, seems not to help. Moreover, there are also polyominoes we cannot build by putting two subpolyominoes together at the same time (see Fig. 10 right).

B. Non-Straight Cuts

Considering any 2-cut makes it more difficult to find cuts, as there are exponential many possible cuts. However, we do not need to consider all cuts. For a given start s and end e on the boundary of a polyomino P , we can show that it is sufficient to consider only one cut connecting s and e . The proof is similar to the one of Lemma 5.

Theorem 5: Given a 2-cuttable polyomino P , we can find a 2-cut in time $O(r^2 N \log N)$, where r is the number of locally reflex tiles in P .

Proof: The idea of this proof is to find $O(r^2)$ 2-cuts which are then tested if they are valid. One necessary criterion is that no cut moves three units to the left or right in case of vertical cuts. This can be achieved with a directed graph D_P . As seen in Fig. 11, we add a set of $O(r)$ vertices that correspond to

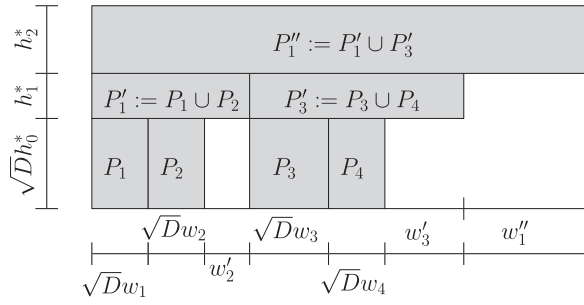


Fig. 12. Block diagram of the workspace to construct a monotone polyomino.

corners of tiles lying on the boundary of P (giving rise to the set V_B), or that correspond to corner tiles not lying on the boundary (resulting in the set V_I). We add edges between adjacent vertices with weight $\frac{1}{2N}$ if both vertices are in V_I . If both vertices are in V_B , then the edge has weight 2, otherwise 1.

A 2-cut is represented by a shortest path of weight at most 2.5 containing exactly two vertices of V_B . If we have at least three vertices of V_B in the shortest path, it has length at least 3. Thus, paths from one vertex in V_B to another vertex of V_B define cuts going through P . Finding all shortest paths from one vertex in V_B lasts $O(N \log N)$ time, as there are $O(N)$ edges in D_P . This implies a total time of $O(rN \log N)$ for finding all shortest paths of length at most 2.5.

Because one cut can make $O(N)$ turns, checking whether the cut is valid takes time $O(N \log N)$. Thus, checking all $O(r^2)$ cuts if they are valid needs time $O(r^2 N \log N)$. \square

All techniques can be generalized for polyominoes with convex holes. However, this increases the number of possible cuts to be checked. In particular, there can be $O(r_h^3)$ possible ways to go through a hole h with r_h locally reflex tiles. Thus, the time to find a cut takes $O(N + r^3 \log r)$ using straight cuts and $O(r^3 N \log N)$ using non-straight cuts.

C. Workspace Size and Number of Obstacles

Theorem 6: Let P be a polyomino. Then, the workspace needed to assemble D copies of P can be put into a rectangle of width $O(w_P \mathcal{L}_P \cdot (\mathcal{C}_P + \sqrt{D}))$ and height $O(h_P \cdot (\mathcal{C}_P + \sqrt{D}))$, where w_P and h_P are the width and height of P , \mathcal{C}_P is the number of movement steps needed, and \mathcal{L}_P is the number of cuts made to decompose P into convex subpolyominoes. Furthermore, we only need $O(N(\mathcal{L}_P + \sqrt{D}))$ obstacles in the workspace.

Proof: Represent each gadget as a block. An example block diagram shown in Fig. 12 illustrates the structure of the workspace with width and height of each stage.

Consider the decomposition tree T of P induced by cuts whose leafs are convex polyominoes. For convex polyominoes we can use the construction from Lemma 6 and for each inner node of T we use the gadgets used in Theorem 1 to combine two subpolyominoes. Let P_1, \dots, P_k be the convex polyominoes in the leafs of T with width w_1, \dots, w_k and height h_1, \dots, h_k . To construct one P_i , we need $O(\sqrt{D}w_i) \times O(\sqrt{D}h_i^*)$ space, where h_i^* is the maximum height of all P_i .

Now consider the j -th stage with $j \leq \mathcal{C}_P$ where some polyominoes are combined. Let P'_1 and P'_2 be two such

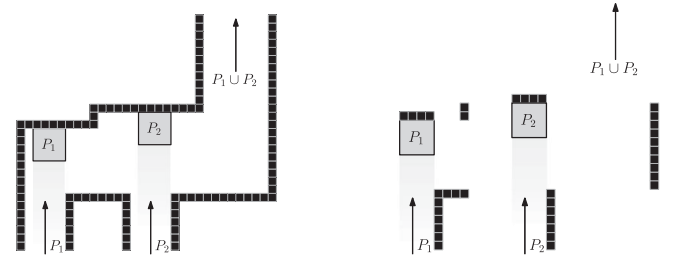


Fig. 13. Gadgets assembling two subpolyominoes. Left: With unnecessary obstacles. Right: Without unnecessary obstacles.

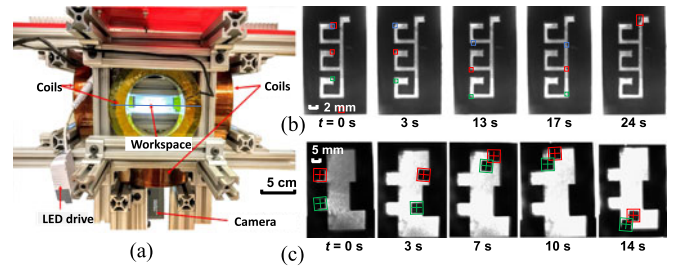


Fig. 14. (a) Magnetic manipulation workspace (b) frames from an assembly of one column of a polyomino. (c) frames from combining two polyominoes.

polyominoes. After assembling these polyominoes the width of the polyomino P''_1 increases to $w''_1 \leq w'_1 + w'_2$. Thus, the width of the workspace increases by at most $w'_1 + w'_2$. We observe that any width of P_1, \dots, P_k appears at most $\mathcal{C}_P + 1$ times. With $w_i \leq w_P$ and $k \in O(\mathcal{L}_P)$, this results in a total width of $\sum_{i=1}^{\mathcal{L}_P} w_P(\sqrt{D} + \mathcal{C}_P + 1) \in O(w_P \mathcal{L}_P(\sqrt{D} + \mathcal{C}_P))$.

For the total height consider the maximum height h_j^* of all polyominoes in stage $j \leq \mathcal{C}_P$. Because we need $O(h_j^*)$ space in the vertical direction for stage j , we have a total height of $h_P \sqrt{D} + \sum_{j=1}^{\mathcal{C}_P} h_j^* \in O(h_P(\sqrt{D} + \mathcal{C}_P))$ resulting in a rectangle of size $O(w_P(\sqrt{D} + \mathcal{C}_P)) \times O(h_P(\sqrt{D} + \mathcal{C}_P))$ enclosing the workspace.

Although the workspace may be large, the number of obstacles needed is smaller. First, ignore any obstacle not needed as a stopper (see Fig. 13). This reduces the number of obstacles to $O(w_P + h_P)$. Because $w_P, h_P \leq N$ this is $O(N)$. The same can be done for building the convex polyominoes. However, to keep the D tiles in a container we need all $O(\sqrt{D})$ obstacles. Thus, we have $O(N\sqrt{D})$ obstacles to build all convex polyominoes and $O(\mathcal{L}_P N)$ obstacles for the gadgets which is in total $O(N(\mathcal{L}_P + \sqrt{D}))$. \square

V. EXPERIMENTAL DEMONSTRATION

We implemented the algorithms for staged assembly at micro and milli scale. A customized setup was used to generate a magnetic field to manipulate the magnetic particles.

1) *Experimental Platform:* The magnetic setup used for the experiments is shown in Fig. 14, consisting of three orthogonal pairs of coils with separation distance equivalent to the outer diameter (127.5 mm) of a coil. The coils (18 AWG, 1200 turns, Custom Coils, Inc) are actuated by six SyRen10-25 motor drivers, and a Tekpower HY3020E is used for the DC power

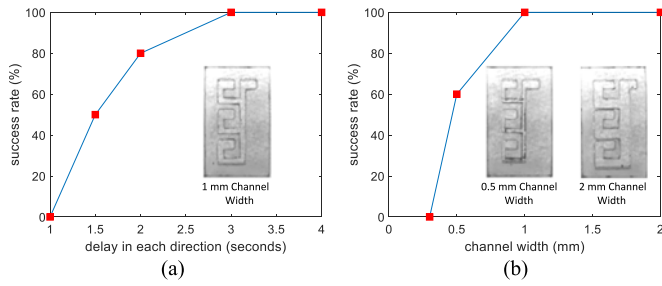


Fig. 15. Results from assembly of a micro-scale three-tile column polyomino. There are 10 trials per data point. (a) Success rate as a function of the duration of control inputs were applied in each of the four directions on a workspace with 1 mm width channels. (b) Success rate as a function of channel widths using control inputs applied for 3 s in each direction.

supply. The electromagnetic platform can provide uniform magnetic fields of up to 101 G, and gradient fields up to 150 mT/m along any horizontal direction in the center of the workspace. With flux concentration cores, up to 900 mT/m gradient fields are observed in the experiment. Each flux concentration core is a solid iron cylinder 73.1 mm in diameter.

The workspaces used to demonstrate the sublinear assembly algorithms were designed to replicate the column assembly in Fig. 1 and the subpolyomino assembly in Fig. 4. Each workspace is made up of two layers of acrylic cut using a Universal Laser Cutter. The base layer is fabricated from 2 mm thick transparent acrylic, and it is glued to 5.5 mm thick acrylic, which acts as an obstacle layout. In each experiment, the workspace is placed in the center of our electromagnetic platform. The particle tiles are composed of nickel-plated neodymium cube-shaped magnets (supermagnetman.com C0010). The magnet cubes have edge lengths of 0.5 mm for micro-scale and 2.88 mm for milli-scale demonstrations. An Arduino Mega 2560 was used to control the current in the coils and the workspaces were observed with a IEEE 1394 camera, captured at 60 fps.

2) *Experimental Results:* In micro-scale experiments, we filled the workspaces with vegetable oil and placed a magnet cube with 0.5 mm edge length in each of the three hoppers. The workspace used in these experiments was 18 mm wide and 30 mm long. To assemble the column polyomino, a gradient magnetic field of 900 mT/m was applied in the direction sequence $\langle d, r, u, l \rangle$. Each direction input was applied for a fixed amount of time specified by a MATLAB program. A successful trial requires that all three components are joined and delivered to the top right of the workspace. Fig. 14(b) shows the completed three-tile polyomino and Fig. 15 shows representative experimental results for the assembly of the column polyomino. Successful assembly depends on the channel widths and the duration of the control inputs. Larger channel widths and longer control durations led to high success rates. Trials were always successful when the magnetic field was applied at least 3 s in each direction and when the channel width was at least 1 mm.

For milli-scale demonstrations we assembled two polyominoes, as shown in Fig. 14(c). Each polyomino is composed of four magnet cubes glued together to form a square shape. The

43 mm \times 62 mm workspace was placed in a uniform, 101 G magnetic field to control the orientation of the polyominoes and then manually tilted in the direction sequence $\langle u, l, d, r, u \rangle$. See video attachment for experimental demonstrations.

VI. CONCLUSION AND FUTURE WORK

A spectrum of future work remains, most notably issues of robustness in the presence of inaccuracies, as well as the extension of our results to three-dimensional shapes. Questions in 2D include the following. Can we guarantee sublinear production times if the polyomino can be scaled by a constant? Are straight cuts sufficient, i.e., if a polyomino P is 2-cuttable, is P also straight 2-cuttable? How hard is it to decide if a polyomino cannot be built at all? Can we efficiently assemble polyomino P' that approximates P ?

REFERENCES

- [1] D. Arbuckle and A. A. Requicha, "Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: Algorithms and simulations," *Auton. Robots*, vol. 28, no. 2, pp. 197–211, 2010.
- [2] A. T. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin, "Reconfiguring massive particle swarms with limited, global control," in *Proc. Int. Symp. Algorithms Exp. Sensor Syst., Wireless Netw. Distrib. Robot.*, 2013, pp. 51–66.
- [3] A. T. Becker *et al.*, "Tilt assembly: Algorithms for micro-factories that build objects with uniform external forces," in *Proc. 28th Int. Symp. Algorithms Comput.*, 2017, vol. 92, pp. 11:1–11:13.
- [4] Z. Butler, K. Kotay, D. Rus, and K. Tomita, "Generic decentralized control for lattice-based self-reconfigurable robots," *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 919–937, 2004.
- [5] S. Cannon *et al.*, "Two hands are better than one (up to constant factors)," in *Proc. Int. Symp. Theor. Aspects Comput. Sci.*, 2013, pp. 172–184.
- [6] C. Chalk, E. Martinez, R. Schweller, L. Vega, A. Winslow, and T. Wylie, "Optimal staged self-assembly of general shapes," *Algorithmica*, vol. 80, pp. 1383–1409, 2016.
- [7] H.-L. Chen and D. Doty, "Parallelism and time in hierarchical self-assembly," *SIAM J. Comput.*, vol. 46, no. 2, pp. 661–709, 2017.
- [8] E. D. Demaine *et al.*, "Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues," *Natural Comput.*, vol. 7, no. 3, pp. 347–370, 2008.
- [9] E. D. Demaine, S. P. Fekete, C. Scheffer, and A. Schmidt, "New geometric algorithms for fully connected staged self-assembly," *Theor. Comput. Sci.*, vol. 671, pp. 4–18, 2017.
- [10] P. S. S. Kim, A. T. Becker, Y. Ou, A. A. Julius, and M. J. Kim, "Imparting magnetic dipole heterogeneity to internalized iron oxide nanoparticles for microorganism swarm control," *J. Nanoparticle Res.*, vol. 17, no. 3, pp. 1–15, 2015.
- [11] S. Manzoor, S. Sheckman, J. Lonsford, H. Kim, M. J. Kim, and A. T. Becker, "Parallel self-assembly of polyominoes under uniform control inputs," *IEEE Robot. Autom. Lett.*, vol. 2, no. 4, pp. 2040–2047, Oct. 2017.
- [12] S. Vassilvitskii, J. Kubica, E. Rieffel, J. Suh, and M. Yim, "On the general reconfiguration problem for expanding cube style modular robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2002, vol. 1, pp. 801–808.
- [13] J. E. Walter, J. L. Welch, and N. M. Amato, "Distributed reconfiguration of metamorphic robot chains," *Distrib. Comput.*, vol. 17, no. 2, pp. 171–189, Aug. 2004. [Online]. Available: <https://doi.org/10.1007/s00446-003-0103-y>
- [14] E. Winfree, "Algorithmic self-assembly of DNA," Ph.D. dissertation, California Institute of Technology, Pasadena, CA, USA, 1998. [Online]. Available: <http://cba.mit.edu/events/03.11.ASE/docs/Winfree.pdf>
- [15] Y. Zhang, X. Chen, H. Qi, and D. Balkcom, "Rearranging agents in a small space using global controls," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2017, pp. 3576–3582.

Coordinated Particle Relocation Using Finite Static Friction with Boundary Walls

Arne Schmidt¹, Victor M. Baez², Aaron T. Becker² and Sándor P. Fekete¹

Abstract—We present theoretical and practical methods for achieving *arbitrary* reconfiguration of a set of objects, based on the use of external forces, such as a magnetic field or gravity: Upon actuation, each object is pushed in the same direction until it collides with an obstruction. This concept can be used for a wide range of applications in which particles do not have their own energy supply.

A crucial challenge for achieving any desired target configuration is breaking global symmetry in a controlled fashion. Previous work made use of specifically placed barriers; however, introducing precisely located obstacles into the workspace is impractical for many scenarios. In this paper, we present a different, less intrusive method: making use of the interplay between static friction with a boundary and the external force to achieve arbitrary reconfiguration. Our key contributions are a precise *theoretical* characterization of the critical coefficient of friction that is sufficient for rearranging two particles in triangles, convex polygons, and regular polygons; a method for reconfiguring multiple particles in rectangular workspaces, and deriving *practical* algorithms for these rearrangements. Hardware experiments show the efficacy of these procedures, demonstrating the usefulness of this novel approach.

Index Terms—Manipulation Planning, Underactuated Robots

I. INTRODUCTION

RECONFIGURING a large set of objects in a prespecified manner is a fundamental task for a large spectrum of applications, including swarm robotics, smart materials and advanced manufacturing. In many of these scenarios, the involved items are not equipped with individual motors or energy supplies, so actuation must be performed from the outside. Moreover, reaching into the workspace to manipulate individual particles of an arrangement is often impractical or even impossible; instead, global external forces (such as gravity or a magnetic force) may be have to employed, targeting each object in the same, uniform manner. These limitations of individual navigation apply even in scenarios of swarm robotics: For example, the well-known kilobots do have individual actuation and energy supply, but often make use of an external light source for navigation [14]; as a consequence, directing a swarm of kilobots by switching on a light beacon works just like activating an external force. This concept of global

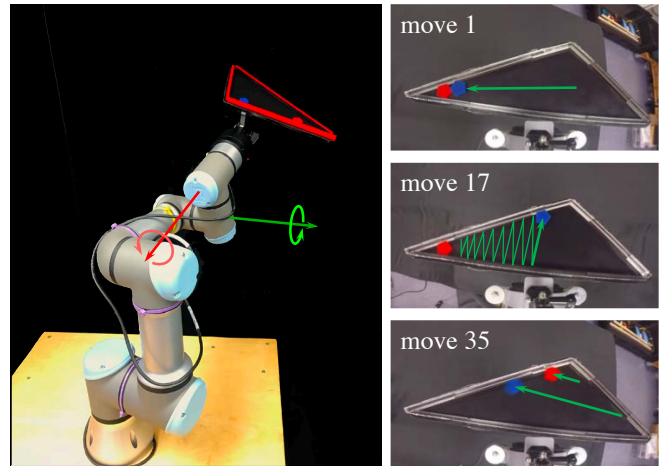


Fig. 1. (Left) Using a robotic apparatus to impose a global force on a configuration of particles. (Right) A reconfiguration sequence that combines global force and local friction to achieve arbitrary repositioning of particles.

control has also been studied for using biological cells as reactive robots controlled by magnetic fields, see Arbuckle and Requicha [3] and Kim et al. [10]. Global control also has applications in assembling nano- and micro-structures. Related work shows how to assemble shapes by adding one particle at a time [7], [4], or combining multiple pairs of subassemblies in parallel in one time step [16].

Considering this approach of navigation by a global external force gives rise to a number of problems, including navigation of one particle from a start to a goal position [11], particle computation [5], [6], or emptying a polygon [2]. Zhang et al. [19], [20] show how to rearrange a rectangle of agents in a workspace that is only constant times larger than the number of agents. Akella et al. [1] consider the problem of reconfiguring an object on a conveyor belt with a simple robot, and Lynch et al. [12] use a mobile robot with a flat pusher plate as the gripper to manipulate objects.

A crucial issue for all these tasks is how to combine the use of a uniform force (which is the same for all involved items) with the individual requirements of object relocation (which may be distinct for different particles): How can we achieve an *arbitrary* arrangement of particles if all of them are subjected to the same external force? Previous work (such as [6]) has shown how arbitrary reconfiguration of an ensemble is possible with the help of specifically placed barriers; however, introducing precisely located obstacles into the workspace is impractical for many scenarios. In this paper, we present a different, less intrusive method: making use of the interplay

Manuscript received: Sept. 19, 2019; Revised Dec. 10, 2019; Accepted Jan. 7, 2020.

This paper was recommended for publication by Editor Dezhen Song upon evaluation of the Associate Editor and Reviewers' comments.

¹Department of Computer Science, TU Braunschweig, 38106 Braunschweig, Germany {s.fekete, arne.schmidt}@tu-bs.de

²Department of Electrical & Computer Engineering, University of Houston, USA. This work was supported by National Science Foundation IIS-1553063 and IIS-1619278. {vjmontan, atbecker}@uh.edu

Digital Object Identifier (DOI): see top of this page.

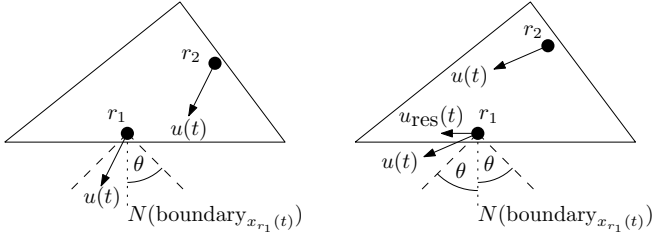


Fig. 2. Left: An input force command $u(t)$ within the cone $\pm\theta$ about the normal to the boundary results in no motion of r_1 . Right: An input force command $u(t)$ outside the cone results in a motion of both particles. Observe that r_1 slides along the boundary with a resulting force $u_{res}(t)$.

between static friction with a boundary of the workspace and the external force to achieve any desired configuration.

A. Our Results.

We provide a fundamentally new approach to manipulating a swarm of objects by an external, global force, demonstrating how static boundary friction can be employed to achieve arbitrary reconfiguration. Our results include the following.

- We show that any two particles in an arrangement can be arbitrarily relocated in a triangle, provided sufficient friction as a function of the triangle geometry.
- More specifically, for a triangle with second smallest angle β , we prove that an angle of friction of $\frac{\pi}{2} - \beta$ is sometimes necessary and always sufficient to guarantee any reconfiguration.
- We also provide procedures for reconfiguring more than two particles, including sorting a line of n particles.
- We provide hardware experiments showing the efficacy of our strategies, as illustrated in Fig. 1.

B. Other Related Work.

Sliding a component using an active tilting tray has a rich history, especially on sensorless part orientation, see [8], [13]. Similar work also applies to using sliding-jaw grippers with low-friction contact surfaces to localize parts without sensing [9]. Shahrokhi et al. [17], [18] considered reconfiguration problems of particles using friction at the walls. However, they assume walls have infinite friction, i.e., a particle lying at a wall cannot be moved when there is a movement parallel to the wall. This differs from the more realistic assumptions in this paper, in which we only consider finite friction. For a theoretical investigation of friction-less sliding tile particles moving on a 2D grid in the presence of obstacles, see the recent paper by Balanza-Martinez et al. [4] and its bibliography.

II. PRELIMINARIES

The *coefficient of friction* is a property of the surfaces of any two materials brought in contact. The coefficient of friction is a ratio of the force required to move a surface horizontally past another and the force with which the materials are pressed together. If a particle is placed on a flat plate that is tilted until the object slides, the tangent of the angle when the sliding commences is the coefficient of friction.

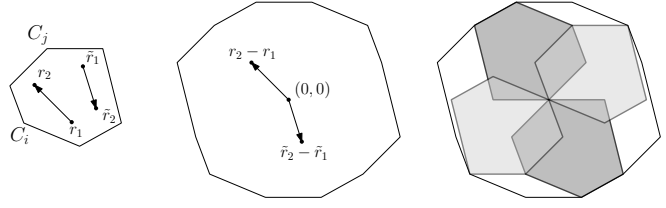


Fig. 3. Left: A six-sided polygon P with start positions r_1 and r_2 for two particles and their goal positions \tilde{r}_1 and \tilde{r}_2 . Middle: The Δ configuration of the polygon and the positions of the start and end configuration. Right: Lightgray (darkgray) area corresponds to the C_i -area (C_j -area, resp.).

Definition 1. Let θ be the angle of friction and $\mu := \tan \theta$ be the coefficient of friction.

See Fig. 2 for an illustration. For a particle r , let $N(\text{boundary}_{x_r(t)})$ be the normal to the boundary at position $x_r(t)$. For notational simplicity, we also use r as the position of the particle. For a force command $u(t)$, if a particle r has position $x_r(t)$ and velocity $\dot{x}_r(t)$ at time t , we assume the following, where $\alpha = \arccos(u(t) \cdot N(\text{boundary}_{x_r(t)}))$ if r lies on the boundary:

$$\dot{x}_r(t) = \begin{cases} 0, & \text{if } x_r(t) \in \text{boundary} \\ & \text{and } \alpha \leq \theta, \\ c_k \|u(t)\| \cdot \sin \alpha, & \text{if } x_r(t) \in \text{boundary} \\ & \text{and } \frac{\pi}{2} \geq \alpha > \theta, \\ \|u(t)\|, & \text{otherwise,} \end{cases}$$

where $c_k < 1$ is some coefficient depending on the kinetic friction. Throughout this paper, we will only consider the first and the third case, i.e., each particle moves at full speed or does not move at all.

Problem 1. Given a workspace, i.e., a convex polygon with n vertices v_1, \dots, v_n , with m particles r_1, \dots, r_m , and an angle of friction θ . Is it possible to reach the configuration $\tilde{r}_1, \dots, \tilde{r}_m$?

In this paper, we do not make any assumption on the initial positions of r_1, \dots, r_m , except that all particles are well separated, i.e., they have a distance $\varepsilon > 0$ to each other.

Definition 2 (Δ Configuration). The Δ configuration space Δ_P of a convex polygon P containing two particles is a polygon obtained by translating n copies of P , such that each vertex of P is moved to the origin, and taking the convex hull of all copies (for an example see Figure 3).

We observe that Δ_P can also be defined by taking the convex hull of differences between each pair of vertices. More formally:

$$\Delta_P := \text{ch}(C_i - C_j \mid C_i, C_j \in P),$$

where $\text{ch}(\cdot)$ denotes the convex hull. From this alternative definition follows that $\Delta_P = \Delta_{-P}$, where $-P$ is P rotated by π . This motivates the following definition.

Definition 3. Let P be a convex polygon and v be a vertex in P . A v -area in Δ_P is the union of P and $-P$ having v centered at the origin (see Figure 3 right).

Note that the union of v -areas for all $v \in P$ equals Δ_P .

III. RECONFIGURATION OF TWO PARTICLES

Just like in the context of sorting algorithms in computer science or discrete mathematics, a critical component for achieving arbitrary reconfiguration of larger ensembles is the ability to rearrange two specific particles. For our purposes of employing external forces and static friction, the additional aspects of geometry and physics have to be considered. These are addressed in this section, before we proceed to show how this can be generalized to large ensembles in the next section.

The main idea for this first step is to try to completely cover the Δ configuration. We start by developing a strategy for separating two particles in Subsection A, which gives us a lower bound for θ for every strategy in this section. This is followed by an upper bound for θ in triangles (Subsection B) and arbitrary convex polygons (Subsection C), i.e., we can guarantee any reconfiguration with any angle of friction higher than this upper bound.

A. Separating two particles

As a first step, we show how to separate two specific particles.

Lemma 1. *Assume particle r_1 is positioned in a corner with angle α , then we can move r_2 to any position in the polygon without moving r_1 , if $\mu > \tan(\frac{\alpha}{2})$, i.e., the angle of friction is greater than $\frac{\alpha}{2}$.*

Proof. We perform a zig-zag move (Figure 5(a) left) that increases the distance between r_1 and r_2 . Consider Figure 4. W.l.o.g., r_1 sits in the corner bounded by segments s_1 and s_2 , while r_2 starts on segment s_1 with distance c to r_1 . We move r_2 to the other segment s_2 with the maximum angle possible. Particle r_2 reaches s_2 with distance b to r_1 . Afterwards, we move r_2 back to s_1 , now having a distance of c' . If θ is sufficiently large, then $c' > c$.

For a given θ , we have

$$b = c \cdot \frac{\sin(\frac{\pi}{2} - \alpha + \theta)}{\sin(\frac{\pi}{2} - \theta)} = c \cdot \frac{\cos(\theta - \alpha)}{\cos(\theta)},$$

and therefore

$$c' = c \cdot \frac{\cos^2(\theta - \alpha)}{\cos^2(\theta)}.$$

If $c' > c$, then $\cos^2(\theta - \alpha) > \cos^2(\theta)$. This is true if $\cos(\theta - \alpha) > \cos(\theta)$. By applying the arccos function, this yields $\alpha - \theta < \theta$, if $\theta < \alpha$, and $\theta - \alpha < \theta$, if $\theta \geq \alpha$. The first case is true iff $\theta > \frac{\alpha}{2}$, the second case is always true for $\alpha > 0$. Hence, for $\theta > \frac{\alpha}{2}$ we can increase the distance between r_1 and r_2 . By moving r_2 by short movements, we can relocate r_2 to any corner of a given polygon. Note that if α is an obtuse angle, the same formula can be derived. \square

Note that in a triangle an angle of friction of $\frac{\alpha}{2}$ is necessary.

B. Reconfiguration of two particles in arbitrary triangles

Let T be a triangle and let A, B and C be the corners with angles α, β and γ . Furthermore, let α be the smallest angle in T and we assume that $\theta > \frac{\alpha}{2}$ is guaranteed. Consider two

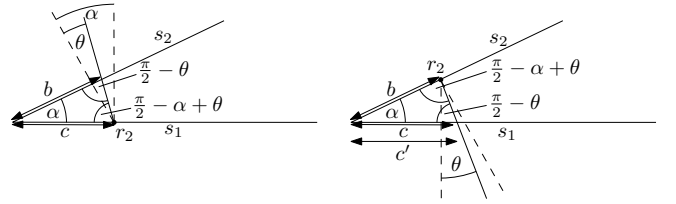
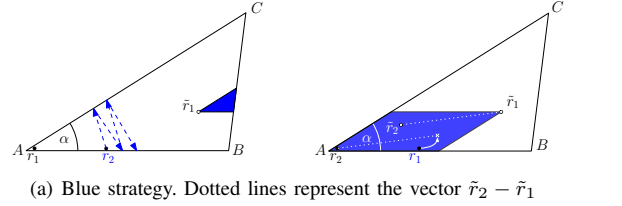
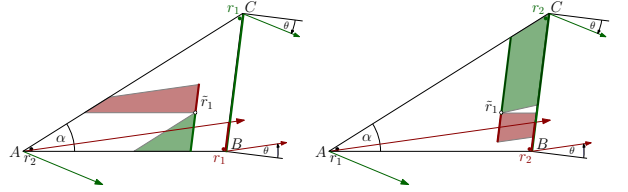


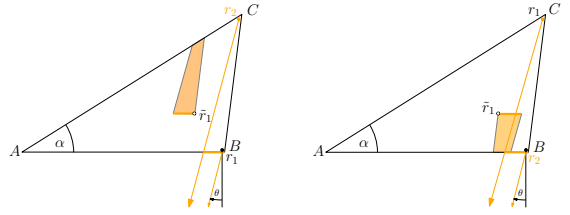
Fig. 4. A corner of a polygon with angle α . Left: r_2 lies on s_1 and is moved to s_2 . Right: r_2 is moved back to s_1 . r_2 can be moved away from the corner if the angle of friction exceeds $\frac{\alpha}{2}$.



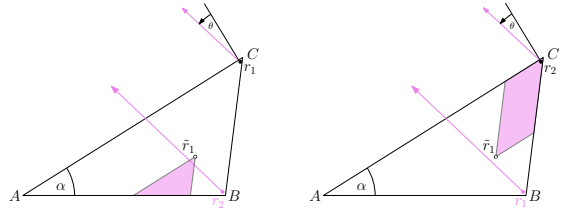
(a) Blue strategy. Dotted lines represent the vector $\tilde{r}_2 - \tilde{r}_1$



(b) Red and green strategy



(c) Orange strategy



(d) Violet strategy

Fig. 5. Illustration of the five strategies. Colored areas correspond to valid goal positions for r_2 , if the goal position of r_1 is \tilde{r}_1 . Left column: We fix r_1 and move r_2 . Right column: We switch intermediate locations of r_1 and r_2 .

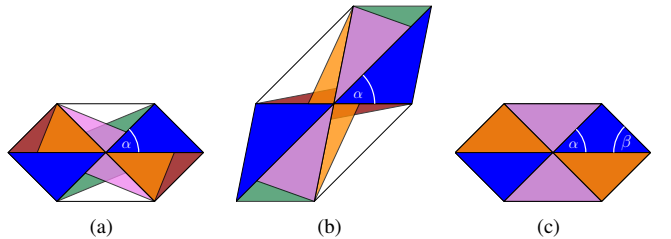


Fig. 6. Shown in (a) and (b) are the Δ configurations of the blue triangle. Both blue triangles correspond to the A -area. Colors represent the areas in the Δ configuration covered by our five strategies with an angle of friction of $\frac{\alpha}{2} + \varepsilon$ for some $\varepsilon > 0$. (a),(b): We observe that every strategy may cover areas not covered by any other strategy. (c): If $\theta > \frac{\pi}{2} - \beta$ then we can guarantee full coverage.

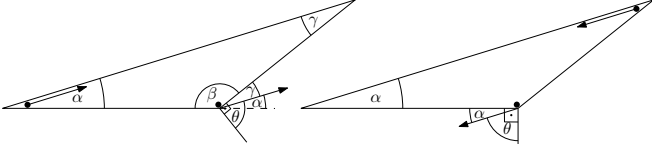


Fig. 7. Left: Computation of minimum θ needed for the red strategy. Right: Computation of minimum θ needed for the final step of the orange strategy.

particles r_1 and r_2 within a triangle and their goal positions \tilde{r}_1 and \tilde{r}_2 . We have the following strategies to reach the goal positions (see also Fig. 5 for a graphical sketch):

Blue: Move r_1 to A . As in Lemma 1, use zig-zag moves to place r_2 in T while r_1 is fixed in A , such that $r_2 - r_1 = \tilde{r}_2 - \tilde{r}_1$. Then, translate r_1 and r_2 to their goal positions.

Red: First, place r_2 in A and move r_1 to B . Then, place r_2 anywhere in the area spanned by \overline{AB} and the angle of friction θ . Afterwards, translate r_1 and r_2 to their goal positions.

Green: First, Place r_2 in A and move r_1 to C . Then, place r_2 in the area spanned by \overline{AC} and the angle of friction θ , such that $r_2 - r_1 = \tilde{r}_2 - \tilde{r}_1$. Afterwards, translate r_1 and r_2 to their goal positions.

Orange: Place r_2 in C and r_1 in B (as we will see later, this is always possible if $\theta > \frac{\alpha}{2}$; see Theorem 2). Then, place r_2 in the area spanned by \overline{BC} and the angle of friction, such that $r_2 - r_1 = \tilde{r}_2 - \tilde{r}_1$. Afterwards, translate both particles to their goal position.

Violet: Place r_2 in B and r_1 in C . Then, place r_2 anywhere in the area spanned by \overline{CB} and the angle of friction, such that $r_2 - r_1 = \tilde{r}_2 - \tilde{r}_1$. Finally, translate both particles to their goal position.

These strategies can also be used by switching the particles r_1 and r_2 . Assume that r_1 lies in corner A . To switch r_1 and r_2 , we separate both particles to corners B and C , then we use strategy orange or violet (depending on which particle is in which corner), and as a last step, we move r_2 to A .

Observation 1. In the Δ configuration, the only strategies that overlap are red with orange and green with violet.

Furthermore, the blue strategy fills out the A -area completely, red and orange fill out parts of the B -area, and green and violet fill out parts of the C -area

Lemma 2. If $\theta > \frac{\pi}{2} - \gamma$, then the area of the red and orange strategy covers the B -area completely.

Proof. W.l.o.g., assume that $\alpha \leq \beta \leq \gamma$. First observe that, if the B -area is covered, then the red or the orange strategy covers the area on its own. Therefore, we search for the minimum angle needed such that one of the two strategies covers the B -area.

Red strategy: Red covers the B -area if we can move r_1 (r_2 , resp.) to C without moving r_2 (r_1 , resp.). To this end, the angle of friction must be $\frac{\pi}{2} - \gamma$ (see Figure 7 left).

Orange strategy: Assume that r_1 and r_2 already lie in B and C , respectively. To cover the B -area, r_2 must be movable to A without moving r_1 . This requires an angle of friction of at least $\frac{\pi}{2} - \alpha$.

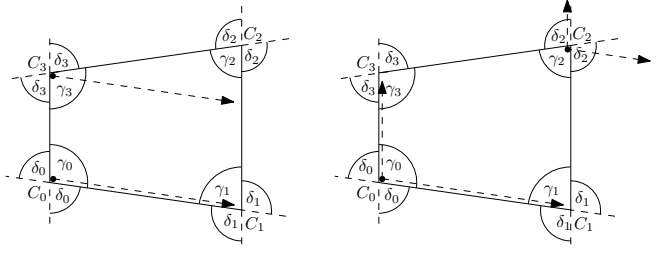


Fig. 8. Left: If we want to move a particle in C_0 without moving a particle in C_3 , some movements are prohibited (even if we have infinite friction), because $\delta_0 < \gamma_3$. Right: However, we can move the particle in C_0 to any place in the polygon without moving a particle in C_2 (unless the friction is too small), because $\delta_3 + \delta_0 > \gamma_2$ and $\delta_1 + \delta_0 > \gamma_2$.

Because $\alpha \leq \gamma$, we only need to consider the red strategy for full coverage and thus, $\theta > \frac{\pi}{2} - \gamma$ is sufficient to cover the B -area. \square

Lemma 3. If $\theta > \frac{\pi}{2} - \beta$, then the area of the green and violet strategy covers the C -area completely.

Proof. The argument of the previous lemma applies. \square

Theorem 1. Let T be a triangle with angles $\alpha \leq \beta \leq \gamma$. If $\theta > \frac{\pi}{2} - \beta$, then we can guarantee any reconfiguration of two particles, i.e., Δ_T is completely covered by our strategies.

Proof. To cover the A -, B -, and C -area of the Δ configuration, the angle of friction θ must be greater than $\max(\frac{\alpha}{2}, \frac{\pi}{2} - \beta, \frac{\pi}{2} - \gamma)$. Because $\beta \leq \gamma$ we have that $\frac{\pi}{2} - \beta \geq \frac{\pi}{2} - \gamma$.

We can rewrite $\frac{\pi}{2} - \beta$ as $\frac{\pi - 2\beta}{2} = \frac{\alpha + \gamma - \beta}{2} \geq \frac{\alpha}{2}$ (because $\gamma - \beta \geq 0$). Therefore, an angle of friction of at least $\frac{\pi}{2} - \beta$ guarantees full coverage of Δ_T . \square

With the following theorem we show that, if θ is slightly larger than $\frac{\alpha}{2}$, then we can guarantee two thirds of all reconfigurations. Furthermore, the proof implies that two particles can be separated to B and C for any $\theta > \frac{\alpha}{2}$.

Theorem 2. For a triangle T with angles $\alpha \leq \beta \leq \gamma$, at least two thirds of all configurations can be guaranteed if $\theta > \frac{\alpha}{2}$.

Proof. Following Lemma 2, we need an angle of friction of at least $\frac{\pi}{2} - \gamma$ to cover the B -area. Because $\frac{\pi}{2} - \gamma = \frac{\pi - 2\gamma}{2} \leq \frac{\pi - \gamma - \beta}{2} = \frac{\alpha}{2}$, the B -area is always covered if $\theta > \frac{\alpha}{2}$. Furthermore, the A -area (covered by the blue strategy) and the B -area are two thirds of Δ_T , and thus, we can guarantee two thirds of all possible configurations. \square

C. Reconfiguration of two particles in convex polygons

Now we proceed to develop strategies to reconfigure two particles in convex polygons by generalizing the strategies for triangles, i.e., for a particle r_1 in corner C_i and a particle r_2 in corner C_j , moving particle r_2 to cover the C_i -area. As shown in Figure 8, we cannot guarantee full coverage with this strategy, because any movement for r_2 in direction to C_1 would also move r_1 . This happens for all pairs of vertices (C_i, C_j) of P , where the segment $C_j C_{j+1}$ has a larger negative slope than the segment $C_i C_{i-1}$.

Definition 4. For a vertex $C_i \in P$, let δ_i be the exterior angle at vertex C_i . Let $P_{i,j}^+ := \{C_i, C_{i+1}, \dots, C_{j-1}, C_j\}$ and $P_{i,j}^- := \{C_i, C_{i-1}, \dots, C_{j+1}, C_j\}$.

Furthermore, let $P_i := \{C_j \in P \mid \sum_{C_k \in P_{i+1,j}^+} \delta_k \geq \gamma_i \wedge \sum_{C_k \in P_{i-1,j-1}^-} \delta_k \geq \gamma_i\}$, i.e., P_i contains every vertex of P such that we can use the strategy described in the beginning of this section. Note that all indices are modulo n .

Lemma 4. For a vertex C_i of P , we have $|P_i| \geq 1$.

Proof. Assume that $|P_i| = 0$. Then, there are two adjacent vertices C_j and $C_{j'}$ such that $\sum_{C_k \in P_{i+1,j}^+} \delta_k < \gamma_i$ and $\sum_{C_k \in P_{i-1,j'}^-} \delta_k < \gamma_i$.

This implies that $2\gamma_i > \sum_{C_k \in P_{i+1,j}^+} \delta_k + \sum_{C_k \in P_{i-1,j'}^-} \delta_k = -\delta_i + \sum_{C_k \in P} \delta_k = -\delta_i + 2\pi > 2\pi - 2\delta_i = 2\gamma_i$. This is a contradiction and therefore $|P_i| \geq 1$. \square

Lemma 5. Let P be a convex polygon with vertices C_0, \dots, C_{n-1} and angles $\gamma_0, \dots, \gamma_{n-1}$. We can cover the C_i -area if $\theta > \min_{j \in P_i} \left(\frac{\gamma_i}{2}, \max \left(\frac{\gamma_j}{2}, \eta_{i,j}^+ - \frac{\pi}{2}, \eta_{i,j}^- - \frac{\pi}{2} \right) \right)$, where $\eta_{i,j}^+ := \sum_{C_k \in P_{i+1,j-1}^+} \delta_k$ and $\eta_{i,j}^- := \sum_{C_k \in P_{i-1,j+1}^-} \delta_k$.

Proof. We consider two strategies to cover the C_i -area. The first strategy keeps one particle in the corner C_i and moves the second particle to any position in the polygon. This requires an angle of friction of more than $\frac{\gamma_i}{2}$.

The second strategy picks one vertex $C_j \in P_i$ and proceeds in two steps. See Fig. 9 for an illustration. In step one, one particle is kept in corner C_j while the other particle is moved to corner C_i . This requires an angle of friction of more than $\frac{\gamma_j}{2}$. In step two, we move the particle from corner C_j to any place in the polygon. We show that an angle of friction of $\max(\eta_{i,j}^+ - \frac{\pi}{2}, \eta_{i,j}^- - \frac{\pi}{2})$ is sufficient to do this.

The segment $C_i C_j$ splits the polygon into subpolygons, i.e., $P_{i,j}^+$ and $P_{i,j}^-$, and splits the angle γ_i (γ_j) into two angles γ_i^+ and γ_i^- (γ_j^+ and γ_j^-). W.l.o.g., consider $P_{i,j}^+$ (calculations for $P_{i,j}^-$ are analogous). To move the particle, say r_1 , from C_j anywhere in $P_{i,j}^+$, it must be possible to move r_1 in direction $\vec{v} = C_{j-1} - C_j$ without moving the particle in C_i . Therefore, θ must be at least the angle that is enclosed by \vec{v} and the orthogonal of the segment $s := \overline{C_i C_{i+1}}$. We observe that the angle between \vec{v} and s is $\pi - \gamma_i^+ - \gamma_j^+$. The sum $\gamma_i^+ + \gamma_j^+$ can be calculated by taking the sum of angles in $P_{i,j}^+$ and subtract every angle of $P_{i,j}^+$ except γ_i^+ and γ_j^+ . More formal: $\gamma_i^+ + \gamma_j^+ = (|P_{i,j}^+| - 2)\pi - \sum_{C_k \in P_{i+1,j-1}^+} \gamma_k = \sum_{C_k \in P_{i+1,j-1}^+} \pi - \gamma_k = \sum_{C_k \in P_{i+1,j-1}^+} \delta_k = \eta_{i,j}^+$.

Because the angle between \vec{v} and s is $\pi - \eta_{i,j}^+$, the angle of friction needed is $\eta_{i,j}^+ - \frac{\pi}{2}$. Thus, θ must be greater than $\max(\frac{\gamma_j}{2}, \eta_{i,j}^+ - \frac{\pi}{2}, \eta_{i,j}^- - \frac{\pi}{2})$ for strategy two by picking one specific vertex of P .

By taking the minimum over all choices for C_j and the minimum of strategies one and two, the claim follows. \square

Combining Lemmas 4 and 5 yields the following theorem.

Theorem 3. Let P be a convex Polygon with vertices C_0, \dots, C_{n-1} and angles $\gamma_0, \dots, \gamma_{n-1}$. If $\theta > \max_{0 \leq i < n} \left(\min_{j \in P_i} \left(\frac{\gamma_i}{2}, \max \left(\frac{\gamma_j}{2}, \eta_{i,j}^+ - \frac{\pi}{2}, \eta_{i,j}^- - \frac{\pi}{2} \right) \right) \right)$, where $\eta_{i,j}^+ := \sum_{C_k \in P_{i+1,j-1}^+} \delta_k$ and $\eta_{i,j}^- := \sum_{C_k \in P_{i-1,j+1}^-} \delta_k$, then every configuration of two particles can be reached.

D. Reconfiguration of two particles in regular n -gons

Theorem 4. If P is a regular polygon with n vertices and if $\mu > \cot(\pi/n)$, then every reconfiguration is possible.

Proof. In a regular polygon, every inner angle is $\frac{n-2}{n}\pi$. We know that $\max \left(\frac{\gamma_i}{2}, \eta_{i,j}^+ - \frac{\pi}{2}, \eta_{i,j}^- - \frac{\pi}{2} \right) \geq \frac{\gamma_i}{2} = \frac{n-2}{2n}\pi$ for every pair (i, j) . Therefore, due to Theorem 3, $\theta > \frac{n-2}{2n}\pi$ is sufficient to cover the whole Δ configuration, and we can guarantee every configuration of two particles. Thus, the coefficient of friction is $\mu = \tan(\theta) > \tan(\frac{n-2}{2n}\pi) = \cot(\frac{\pi}{n})$. \square

IV. RECONFIGURATION OF MANY PARTICLES

In this section, we consider more than two particles. We show further limitations by demonstrating that not every reconfiguration of three particles may be possible. On the positive side, we show that we can perform arbitrary permutations for a line of n particles.

Theorem 5. Consider the class \mathcal{C} of configurations of three particles in a square, where one of the particles lies within the bounding rectangle of the other two particles. If $\theta > \frac{\pi}{2}$, then we can reconfigure any configuration to any configuration of \mathcal{C} . Furthermore, \mathcal{C} contains $\frac{1}{3}$ of all possible configurations.

Proof. W.l.o.g., let r_1, r_2 and r_3 be the three particles such that the x - and y -coordinates in the goal configuration are monotonically increasing, i.e., $r_1.x \leq r_2.x \leq r_3.x$ and $r_1.y \leq r_2.y \leq r_3.y$. We can also assume that the same holds for the x -coordinates in the start configuration (or else we start using the swap strategy from the last section). Proceed as follows: (1) Move r_1 to the lower left corner of the square and (2) use zig-zag moves to move r_3 to the top right corner. Then, (3) we can use zig-zag moves to move r_2 to a position, such that r_1 and r_2 have the same relative position as in the goal configuration. (4) Translate r_1 together with r_2 such that r_2 and r_3 have the correct relative position. As a last step (5) we can translate all three particles to the desired goal configuration.

To show that this strategy is correct, we show that we can carry out all five steps. We can do step (1) by simply translating all particles. We show that we can do step (2) in the previous section. For step (3), assume that we move r_2 further away from r_1 . This means the zig-zag moves cannot affect r_3 . Due to the angle of friction of $\theta > \frac{\pi}{2}$ we can move r_2 without moving r_1 . Step (4) and (5) are simple translations and can therefore be performed.

Now, it is left to show that \mathcal{C} contains $\frac{1}{3}$ of all configurations. There are 12 choices for the two particles that define the bounding rectangle and for a fixed choice these are

$$\int_0^1 \int_0^1 \int_0^{x_1} \int_0^{y_1} (x_1 - x_2)(y_1 - y_2) dy_2 dx_2 dy_1 dx_1 = \frac{1}{36}$$

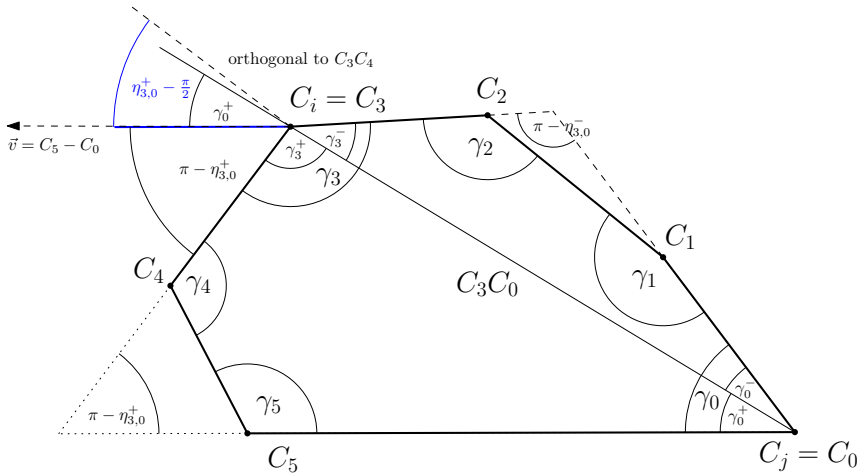


Fig. 9. Second strategy to cover the C_i -area (Lemma 5). Assuming one particle in C_3 and one particle in C_0 . Blue angle is sufficient as angle of friction to reach any position in $P_{3,0}^+ = C_3C_4C_5C_0$ with the particle in C_0 without moving the particle in C_3 .

of all configurations. Therefore, in total \mathcal{C} contains $\frac{1}{3}$ of all configurations. \square

Theorem 6. *There are configurations of three particles in a square we cannot reach, unless we have infinite friction.*

Proof. Consider the goal configuration with particle r_1 in the top left corner, particle r_2 in the bottom left corner, and particle r_3 in the middle of the right side of the square. Assume r_1 is the last particle reaching the goal position (or together with the other particles). Then, the last moving direction would also move r_3 away from its goal position. Therefore, the assumption is wrong. The same holds for particle r_2 . If r_3 is the last particle reaching the goal position, then any of the last moving direction would also move r_1 or r_2 away from their goal position. Because no particle can be the last particle reaching its goal position, we can not reach the desired goal configuration.

Theorem 7. Consider n particles in a square with distance d between adjacent particles. If the angle of friction $\theta > \frac{\pi}{4}$ then we can reorder the particles.

Proof. Consider some permutation Π of the particles. The idea is to move the $\Pi(n-i)$ -th particle to the left side of the current line in round i , thus performing a mix of selection sort and insertion sort.

Assume the line lies horizontally within the square. Then we push the line to the left until the first particle hits the wall. We start to move all particles with a diagonal down-left movement (see Fig. 11(a)). This only moves particles that are not placed on a wall. We stop the movement when the $\Pi(n-i)$ -th particle p hits the wall (see Fig. 11(b)). After translating all particles such that p gets trapped in the lower left corner, we perform a diagonal right-down movement until the former left neighbor of p has position $(\frac{d}{2}, \frac{d}{2})$ (see Fig. 11(c)). Then, we move all particles except p with zig-zag moves to the top wall, where we can rebuild a line of $n-1$ particles by repeating top-right,

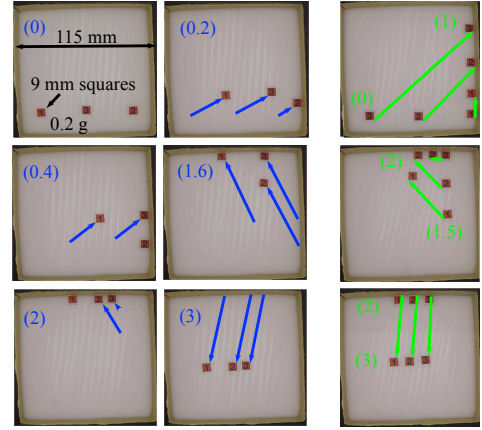


Fig. 10. Sorting multiple particles (hardware experiments, see video attachment [15]). In blue (1,3,2)→(1,2,3). In green (3,2,1)→(1,2,3). All particles move when commanded, unless friction with the boundary prevents motion. Boundary is coated with 220 grit sandpaper, giving $\theta_B = 20.2^\circ \pm 2.22^\circ$ and the acetal floor $\theta_F = 55.4^\circ \pm 4.9^\circ$.

down-left, and zig-zag moves (see Fig. 11(d)-(f)). With simple translations we can add p to the left side with distance d . Therefore, after i repetitions of this strategy, the left i particles of the current line are sorted in ascending order. \square

See Figure 10 for a real-world demonstration of these arguments, showing their practical usefulness.

V. HARDWARE EXPERIMENTS

To show the practical usefulness of our theoretical work, we built 2D workspaces containing two sliders, with gravity as external force. The workspace was tilted by a robot arm. See our video [15] for animation and explanations.

A. Hardware platform and workspace

The triangular workspace has side walls of length {270,198,126} mm. Our workspace floor was made of nonstick teflon oven liners and the boundary walls were made of laser-cut acrylic. The pentagonal particles are laser-cut acrylic with side lengths of 3 mm with teflon tape on their underside.

The workspace is held by the gripper of a UR-3 robotic arm. The 4th and 6th joints are used to tilt the workspace in arbitrary directions, with the 5th joint oriented at 90°.

The first sections of this paper assumed a single, constant coefficient of friction of μ , where $\mu \in [0, \infty]$. A particle slides if the workspace is tilted beyond the angle $\arctan(\mu)$. The wall's coefficient of static friction (acrylic on acrylic) is approximately $\mu_w = 0.61$ ($\theta = 31.4^\circ$) (measured by placing the particle on this surface and tilting until the particle first slides). The floor's coefficient of static friction (teflon tape on teflon oven liner) is approximately $\mu_f = 0.207$ ($\theta = 11.7^\circ$).

The composite force of static friction is a function of table tilt. The force causing the particle to slide is opposed by the static friction with the floor and with the wall.

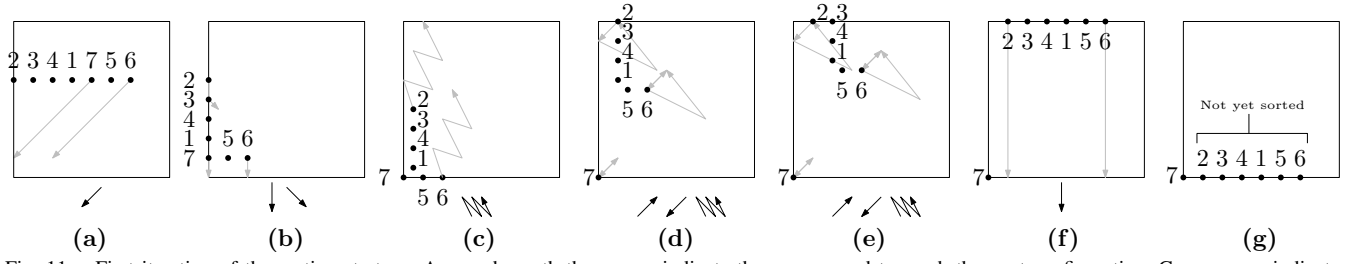


Fig. 11. First iteration of the sorting strategy. Arrows beneath the square indicate the moves used to reach the next configuration. Gray arrows indicate trajectories for particles 2, 6 and 7. (a): A line with desired ordering. (b),(c): Extracting largest number. (d)-(f): Rebuilt line with remaining particles. (g): Add particle to line.

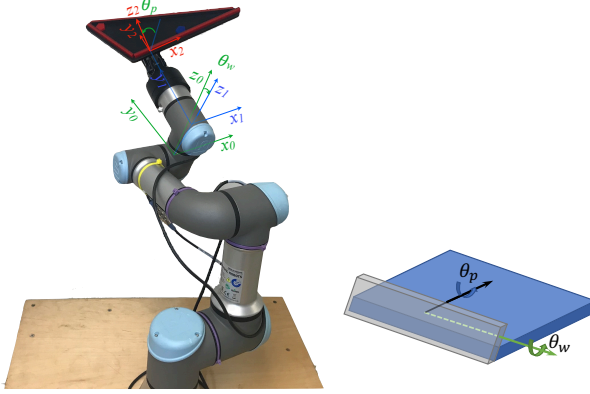


Fig. 12. The workspace (black triangle) is tilted by the fourth and sixth links of a UR-3 robot. The workspace walls have a higher coefficient of friction than the workspace floor.

B. Model for wall and floor friction

Any tilt of a 2D workspace can be described by first a tilt θ_w about the axis parallel to the boundary wall (such that positive θ_w slopes the workspace toward the wall), followed by a tilt θ_p about the axis perpendicular to the first tilt and the original gravity axis.

We can therefore first apply a rotation about the world gravity axis (the z -axis) to align the boundary wall with the world x -axis, rotate θ_w about the current x -axis, and rotate θ_p about the current y -axis to complete the composite tilt. The composite rotation is

$$R_{z,\phi} R_{x,\theta_w} R_{y,\theta_p} = R_{z,\phi} \begin{bmatrix} c_{\theta_p} & 0 & s_{\theta_p} \\ s_{\theta_w} s_{\theta_p} & c_{\theta_w} & -c_{\theta_p} s_{\theta_w} \\ -c_{\theta_w} s_{\theta_p} & s_{\theta_w} & c_{\theta_w} c_{\theta_p} \end{bmatrix}. \quad (1)$$

Here we use the shorthand $\sin(x) = s_x$ and $\cos(x) = c_x$. For simplicity, the following analysis will ignore the initial rotation about the z -axis. The third row describes how the components of the original gravity vector are distributed along the boundary wall ($-c_{\theta_w} s_{\theta_p}$), perpendicular to the wall (s_{θ_w}) and into the floor ($c_{\theta_w} c_{\theta_p}$). For simplicity, assume the force of gravity on the particle is 1N: $f_g = [0, 0, -1]^T$. To contact the floor, both θ_w and θ_p must have magnitude less than $\pi/2$. The normal force from the tilted floor is $f_{N,\text{floor}} = c_{\theta_w} c_{\theta_p}$. If a particle is touching a wall *and* the tilt $\theta_w > 0$ and thus pushes the particle against the wall, then the wall generates a normal force

$$f_{N,\text{wall}} = \begin{cases} s_{\theta_w}, & \theta_w > 0 \\ 0, & \text{else.} \end{cases} \quad (2)$$

The force after accounting for the normal force is

$$f_{\text{slide}} = f_g - f_{N,\text{floor}} - f_{N,\text{wall}}. \quad (3)$$

The static friction force is proportional to the normal force. The particle will only slide if f_{slide} is greater than the static friction force, i.e.,

$$|f_{\text{slide}}| > \mu_f |f_{N,\text{floor}}| + \mu_w |f_{N,\text{wall}}|. \quad (4)$$

The particle slides if the following quantity is positive:

$$\begin{cases} |c_{\theta_w} s_{\theta_p}| - \mu_f c_{\theta_w} c_{\theta_p} - \mu_w s_{\theta_w} & \theta_w > 0 \\ \sqrt{1 - c_{\theta_w}^2 c_{\theta_p}^2} - \mu_f c_{\theta_w} c_{\theta_p} & \text{else} \end{cases}. \quad (5)$$

a) Conversion to rotation about x and y axes: The two-links of our robot generate a rotation about the global x -axis, followed by a rotation about the current y -axis: $R_{x,\theta_x} R_{y,\theta_y}$. To generate the appropriate gravitational force described by a z rotation of ϕ followed by θ_w about the wall and θ_p perpendicular to the wall, we only need to reproduce the third column of (1), and select

$$\theta_y = \arcsin(c_{\phi} s_{\theta_p} + c_{\theta_p} s_{\theta_w} s_{\phi}) \quad (6)$$

$$\theta_x = \arcsin\left(\frac{s_{\theta_w} c_{\theta_p} c_{\phi} - s_{\theta_p} s_{\phi}}{c_{\theta_y}}\right) \quad (7)$$

b) Verification of model: The slipping force from (5) is the left plot of Fig. 13. Particles not touching a wall slip outside the green circle; particles touching a wall only slip in the region below the red line. The required angle of friction to avoid slipping is shown in the left plot of Fig. 13.

$$\text{Angle of Friction} = \frac{\pi}{2} - \arctan(s_{\theta_p} c_{\theta_w}, s_{\theta_w}) \quad (8)$$

C. Demonstration: placing particles in opposite corners of a triangular workspace

For this demonstration, two pentagonal particles positions were placed into opposing corners of a triangle. A motion sequence using the blue strategy from Section III was used to hold one particle in the left corner while the other was moved to the right side. Then the red strategy was used to swap the particle's positions. Repeating the procedure iterates between placing the particles in opposite corners every 36 moves using the procedure. Representative screenshots of a rearrangement procedure are shown in Fig. 14. The tilt used to move both particles were $(\theta_w, \theta_p) = (0^\circ, 20^\circ)$, shown in Fig. 13 by a blue point. The particles both move, since this

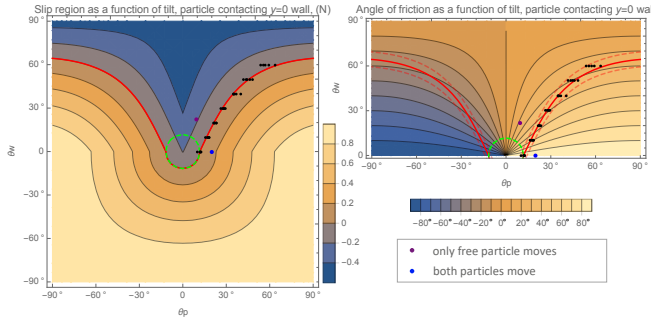


Fig. 13. (Left) Contour plot showing slipping force due to gravity minus static friction forces from the wall and floor. Regions with positive values will slip. Particles not at a wall will slip outside the green circle and particles at the wall will slip below the red line. (Right) Contour plot showing angle of friction. The 35 data points overlaid show where components slipped, as a function of tilt about the wall θ_w and perpendicular to the wall θ_p . The experiments moved both particles using the tilt marked by the blue point, and the purple point moved only the free particle.

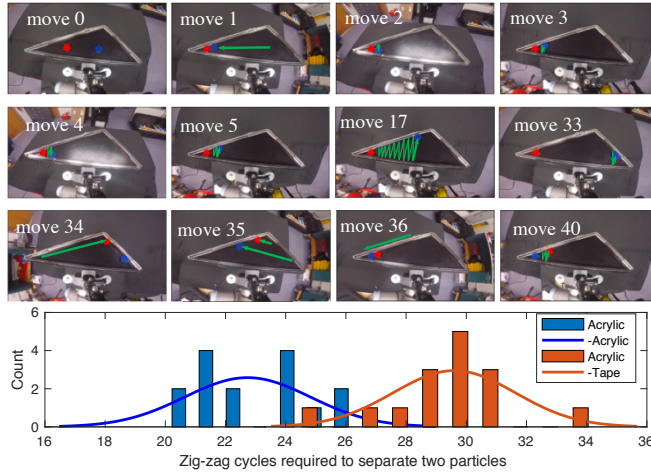


Fig. 14. (top) Two pentagonal particles were placed into opposing corners of a triangle, and their positions are switched every 36 moves using the procedure from Section III (see video attachment <https://youtu.be/hSa4EmjHXAI>) [15]. (bottom) Histogram data on required number of zigzag movements required for two combinations of wall and particle materials.

would require an angle of friction of 90° . To move one particle we used $(\theta_w, \theta_p) = (22^\circ, 9.5^\circ)$, shown in Fig. 13 by a purple point, which had an angle of friction of 22.2° . The tilts were performed at $66^\circ/s$.

To measure the repeatability of this setup, we counted the number of zigzag cycles required to move one particle from touching the first particle in the left corner to touching the opposite triangle corner while the first particle stays stationary. Figure 14 shows a normal distribution fit to counts from 15 trials for two different boundary materials: acrylic and electrical tape over acrylic. When the particle impacts an acrylic boundary, it tends to slide along the edge, resulting in less required cycles than if the boundary is covered with electrical tape. We reject the null hypothesis that the different surfaces require the same number of cycles with p -value 6.7×10^{-10} .

VI. CONCLUSION

We introduced a novel approach for rearranging the positions of particles by applying global uniform forces, making use of

different local static friction to achieve arbitrary goal positions. We provided strategies enabling arbitrary rearrangements of two particles in a triangle, giving a characterization of the critical coefficient of friction in terms of the boundary geometry. These results are extended to convex polyominoes, and for rearranging larger numbers of particles, and employed for practical experiments. Future work can now investigate optimal motion planning (shortest paths, reproducibility, throughput), as well as coupling these results with orientation control and possible applications in part assembly.

REFERENCES

- [1] S. Akella, W. H. Huang, K. M. Lynch, and M. T. Mason, "Parts feeding on a conveyor with a one joint robot," *Algorithmica*, vol. 26, no. 3-4, pp. 313-344, 2000.
- [2] G. Aloupis, J. Cardinal, S. Collette, F. Hurtado, S. Langerman, and J. O'Rourke, "Draining a polygon—or—rolling a ball out of a polygon," *Computational Geometry*, vol. 47, no. 2, pp. 316-328, 2014.
- [3] D. Arbutkale and A. A. Requicha, "Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations," *Autonomous Robots*, vol. 28, no. 2, pp. 197-211, 2010.
- [4] J. Balanza-Martinez, A. Luchsinger, D. Caballero, R. Reyes, A. A. Cantu, R. Schweller, L. A. Garcia, and T. Wylie, "Full tilt: Universal constructors for general shapes with uniform external forces," in *SODA*, 2019, pp. 2689-2708.
- [5] A. Becker, E. D. Demaine, S. P. Fekete, and J. McLurkin, "Particle computation: Designing worlds to control robot swarms with only global signals," in *IEEE ICRA*, 2014, pp. 6751-6756.
- [6] A. T. Becker, E. D. Demaine, S. P. Fekete, J. Lonsford, and R. Morris-Wright, "Particle computation: complexity, algorithms, and logic," *Natural Computing*, vol. 18, no. 1, pp. 181-201, 2019.
- [7] A. T. Becker, S. P. Fekete, P. Keldenich, D. Krupke, C. Rieck, C. Scheffer, and A. Schmidt, "Tilt assembly: algorithms for micro-factories that build objects with uniform external forces," *Algorithmica*, pp. 1-23, 2017.
- [8] M. A. Erdmann and M. T. Mason, "An exploration of sensorless manipulation," *IEEE J. Robot. Autom.*, vol. 4, no. 4, pp. 369-379, 1988.
- [9] K. Y. Goldberg, "Orienting polygonal parts without sensors," *Algorithmica*, vol. 10, no. 2-4, pp. 201-225, 1993.
- [10] P. S. S. Kim, A. T. Becker, Y. Ou, A. A. Julius, and M. J. Kim, "Imparting magnetic dipole heterogeneity to internalized iron oxide nanoparticles for microorganism swarm control," *Journal of Nanoparticle Research*, vol. 17, no. 3, pp. 1-15, 2015.
- [11] J. S. Lewis and J. M. O'Kane, "Planning for provably reliable navigation using an unreliable, nearly sensorless robot," *Int J Robot Res.*, vol. 32, no. 11, pp. 1342-1357, 2013.
- [12] K. M. Lynch and M. T. Mason, "Stable pushing: Mechanics, controllability, and planning," *Int J Robot Res.*, vol. 15, no. 6, pp. 533-556, 1996.
- [13] P. Mannam, A. V. Volkov, R. Paolini, G. Chirikjian, and M. T. Mason, "Sensorless pose determination using randomized action sequences," *Entropy*, vol. 21, no. 2, p. 154, 2019.
- [14] M. Rubenstein, C. Ahler, N. Hoff, A. Cabrera, and R. Nagpal, "Kilobot: A low cost robot with scalable operations designed for collective behaviors," *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 966-975, 2014.
- [15] A. Schmidt, V. M. Baez, A. T. Becker, and S. P. Fekete, "Particle relocation," 2019. [Online]. Available: <https://youtu.be/hSa4EmjHXAI>
- [16] A. Schmidt, S. Manzoor, L. Huang, A. T. Becker, and S. P. Fekete, "Efficient parallel self-assembly under uniform control inputs," *IEEE RA-L*, vol. 3, no. 4, pp. 3521-3528, 2018.
- [17] S. Shahrokhi, A. Mahadev, and A. T. Becker, "Algorithms for shaping a particle swarm with a shared input by exploiting non-slip wall contacts," in *IEEE/RSJ IROS*, 2017, pp. 4304-4311.
- [18] S. Shahrokhi, J. Shi, B. Isichei, and A. T. Becker, "Exploiting nonslip wall contacts to position two particles using the same control input," *IEEE Trans. Robot.*, pp. 1 - 12, 2019.
- [19] Y. Zhang, X. Chen, H. Qi, and D. Balkcom, "Rearranging agents in a small space using global controls," in *IEEE/RSJ IROS*, 2017, pp. 3576-3582.
- [20] Y. Zhang, E. Whiting, and D. Balkcom, "Assembling and disassembling planar structures with divisible and atomic components," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 3, pp. 945-954, 2018.

CADbots: Algorithmic Aspects of Manipulating Programmable Matter with Finite Automata*

Sándor P. Fekete¹, Robert Gmyr², Sabrina Hugo¹,
Phillip Keldenich¹, Christian Scheffer¹, and Arne Schmidt¹

¹ Department of Computer Science, TU Braunschweig, Germany

² Department of Computer Science, University of Houston, USA

Abstract. We contribute results for a set of fundamental problems in the context of programmable matter by presenting algorithmic methods for evaluating and manipulating a collective of particles by a finite automaton that can neither store significant amounts of data, nor perform complex computations, and is limited to a handful of possible physical operations. We provide a toolbox for carrying out fundamental tasks on a given arrangement of tiles, using the arrangement itself as a storage device, similar to a higher-dimensional Turing machine with geometric properties. Specific results include time- and space-efficient procedures for bounding, counting, copying, reflecting, rotating or scaling a complex given shape.

1 Introduction

When dealing with classic challenges of robotics, such as exploration, evaluation and manipulation of objects, traditional robot models are based on relatively powerful capabilities, such as the ability (1) to collect and store significant amounts of data, (2) perform intricate computations, and (3) execute complex physical operations. With the ongoing progress in miniaturization of devices, new possibilities emerge for exploration, evaluation and manipulation. However, dealing with micro- and even nano-scale dimensions (as present in the context of programmable matter) introduces a vast spectrum of new difficulties and constraints. These include significant limitations to all three of the mentioned capabilities; challenges get even more pronounced in the context of complex nanoscale systems, where there is a significant threshold between “internal” objects and sensors and “external” control entities that can evaluate gathered data, extract important information and provide guidance.

In this paper, we present algorithmic methods for evaluating and manipulating a collective of particles by agents of the most basic possible type: finite automata that can neither store significant amounts of data, nor perform complex computations, and are limited to a handful of possible physical operations. The objective is to provide a toolbox for carrying out fundamental tasks on a given

* A full version has been made available on arXiv [7]

arrangement of particles, such as bounding, counting, copying, reflecting, rotating or scaling a large given shape. The key idea is to use the arrangement itself as a storage device, similar to a higher-dimensional Turing machine with geometric properties.

1.1 Our Results

We consider an arrangement P of N pixel-shaped particles, on which a single finite-state robot can perform a limited set of operations; see Section 2 for a precise model description. Our goal is to develop strategies for evaluating and modifying the arrangement by defining sequences of transformations and providing metrics for such sequences. In particular, we present the following; a full technical overview is given in Table 1.

- We give a time- and space-efficient method for determining the bounding box of P .
- We show that we can simulate a Turing machine with our model.
- We provide a counter for evaluating the number N of tiles forming P , as well as the number of corner pixels of P .
- We develop time- and space-efficient algorithms for higher-level operations, such as copying, reflecting, rotating or scaling P .

| Problem | Tile Complexity | Time Complexity | Space Complexity |
|----------------|-----------------------------|--|-----------------------------------|
| Bounding Box | $\mathcal{O}(\partial P)$ | $\mathcal{O}(wh \max(w, h))$ | $\mathcal{O}(w + h)$ |
| Counting: | | | |
| N Tiles | $\mathcal{O}(\log N)^*$ | $\mathcal{O}(\max(w, h) \log N + N \min(w, h))$ | $\mathcal{O}(\max(w, h))$ |
| k Corners | $\mathcal{O}(\log k)^*$ | $\mathcal{O}(\max(w, h) \log k + k \min(w, h) + wh)$ | $\mathcal{O}(\max(w, h))$ |
| Function: | | | |
| Copy | $\mathcal{O}(N)^*$ | $\mathcal{O}(wh^2)$ | $\mathcal{O}(wh)$ |
| Reflect | $\mathcal{O}(\max(w, h))^*$ | $\mathcal{O}((w + h)wh)$ | $\mathcal{O}(w + h)$ |
| Rotate | $\mathcal{O}(w + h)^*$ | $\mathcal{O}((w + h)wh)$ | $\mathcal{O}(w - h \max(w, h))$ |
| Scaling by c | $\mathcal{O}(cN)$ | $\mathcal{O}((w^2 + h^2)c^2N)$ | $\mathcal{O}(cwh)$ |

Table 1: Results of this paper. N is the number of tiles in the given shape P , w and h its width and height. $(*)$ is the number of auxiliary tiles after constructing the bounding box.

1.2 Related Work

There is a wide range of related work; due to space limitations, we can only present a small selection.

A very successful model considers self-assembling DNA tiles (e.g., [5, 15]) that form complex shapes based on the interaction of different glues along their edges; however, no active agents are involved, and composition is the result of chemical and physical diffusion.

The setting of a finite-automaton robot operating on a set of tiles in a grid was introduced in [10], where the objective is to arrange a given set of tiles

into an equilateral triangle. An extension studies the problem of recognizing certain shapes [9]. We use a simplified variant of the model underlying this line of research that exhibits three main differences: First, for ease of presentation we consider a square grid instead of a triangular grid. Second, our model is less restrictive in that the robot can create and destroy tiles at will instead of only being able to transport tiles from one position to another. Finally, we allow the robot to move freely throughout the grid instead of restricting it to move along the tile structure.

Other previous related work includes shape formation in a number of different models: in the context of agents walking DNA-based shapes [17, 20, 22]; in the Amoebot model [6]; in modular robotics [11]; in a variant of population protocols [13]; in the nubot model [23]. Work focusing on a setting of robots on graphs includes network exploration [3], maze exploration [1], rendezvous search [16], intruder capture and graph searching [2, 8]. For a connection to other approaches to agents moving tiles, e.g., see [4, 19].

Although the complexity of our model is very restricted, actually realizing such a system, for example using complex DNA nanomachines, is currently still a challenging task. However, on the practical side, recent years have seen significant progress towards realizing systems with the capabilities of our model. For example, it has been shown that nanomachines have the ability to act like the head of a finite automaton on an input tape [17], to walk on a one- or two-dimensional surface [12, 14, 22], and to transport cargo [18, 20, 21].

2 Preliminaries

We consider a single *robot* that acts as a *deterministic finite automaton*. The robot moves on the (infinite) *grid* $G = (\mathbb{Z}^2, E)$ with edges between all pairs of nodes that are within unit distance. The nodes of G are called *pixels*. Each pixel is in one of two *states*: It is either *empty* or *occupied* by a particle called *tile*. A *diagonal pair* is a pair of two pixels $(x_1, y_1), (x_2, y_2)$ with $|x_1 - x_2| = |y_1 - y_2| = 1$ (see Fig. 1, left and middle). A *polyomino* is a connected set of tiles $P \subset \mathbb{Z}^2$ such that for all diagonal pairs $p_1, p_2 \in P$ there is another tile $p \in P$ that is adjacent to p_1 and adjacent to p_2 (see Fig. 1 middle and right). We say that P is *simple* if it has no holes, i.e., if there is no empty pixel $p \in G$ that lies inside a cycle formed by occupied tiles. Otherwise, P is *non-simple*.

The *a-row* of P is the set of all pixels $(x, a) \in P$. We say that P is *y-monotone* if the *a-row* of P is connected in G for each $a \in \mathbb{Z}$ (see Fig. 1 right). Analogously, the *a-column* of P is the set of all pixels $(a, y) \in P$ and P is called *x-monotone* if the *a-column* of P is connected in G for each $a \in \mathbb{Z}$. The *boundary* ∂P of P is the set of all tiles of P that are adjacent to an empty pixel or build a diagonal pair with an empty pixel (see Fig. 1 right).

A *configuration* consists of the states of all pixels and the robot's location and state. The robot can transform a configuration into another configuration using a sequence of look-compute-move steps as follows. In each step, the robot acts according to a *transition function* δ . This transition function maps a pair

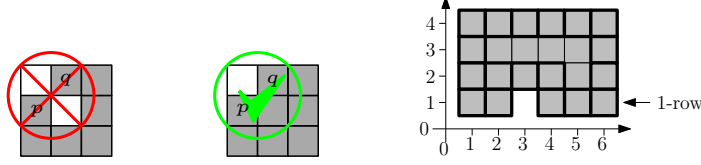


Fig. 1: Left: An illegal diagonal pair (p, q) . Middle: An allowed diagonal pair (p, q) . Right: A polyomino P with its boundary ∂P (tiles with bold outline). P is x -monotone but not y -monotone, because the 1-row is not connected.

(p, b) containing the state of the robot and the state of the current pixel to a triple (q, c, d) , where q is the new state of the robot, c is the new state of the current pixel, and $d \in \{\text{up, down, left, right}\}$ is the direction the robot moves in. In other words, in each step, the robot checks its state p and the state of the current pixel b , computes $(q, c, d) = \delta(p, b)$, changes into state q , changes the state of the current pixel to c if $c \neq b$, and moves one step into direction d .

Our goal is to develop robots transforming an unknown initial configuration S into a target configuration $T(S)$. We assess the efficiency of a robot by several metrics: (1) *Time complexity*, i.e., the total number of steps performed by the robot until termination, (2) *space complexity*, i.e., the total number of visited pixels outside the bounding box of the input polyomino P , and (3) *tile complexity*, i.e., the maximum number of tiles on the grid minus the number of tiles in the input polyomino P .

3 Basic Tools

A robot can check the states of all pixels within a constant distance by performing a tour of constant length. Thus, from now on we assume that a robot can check within a single step all eight pixels that are adjacent to the current position r or build a diagonal pair with r .

3.1 Bounding Box Construction

We describe how to construct a bounding box of a given polyomino P in the form of a zig-zag as shown in Fig. 2. We can split the bounding box into an *outer lane* and an *inner lane* (see Fig. 2). This allows distinguishing tiles of P and tiles of the bounding box.

Our construction proceeds in three phases. (i) We search for an appropriate starting position. (ii) We wrap a zig-zag path around P . (iii) We finalize this path to obtain the bounding box.

For **phase (i)**, we simply search for a local minimum in the y -direction. We break ties by taking the tile with minimal x -coordinate, i.e., the leftmost tile. From the local minimum, we go two steps further to the left. If we land on a tile, this tile belongs to P and we restart phase (i) from this tile. Otherwise we have found a possible *start position*.

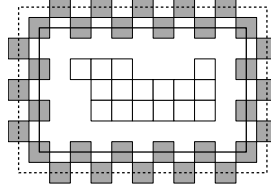


Fig. 2: A Polyomino P (white) surrounded by a bounding box of tiles (gray) on the inner lane (solid line) and tiles on the outer lane (dashed line).

In **phase (ii)**, we start constructing a path that will wrap around P . We start placing tiles in a zig-zag manner in the upwards direction. While constructing the path, three cases may occur.

1. At some point we lose contact with P , i.e., there is no tile at distance two from the inner lane. In this case, we do a right turn and continue our construction in the new direction.
2. Placing a new tile produces a conflict, i.e., the tile shares a corner or a side with a tile of P . In this case, we shift the currently constructed side of the bounding box outwards until no more conflict occurs. This shifting process may lead to further conflicts, i.e., we may not be able to further shift the current side outwards. If this happens, we deconstruct the path until we can shift it outwards again (see Fig. 3). In this process, we may be forced to deconstruct the entire bounding box we have built so far, i.e., we remove all tiles of the bounding box including the tile in the starting position. In this case we know that there must be a tile of P to the left of the start position; we move to the left until we reach such a tile and restart phase (i).
3. Placing a new tile closes the path, i.e., we reach a tile of the bounding box we have created so far. We proceed with phase (iii).

Phase (iii): Let t be the tile that we reached at the end of phase (ii). We can distinguish two cases: (i) At t , the bounding box splits into three different directions (shown as the tile with black-and-white diagonal stripes in Fig. 4), and (ii) the bounding box splits into two different directions. In the latter case we are done, because we can only reach the bottom or left side of the bounding box. In the first case we have reached a right or top side of the bounding box. From t we can move to the tile where we started the bounding box construction, and remove the bounding box parts until we reach t . Now the bounding box splits in two directions at t . Because we have reached a left or top side of the bounding box, we may not have built a convex shape around P (see Fig. 4 left). This can be dealt with by straightening the bounding box in an analogous fashion, e.g., by pushing the line to the right of t down.

Theorem 1. *The described strategy builds a bounding box that encloses the given polyomino P of width w and height h . Moreover, the strategy terminates after $\mathcal{O}(\max(w, h) \cdot wh)$ steps, using at most $\mathcal{O}(|\partial P|)$ auxiliary tiles and only $\mathcal{O}(w + h)$ of additional space. The running time in the best case is $\mathcal{O}(wh)$.*

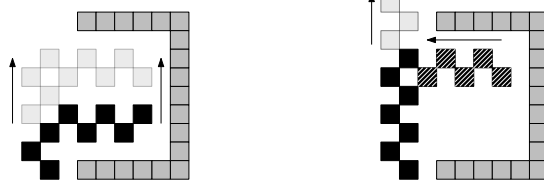


Fig. 3: Left: Further construction is not possible. Therefore, we shift the line upwards (see light gray tiles). Right: A further shift produces a conflict with the polyomino. Thus, we remove tiles (diagonal stripes) and proceed with the shift (light gray) when there is no more conflict.

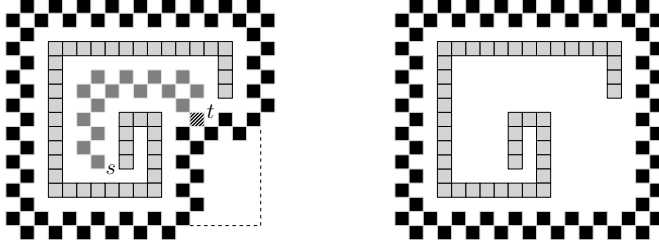


Fig. 4: Left: During construction, starting in s , we reach a tile t (diagonal stripes) at which the bounding box is split into three directions. The part between s and t (dark gray tiles) can be removed, followed by straightening the bounding box along the dashed line. Right: The final bounding box.

Proof. Correctness: We show that (a) the bounding box will enclose P ; and (b) whenever we make a turn or shift a side of the bounding box, we find a tile with distance two from the bounding box, i.e., we will not build an infinite line.

First note that we only make a turn after encountering a tile with distance two to the inner lane. This means that we will “gift wrap” the polyomino until we reach the start. When there is a conflict (i.e., we would hit a tile of P with the current line), we shift the current line. Thus, we again find a tile with distance two to the inner lane. This also happens when we remove the current line and extend the previous one. After a short extension we make a turn and find a tile with distance two to the inner lane, meaning that we make another turn at some point. Therefore, we do not construct an infinite line, and the construction remains finite.

Time: To establish a runtime of $\mathcal{O}(wh)$, we show that each pixel lying in the final bounding box is visited only a constant number of times. Consider a tile t that is visited for the first time. We know that t gets revisited again if the line through gets shifted or removed. When t is no longer on the bounding box, we can visit t again while searching for the start tile. Thus, t is visited at most four times by the robot, implying a running time of $\mathcal{O}(wh)$ unit steps. However, it may happen that we have to remove the bounding box completely and have to restart the local minimum search. In this case, there may be tiles that can

be visited up to $\max(w, h)$ times (see Fig. 5). Therefore, the running time is $\mathcal{O}(\max(w, h) \cdot wh)$ in the worst-case.

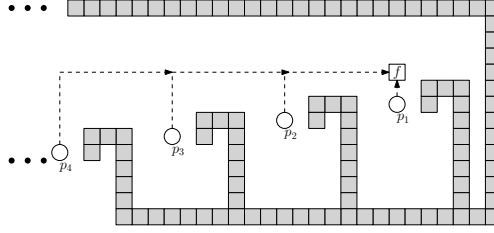


Fig. 5: A worst-case example for building a bounding box. The positions p_1 to p_4 denote the first, second, third and fourth starting position of the robot during bounding box construction. With each restart, the pixel f is visited at least once. Therefore, f can be visited $\Omega(w)$ times.

Auxiliary Tiles: We now show that we need at most $\mathcal{O}(|\partial P|)$ many auxiliary tiles at any time. Consider a tile t of ∂P from which we can shoot a ray to a tile of the bounding box (or the intermediate construction), such that no other tile of P is hit. For each tile t of ∂P , there are at most four tiles t_1, \dots, t_4 of the bounding box. We can charge the cost of t_1, \dots, t_4 to t , which is constant. Thus, each tile in ∂P has been charged by $\mathcal{O}(1)$. However, there are still tiles on the bounding box that have not been charged to a tile of ∂P , i.e., tiles that lie in a curve of the bounding box.

Consider a locally convex tile t , i.e., a tile at which we can place a 2×2 square solely containing t . Each turn of the bounding box can be charged to a locally convex tile. Note that there are at most four turns that can be charged to a locally convex tile. For each turn, there are at most four tiles that are charged to a locally convex tile. Thus, each tile of ∂P has constant cost, i.e., we need at most $\mathcal{O}(\partial P)$ auxiliary tiles.

It is straightforward to see that we need $\mathcal{O}(w + h)$ additional space. \square

3.2 Binary Counter

For counting problems, a binary counter is indispensable, because we are not able to store non-constant numbers. The binary counter for storing an n -bit number consists of a base-line of n tiles. Above each tile of the base-line there is either a tile denoting a 1, or an empty pixel denoting 0. Given an n -bit counter we can *increase* and *decrease* the counter by 1 (or by any constant c), and *extend* the counter by one bit. The latter operation will only be used in an increase operation. In order to perform an increase or decrease operation, we move to the least significant bit and start flipping the bit (i.e., remove or place tiles).



Fig. 6: Left: A 4-bit counter with decimal value 10 (1010 in binary). Middle: The binary counter increased by one. Right: The binary counter decreased by one.

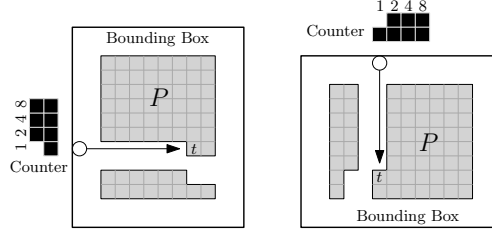


Fig. 7: An 8×8 square P in a bounding box, being counted row-by-row (left) or column-by-column (right). The robot has already counted 14 tiles of P and stored this information in the binary counter. The arrow shows how the robot (\circ) moves to count the next tile t .

4 Counting Problems

The constant memory of our robot poses several challenges when we want to count certain elements of our polyomino, such as tiles or corners. Because these counts can be arbitrarily large, we cannot store them in the state space of our robot. Instead, we have to make use of a binary counter. This requires us to move back and forth between the polyomino and the counter. Therefore, we must be able to find and identify the counter coming from the polyomino and to find the way back to the position where we stopped counting.

This motivates the following strategy for counting problems. We start by constructing a slightly extended bounding box. We perform the actual counting by shifting the polyomino two units downwards or to the left, one tile at a time. After moving each tile, we return to the counter and increase it if necessary. We describe further details in the next two sections, where we present algorithms for counting the number of tiles or corners in a polyomino.

4.1 Counting Tiles

The total number of tiles in our polyomino can be counted using the strategy outlined above, increasing the counter by one after each moved tile.

Theorem 2. *Let P be a polyomino of width w and height h with N tiles for which the bounding box has already been created. Counting the number of tiles in P can be done in $\mathcal{O}(\max(w, h) \log N + N \min(w, h))$ steps using $\mathcal{O}(\max(w, h))$ of additional space and $\mathcal{O}(\log N)$ auxiliary tiles.*

Proof. In a first step, we determine whether the polyomino's width is greater than its height or vice versa. We can do this by moving to the lower left corner

of the bounding box and then alternately moving up and right one step until we meet the bounding box again. We can recognize the bounding box by a local lookup based on its zig-zag shape that contains tiles only connected by a corner, which cannot occur in a polyomino. The height is at least as high as the width if we end up on the right side of the bounding box. In the following, we describe our counting procedure for the case that the polyomino is higher than wide; the other case is analogous.

We start by extending the bounding box by shifting its bottom line down by two units. Afterwards we create a vertical binary counter to the left of the bounding box. We begin counting tiles in the bottom row of the polyomino. We keep our counter in a column to the left of the bounding box such that the least significant bit at the bottom of the counter is in the row, in which we are currently counting. We move to the right into the current row until we find the first tile. If this tile is part of the bounding box, the current row is done. In this case, we move back to the counter, shift it upwards and continue with the next row until all rows are done. Otherwise, we simply move the current tile down two units, return to the counter and increment it. For an example of this procedure, refer to Fig. 7.

For each tile in the polyomino, we use $\mathcal{O}(\min(w, h))$ steps to move to the tile, shift it and return to the counter; incrementing the counter itself only takes $\mathcal{O}(1)$ amortized time per tile. For each empty pixel we have cost $\mathcal{O}(1)$. In addition, we have to shift the counter $\max(w, h)$ times. Thus, we need $\mathcal{O}(\max(w, h) \log N + \min(w, h)N + wh) = \mathcal{O}(\max(w, h) \log N + \min(w, h)N)$ unit steps. We only use $\mathcal{O}(\log N)$ auxiliary tiles in the counter, in addition to the bounding box.

In order to achieve $\mathcal{O}(1)$ additional space, we modify the procedure as follows. Whenever we move our counter, we check whether it extends into the space above the bounding box. If it does, we reflect the counter vertically, such that the least significant bit is at the top in the row, in which we are currently counting. This requires $\mathcal{O}(\log^2 N)$ extra steps and avoids using $\mathcal{O}(\log N)$ additional space above the polyomino.

4.2 Counting Corners

In this section, we present an algorithm to count reflex or convex corners of a given polyomino.

Theorem 3. *Let P be a polyomino of width w and height h with n convex (reflex) corners for which the bounding box has already been created. Counting the number of k convex (reflex) corners in P can be done in $\mathcal{O}(\max(w, h) \log k + k \min(w, h) + wh)$ steps, using $\mathcal{O}(\max(w, h))$ of additional space and $\mathcal{O}(\log k)$ auxiliary tiles.*

Proof. We use the same strategy as for counting tiles in Theorem 2 and retain the same asymptotic bounds on running time, auxiliary tiles and additional space. It remains to describe how we recognize convex and reflex corners.

Whenever we reach a tile t that we have not yet considered so far, we examine its neighbors x_1, \dots, x_6 , as shown in Fig. 8. The tile t has one convex corner for

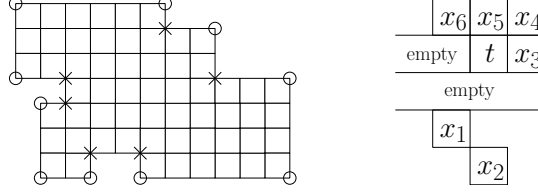


Fig. 8: *Left:* A polyomino and its convex corners (\circ) and reflex corners (\times). *Right:* After reaching a new tile t , we have to know which of the pixel x_1 to x_6 are occupied to decide how many convex (reflex) corners t has.

each pair $(x_1, x_2), (x_2, x_3), (x_3, x_5), (x_5, x_1)$ that does not contain a tile, and no convex corner is contained in more than one tile of the polyomino. As there are at most four convex corners per tile, we can simply store this number in the state space of our robot, return to the counter, and increment it accordingly.

A reflex corner is shared by exactly three tiles of the polyomino, so we have to ensure that each corner is counted exactly once. We achieve this by only counting a reflex corner if it is on top of our current tile t and was not already counted with the previous tile x_1 . In other words, we count the upper right corner of t as a reflex corner if exactly two of x_3, x_4, x_5 are occupied; we count the upper left corner of t as reflex corner if x_6 and x_5 are present and x_1 is not. In this way, all reflex corners are counted exactly once.

5 Transformations with Turing machines

In this section we develop a robot that transforms a polyomino P_1 into a required target polyomino P_2 . In particular, we encode P_1 and P_2 by strings $S(P_1)$ and $S(P_2)$ whose characters are from $\{0, 1, \sqcup\}$ (see Fig. 9 left and Definition 1). If there is a Turing machine transforming $S(P_1)$ into $S(P_2)$, we can give a strategy that transforms P_1 into P_2 (see Theorem 4).

We start with the definition of the encodings $S(P_1)$ and $S(P_2)$.

Definition 1. Let $R := R(P)$ be the polyomino forming the smallest rectangle containing a given polyomino P (see Fig. 9 right). We represent P by the concatenation $S(P) := S_1 \sqcup S_2 \sqcup S_3$ of three bit strings S_1 , S_2 , and S_3 separated by blanks \sqcup . In particular, S_1 and S_2 are the height and width of R . Furthermore, we label each tile of R by its rank in the lexicographic decreasing order w.r.t. y - and x -coordinates with higher priority to y -coordinates (see Fig. 9 right). Finally, S_3 is an $|R|$ -bit string $b_1, \dots, b_{|R|}$ with $b_i = 1$ if and only if the i th tile in R also belongs to P .

Theorem 4. Let P_1 and P_2 be two polyominos with $|P_1| = |P_2| = N$. There is a strategy transforming P_1 into P_2 if there is a Turing machine transforming $S(P_1)$ into $S(P_2)$. The robot needs $\mathcal{O}(\partial P_1 + \partial P_2 + S_{TM})$ auxiliary tiles, $\mathcal{O}(N^4 + T_{TM})$ steps, and $\Theta(N^2 + S_{TM})$ of additional space, where T_{TM} and S_{TM} are the number of steps and additional space needed by the Turing machine.

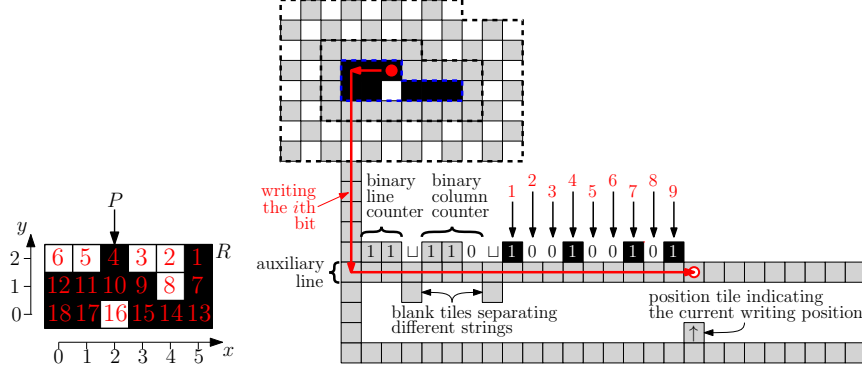


Fig. 9: Left: An example showing how to encode a polyomino of height $h = 3$ ($S_1 = 11$ in binary) and width $w = 6$ ($S_2 = 110$ in binary) by $S(P) = S_1 \sqcup S_2 \sqcup S_3 = 11 \sqcup 110 \sqcup 10010010111111011$, where S_3 is the string of 18 bits that represent tiles (black, 1 in binary) and empty pixel (white, 0 in binary), proceeding from high bits to low bits. Right: Phase (2) writing $S(P)$ (currently the 10th bit) on a horizontal auxiliary line.

Proof. Our robot works in five phases: Phase (1) constructs a slightly modified bounding box of P_1 (see Fig. 9). Phase (2) constructs a shape representing $S(P_1)$. Bit-by-bit, the robot writes $S(P_1)$ onto a horizontal auxiliary line (see Fig 9). In order to remember the position, at which the previous bit was written, we use a *position tile* placed on another horizontal auxiliary line. Phase (3) simulates the Turing machine that transforms $S(P_1)$ into $S(P_2)$. Phase (4) is the reversed version of Phase (2), and Phase (5) is the reversed version of Phase (1). As Phase (4) and Phase (5) are the reversed version of Phase (1) and Phase (2), we still have to discuss how to realize Phases (1), (2), and (3), for which we refer to the full version [7]. This concludes the proof of Theorem 4.

6 CAD Functions

As already seen, we can transform a given polyomino by any computable function by simulating a Turing machine. However, this requires a considerable amount of space. In this section, we present realizations of common functions in a more space-efficient manner.

6.1 Copying a Polyomino

Copying a polyomino P has many application. For example, we can apply algorithms that may destroy P after copying P into free space. In the following, we describe the *copy* function that copies each column below the bounding box, as seen in Fig. 10 (a row-wise copy can be described analogously).

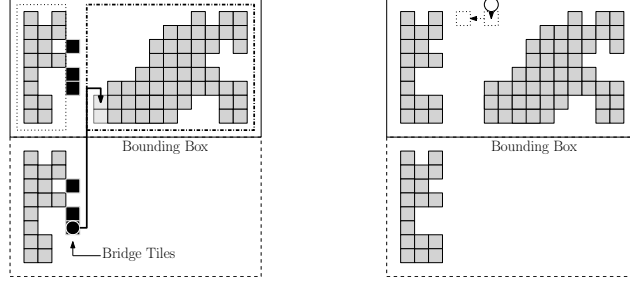


Fig. 10: Left: Intermediate step while copying a column of a polyomino (gray tiles) with bridges (black tiles). Tiles in the left box (dotted) are already copied, the tile of P in the right box (dash-dotted) are not copied yet. The robot (\circ) moves to next pixel. Right: When the column is copied the bridges get removed and the robot proceeds with the next column.

Our strategy to copy a polyomino is as follows: After constructing the bounding box, we copy each column of the bounding box area into free space. This, however, comes with a problem. As the intersection of the polyomino with a column may consist of more than one component, which may be several units apart, we have to be sure that the right amount of empty pixels are copied. This problem can be solved by using *bridges*, i.e., auxiliary tiles denoting empty pixels. To distinguish between tiles of P and bridges, we use two lanes (i) a left lane containing tiles of P and (ii) a right lane containing bridge tiles (see Fig. 10).

To copy an empty pixel, we place a tile two unit steps to the left. This marks a part of a bridge. Then we move down and search for the first position, at which there is no copied tile of P to the left, nor a bridge tile. When this position is found, we place a tile, denoting an empty position.

To copy a tile t of P , we move this tile three unit steps to the left. Afterwards, we move downwards following the tiles and bridges until we reach a free position and place a tile, denoting a copy of t . When we reach the bottom of a column, we remove any tile representing a bridge tile and proceed with the next column (see Fig. 10 right).

Theorem 5. *Copying a polyomino P column-wise can be done within $\mathcal{O}(wh^2)$ unit steps using $\mathcal{O}(N)$ of auxiliary tiles and $\mathcal{O}(h)$ additional space.*

Proof. Consider the strategy described above. It is straightforward to see that by maintaining the bridges, we can ensure that each tile of P is copied to the correct position. As there are $\mathcal{O}(wh)$ many pixels that we have to copy with cost of $\mathcal{O}(h)$ per pixel, the strategy needs $\mathcal{O}(wh^2)$ unit steps to terminate.

Now consider the number of auxiliary tiles. We need N tiles for the copy and $\mathcal{O}(h)$ tiles for bridges, which are reused for each column. Thus, we need $\mathcal{O}(N)$ auxiliary tiles in total. Because we place the copied version of P beneath P , we need $\mathcal{O}(h)$ additional space in the vertical direction. \square

6.2 Reflecting a Polyomino

In this section we show how to perform a vertical reflection on a polyomino P (a horizontal reflection is done analogously). Assume that we already built the bounding box. Then we shift the bottom side of the bounding box one unit down, such that we have two units of space between the bounding box and P . We start with the bottom-most row r and copy r in reversed order beneath the bounding box using bridges, as seen in the previous section. After the copy process, we delete r from P and shift the bottom side of the bounding box one unit up. Note that we still have two units of space between the bounding box and P . We can therefore repeat the process until no tile is left.

Theorem 6. *Reflecting a polyomino P vertically can be done in $\mathcal{O}(w^2h)$ unit steps, using $\mathcal{O}(w)$ of additional space and $\mathcal{O}(w)$ auxiliary tiles.*

Proof. For each pixel within the bounding box we have to copy this pixel to the desired position. This costs $\mathcal{O}(w)$ steps, i.e., we have to move to the right side of the boundary, move a constant number of steps down and move $\mathcal{O}(w)$ to the desired position. These are $\mathcal{O}(w)$ steps per pixel, $\mathcal{O}(w^2h)$ unit steps in total.

It can be seen in the described strategy that we only need a constant amount of additional space in one dimension because we are overwriting the space that was occupied by P . This implies a space complexity of $\mathcal{O}(w)$. Following the same argumentation of Theorem 5, we can see that we need $\mathcal{O}(w)$ auxiliary tiles. \square

Corollary 1. *Reflecting a polyomino P vertically and horizontally can be done in $\mathcal{O}(wh(w+h))$ unit steps, using $\mathcal{O}(w+h)$ additional space and $\mathcal{O}(w)$ auxiliary tiles.*

6.3 Rotating a Polyomino

Rotating a polyomino presents some additional difficulties, because the dimension of the resulting polyomino has dimensions $h \times w$ instead of $w \times h$. Thus, we may need non-constant additional space in one dimension, e.g., if one dimension is large compared to the other dimension. A simple approach is to copy the rows of P bottom-up to the right of P . This allows us to rotate P with $\mathcal{O}(wh)$ additional space. For now, we assume that $h \geq w$.

We now propose a strategy that is more compact. The strategy consists of two phases: First build a reflected version of our desired polyomino, then reflect it to obtain the correct polyomino.

After constructing the bounding box, we place a tile t_1 in the bottom-left corner of the bounding box, which marks the bottom-left corner of P (see Fig. 11). We also extend the bounding box at the left side and the bottom side by six units. This gives us a width of seven units between the polyomino and the bounding box at the left and bottom side. Now we can move the first column rotated in clockwise direction five units below P (which is two units above the bounding box), as follows.

We use two more tiles t_c denoting the column, at which we have to enter the bounding box, and t_2 marking the pixel that has to be copied next (see Fig. 11). We can maintain these tiles as in a copy process to always copy the next pixel: If t_2 reached t_1 , i.e., we would place t_2 on t_1 , then we know that we have finished the column and proceed with the row by moving t_2 to the right of t_1 . We can copy this pixel to the desired position again by following the bridges and tiles that now make a turn at some point (see Fig. 11 for an example). Note that we cannot place the row directly on top of the column or else we may get conflicts with bridges. We therefore build the row one unit to the left and above the column. Also note that during construction of the first column or during the shifting we may hit the bounding box. In this case we extend the bounding box by one unit.

After constructing a column and a row, we move the constructed row one unit to the left and two units down, and we move the column one unit down and two units left. This gives us enough space to construct the next column and row in the same way.

When all columns and rows are constructed, we obtain a polyomino that is a reflected version of our desired polyomino. It is left to reflect horizontally to obtain a polyomino rotated in counter-clockwise direction, or vertically to obtain a polyomino that is rotated in clockwise direction. This can be done with the strategy described above.

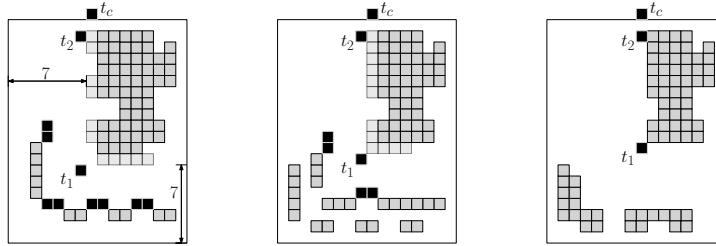


Fig. 11: Left: Constructing the first column and row (light gray in the polyomino). Middle: First row and column have been moved away by a constant number of steps to make space for the next row and column. Right: Merging the first two columns and rows.

Theorem 7. *There is a strategy to rotate a polyomino P by 90° within $\mathcal{O}((w+h)wh)$ unit steps, using $\mathcal{O}(|w-h|h)$ of additional space and $\mathcal{O}(w+h)$ auxiliary tiles.*

Proof. Like in other algorithms, the number of steps to copy the state of a pixel to the desired position is bounded by $\mathcal{O}(w+h)$. Shifting the constructed row and column also takes the same number of steps. Therefore, constructing the reflected version of our desired polyomino needs $\mathcal{O}((w+h)wh)$ unit steps. Additionally we may have to extend one side of the bounding box. This can happen at most

$\mathcal{O}(|w - h|)$ times, with each event needing $\mathcal{O}(h)$ unit steps. Because $\mathcal{O}(|w - h|)$ can be bounded by $\mathcal{O}(w + h)$, this does not change the total time complexity.

Because the width of the working space increases by $\mathcal{O}(|w - h|)$ and the height only increases by $\mathcal{O}(1)$, we need $\mathcal{O}(|w - h|h)$ of additional space. It is straightforward to see that we need a total of $\mathcal{O}(\max(w + h))$ auxiliary tiles. \square

6.4 Scaling a Polyomino

Scaling a polyomino P by a factor c replaces each tile by a $c \times c$ square. This can easily be handled by our robot.

Theorem 8. *Given a constant c , the robot can scale the polyomino by c within $\mathcal{O}((w^2 + h^2)c^2N)$ unit steps using $\mathcal{O}(c^2wh)$ additional tiles and $\mathcal{O}(c^2N)$ additional space.*

Proof. After constructing the bounding box, we place a tile denoting the current column. Suppose we are in the i -th column C_i . Then we shift all columns that lie to the right of C_i by c units to the right with cost of $\mathcal{O}((w - i) \cdot \sum_{j=i+1}^h cN_{C_j}) \subseteq \mathcal{O}(wcN)$, where N_{C_j} is the number of tiles in the j th column. Because c is a constant, we can always find the next column that is to be shifted. Afterwards, we copy C_i exactly $c - 1$ times to the right of C_i , which costs $\mathcal{O}(N_{C_i}c)$ unit steps. Thus, extending a single row costs $\mathcal{O}(c \cdot (wN + N_{C_i}))$ and hence extending all rows costs $\mathcal{O}(cw^2N)$ unit steps in total.

Extending each row is done analogously. However, because each tile has already been copied c times, we obtain a running time of $\mathcal{O}(c^2h^2N)$. This implies a total running time of $\mathcal{O}((w^2 + h^2)c^2N)$. The proof for the tile and space complexity is straightforward. \square

7 Conclusion

We have given a number of tools and functions for manipulating an arrangement of tiles by a single robotic automaton. There are various further questions, including more complex operations, as well as sharing the work between multiple robots. These are left for future work.

References

1. M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. 19th Annual Symposium on Foundations of Computer Science*, pages 132–142, 1978.
2. A. Bonato and R. J. Nowakowski. *The Game of Cops and Robbers on Graphs*. AMS, 2011.
3. S. Das. Mobile agents in distributed computing: Network exploration. *Bulletin of the European Association for Theoretical Computer Science*, 109:54–69, 2013.
4. E. Demaine, M. Demaine, M. Hoffmann, and J. O’Rourke. Pushing blocks is hard. *Computational Geometry*, 26(1):21–36, 2003.

5. E. D. Demaine, S. P. Fekete, C. Scheffer, and A. Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Computer Science*, 671:4–18, 2017.
6. Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal shape formation for programmable matter. In *Proc. 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 289–299, 2016.
7. S. P. Fekete, R. Gmyr, S. Hugo, P. Keldenich, C. Scheffer, and A. Schmidt. CADbots: Algorithmic Aspects of Finite Automata for Manipulating Programmable Matter. *arXiv preprint arXiv:1810.06360*, 2018.
8. F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, jun 2008.
9. 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*, 2018. To appear.
10. R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler, and T. Strothmann. Forming tile shapes with simple robots. In *23rd International Conference on DNA Computing and Molecular Programming*, 2018. To appear.
11. F. Hurtado, E. Molina, S. Ramaswami, and V. Sacristán. Distributed reconfiguraiton of 2D lattice-based modular robotic systems. *Autonomous Robots*, 38(4):383–413, 2015.
12. K. Lund, A. Manzo, N. Dabby, N. Michelotti, A. Johnson-Buck, J. Nangreave, S. Taylor, R. Pei, M. Stojanovic, N. Walter, and E. Winfree. Molecular robots guided by prescriptive landscapes. *Nature*, 465(7295):206–210, 2010.
13. O. Michail and P. G. Spirakis. Simple and efficient local codes for distributed stable network construction. *Distributed Computing*, 29(3):207–237, 2016.
14. T. Omabegho, R. Sha, and N. Seeman. A bipedal DNA Brownian motor with coordinated legs. *Science*, 324(5923):67–71, 2009.
15. M. J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.
16. A. Pelc. Deterministic rendezvous in networks: A comprehensive survey. *Networks*, 59(3):331–347, 2012.
17. J. H. Reif and S. Sahu. Autonomous programmable DNA nanorobotic devices using dnazymes. *Theoretical Computer Science*, 410:1428–1439, 2009.
18. J. Shin and N. Pierce. A synthetic DNA walker for molecular transport. *Journal of the American Chemical Society*, 126:4903–4911, 2004.
19. Y. Terada and S. Murata. Automatic modular assembly system and its distributed control. *International Journal of Robotics Research*, 27(3–4):445–462, 2008.
20. A. Thubagere, W. Li, R. Johnson, Z. Chen, S. Doroudi, Y. Lee, G. Izatt, S. Wittman, N. Srinivas, D. Woods, E. Winfree, and L. Qian. A cargo-sorting DNA robot. *Science*, 357(6356), 2017.
21. Z. Wang, J. Elbaz, and I. Willner. A dynamically programmed DNA transporter. *Angewandte Chemie International Edition*, 51(48):4322–4326, 2012.
22. S. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A. Turberfield. A DNA-based molecular motor that can navigate a network of tracks. *Nature Nanotechnology*, 7(3):169–173, 2012.
23. 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 *Proc. 4th Conference of Innovations in Theoretical Computer Science*, pages 353–354, 2013.

CADbots: Algorithmic Aspects of Manipulating Programmable Matter with Finite Automata

(Preprint — Submitted for journal publication to *Algorithmica*)

Sándor P. Fekete^{1*} · Robert Gmyr² ·
Sabrina Hugo¹ · Phillip Keldenich¹ ·
Christian Scheffer¹ · Arne Schmidt¹

Abstract We contribute results for a set of fundamental problems in the context of programmable matter by presenting algorithmic methods for evaluating and manipulating a collective of particles by a finite automaton that can neither store significant amounts of data, nor perform complex computations, and is limited to a handful of possible physical operations. We provide a toolbox for carrying out fundamental tasks on a given arrangement of particles, using the arrangement itself as a storage device, similar to a higher-dimensional Turing machine with geometric properties. Specific results include time- and space-efficient procedures for bounding, counting, copying, reflecting, rotating or scaling a complex given shape.

1 Introduction

When dealing with classic challenges of robotics, such as exploration, evaluation and manipulation of objects, traditional robot models are based on relatively powerful capabilities, such as the ability (1) to collect and store significant amounts of data, (2) perform intricate computations, and (3) execute complex physical operations. With the ongoing progress in miniaturization of devices, new possibilities emerge for exploration, evaluation and manipulation. However, dealing with small dimensions (as present in the context of micro-robots and even programmable matter) or large dimensions (as in the construction of large-scale structures in space) introduces a vast spectrum of

A preliminary extended abstract appears in the Proceedings of the 13th Workshop on Algorithmic Foundations of Robotics (WAFR), 2018 [16]. Phillip Keldenich was supported by the DFG Research Unit “Controlling Concurrent Change”, funding number FOR 1800, project FE407/17-2, “Conflict Resolution and Optimization”.

¹Department of Computer Science, TU Braunschweig, Germany. {fekete, hugo, keldenich, scheffer, aschmidt}@ibr.cs.tu-bs.de

²Department of Computer Science, University of Paderborn, Germany. robert@gmyr.net

*Corresponding author.

new difficulties and constraints. These include significant limitations to all three of the mentioned capabilities; challenges get even more pronounced in the context of complex small-scale or far-away systems, where there is a significant threshold between “internal” objects and sensors, and “external” control entities [2]. that can evaluate gathered data, extract important information, and provide guidance.

In this paper, we present algorithmic methods for evaluating and manipulating a collective of particles by agents of the most basic possible type: finite automata that can neither store significant amounts of data, nor perform complex computations, and are limited to a handful of possible physical operations. The objective is to provide a toolbox for carrying out fundamental tasks on a given arrangement of particles, such as bounding, counting, copying, reflecting, rotating or scaling a large given shape. A key idea is to use the arrangement itself as a storage device, similar to a higher-dimensional Turing machine with geometric properties.

1.1 Our Results

We consider an arrangement P of N square-shaped particles, on which a single finite-state robot can perform a limited set of operations; see Section 2 for a precise model description. Our goal is to develop first approaches for evaluating and modifying the arrangement by defining sequences of transformations and providing metrics for such sequences. In particular, we present the following; a full technical overview is given in Table 1.

- We give a time- and space-efficient method for determining the bounding box of P , i.e., the smallest axis-aligned box containing P .
- We show that we can simulate a Turing machine in our model.
- We provide a counter for evaluating the number N of particles forming P , as well as the number of corner pixels of P .
- We develop time- and space-efficient algorithms for higher-level operations also used in computer-aided design (CAD), such as copying, reflecting, rotating or scaling P .

1.2 Related Work

Practical motivation for our work arises both at very small and very large dimensions. See the overview in [2] for a discussion of work on particle computation and programmable matter, and [1, 24] for a description of possible applications for building large-scale structures in space.

There have also been a number of different, related basic models. Derakhshandeh et al. [9] introduced a fundamental concept for studying algorithmic approaches for extremely simple robots, called the *Amoebot model*. In this

| Problem | Particle Complexity | Time Complexity |
|--|--|--|
| Bounding Box | $\mathcal{O}(\partial P)$ | $\mathcal{O}(wh \max(w, h))$ |
| Counting: N particles k Corners | $\mathcal{O}(\log N)^*$ $\mathcal{O}(\log k)^*$ | $\mathcal{O}(\max(w, h) \log N + N \min(w, h))$ $\mathcal{O}(\max(w, h) \log k + k \min(w, h) + wh)$ |
| Function: Copy Reflect Rotate Scaling by c | $\mathcal{O}(N)^*$ $\mathcal{O}(\max(w, h))^*$ $\mathcal{O}(w + h)^*$ $\mathcal{O}(cN)$ | $\mathcal{O}(wh^2)$ $\mathcal{O}((w + h)wh)$ $\mathcal{O}((w + h)wh)$ $\mathcal{O}((w^2 + h^2)c^2 N)$ |

| Problem | Space Complexity | Dimension Complexity |
|--|---|---|
| Bounding Box | $\mathcal{O}(w + h)$ | $\mathcal{O}(1)$ |
| Counting: N particles k Corners | $\mathcal{O}(\max(w, h))$ $\mathcal{O}(\max(w, h))$ | $\mathcal{O}(1)$ $\mathcal{O}(1)$ |
| Function: Copy Reflect Rotate Scaling by c | $\mathcal{O}(wh)$ $\mathcal{O}(w + h)$ $\mathcal{O}(w + h + w - h \max(w, h))$ $\mathcal{O}(c^2 wh)$ | $\mathcal{O}(h)$ $\mathcal{O}(1)$ $\mathcal{O}(w - h + 1)$ $\mathcal{O}(c(w + h))$ |

Table 1 Results of this paper. N is the number of particles in the given shape P , w and h its width and height, and ∂P denotes the boundary of P . (*) is the particle complexity after constructing the bounding box. Particle Complexity denotes the maximum number of particles placed on the grid minus N during construction, time complexity denotes the total number of steps needed, space complexity denotes the total number of visited pixels outside the bounding box of P , and dimension complexity denotes the total number of rows and column visited outside the bounding box of P .

model, active particles move on hexagonal cells by expanding (and thus occupying two cells) and contracting. With only a few rules how the particles have to move, the particles can form shapes like lines, triangles or hexagons [11]. By first performing a leader election, further operations are possible [6, 14]. An algorithm for universal shapes formation was presented by Di Luna et al. [15] and by Derakhshandeh et al. [12]. The problem of coating an arbitrarily shaped object by particles was solved by Derakhshandeh et al. [13] and analyzed in [10]. Further models involving active particles only were introduced by Woods et al. [35] (*Nubot model*) and by Hurtado et al. [20] (modular robots). Other related work includes shape formation in a number of different models, such as agents walking on DNA-based shapes [28, 32, 34], or variants of population protocols [23].

The setting of a finite-automaton robot (as an active particle) operating on a set of passive particles in a grid, called the *hybrid model*, was introduced in [19], where the objective is to arrange a given set of particles into an equilateral triangle. An extension studies the problem of recognizing certain shapes [18]. We use a simplified variant of the model underlying this line of research that exhibits three main differences: First, for ease of presentation we consider a square grid instead of a triangular grid. Second, our model is less restrictive in that the robot can create and destroy particles at will instead of

only being able to transport particles from one position to another. It appears straightforward to adapt all our algorithms to the case where a robot has to bring particle to and from a location from and to a particle depot, albeit at potentially considerable overhead for additional travel time. Finally, we allow the robot to move freely throughout the grid instead of restricting it to move along the particle structure. As shown in [24], we are able to adapt all our presented algorithms to maintain connectivity during the runtime, provided that we are allowed to use at least two robots or one robot and a special marker.

Models based on automate and movable objects have also been studied in the context of one-dimensional arrays, e.g., pebble automata [29]. Work focusing on a setting of robots on graphs includes network exploration [5], maze exploration [3], rendezvous search [27], intruder capture and graph searching [4, 17], and black hole search [22]. For a connection to other approaches to agents moving tiles, e.g., see [7, 31].

Another widely considered model considers self-assembling DNA tiles (e.g., [8, 26]) that form complex shapes based on the interaction of different glues along their edges; however, no active agents are involved, and composition is the result of chemical and physical diffusion. Although the complexity of our model is very restricted, actually realizing such a system, for example using complex DNA nanomachines, is currently still a challenging task. However, on the practical side, recent years have seen significant progress towards realizing systems with the capabilities of our model. For example, it has been shown that nanomachines have the ability to act like the head of a finite automaton on an input tape [28], to walk on a one- or two-dimensional surface [21, 25, 34], and to transport cargo [30, 32, 33].

2 Preliminaries

We consider a single *robot* that acts as a *deterministic finite automaton*. The robot moves on the (infinite) *grid* $G = (\mathbb{Z}^2, E)$ with edges between all pairs of nodes that are within unit distance using Manhattan metric. The nodes of G are called *pixels*. Each pixel is in one of two *states*: It is either *empty* or *occupied*. A *particle* denotes an occupied pixel. A *diagonal pair* is a pair of two pixels $(x_1, y_1), (x_2, y_2)$ with $|x_1 - x_2| = |y_1 - y_2| = 1$ (see Fig. 1, left and middle). A *polyomino* is a connected set of particles $P \subset \mathbb{Z}^2$ such that for all diagonal pairs $p_1, p_2 \in P$ there is another particle $p \in P$ that is adjacent to p_1 and adjacent to p_2 (see Fig. 1 middle and right). We say that P is *simple* if it has no holes, i.e., if $G \setminus P$ is connected. Otherwise, P is *non-simple*.

The *a-row* of P is the set of all pixels $(x, a) \in P$. We say that P is *y-monotone* if the *a-row* of P is connected in G for each $a \in \mathbb{Z}$ (see Fig. 1 right). Analogously, the *a-column* of P is the set of all pixels $(a, y) \in P$ and P is called *x-monotone* if the *a-column* of P is connected in G for each $a \in \mathbb{Z}$. The *boundary* ∂P of P is the set of all particles of P that are adjacent to an empty pixel or that build a diagonal pair with an empty pixel (see Fig. 1 right). The

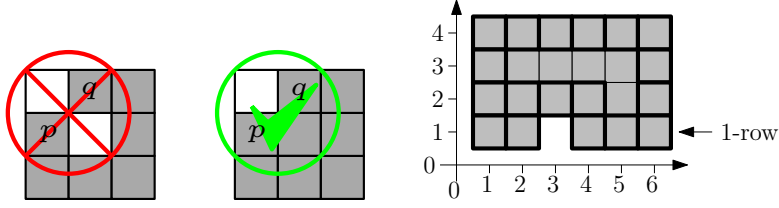


Fig. 1 Left: An illegal diagonal pair (p, q) . Middle: An allowed diagonal pair (p, q) . Right: A polyomino P with its boundary ∂P (particles with bold outline). P is x -monotone but not y -monotone, because the 1-row is not connected. Particles at position $(2,2)$ and $(4,2)$ are both building a diagonal pair with an empty pixel at position $(3,1)$ and therefore also belong to ∂P .

bounding box of a given polyomino P is defined as the smallest axis-aligned rectangle enclosing P .

A *configuration* consists of the states of all pixels and the robot's location and state. Initially, the robot is placed on a particle. We assume that the robot uses a local compass to distinguish the four directions the robot can move to. Because we are only using one robot, we may assume that this local compass matches the global compass. The robot can transform a configuration into another configuration using a sequence of look-compute-move steps as follows. In each step, the robot acts according to a *transition function* δ . This transition function maps a pair (p, b) containing the state of the robot and the state of the current pixel to a triple (q, c, d) , where q is the new state of the robot, c is the new state of the current pixel, and $d \in \{\text{up, down, left, right}\}$ is the direction the robot moves in. In other words, in each step, the robot checks its state p and the state of the current pixel b , computes $(q, c, d) = \delta(p, b)$, changes into state q , changes the state of the current pixel to c if $c \neq b$, and moves one step into direction d .

Our goal is to develop robots transforming an unknown initial configuration P into a target configuration $T(P)$ by moving, creating, and deleting particles. We assess the efficiency of a robot by several metrics:

Time complexity: The total number of steps performed by the robot until termination.

Dimension complexity: The number of rows and columns we visit outside the bounding box of P . (Having this complexity can help having multiple robots working on different polyominoes at the same time, e.g. by separating the polyominoes by enough space so the robots do not use workspace of other robots.)

Space complexity: the total number of visited pixels outside the bounding box of the input polyomino P . (This gives a slightly more precise analysis of the used space, e.g., if we need only $O(1)$ extra rows and columns.)

Particle complexity: The maximum number of particles on the grid minus the number of particles in the input polyomino P at any given time. (The idea is that, once created, particles can be stored at some place, where robots can pick up and store particles whenever needed, and thus, already

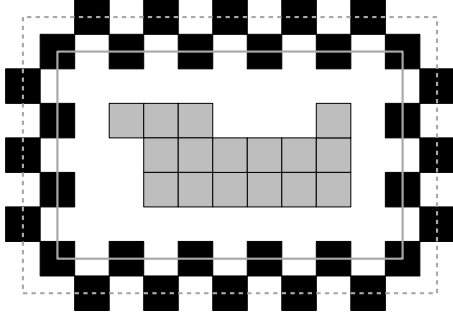


Fig. 2 A Polyomino P (gray) surrounded by a bounding box of particles (black) on the inner lane (solid line) and particles on the outer lane (dashed line).

created particles can be reused. Carrying out these operations efficiently is a separate algorithmic problem that is not at the heart of our presented work and will be dealt with separately.)

3 Basic Tools

A robot can check the states of all pixels within a constant distance by performing a tour of constant length. Thus, from now on we assume that a robot can check within a single step all eight pixels that are adjacent to the current pixel p or build a diagonal pair with p .

3.1 Bounding Box Construction

Because the bounding box and the polyomino are comprised of identical tiles, we need a gap between both to differentiate the two structures. We describe how to construct a bounding box of a given, not necessarily simple polyomino P in the checkered pattern shown in Fig. 2. We can split the bounding box into an *outer lane* and an *inner lane* (see Fig. 2). This allows distinguishing particles of P and particles of the bounding box. We assume that the robot starts anywhere on P .

Our construction proceeds in three phases. (i) We search for an appropriate starting position. (ii) We wrap a checkered path around P . (iii) We finalize this path to obtain the bounding box.

For **phase (i)**, we simply search for a local minimum in the y -direction:

1. Move to the leftmost particle, i.e., move left until the next pixel is empty.
2. Move to the right until there is a particle to the bottom. If there is no such particle move back to the leftmost particle, i.e., the leftmost local minimum, and end this search.
3. Move down until there is no more particle below the current position. Go back to step 1.

From the local minimum, we go two steps further to the left. If we land on a particle, this particle belongs to P and we restart phase (i) from this particle. Otherwise we have found a possible *starting position*.

In **phase (ii)**, we start constructing a path that will wrap around P . We start placing particles in a checkered pattern in the upwards direction. While constructing the path, three cases may occur.

1. At some point we lose contact with P , i.e., there is no particle at distance two from the inner lane. In this case, we do a right turn and continue our construction in the new direction.
2. Placing a new particle produces a conflict, i.e., the particle shares a corner or a side with a particle of P . In this case, we shift the currently constructed side of the bounding box outwards until no more conflict occurs. This shifting process may lead to further conflicts, i.e., we may not be able to further shift the current side outwards. If this happens, we deconstruct the path until we can shift it outwards again (see Fig. 3). In this process, we may be forced to deconstruct the entire bounding box we have built so far, i.e., we remove all particles of the bounding box including the particle in the starting position. In this case we know that there must be a particle of P to the left of the start position; we move to the left until we reach such a particle and restart phase (i).
3. Placing a new particle closes the path, i.e., we reach a particle of the bounding box we have created so far. We proceed with phase (iii).

Phase (iii): Let t be the particle that we reached at the end of phase (ii). We can distinguish two cases: (i) At t , the bounding box splits into three different directions (shown as the particle with black-and-white diagonal stripes in Fig. 4), and (ii) the bounding box splits into two different directions. In the latter case we are done, because we can only reach the bottom or left side of the bounding box. In the first case, we can move from t to the particle where we started the bounding box construction by doing a right turn and following the checkered path, and remove the bounding box parts until we reach t again. Now the bounding box splits in two directions at t . However, we may not have built a convex shape around P (see Fig. 4 left). This can be dealt with by straightening the bounding box in an analogous fashion like shifting in phase (2), e.g., by pushing the line to the right of t down.

Theorem 1 *The described strategy builds a bounding box that encloses the given polyomino P of width w and height h . Moreover, the strategy terminates after $\mathcal{O}(\max(w, h) \cdot wh)$ steps, using at most $\mathcal{O}(|\partial P|)$ auxiliary particles and $\mathcal{O}(w + h)$ of additional space in $\mathcal{O}(1)$ extra columns and rows. The running time in the best case is $\mathcal{O}(wh)$.*

Proof Correctness: We show that (a) the bounding box will enclose P , and (b) whenever we make a turn or shift a side of the bounding box, we find a particle with distance two from the bounding box, i.e., we will not build an infinite line.

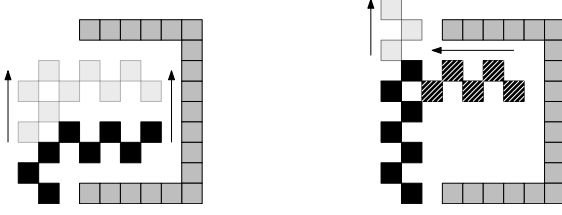


Fig. 3 Left: Further construction is not possible. Therefore, we shift the line upwards (see light gray particles). Right: A further shift produces a conflict with the polyomino. Thus, we remove particles (diagonal stripes) and proceed with the shift (light gray) when there is no more conflict.

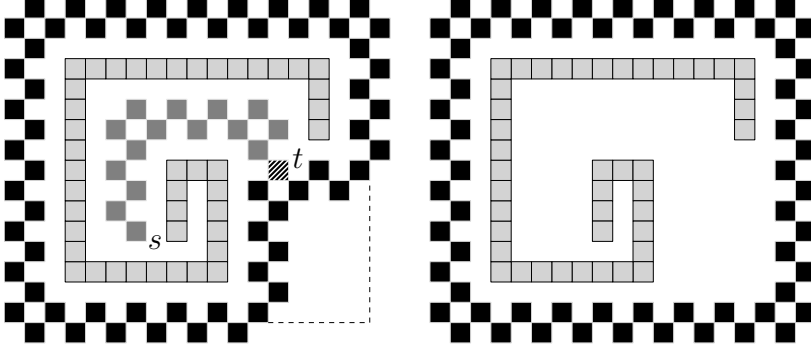
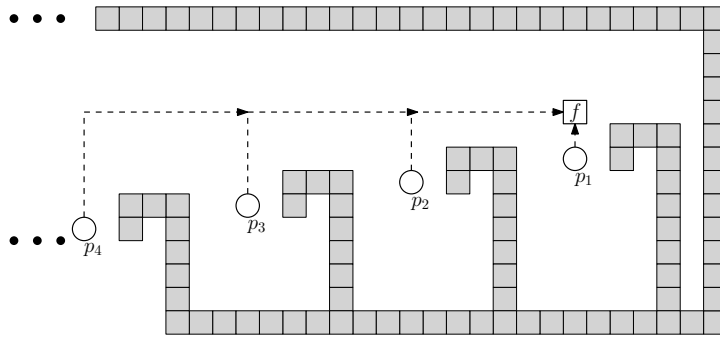


Fig. 4 Left: During construction, starting in s , we reach a particle t (diagonal stripes) at which the bounding box is split into three directions. The part between s and t (dark gray particles) can be removed, followed by straightening the bounding box along the dashed line. Right: The final bounding box.

First note that we only make a turn after encountering a particle with distance two to the inner lane. This means that we will “gift wrap” the polyomino until we reach the start. When there is a conflict (i.e., we would hit a particle of P with the current line), we shift the current line. Thus, we again find a particle with distance two to the inner lane. This also happens when we remove the current line and extend the previous one. After a short extension we make a turn and find a particle with distance two to the inner lane, meaning that we make another turn at some point. Therefore, we do not construct an infinite line, and eventually close the loop at some point.

Time: Every time we enter phase (1), we visit each particle of P at most $\mathcal{O}(1)$ times. When comparing to runs of phase (1), we observe that no particle of P is visited in both runs. Therefore, phase (1) costs $\mathcal{O}(N)$ time for the whole algorithm.

To establish a runtime of $\mathcal{O}(wh)$ for phase (2), we show that each pixel lying in the final bounding box is visited only a constant number of times. Consider a particle t that is visited for the first time. We know that t gets revisited again if the line through gets shifted or removed. When t is no longer on the bounding box, we can visit t again while searching for the start particle. Thus, t is visited at most four times by the robot, implying a running time of $\mathcal{O}(wh)$ unit steps. However, it may happen that we have to remove the



Because we never move further away from the polyomino than the resulting bounding box, we need $\mathcal{O}(w + h)$ additional space in $\mathcal{O}(1)$ extra rows and columns. \square

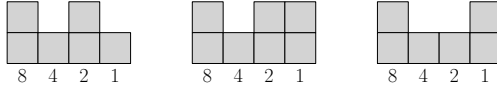


Fig. 6 Left: A 4-bit counter with decimal value 10 (1010 in binary). Middle: The binary counter increased by one. Right: The binary counter decreased by one.

Even if the starting configuration P contains a forbidden configuration, i.e., there is a diagonal pair of particles having no common adjacent particle, we can construct the bounding box in the same way. However, the test if a particle belongs to P or to the bounding box is slightly different: Let p_1, p_2 be a diagonal pair with no common adjacent tile. Then, p_1 and p_2 have at least one adjacent particle p_3 and p_4 , resp. Therefore, the robot can perform a local lookup if one of p_3 or p_4 exists and we get the following observation.

Observation 2 *We can build a bounding box that encloses a given connected starting configuration P of width w and height h , i.e., a polyomino that allows forbidden configurations. Moreover, the strategy terminates after $\mathcal{O}(\max(w, h) \cdot wh)$ steps, using at most $\mathcal{O}(|\partial P|)$ auxiliary particles and $\mathcal{O}(w+h)$ of additional space in $\mathcal{O}(1)$ extra columns and rows. The running time in the best case is $\mathcal{O}(wh)$.*

3.2 Binary Counter

For counting problems, a particle-based binary counter is indispensable, because the robot is not able to store non-constant numbers. The binary counter for storing an n -bit number consists of a base-line of n particles. Above each particle of the base-line there is either a particle denoting a 1, or an empty pixel denoting 0. Given an n -bit counter we can *increase* and *decrease* the counter by 1 (or by any constant c), and *extend* the counter by one bit. The latter operation will only be used in an increase operation.

In order to perform an increase operation, we firstly move to the least significant bit, and secondly start flipping 1s to 0s we flip a 0 to a 1. If we have run through the counter, we extend the counter by one bit.

For the decrease operation we again start at the least significant bit, and start flipping 0s to 1s until we flip a 1 to a 0. Note that this operation only works correctly if the counter contains at least one bit set to 1.

4 Counting Problems

The constant memory of our robot poses several challenges when we want to count certain elements of our polyomino, such as particles or corners. Because these counts can be arbitrarily large, we cannot store them in the state space of our robot. Instead, we have to make use of a binary counter. This requires us to move back and forth between the polyomino and the counter. Therefore,

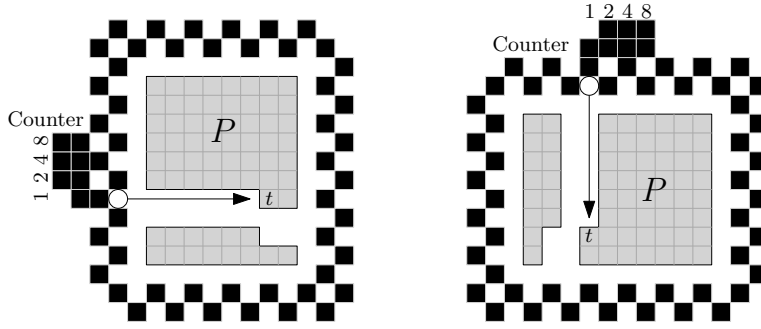


Fig. 7 An 8×8 square P in a bounding box, being counted row-by-row (left) or column-by-column (right). The robot has already counted 14 particles of P and stored this information in the binary counter. The arrow shows how the robot (\circ) moves to count the next particle t .

we must be able to find and identify the counter coming from the polyomino and to find the way back to the position where we stopped counting.

This motivates the following strategy for counting problems. We start by constructing a slightly extended bounding box and moving the robot to its bottom-left corner. From there, we perform the actual counting by shifting the polyomino two units downwards or to the left, one particle at a time. After moving each particle, we return to the counter, increasing it if necessary. We describe further details in the next two sections, where we present algorithms for counting the number of particles or corners in a polyomino.

4.1 Counting particles

The total number of particles in our polyomino can be counted using the strategy outlined above, increasing the counter by one after each moved particle.

Theorem 3 *Let P be a polyomino of width w and height h with N particles for which the bounding box has already been created. Counting the number of particles in P can be done in $\mathcal{O}(\max(w, h) \log N + N \min(w, h))$ steps using $\mathcal{O}(\max(w, h))$ of additional space in $\mathcal{O}(1)$ extra rows and columns and $\mathcal{O}(\log N)$ auxiliary particles.*

Proof In a first step, we determine whether the polyomino's width is greater than its height or vice versa. We can do this by starting from the lower left corner of the bounding box and then alternately moving up and right one step until we meet the bounding box again. We can recognize the bounding box by a local lookup based on its zig-zag shape that contains particles only connected by a corner, which cannot occur in a polyomino. The height is at least as high as the width if we end up on the right side of the bounding box. In the following, we describe our counting procedure for the case that the polyomino is higher than it is wide; the other case is analogous.

We start by extending the bounding box by shifting its bottom line down by two units. Afterwards we create a vertical binary counter to the left of the bounding box. We begin counting particles in the bottom row of the polyomino. We keep our counter in a column to the left of the bounding box such that the least significant bit at the bottom of the counter is in the row in which we are currently counting. We move to the right into the current row until we find the first particle. If this particle is part of the bounding box, the current row is done. In this case, we move back to the counter, shift it upwards and continue with the next row until all rows are done. Otherwise, we simply move the current particle down two units, return to the counter and increment it. For an example of this procedure, refer to Fig. 7.

For each particle in the polyomino, we use $\mathcal{O}(\min(w, h))$ steps to move to the particle, shift it and return to the counter; incrementing the counter itself only takes $\mathcal{O}(1)$ amortized time per particle. For each empty pixel we have cost $\mathcal{O}(1)$. In addition, we have to shift the counter $\max(w, h)$ times. Thus, we need $\mathcal{O}(\max(w, h) \log N + \min(w, h)N + wh) = \mathcal{O}(\max(w, h) \log N + \min(w, h)N)$ unit steps. We only use $\mathcal{O}(\log N)$ auxiliary particles in the counter, in addition to the bounding box.

In order to achieve $\mathcal{O}(1)$ extra rows and columns, we modify the procedure as follows. Whenever we move our counter, we check whether it extends into the space above the bounding box. If it does, we reflect the counter vertically, such that the least significant bit is at the top in the row, in which we are currently counting. This requires $\mathcal{O}(\log^2 N)$ extra steps and avoids using $\mathcal{O}(\log N)$ additional rows above the polyomino. \square

4.2 Counting Corners

The counting approach described in the previous section is not restricted to counting particles. Various other features of the polyomino can be counted using the same (asymptotic) number of steps, extra particles, space and rows or columns. Essentially, the only precondition is that we must be able to identify the feature based on a local lookup. For instance, we make the following observation.

Observation 4 *Let P be a polyomino of width w and height h with k convex (reflex) corners for which the bounding box has already been created. Counting the number of convex (reflex) corners in P can be done in $\mathcal{O}(\max(w, h) \log k + k \min(w, h) + wh)$ steps, using $\mathcal{O}(\max(w, h))$ of additional space in $\mathcal{O}(1)$ extra rows and columns, and $\mathcal{O}(\log k)$ auxiliary particles.*

Proof We use the same strategy as for counting particles in Theorem 3 and retain the same asymptotic bounds on running time, auxiliary particles and additional space and rows or columns. It remains to describe how we recognize convex and reflex corners.

Whenever we reach a particle t that we have not yet considered so far, we examine its neighbors x_1, \dots, x_6 , as shown in Fig. 8. The particle t has one

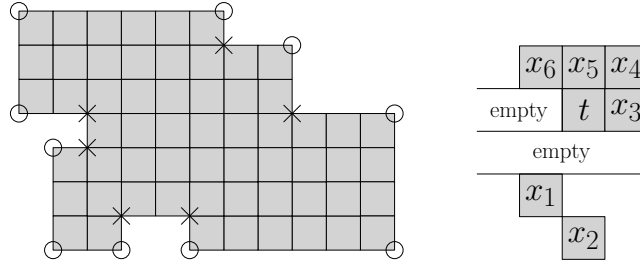


Fig. 8 *Left:* A polyomino and its convex corners (\circ) and reflex corners (\times). *Right:* After reaching a new particle t , we have to know which of the pixels x_1 to x_6 are occupied to decide how many convex (reflex) corners t has.

convex corner for each pair (x_1, x_2) , (x_2, x_3) , (x_3, x_5) , (x_5, x_1) that does not contain a particle, and no convex corner is contained in more than one particle of the polyomino. As there are at most four convex corners per particle, we can simply store this number in the state space of our robot, return to the counter, and increment it accordingly.

A reflex corner is shared by exactly three particles of the polyomino, so we have to ensure that each corner is counted exactly once. We achieve this by only counting a reflex corner if it is on top of our current particle t and was not already counted with the previous particle x_1 . In other words, we count the upper right corner of t as a reflex corner if exactly two of x_3, x_4, x_5 are occupied; we count the upper left corner of t as reflex corner if x_6 and x_5 are present and x_1 is not. In this way, we count all reflex corners exactly once. \square

5 Transformations with Turing Machines

In this section we develop a robot that transforms a polyomino P_1 into a required target polyomino P_2 . In particular, we encode P_1 and P_2 by strings $S(P_1)$ and $S(P_2)$ whose characters are from $\{0, 1, \sqcup\}$ (see Fig. 9 left and Definition 1). If there is a Turing machine transforming $S(P_1)$ into $S(P_2)$, we can give a strategy that transforms P_1 into P_2 (see Theorem 5).

Note that while it may be intuitively plausible that such a relationship exists, a Turing machine itself (which works in a one-dimensional environment) does not encounter some of the issues of two-dimensional geometry, such as the more complex topology involved in determining location and boundaries. Our Theorem 5 shows that these geometric issues can be handled, paving the way for the power of Turing Machines. For a visualization of the basic approach, see our video [1], in particular, the part starting at 5:04.

Also note that the Turing Machine emulation described in this section course also could be used, e.g., to copy polyominoes, instead of the algorithm from the next section. In practice, many applications would require a very large transition table which may not fit into the robots limited memory despite being of constant size; in these applications it is reasonable to assume that it would

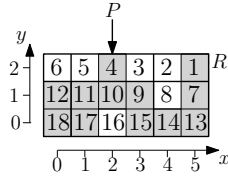


Fig. 9 An example showing how to encode a polyomino of height $h = 3$ ($S_1 = 11$ in binary) and width $w = 6$ ($S_2 = 110$ in binary) by $S(P) = S_1 \sqcup S_2 \sqcup S_3 = 11 \sqcup 110 \sqcup 100100101111111011$, where S_3 is the string of 18 bits that represent particles (black, 1 in binary) and empty pixel (white, 0 in binary), proceeding from high bits to low bits.

be more efficient w.r.t. both memory and time to use counters — and thus $\mathcal{O}(\log n)$ space — or, if the input is too large, a specialized algorithm such as the algorithms from the following Section 6.

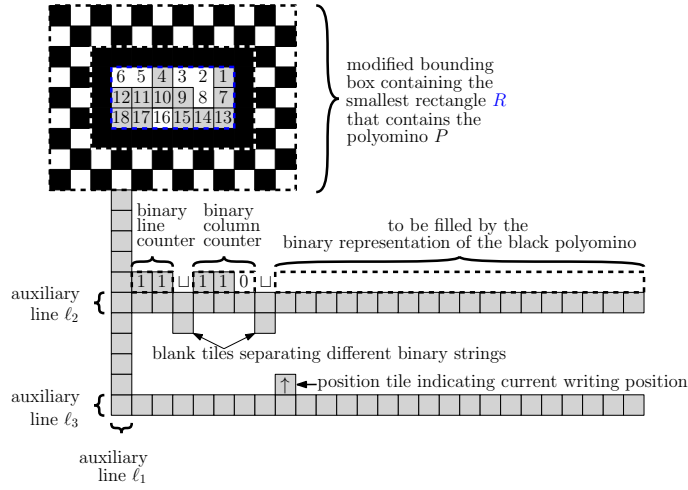
We start with the definition of the encodings $S(P_1)$ and $S(P_2)$.

Definition 1 Let $R := R(P)$ be the polyomino forming the smallest rectangle containing a given polyomino P (see Fig. 9). We represent P by the concatenation $S(P) := S_1 \sqcup S_2 \sqcup S_3$ of three bit strings S_1 , S_2 , and S_3 separated by blanks \sqcup . In particular, S_1 and S_2 are the height and width of R . Furthermore, we label each pixel of R by its rank in the lexicographic decreasing order w.r.t. y - and x -coordinates with higher priority to y -coordinates (see Fig. 9). Finally, S_3 is an $|R|$ -bit string $b_1, \dots, b_{|R|}$ with $b_i = 1$ if and only if the i th pixel in R is a particle of P .

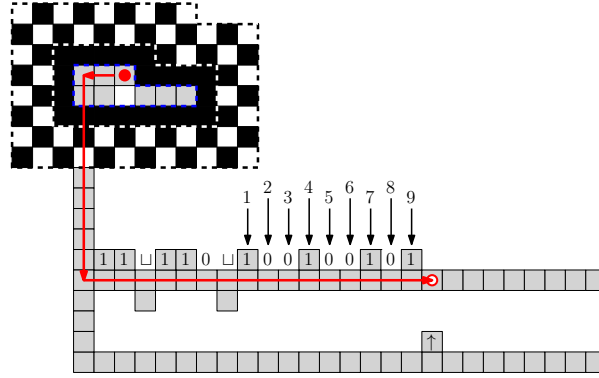
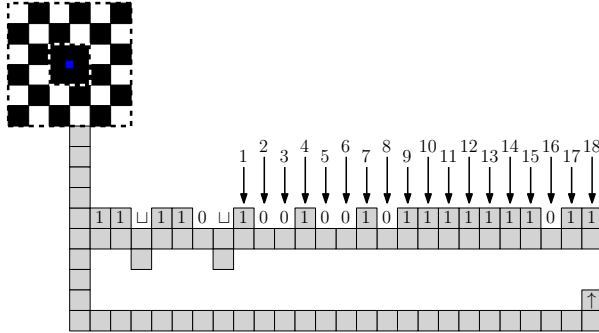
Theorem 5 Let P_1 and P_2 be two polyominoes with $|P_1| = |P_2| = N$. There is a strategy transforming P_1 into P_2 if there is a Turing machine transforming $S(P_1)$ into $S(P_2)$. The robot needs $\mathcal{O}(\partial P_1 + \partial P_2 + S_{TM})$ auxiliary particles, $\mathcal{O}(N^4 + T_{TM})$ steps, and $\Theta(N^2 + S_{TM})$ of additional space, where T_{TM} and S_{TM} are the number of steps and additional space needed by the Turing machine.

Proof Our strategy works in five phases: Phase (1) constructs a slightly modified bounding box of P_1 (see Fig. 10(a)). Phase (2) constructs a shape representing $S(P_1)$. In particular, the robot writes $S(P_1)$ onto an auxiliary line ℓ_2 in a bit-by-bit fashion (see Fig 10(b)). In order to remember the position, to which the previous bit was written, we use an additional particle called *position particle* which is not part of P_1 and placed onto another auxiliary line ℓ_3 . Phase (3) simulates the Turing machine that transforms $S(P_1)$ into $S(P_2)$. Phase (4) is the reversed version of Phase (2), and Phase (5) is the reversed version of Phase (1), so we only need to discuss how to realize Phases (1), (2), and (3).

Phase (1): We apply a slightly modified version of the approach of Theorem 1. In particular, let R be the smallest rectangle enclosing the polyomino. First, we fill all pixels that lie not inside R and adjacent to the boundary of R by auxiliary particles whose union we call *shell*. Second, we construct the



(a) State after counting numbers of lines and columns.

(b) Intermediate state of Phase (2) while writing the string representation $S(P_1)$ of the input polyomino P_1 by placing R line by line onto ℓ_1 .

(c) Final state of Phase (2) after writing the binary representation of the input polyomino.

Fig. 10 Phase (2) of transforming a (black) polyomino by simulating a Turing Machine.

bounding box as described above around the previously constructed shell. This results in a third additional layer of the bounding box (see Fig. 10(a)). Note that the robot recognizes that the particles of the third layer do not belong to the input polyomino P , because these particles of the third layer lie adjacent to the original bounding box.

Phase (2): Initially, we construct a vertical auxiliary line ℓ_1 of constant length, seeding two horizontal auxiliary lines ℓ_2 and ℓ_3 that are simultaneously constructed by the robot during Phase (2). The robot constructs the representation of $S(P_1)$ iteratively bit-by-bit on ℓ_2 and stores on ℓ_3 a “position particle” indicating the next pixel on ℓ_2 to which the robot has to write the next bit of the representation of $S(P_1)$.

Next we apply the approach of Theorem 3 twice in order to successively construct on ℓ_2 the binary representations of the numbers of lines and the number of columns of which R is made up (see Fig. 10).

Finally, the robot places consecutively and in decreasing order w.r.t. y -coordinates all lines of R onto ℓ_2 . In particular, the pixel of the current line ℓ of R are processed from right to left as follows. For each pixel $t \in \ell$, the robot alternates between ℓ and the auxiliary lines ℓ_2 and ℓ_3 in order to check whether t belongs to P_1 or not.

In order to reach the first pixel of the topmost line, the robot moves in a vertical direction on the lane induced by the vertical line ℓ_1 to the topmost line of R and then to the rightmost pixel t of the topmost line (see Fig. 10(b)). In order to ensure that in the next iteration of Phase (2), the next pixel of R is reached, the robot deletes a particle from t if there is a particle on pixel t from R and shrinks the modified bounding box such that it is the modified bounding box of $R \setminus \{t\}$ (see Fig. 10(b)). This involves only a constant-sized neighborhood of t and thus can be realized by a robot with constant memory. Next, the robot moves to the first free position of auxiliary line ℓ_2 by searching for the position particle on auxiliary line ℓ_3 . As the vertical distances of ℓ_2 and ℓ_3 to the lowest line of the bounding box are of constant values, the robot can change vertically between ℓ_2 and ℓ_3 by simply remembering the robot’s vertical distance to the lowest line of the bounding box. The robot concludes the current iteration of Phase (2) by moving the position particle one step to the right.

The approach of Phase (2) is repeated until all particles of P_1 are removed from R . In order to recognize that all pixels from R are removed, the robot checks whether $R = \emptyset$ in each iteration of Phase (2). This check can simply be done by checking whether the entire extended bounding box lies inside a 6×6 -rectangle, as illustrated in Figure 10(c). Hence, the robot simply has to scan an environment of constant size.

Phase (3): We simply apply the algorithm of the Turing machine by moving the robot correspondingly to the movement of the head of the Turing machine.

Finally, we analyze the entire approach of simulating a Turing machine as described above. From the discussion of Phase (2) we conclude that the construction of the auxiliary lines ℓ_1 , ℓ_2 , and ℓ_3 are not necessary, because the

vertical length of the entire construction below the bounding is a constant, i.e., in each iteration when the robot approaches the current writing position, the robot moves to the right until it arrives at the current writing position. Thus, the horizontal length of ℓ_2, ℓ_3 do not have an influence to the memory needed by the robot. Furthermore, the current writing position on ℓ_2 is indicated by a single position particle on ℓ_3 . As no auxiliary particles are needed for ℓ_1, ℓ_2, ℓ_3 , the only auxiliary particles needed are the current position particle and the particles needed for the modified bounding box which lie in $\mathcal{O}(\partial P_1 + \partial P_2)$. Hence, the sizes of the bounding boxes needed in Phase (2) and Phase (4) and the additional space $\mathcal{O}(S_{TM})$ needed by the Turing machine dominate the number of used auxiliary particles, which is in $\mathcal{O}(\partial P_1 + \partial P_2 + S_{TM})$. Furthermore, Phase (2) and Phase (4) need $\mathcal{O}(N^2)$ steps. In particular, counting the number of rows and counting the numbers of columns of the polyominoes can be done in $\mathcal{O}(N \log N + N^2)$ by using the approach of Theorem 3 because the polyominoes P_1 and P_2 may be single lines of width or height N . Furthermore, scanning a pixel, walking a distance of $\mathcal{O}(N)$ to the current writing position on ℓ_2 , writing the current pixel, and walking to the next pixel inside the modified bounding box takes $\mathcal{O}(N)$ time. Hence, writing N pixels takes $\mathcal{O}(N^2)$ time. Furthermore, Phase (3) needs $\mathcal{O}(T_{TM})$ steps, where T_{TM} is the number of steps needed by the Turing machine. Thus, the total time needed in the entire approach is in $\mathcal{O}(N^2 + T_{TM})$. Finally, the additional space is in $\Theta(N^2 + S_{TM})$. This concludes the proof of Theorem 5. \square

6 Custom-Made Functions

As already seen, we can transform a given polyomino by any computable function by simulating a Turing machine. However, this requires a considerable amount of space. In this section, we present realizations of common operations in a more space-efficient manner, reflecting the idea that a usable toolbox for such transformations is analogous to the functions in a software package for computer-aided design (CAD).

6.1 Copying a Polyomino

Copying a polyomino P has many applications. For example, we can reproduce a given shape, or apply algorithms that may destroy P after copying P into free space. In the following, we describe the *copy* function that copies P below the bounding box in a column-wise fashion, as seen in Fig. 12 (a row-wise copy can be described analogously).

To copy a polyomino P we proceed in two phases: (1) A preprocessing step that includes building the bounding box and extending the left side of the bounding box by three units. (2) Copying pixels column-wise below the bounding box. Note that P is contained in the first $w + 1$ columns within

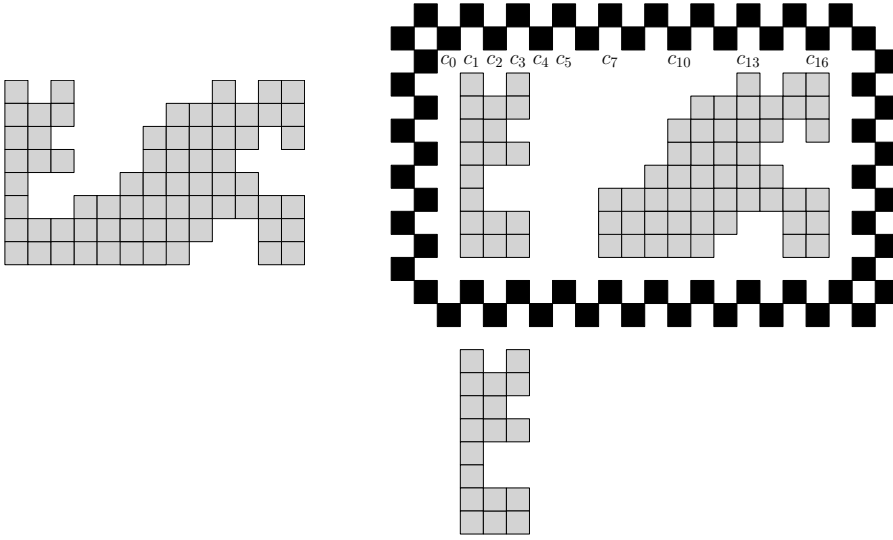


Fig. 11 Left: A polyomino P . Right: Intermediate situation during the copy process. Columns c_1 to c_3 contain the first three columns of P that are already copied below the bounding box. c_4 and c_5 will be used to copy the fourth column of P . Columns c_7 to c_{16} contain particles of P that still need to be copied.

the bounding box from the right. Therefore, we only need to start the copy procedure from the fifth column from the left.

After phase (1), we can split the bounding box into components that will be maintained by the robot: The first $k+1$ columns c_0, \dots, c_k of the bounding box from the left contain the first k columns of P already copied, then two columns c_{k+1}, c_{k+2} are used to process the next column of P , and after an empty column, the next $w-k$ columns c_{k+4}, \dots, c_{w+4} contain the rest of P that still needs to be copied. (see Fig. 11)

We now describe how to copy the $(k+1)$ -st column of P . A problem we have to deal with is that the connected components of the intersection of one column with P may be several units apart (e.g., in Figure 11 column c_3 has three components with one and two units space between them), and/or that the distance from the first/last particle of P to the bounding box can be arbitrarily large (e.g., in Figure 11, column c_7 has several empty pixel until the first particle of P). Thus, we need an auxiliary structure for determining whether a pixel is empty.

We solve this by using *bridges*, i.e., auxiliary particles denoting empty pixels. To distinguish between particles of P and bridges, we use column c_{k+1} to denote particles of P , and column c_{k+2} to denote bridge particles (see Fig. 12).

To copy an empty pixel, we place a particle two unit steps to the left, i.e., in column c_{k+2} . Then we move straight down and search for the first position, at which there is no copied particle of P to the left, nor a bridge particle. When this position is found, we place a particle, denoting an empty position.

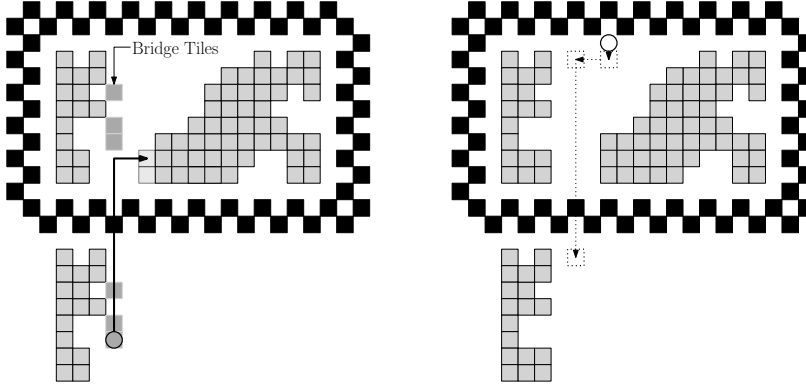


Fig. 12 Left: Intermediate step while copying the third column of a polyomino (gray particles) with bridges (dark gray particles). The robot (\circ) moves to next pixel along the black line. Right: When the column is copied the bridges get removed and the robot proceeds with the next column. In this case the robot finds an empty pixel, places a bridge particle two steps to the left and a bridge particle below the bounding box.

To copy a particle t of P , we pick up and move this particle three unit steps to the left, i.e., to column c_{k+1} . Afterwards, we move straight down following the particles and bridges until we reach a free position, i.e., there is no bridge particle to the right nor a copied polyomino particle, and place a particle, denoting a copy of t .

After placing the copy of a pixel, we move straight up until we find the first particle in c_{k+1} or c_{k+2} . From there we move two or three steps to the right (depending on whether we placed a bridge particle or not) and start to copy the next pixel. In case we reached the bottom of a column, we remove any particle representing a bridge particle and proceed with the next column (see Fig. 12 right).

Theorem 6 *Copying a polyomino P column-wise can be done within $\mathcal{O}(wh^2)$ unit steps using $\mathcal{O}(N)$ of auxiliary particles and $\mathcal{O}(wh)$ additional space in $\mathcal{O}(h)$ extra rows and columns.*

Proof Consider the strategy described above. Because the robot copies occupied pixels and empty pixels by using bridges, the robot is always able to find the next position to place the next particle (either a copied particle of P or a bridge particle). Thus, at the end of the algorithm, we obtain a valid copy of P .

As there are $\mathcal{O}(wh)$ many pixels that we have to copy with cost of $\mathcal{O}(h)$ per pixel, the strategy needs $\mathcal{O}(wh^2)$ unit steps to terminate.

Now, consider the number of auxiliary particles. We need N particles for the copy and $\mathcal{O}(h)$ particles for bridges, which are reused for each column. Thus, we need $\mathcal{O}(N)$ auxiliary particles in total. Because we place the copied version of P beneath P , we need $\mathcal{O}(wh)$ additional space in $\mathcal{O}(h)$ extra rows and columns. \square

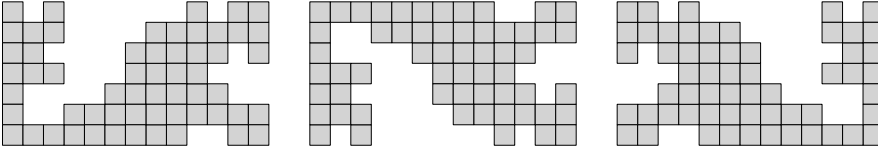


Fig. 13 Left: A polyomino P . Middle: Vertical reflection of P . Right: Horizontal reflection of P .

6.2 Reflecting a Polyomino

In this section, we show how to perform a horizontal reflection on a polyomino P (a vertical reflection is done analogously). Assume that we already built the bounding box. Then we shift the bottom side of the bounding box one unit down, such that we have two units of space between the bounding box and P . We start with the bottom-most row r and build r in reversed order beneath the bounding box using bridges, as seen in the previous section. To do so, we use the row below r to mark empty pixels with bridge particles (see Figure 14).

To copy the state of a pixel p , we pick up the particle (from P or from the bridge), move to the right until we reach the bounding box, move two or three steps below the bounding box (depending on whether we place a bridge particle next or not) and move to the first position where no copied particle of P nor a bridge particle exists.

To find the next pixel to copy, we move back to the right until we reach the end (i.e., the next column has no particle from P nor a bridge particle), move two steps upwards within bounding box, and search for the leftmost pixel we have not copied yet. Again, this pixel is identified, when a further step has no particle from P nor a bridge particle.

After the reflection of r , we shift the bottom side of the bounding box one unit up and remove bridge particles. The rows that were reflected so far are not moved upwards, so we have again two units space below the bounding box. Also note that we again have two units of space between the bounding box and the rest of P . We can therefore repeat the process until no particle is left.

Theorem 7 *Reflecting a polyomino P horizontally can be done in $\mathcal{O}(w^2h)$ unit steps, using $\mathcal{O}(w)$ of additional space and $\mathcal{O}(w)$ auxiliary particles.*

Proof For each pixel within the bounding box we have to copy this pixel to the desired position. This costs $\mathcal{O}(w)$ steps, i.e., we have to move to the right side of the boundary, move a constant number of steps down and move $\mathcal{O}(w)$ to the desired position. These are $\mathcal{O}(w)$ steps per pixel, thus, $\mathcal{O}(w^2h)$ unit steps in total.

It can be seen in the described strategy that we only need a constant amount of additional space in one dimension because we are overwriting the space that was occupied by P . This implies a space complexity of $\mathcal{O}(w)$. Following the same argumentation of Theorem 6, we can see that we need $\mathcal{O}(w)$ auxiliary particles. \square

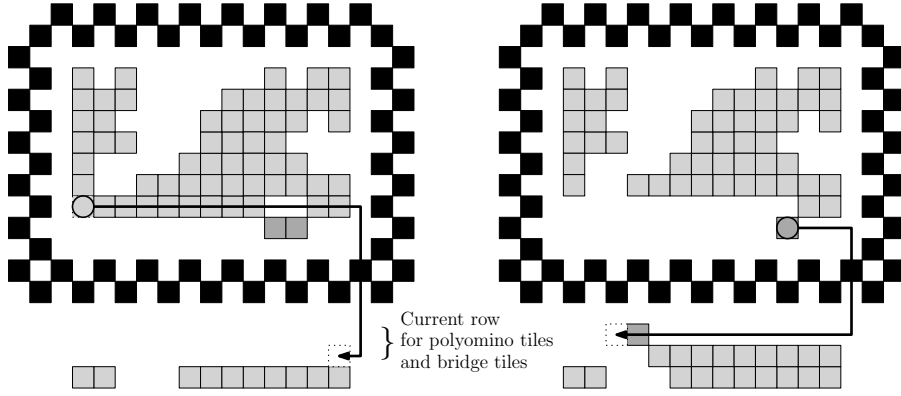


Fig. 14 Left: Beginning of reflecting the second row. Because the current pixel is occupied, the robot moves the particle to the right below the bounding box (particle with dotted border). Note that the extra space between P and the bounding box is used by bridge particles (dark gray) to denote empty pixels above. Right: Intermediate step of reflecting the second row. The next position is empty, thus, the robot will move the corresponding bridge particle next.

Corollary 1 *Reflecting a polyomino P vertically and horizontally can be done in $\mathcal{O}(wh(w+h))$ unit steps, using $\mathcal{O}(w+h)$ additional space in $\mathcal{O}(1)$ extra rows and columns and $\mathcal{O}(w+h)$ auxiliary particles.*

6.3 Rotating a Polyomino

Rotating a polyomino by $\pm \frac{\pi}{2}$ presents some additional difficulties, because the dimension of the resulting polyomino has dimensions $h \times w$ instead of $w \times h$. Thus, we may need non-constant additional space in one dimension, e.g., if one dimension is large compared to the other dimension. A simple approach is to copy the rows of P bottom-up to the right of P . This allows us to rotate P with $\mathcal{O}(wh)$ additional space. For now, we assume that $h \geq w$.

We propose a strategy that is more compact. The strategy consists of two phases: First, reflect P over the $x = y$ line. Then do a second reflection over the y -axis, i.e., a horizontal reflection. Because

$$\underbrace{\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}}_{\text{Reflection over } y\text{-axis}} \underbrace{\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}}_{\text{Reflection over } x=y \text{ line}} P = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} P,$$

we effectively perform a clockwise rotation by $\frac{\pi}{2}$ (to perform a counter-clockwise rotation we do a vertical reflection instead of a horizontal reflection).

After constructing the bounding box, we place a particle t_1 in the bottom-left corner of the bounding box, which marks the bottom-left corner of P (see

Fig. 15(a) left). We further add two more particles t_c outside the bounding box marking the position of the leftmost column of P , and t_2 inside the bounding box marking the topmost row of P (see Fig. 15(a)). Afterwards, we extend the bounding box at the left side and the bottom side by six units without moving t_1 , t_2 or t_c . This gives us a width of seven units between the polyomino and the bounding box at the left and bottom side (see Fig. 15(a)). Now we can move the first column rotated in clockwise direction five units below P (which is two units above the bounding box), as follows.

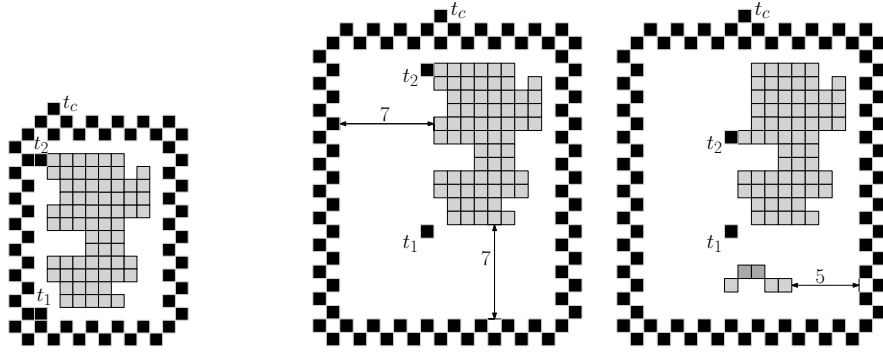
To find the next pixel whose state we want to copy, we move along the bounding box until we find t_c . Then we move down until we reach t_2 , i.e., the robot finds a particle to the west. After storing the state s_p of the pixel p and removing its particle (if it exists), we move t_2 one position down. (If t_2 reached t_1 , i.e., we would place t_2 on t_1 , then we know that we have finished the column and proceed with the row by moving t_2 to the right of t_1 in the next iteration.) To place a particle denoting s_p , we move over t_1 to the right side of the bounding box and three units down and then six units to the left. From there, we can follow bridge particles and particles from P , until we reach a free position. We place a particle on this position if p was empty, or a particle below if p was occupied. In case we copy a row-pixel, we do a turn upwards and follow bridges and particles of P again. See Fig. 15. Note that we cannot place the row directly on top of the column or else we may get conflicts with bridges. We therefore build the row one unit to the left and above the column. Also note that during construction of the first column we may hit the left side of the bounding box. In this case we extend this side by one unit (see Fig. 15(b)).

After constructing a column and a row, we move the constructed row one unit to the left and two units down, we move the column one unit down and two units left, and delete the bridge particles on the fly. This gives us enough space to construct the next column and row in the same way (see Fig. 15(c)). Then we move t_1 one unit upwards and one unit to the right, we move t_2 four units below t_c (i.e., to the topmost row of P), and t_c one unit to the right.

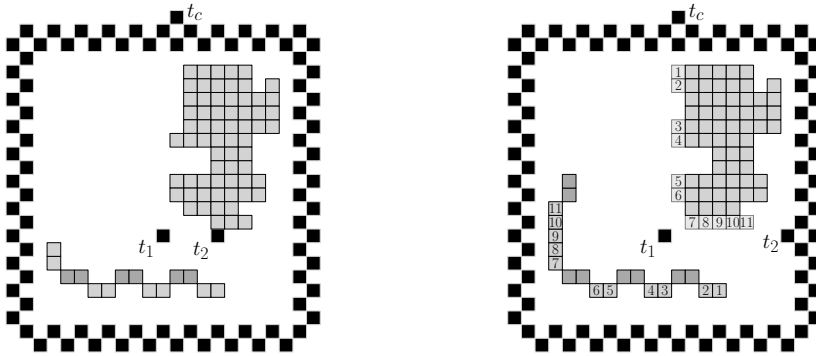
When all columns and rows are constructed, we obtain a polyomino that is a reflected version of our desired polyomino. It is left to reflect vertically to obtain a polyomino rotated in counter-clockwise direction, or horizontally to obtain a polyomino that is rotated in clockwise direction. This can be done with the strategy described in Section 6.2.

Theorem 8 *There is a strategy to rotate a polyomino P by $\pm \frac{\pi}{2}$ within $\mathcal{O}((w+h)wh)$ unit steps, using $\mathcal{O}(w+h+|w-h|h)$ of additional space in $\mathcal{O}(|w-h|+1)$ extra rows and columns and $\mathcal{O}(w+h)$ auxiliary particles.*

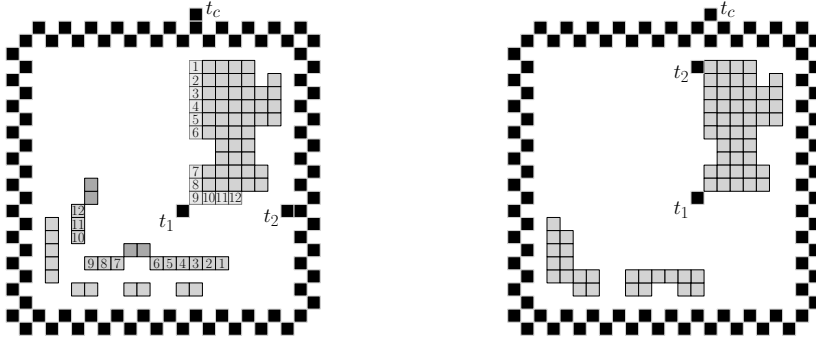
Proof Like in our other algorithms, the number of steps to copy the state of a pixel to the desired position is bounded by $\mathcal{O}(w+h)$. Shifting the constructed row and column also takes the same number of steps. Therefore, constructing the reflected version of our desired polyomino needs $\mathcal{O}((w+h)wh)$ unit steps. Additionally we may have to extend one side of the bounding box. This can happen at most $\mathcal{O}(|w-h|)$ times, with each event needing $\mathcal{O}(h)$ unit steps.



(a) Left: Intermediate construction before expanding the bounding box. Markers t_1 , t_2 and t_c are already placed. Middle: Construction after expanding the bounding box. Right: Intermediate step while reflecting the first column over the $x = y$ line. Note that we start the column with distance five to the bounding box. This is needed to build the row upwards without hitting P if w is close to h .



(b) Left: Intermediate step while building the first reflected row. Note that the left side of the bounding box has been extended to make more space. Right: Intermediate step after reflecting the first column and row (original column and row are marked in light gray). Numbers show the matching between original position and new position of particles.



(c) Left: First row and column have been moved down and left to make space for the next row and column. Right: Merging the first two columns and rows.

Fig. 15 Intermediate steps of reflecting a polyomino over the $x = y$ line.

Because $\mathcal{O}(|w - h|)$ can be bounded by $\mathcal{O}(w + h)$, this does not change the total time complexity.

Because the width of the working space increases by $\mathcal{O}(|w - h|)$ and the height only increases by $\mathcal{O}(1)$, we need $\mathcal{O}(|w - h|h)$ of additional space. For the number of auxiliary particles, consider the number of particles used for the bounding box, for bridges, and marker particles. The bounding box is made from $\mathcal{O}(w + h + |w - h|) = \mathcal{O}(w + h)$ particles, and there are only $\mathcal{O}(w + h)$ bridge particles. As there are only $\mathcal{O}(1)$ marker particles, we need in total $\mathcal{O}(w + h)$ auxiliary particles. \square

6.4 Scaling a Polyomino

Scaling a polyomino P by a factor c replaces each particle by a $c \times c$ square. This can easily be handled by our robot.

Theorem 9 *Given a constant c , the robot can scale the polyomino by c within $\mathcal{O}((w^2 + h^2)c^2N)$ unit steps using $\mathcal{O}(c^2wh)$ of additional space in $\mathcal{O}(c(w + h))$ additional rows and columns, using $\mathcal{O}(c^2N)$ auxiliary particles.*

Proof After constructing the bounding box, we place a particle denoting the current column. Suppose we are in the i -th column C_i . Then we shift all columns that lie to the right of C_i by c units to the right with cost of $\mathcal{O}((w - i) \cdot \sum_{j=i+1}^h cN_{C_j}) \subseteq \mathcal{O}(wcN)$, where N_{C_j} is the number of particles in the j th column. Because c is a constant, we can always find the next column to shift. Afterwards, we copy C_i exactly $c - 1$ times to the right of C_i , which costs $\mathcal{O}(N_{C_i}c)$ unit steps. Thus, extending a single column costs $\mathcal{O}(c \cdot (wN + N_{C_i}))$ and hence extending all columns costs $\mathcal{O}(cw^2N)$ unit steps in total.

Extending each row is done analogously. However, because each particle has already been copied c times, we obtain a running time of $\mathcal{O}(c^2h^2N)$. This implies a total running time of $\mathcal{O}((w^2 + h^2)c^2N)$. The proof for the particle and space complexity is straightforward. \square

7 Conclusion

We have given a number of tools and functions for manipulating an arrangement of particles by a single robotic automaton with constant memory.

In our work, we focused on arrangements of square-shaped particles; it seems straightforward to apply our ideas to other arrangements, such as hexagonal particles. There are various further challenges, including more complex operations, explicitly dealing with the logistics of obtaining and moving new tiles from depots to their final destinations, as well as methods for sharing the work between multiple robots. Another extension is to consider three-dimensional arrangements of voxels, with robots restricted to motions over the surface, or being able to move through the interior. An additional set of problems arises by requiring the set of tiles and the robot location to stay

connected; this is motivated by scenarios in which disconnected pieces may drift apart, e.g., in space. These and other questions are left for future work.

References

1. A. Abdel-Rahman, A. T. Becker, D. E. Biediger, K. C. Cheung, S. P. Fekete, N. A. Gershfeld, S. Hugo, B. Jenett, P. Keldenich, E. Niehs, C. Rieck, A. Schmidt, C. Scheffer, and M. Yannuzzi. Space ants: Constructing and reconfiguring large-scale structures with finite automata. In *Proc. Symposium on Computations Geometry (SoCG)*, 2020. Submitted; video at <https://www.youtube.com/watch?v=12j7G1TXKEo>.
2. A. T. Becker, E. D. Demaine, S. P. Fekete, J. Lonsford, and R. Morris-Wright. Particle computation: complexity, algorithms, and logic. *Natural Computing*, 18(1):181–201, 2019.
3. M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 132–142, 1978.
4. A. Bonato and R. J. Nowakowski. *The Game of Cops and Robbers on Graphs*. AMS, 2011.
5. S. Das. Mobile agents in distributed computing: Network exploration. *Bulletin of the European Association for Theoretical Computer Science*, 109:54–69, 2013.
6. J. J. Daymude, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Improved leader election for self-organizing programmable matter. In *Proc. Int. Symp. Algor. and Experim. Wirel. Sens. Netw. (ALGOSENSORS)*, pages 127–140, 2017.
7. E. Demaine, M. Demaine, M. Hoffmann, and J. O’Rourke. Pushing blocks is hard. *Computational Geometry*, 26(1):21–36, 2003.
8. E. D. Demaine, S. P. Fekete, C. Scheffer, and A. Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Computer Science*, 671:4–18, 2017.
9. Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Brief announcement: Amoebot – a new model for programmable matter. In *Proc. ACM Symp. Parallel Algor. and Archit. (SPAA)*, pages 220–222, 2014.
10. Z. Derakhshandeh, R. Gmyr, A. Porter, A. W. Richa, C. Scheideler, and T. Strothmann. On the runtime of universal coating for programmable matter. In *DNA Computing and Molecular Programming*, pages 148–164, 2016.
11. 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*, page 21, 2015.
12. 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*, pages 289–299, 2016.
13. Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal coating for programmable matter. *Theoretical Computer Science*, 671:56–68, 2017.
14. Z. Derakhshandeh, R. Gmyr, T. Strothmann, R. Bazzi, A. W. Richa, and C. Scheideler. Leader election and shape formation with self-organizing programmable matter. In *DNA Computing and Molecular Prog.*, pages 117–132, 2015.
15. G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. Shape formation by programmable particles. *Distributed Computing*, 0:1–33, 2019.
16. S. P. Fekete, R. Gmyr, S. Hugo, P. Keldenich, C. Scheffer, and A. Schmidt. CADbots: Algorithmic Aspects of Manipulating Programmable Matter with Finite Automata. In *Proceedings of the 13th Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2018. To appear.
17. F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, jun 2008.
18. R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, and C. Scheideler. Shape recognition by a finite automaton robot. In *Proceedings of 43rd International Symposium on Mathematical Foundations of Computer Science*, pages 52:1–52:15, 2018.

19. R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler, and T. Strothmann. Forming tile shapes with simple robots. In *Proceedings of the 23rd International Conference on DNA Computing and Molecular Programming*, pages 122–138, 2018.
20. F. Hurtado, E. Molina, S. Ramaswami, and V. Sacristán. Distributed reconfiguraiton of 2D lattice-based modular robotic systems. *Autonomous Robots*, 38(4):383–413, 2015.
21. K. Lund, A. Manzo, N. Dabby, N. Michelotti, A. Johnson-Buck, J. Nangreave, S. Taylor, R. Pei, M. Stojanovic, N. Walter, and E. Winfree. Molecular robots guided by prescriptive landscapes. *Nature*, 465(7295):206–210, 2010.
22. E. Markou. Identifying hostile nodes in networks using mobile agents. *Bulletin of the European Association for Theoretical Computer Science*, 108:93–129, 2012.
23. O. Michail and P. G. Spirakis. Simple and efficient local codes for distributed stable network construction. *Distributed Computing*, 29(3):207–237, 2016.
24. E. Niehs, A. Schmidt, C. Scheffer, D. E. Biediger, M. Yannuzzi, B. Jenett, A. Abdel-Rahman, K. C. Cheung, A. T. Becker, and S. P. Fekete. Recognition and reconfiguration of lattice-based cellular structures by simple robots. In *Proc. 37th International Conference on Robotics and Automation (ICRA)*, 2020. To appear.
25. T. Omabegho, R. Sha, and N. Seeman. A bipedal DNA Brownian motor with coordinated legs. *Science*, 324(5923):67–71, 2009.
26. M. J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.
27. A. Pelc. Deterministic rendezvous in networks: A comprehensive survey. *Networks*, 59(3):331–347, 2012.
28. J. H. Reif and S. Sahu. Autonomous programmable DNA nanorobotic devices using dnazymes. *Theoretical Computer Science*, 410:1428–1439, 2009.
29. A. N. Shah. Pebble automata on arrays. *Computer Graphics and Image Processing*, 3(3):236–246, 1974.
30. J. Shin and N. Pierce. A synthetic DNA walker for molecular transport. *Journal of the American Chemical Society*, 126:4903–4911, 2004.
31. Y. Terada and S. Murata. Automatic modular assembly system and its distributed control. *International Journal of Robotics Research*, 27(3–4):445–462, 2008.
32. A. Thubagere, W. Li, R. Johnson, Z. Chen, S. Doroudi, Y. Lee, G. Izatt, S. Wittman, N. Srinivas, D. Woods, E. Winfree, and L. Qian. A cargo-sorting DNA robot. *Science*, 357(6356), 2017.
33. Z. Wang, J. Elbaz, and I. Willner. A dynamically programmed DNA transporter. *Angewandte Chemie International Edition*, 51(48):4322–4326, 2012.
34. S. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A. Turberfield. A DNA-based molecular motor that can navigate a network of tracks. *Nature Nanotechnology*, 7(3):169–173, 2012.
35. 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 of Innovations in Theoretical Computer Science*, pages 353–354, 2013.

Recognition and Reconfiguration of Lattice-Based Cellular Structures by Simple Robots

Eike Niehs^{1,*}, Arne Schmidt^{1,*}, Christian Scheffer¹, Daniel E. Biediger², Michael Yannuzzi², Benjamin Jenett^{3,4}, Amira Abdel-Rahman⁴, Kenneth C. Cheung³, Aaron T. Becker², Sándor P. Fekete¹

Abstract—We consider recognition and reconfiguration of lattice-based cellular structures by very simple robots with only basic functionality. The underlying motivation is the construction and modification of space facilities of enormous dimensions, where the combination of new materials with extremely simple robots promises structures of previously unthinkable size and flexibility; this is also closely related to the newly emerging field of programmable matter. Aiming for large-scale scalability, both in terms of the number of the cellular components of a structure, as well as the number of robots that are being deployed for construction requires simple yet robust robots and mechanisms, while also dealing with various basic constraints, such as connectivity of a structure during reconfiguration. To this end, we propose an approach that combines ultra-light, cellular building materials with extremely simple robots. We develop basic algorithmic methods that are able to detect and reconfigure arbitrary cellular structures, based on robots that have only constant-sized memory. As a proof of concept, we demonstrate the feasibility of this approach for specific cellular materials and robots that have been developed at NASA.

I. INTRODUCTION

Building and modifying large-scale structures is an important and natural objective in a vast array of applications. In many cases, the use of autonomous robots promises significant advantages, but also a number of additional difficulties. This is particularly true in space, where the difficulties of expensive supply chains, scarcity of building materials, dramatic costs and consequences of even small errors, and the limitations of outside intervention in case of malfunctions pose a vast array of extreme challenges. Nevertheless, the unquestionable long-term benefits and perspectives of large-scale facilities in space have lead to enormous investments in terms of funds, time, and human capital [3]. As a consequence, the use of advanced methods for autonomous construction in space is indispensable for further progress.

In recent years, a number of significant advances have been made to facilitate overall breakthroughs. One important step has been the development of ultra-light and scalable composite lattice materials [35] that allow the construction of modular, reconfigurable, lattice-based structures [42]; see

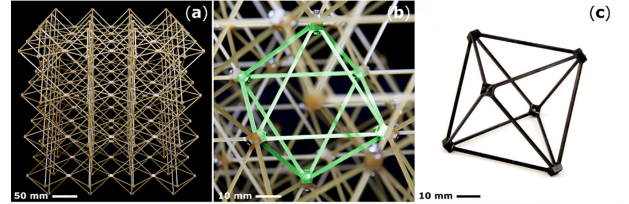


Fig. 1. (a) An assembled cuboctahedral lattice specimen, made from (b) Ultem 2200 (20% glass fiber reinforced polyetherimide) octahedral unit cells (highlighted), termed voxels. (c) A single monolithic RTP 2187 (40% carbon fiber reinforced polyetherimide) injection molded voxel. (See [35].)

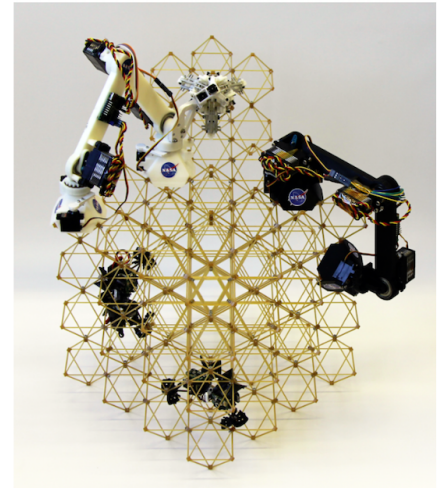


Fig. 2. Modular reconfigurable 3D lattice structure and mobile robots, showing the small size of the robots relative to the structure that they work on, and the parallel use of multiple robots. (See [9].)

Figure 1. A second step has been the design of simple autonomous robots [38], [41] that are able to move on the resulting lattice structures and move their elementary cell components, thereby allowing the reconfiguration of the overall edifice; see Figure 2 and Figure 3. Combining such materials and robots in space promises to vastly increase the dimensions of constructible facilities and spacecraft, as well as offering to extend mission capabilities with reconfiguration and re-use [34].

In this paper, we address the next step in this hierarchy: Can we enable extremely simple robots to perform a more complex spectrum of construction tasks for cellular structures in space, such as patrolling and marking the perimeter,

* Both authors contributed equally to this document.

¹Department of Computer Science, TU Braunschweig, Germany. {s.fekete, e.niehs, c.scheffer, arne.schmidt}@tu-bs.de

²Department of Electrical and Computer Engineering, University of Houston, USA. {atbecker,dbiediger}@uh.edu

³NASA Ames Research Center, Coded Structures Lab (CSL), Moffett Field, CA, USA. kenny@nasa.gov

⁴Center for Bits and Atoms (CBA), Massachusetts Institute of Technology, Cambridge, MA, USA. bej@mit.edu, amira.abdel-rahman@cba.mit.edu

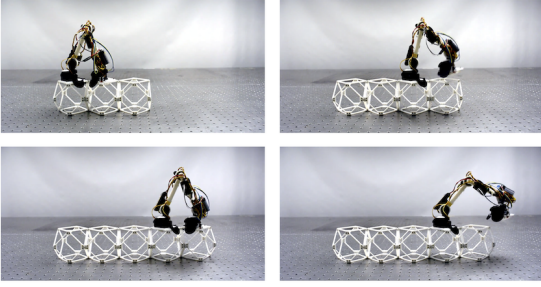


Fig. 3. A BILL-E robot moving on an expanding row of voxels. (See [39].)

scaling up a given seed construction, and a number of other design operations? In the harsh and remote environment of space, relying on powerful CPUs is a serious limitation, not only because of the vulnerability of complex computer hardware, but also because its availability becomes a limiting factor to mass-producing a large number of simple, modular robots in space. As we demonstrate, sophisticated computing hardware is not necessary. Even the extremely limited capabilities of machines with a finite number of states suffice for these tasks. To this end, we provide a suite of algorithmic methods, and demonstrate the feasibility of the resulting approach by implementing it on actual ultra-light material with simple mobile manipulators.

A. Our Results

We present the following results.

- 1) We show how just two finite-state robots suffice to construct a bounding box for a given connected planar arrangement of grid cells (a *polyomino* P) in a limited number of steps.
- 2) We provide an algorithmic methods that enables two finite-state robots to construct for a polyomino P with a surrounding bounding box a scaled-up copy of P in a limited number of steps, while preserving connectivity of intermediate arrangements.
- 3) We also sketch how other basic operations of Computer-Aided Design (such as copying or rotating) can be performed in similar ways.
- 4) We demonstrate how these methods can be implemented on actual lattice-based structures with simple robots.
- 5) We provide experimental outcomes for the resulting processing times.

B. Related Work

The structures considered in this work are based on **ultra-light material**, as described by Cheung and Gershenfeld [8] and Gregg et al. [35]. Modular two-dimensional elements mechanically link in 3D to form reversibly assembled composite lattices. This process is not limited by scale, and it enables disassembly and reconfiguration. As shown by Cramer et al. [10] and Jenett et al. [40], large but light-weight structures can be built from these components. Jenett et al. have developed autonomous robots that move on the

surface [38], [39] or within the cellular structure [41]. With the help of these robots, individual cells can be attached to an existing assembly, or moved to a different location. An approach for global optimization of a corresponding motion plan has been described by Costa et al. [9], while the design of hierarchical structures was addressed by Jenett et al. [43].

Assembly by simple robots has also been considered at the micro scale, where global control is used for supplying the necessary force for moving agents, e.g., see Becker et al. [2] for the corresponding problem of motion planning, Schmidt et al. [47] for using this model for assembling structures, and Balanza-Martinez et al. [1] for theoretical characterizations. Distributed self-assembly for modular robots with limited computing resources was studied by Tucci et al. [48]. Self-configuration of robots themselves has been considered by Naz et al. [44]. A basic model in which robots themselves are used as building material was introduced by Derakhshandeh et al. [16], [17]. This resembles Claytronics robots like Catoms, see Goldstein and Mowry [33].

A large spectrum of methods for **tile-based assembly in biology** has also been considered. Examples include DNA self-assembly, introduced by Winfree [49], [50] and extended to a variety of models [7], [13]. See Patitz [46] for a survey.

On the algorithmic side, there has been a considerable amount of work dealing with **robots or agents on graphs**. Blum and Kozen [5] showed that two finite automata can jointly search any unknown maze. Other work has focused on exploring general graphs (e.g., [24], [27], [45]), as a distributed or collaborative problem using multiple agents (e.g. [4], [6], [11], [25]) or with space limitations (e.g. [21], [26]–[29]).

From an algorithmic view, we are interested in **different models representing programmable matter** and further recent results. Inspired by the single-celled amoeba, Derakhshandeh et al. introduced the Amoebot model [14] and later a generalized variant, the general Amoebot model [19]. The Amoebot model provides a framework based on an equilateral triangular graph and active particles that can occupy a single vertex or a pair of adjacent vertices within that graph. With just a few possible movements, these particles can be formed into different shapes like lines, triangles or hexagons [16], and a leader out of all particles can be elected [12], [19]. A universal shape formation algorithm within the Amoebot model was described by Di Luna et al. in [20]. An algorithm for solving the problem of coating an arbitrarily shaped object with a layer of self-organizing programmable matter was presented in [18] and analyzed in [15]. Other models with active particles were introduced in [51] as the Nubot model and in [37] with modular robots. In [30], Gmyr et al. introduced a model with two types of particles: active robots acting like a deterministic finite automaton and passive tile particles. Furthermore, they presented algorithms for shape formation [32] and shape recognition [31] using robots on tiles. Using the same “Robot-on-Tiles” model as we do in our work, Hugo presented algorithms for recognizing convexity of a given polyomino and counting the number of tiles or

corners in her Bachelor thesis [36]. Finally, Fekete et al. introduced more complex geometric algorithms for copying, reflecting, rotating and scaling a given polyomino as well as an algorithm for constructing a bounding box surrounding a polyomino in [22].

II. PRELIMINARIES

In the following, we introduce models, general definitions as well as a description of the underlying limitations.

A. Model

We consider an infinite *square grid graph* G , where \mathbb{Z}^2 defines the *vertices*, and for every two vertices with distance one there is a corresponding *edge* in G . We use the compass directions (N, E, S, W) for orientation when moving on on the grid and may use *up, right, down* and *left* synonymously.

Every vertex of G is either *occupied* by a tile or *unoccupied*. Tiles represent passive particles of programmable matter that cannot move or manipulate themselves. The maximal connected set of occupied vertices is called *polyomino*.

The *boundary* of a polyomino P is denoted by ∂P and includes all tiles of P that are adjacent to an empty vertex. For simplicity we show the boundary as a red line around P (see also Figure 4 (a)). Polyominoes can have *holes*, i.e., finite maximal connected sets of empty vertices. Polyominoes without holes are called *simple*; otherwise, they are *non-simple*. The bounding box of a given polyomino P is defined as the boundary of the smallest axis-aligned rectangle enclosing but not touching P ; it will be denoted by $bb(P)$ (see Fig. 5). Because the bounding box and polyomino are comprised of identical tiles, a gap is necessary to differentiate the two.

We use *robots* as active particles in our model. These robots work like *finite deterministic automata* that can move around on the grid and manipulate the polyomino. A robot has the abilities to move along the edges of the grid graph and to change the state of the current vertex by placing or removing a tile on it. The robots work in a series of Look-Compute-Move (LCM) steps. Depending on the current state of the robot and the vertex it is positioned on (Look), the next step is computed according to a specific transition function

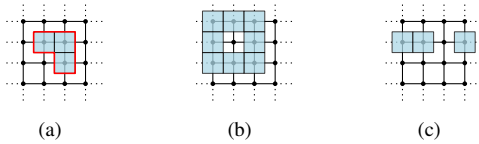


Fig. 4. (a) The red line indicates the boundary of P , denoted by ∂P . (b) A non-simple polyomino with one hole. (c) The tiles on the grid induce two separate connected components.

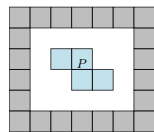


Fig. 5. A polyomino P and its gray colored bounding box surrounding P .

δ (Compute), which determines the future state of robot and vertex and the actual movement (Move). In the case of multiple robots (Figure 6 (b)), we assume, that they cannot be placed on the same vertex at the same time. Communication between robots is limited to adjacent vertices and can be implemented by expanding the Look phase by the states of all adjacent robots.

Connectivity in the sense of this work is ensured if the union of all placed tiles and all used robots is completely connected. Accordingly, a robot can hold two components together, e.g., as shown in Figure 6 (c).

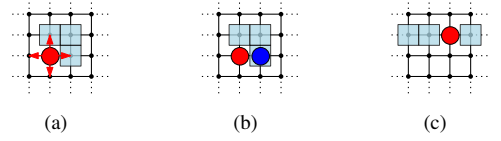


Fig. 6. (a) One robot and its possible moves. (b) Two robots on the grid. (c) Robots can hold separate connected components together.

III. CONSTRUCTING A BOUNDING BOX

Fekete et al. [22] showed how to construct a bounding box around a polyomino. However, that algorithm does not guarantee connectivity. We describe an algorithm to construct the bounding box keeping connectivity after each step. Due to space constraints, we only sketch technical details; see the full version of the paper [23] for a full description. To accomplish the required connectivity we specify without any loss of generality that the connection between $bb(P)$ and P must be on the south side of the boundary. For ease of presentation, the polyomino is shown in blue and the bounding box in gray; the robots cannot actually distinguish between those tiles.

In the following, we assume that two robots are placed adjacent to each other on an arbitrary tile of the polyomino P , and that the first robot R_1 (marked red) is the leader.

The construction can be split into three phases: (1) finding a start position, (2) constructing the bounding box, and (3) the clean-up. To find a suitable start position, we search for a locally y -minimal vertex that is occupied by a tile. This can be done by scanning the current row and moving downwards whenever possible. The search is done by the leader robot R_1 , followed by R_2 . Then R_2 positions itself on the first vertex beneath this locally y -minimal vertex. Afterwards, R_1 starts the bounding box construction one vertex further down. This brings us to phase (2).

The construction of the bounding box is performed clockwise around P , i.e., whenever possible, R_1 makes a right turn. At some point, R_1 finds a tile either belonging to P or to the bounding box. In the latter case we are done with phase (2) and can proceed to phase (3). If it is a tile belonging to P , we need to shift the current line outwards until there is no more conflict, then continue the construction (see Fig. 7). If the line to shift is the first line of the constructed bounding box, we know that there exists a tile of P that has a lower y -coordinate than the current starting position. Therefore,

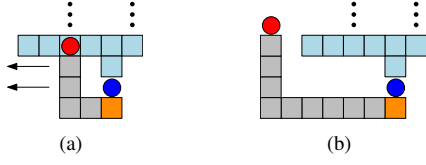


Fig. 7. (a) R_1 hits a tile belonging to P . (b) The triggered shifting process is finished.

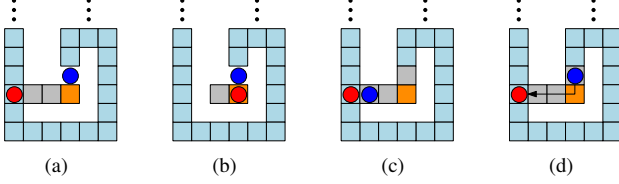


Fig. 8. Traversing a gap by building a bridge

we build a bridge to traverse this gap, as shown in Fig. 8. Afterwards, we can restart from phase (1).

To decide whether a tile t belongs to P or the current bounding box, we start moving around the boundary of the shape t belongs to. At some point, R_1 reaches R_2 . If R_1 is above R_2 then t is a tile of P , otherwise t is a tile of the bounding box.

For phase (3), consider the case when R_1 reaches a tile from the bounding box. If the hit tile is not a corner tile, the current line needs to be shifted outwards until the next corner is reached (see Figure 9(a)). Then we can search for another suitable connection between P and $bb(P)$, place a tile there, and get to R_2 to remove unnecessary parts of the bounding box (see Figure 9(b)-(d)). Because $bb(P)$ has only one tile with three adjacent tiles left, we can always find the connection between P and $bb(P)$.

Theorem 1: Given a polyomino P of width w and height h , building a bounding box surrounding P with the need that boundary and P are always connected, can be done with two robots in $O(\max(w, h) \cdot (wh + k \cdot |\partial P|))$ steps, where k is the number of convex corners in P .

The proof of this theorem is analogous to that from [22]; see [23] for full details.

If we know in advance that the given polyomino contains no holes, then we can build a non-simple bounding box. This requires only one robot, because we can at any moment distinguish the polyomino from the bounding box without having a second robot holding both parts together. This yields the following corollary.

Corollary 1: Given a simple polyomino P of width w and height h , building a bounding box surrounding P with the need that boundary and P are always connected, can be done with one robot in $O(\max(w, h) \cdot wh)$ steps.

IV. SCALING POLYOMINOES

Now we consider scaling a given shape by a factor c . Note that reducing the size of a polyomino by a factor (“scaling down”) can then be done in a similar fashion. In

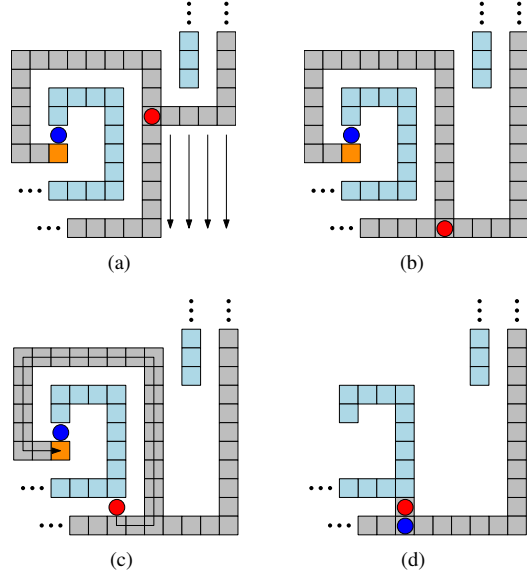


Fig. 9. The second case of finishing the bounding box. (a) An already constructed part of the bounding box is hit. (b) The last boundary side is shifted. (c) R_1 found a suitable new connectivity vertex above the southern side, places a tile and retraces its path to the initial starting position. (d) The unnecessary part of the bounding box is removed and both robots catch up to the new connection.

the following we assume that the robot R_1 already built the bounding box and is positioned on one of its tiles.

A. Scaling

The scaling process can be divided into two phases: (1) the preparation phase, and (2) the scaling phase. In phase (1) we fill up the last, i.e., rightmost column within $bb(P)$, add a tile in the second last column above the south side of $bb(P)$, and remove the lowest tile (called *column marker*) and third lowest tile (called *row marker*) on the east side of $bb(P)$ (see Fig. 10(a)). This gives us three columns within the bounding box (including $bb(P)$ itself). The first (from west to east) is the current column of P to scale. The second column, which is filled with tiles excepting the topmost row, is used to ensure connectivity and helps to recognize the end of the current column. The third column marks the current overall progress, i.e., we can find the tile in the correct current column and row that we want to scale next.

In phase (2), we simply search for the tile to scale, and place the row marker one vertex upwards. For possible cases, see Fig. 10. When we reach the top row of the bounding box, we move the column marker one vertex to the left and place a new row marker. Then we add a $c \times c$ square to the left of $bb(P)$. If we did not move the column marker, we move left from the south side of $bb(P)$ until we reach an end and start moving up until we find the place to build the $c \times c$ -square. Otherwise, we do not move upwards and build the square after the leftward moves. If the vertex to scale is empty, then we leave out one tile within the square.

After scaling a column that only contained empty vertices, we know that we are done with scaling. Thus, we can start removing all tiles, proceeding columnwise within $bb(P)$ from

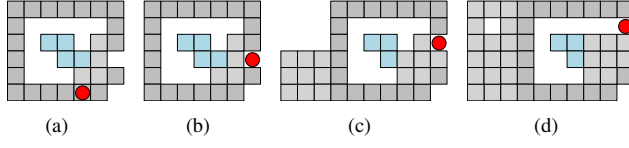


Fig. 10. (a) Configuration after the preparation phase. (b)-(d) Cases that appear during the scaling: (b) Scaling an occupied vertex; (c) scaling an empty vertex; (d) reaching the end of a column.

right to left. If necessary, all scaled empty tiles can also be removed by one scan through the scaled field.

Theorem 2: After building $bb(P)$, scaling a polyomino P of width w and height h by a constant scaling factor c without loss of connectivity can be done with one robot in $O(wh \cdot (c^2 + cw + ch))$ steps.

Proof: Correctness: We scan through the whole bounding box of P and scale every position. This implies that we scale every tile of P . Because we scale columnwise, we ensure that every scaled tile is built at the correct position. Connectivity is guaranteed because we never remove a tile that is necessary to have connectivity.

Time: Each of the $w \cdot h$ vertices within the bounding box of P is scaled. To this end, the robot has to move $O(c(w + h))$ steps to reach the position, where the scaled vertex needs to be constructed. A further $O(c^2)$ steps are needed to construct the tile. Finding the next vertex to scale takes $O(c(w + h))$ steps, including going back, find the correct column and row, and moving the row and column marker. In total we have a runtime of $O(wh(c^2 + cw + ch))$. ■

B. Adapting Algorithms

As shown in Fig. 11, there are algorithms that may not guarantee connectivity. An immediate consequence of being able to scale a given shape is that we can simulate any algorithm \mathcal{A} within the Robot-on-Tiles model while guaranteeing connectivity: We first scale the polyomino by three and then execute \mathcal{A} by always performing three steps into one direction if \mathcal{A} does one step. If at some point the robot needs to move through empty vertices, then we place a 3×3 -square with the middle vertex empty (if a clean up is desired at the end of \mathcal{A} , i.e., removing all scaled empty vertices, we fill up the complete row/column with these squares). This guarantees connectivity during the execution and we obtain the following theorem.

Theorem 3: If there is an algorithm \mathcal{A} for some problem Π in the Robots-on-Tiles model with runtime $\mathcal{T}(\mathcal{A})$, such that the robot moves within a $w' \times h'$ rectangle, then there is an algorithm \mathcal{A}' for Π with runtime $O(wh \cdot (c^2 + cw + ch) + \max((w' - w)h', (h' - h)w') + c \cdot \mathcal{T}(\mathcal{A}))$ guaranteeing connectivity during execution.

V. SIMULATION

As a key step in realizing a full robotic implementation of the bounding box construction, a tile-based simulator was developed¹. This python-based simulation allows for a

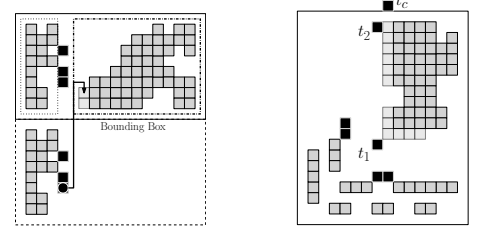


Fig. 11. Figures from [22] showing how to copy (left) or rotate (right) a given shape. We can clearly see that the tiles are not connected. Theorem 3 guarantees that this kind of construction can be modified to be performed in a connected fashion.

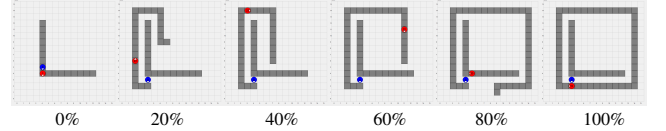


Fig. 12. 2D simulation snapshots from building a bounding box for an L-shaped polyomino.

clearer understanding of the subtle details in the execution of the approach. It models the states of the tiles as well as the robots and elucidates the state transitions. The visualization provides a snapshot of these states and how they unfold. It aides in understanding the scalability of the approach as well as the behavior of the approach on different shapes. It also provided the motion plan for the full 3D simulation of the algorithm with the virtual BILL-E robot.

The implemented state machine has 34 states, which include some optimizations for shifting long stretches of tiles. When the robots are in close proximity, they behave synchronously. In each state the robots first sense one unit in the directions left, up, right, and down. Each step in the simulation consists of one or all of the following actions: {look; communicate; turn; move one space; add, move or delete a tile; turn}. A build sequence from the simulator is shown in Fig. 12.

A plot showing the percentage of steps spent in four classes of states is shown in Fig. 14, for constructing bounding boxes around L-shaped polyominoes. The state classes for these plots are *Initial Search* (which searches for the local y -minimum of the seed polyomino), *Add/Shift Tiles* (which either adds a tile or shifts an already placed tile by 90°), *Delete Tiles* (which removes a tile), or *Move/Search* (which either moves to the end of the bounding box under construction or searches to see if the current tile is part of the bounding box or the seed polyomino).

As the size of the polyomino grows, the number of steps required grows quadratically. The time required to *Move/Search* dominates the other classes. The number of steps required to construct a bounding box around six simple shapes of polyominoes are compared in Fig. 15. Filled squares require the fewest steps to construct a bounding box, while L-shaped sets require considerably more time. Most of this extra time is spent searching to determine if an encountered tile is part of the polyomino or bounding box. Four classes of shapes that are identical up to a rotation are

¹<https://github.com/AlienHunterD/2DTileRobot>

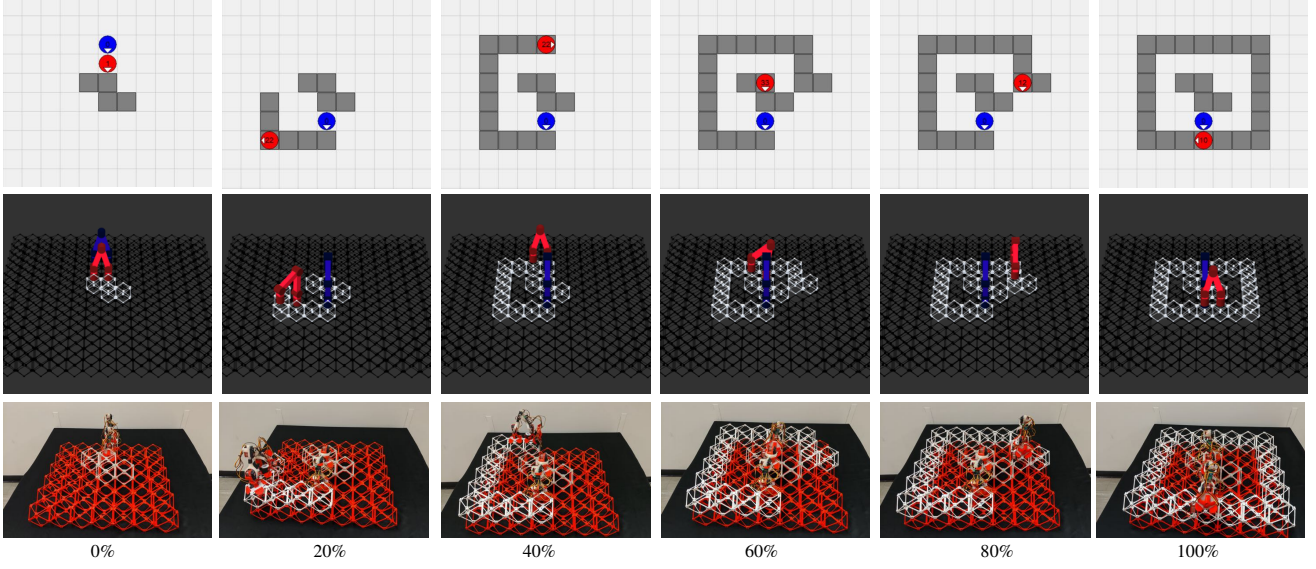


Fig. 13. Snapshots from building a bounding box for z-shaped polyomino using 2D simulator, 3D simulator, and staged hardware robots, synchronized so all are shown at steps $\{0, 24, 48, 72, 96, 120\}$.

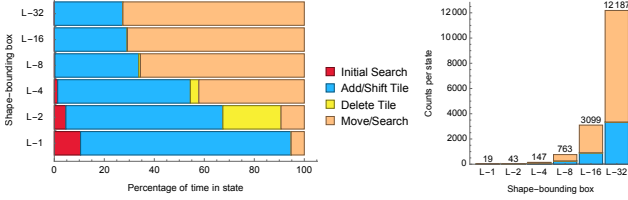


Fig. 14. Percentage of time spent in states for an L-shaped seed polyomino. The L-1 is a single tile, while the L-32 extends from the bottom-left tile a single row of tiles 32 wide and a single column 32 tiles tall.

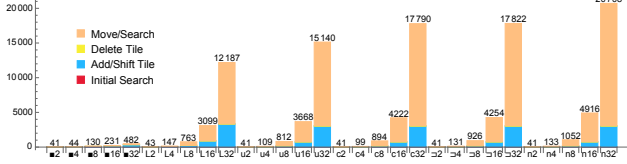


Fig. 15. Steps required for six canonical shapes. All shapes fill an $n \times n$ bounding box, with $n \in \{2, 4, 8, 16, 32\}$. Shapes include filled squares, and five shapes composed of single width polyomino lines: L-shapes, u-shapes, c-shapes, \sqcap -shapes, and n-shapes. The time requirements increase from left to right.

u, c, \sqcap , and n-shaped sets. The differences in time for these classes reflect the arbitrary choice to turn clockwise when determining if a tile belongs to the polyomino. These simulations show many opportunities for improved efficiency. The time for L-shapes is dominated by running the routine to determine if the tile encountered is the boundary or the seed polyomino. Adding more states could approximate wall following.

The tile-based simulator was used to construct a 3D simulation using the BILL-E robots and magnetic voxels. The tile-based simulator, the 3D simulator, and two hardware robots constructing a bounding box are shown in Fig. 13. Moving from the tile-based approach required modifications.

Among those are: (1) The robot must physically reach out to look at a neighboring voxel (and thus must not collide with another robot). (2) The robot can only move from one tile to the neighbor by performing a “cartwheel” move. In practice, performing an inch-worm maneuver that places a leg two tiles away, then moves the trailing leg forward, moves the robot more quickly. (3) At this point, the implementation details of obtaining a voxel to add or deleting a voxel are omitted. In the future, robots may carry a supply of voxels, move them to or from appropriately placed depots when this supply is exhausted, be supplied by a number of dedicated gopher robots, or may even construct and consume voxels (i.e. 3D printing/ filament recycling).

VI. CONCLUSION

We demonstrated how geometric algorithms for finite automata can be used to enable very simple robots to perform a number of fundamental but non-trivial construction tasks, such as building a bounding box and scaling a given shape by some constant, that guarantee connectivity between all tiles and robots during their execution, and also provided a practical realization.

There is a whole range of possible extensions. Is it possible to scale general polyominoes without the preceding bounding box construction? A possible approach could be to cut the polyomino into a subset of monotone polyominoes, which could be handled separately. Expanding the existing repertoire of operations to three-dimensional configurations and operations is another logical step. An equally relevant challenge is to develop distributed algorithms with multiple robots that are capable of solving a range of problems with the requirement of connectivity, without having to rely on the preceding scaling procedure that we used in our work. Other questions arise from additional requirements of real-world applications, such as the construction and reconfiguration of space habitats.

REFERENCES

- [1] J. Balanza-Martinez, A. Luchsinger, D. Caballero, R. Reyes, A. A. Cantu, R. Schweller, L. A. Garcia, and T. Wylie, "Full tilt: universal constructors for general shapes with uniform external forces," in *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019, pp. 2689–2708.
- [2] A. T. Becker, S. P. Fekete, P. Keldenich, D. Krupke, C. Rieck, C. Scheffer, and A. Schmidt, "Tilt assembly: algorithms for micro-factories that build objects with uniform external forces," *Algorithmica*, pp. 1–23, 2017.
- [3] W. K. Belvin, W. R. Doggett, J. J. Watson, J. T. Dorsey, J. E. Warren, T. C. Jones, E. E. Komendera, T. Mann, and L. M. Bowman, "In-space structural assembly: Applications and technology," in *3rd AIAA Spacecraft Structures Conference*, 2016, p. 2163.
- [4] M. A. Bender and D. K. Slonim, "The power of team exploration: two robots can learn unlabeled directed graphs," in *35th Annual Symposium on Foundations of Computer Science (FOCS)*, 1994, pp. 75–85. [Online]. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=365703>
- [5] 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 (FOCS)*, 1978, pp. 132–142. [Online]. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4567972>
- [6] P. Brass, F. Cabrera-Mora, A. Gasparri, and J. Xiao, "Multirobot tree and graph exploration," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 707–717, Aug 2011.
- [7] C. Chalk, E. Martinez, R. Schweller, L. Vega, A. Winslow, and T. Wylie, "Optimal staged self-assembly of general shapes," *Algorithmica*, vol. 80, no. 4, pp. 1383–1409, 2018.
- [8] K. C. Cheung and N. Gershenfeld, "Reversibly assembled cellular composite materials," *Science*, vol. 341, no. 6151, pp. 1219–1221, 2013.
- [9] A. Costa, A. Abdel-Rahman, B. Jenett, N. Gershenfeld, I. Kostitsyna, and K. Cheung, "Algorithmic approaches to reconfigurable assembly systems," in *IEEE Aerospace Conference*, 2019, pp. 1–8.
- [10] N. B. Cramer, D. W. Cellucci, O. B. Formoso, C. E. Gregg, B. E. Jenett, J. H. Kim, M. Lendraitis, S. S. Swee, G. T. Trinh, K. V. Trinh *et al.*, "Elastic shape morphing of ultralight structures by programmable assembly," *Smart Materials and Structures*, vol. 28, no. 5, p. 055006, 2019.
- [11] S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro, "Map construction of unknown graphs by multiple agents," *Theoretical Computer Science*, vol. 385, no. 1, pp. 34 – 48, 2007.
- [12] J. J. Daymude, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann, "Improved leader election for self-organizing programmable matter," in *Algorithms for Sensor Systems*, Cham, 2017, pp. 127–140.
- [13] E. D. Demaine, M. L. Demaine, S. P. Fekete, M. Ishaque, E. Rafalin, R. T. Schweller, and D. L. Souvaine, "Staged self-assembly: nanomanufacture of arbitrary shapes with $o(1)$ glues," *Natural Computing*, vol. 7, no. 3, pp. 347–370, 2008.
- [14] Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann, "Brief announcement: Amoebot – a new model for programmable matter," in *26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2014, pp. 220–222.
- [15] Z. Derakhshandeh, R. Gmyr, A. Porter, A. W. Richa, C. Scheideler, and T. Strothmann, "On the runtime of universal coating for programmable matter," in *22nd International Conference on DNA Computing and Molecular Programming (DNA)*, 2016, pp. 148–164.
- [16] 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 *2nd International Conference on Nanoscale Computing and Communication (NANOCOM)*, 2015, pp. 21:1–21:2.
- [17] —, "Universal shape formation for programmable matter," in *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, 2016, pp. 289–299.
- [18] —, "Universal coating for programmable matter," *Theoretical Computer Science*, vol. 671, pp. 56–68, 2017.
- [19] Z. Derakhshandeh, R. Gmyr, T. Strothmann, R. Bazzi, A. W. Richa, and C. Scheideler, "Leader election and shape formation with self-organizing programmable matter," in *21st International Conference on DNA Computing and Molecular Programming (DNA)*, 2015, pp. 117–132.
- [20] G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi, "Shape formation by programmable particles," *Distributed Computing*, vol. 33, no. 1, pp. 69–101, 2020.
- [21] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc, "Tree exploration with little memory," *Journal of Algorithms*, vol. 51, no. 1, pp. 38 – 63, 2004.
- [22] S. P. Fekete, R. Gmyr, S. Hugo, P. Keldenich, C. Scheffer, and A. Schmidt, "CADbots: Algorithmic aspects of manipulating programmable matter with finite automata," *CoRR*, vol. abs/1810.06360, 2018. [Online]. Available: <https://arxiv.org/pdf/1810.06360.pdf>
- [23] S. P. Fekete, E. Niehs, C. Scheffer, and A. Schmidt, "Connected assembly and reconfiguration by finite automata," *CoRR*, 2019. [Online]. Available: <https://arxiv.org/pdf/1909.03880.pdf>
- [24] R. Fleischer and G. Trippen, "Exploring an unknown graph efficiently," in *13th European Symposium on Algorithms (ESA)*, 2005, pp. 11–22.
- [25] P. Fraigniaud, L. Gasieniec, D. R. Kowalski, and A. Pelc, "Collective tree exploration," *Networks*, vol. 48, no. 3, pp. 166–177, 2006.
- [26] P. Fraigniaud and D. Ilcinkas, "Digraphs exploration with little memory," in *21st Symposium on Theoretical Aspects of Computer Science (STACS)*, 2004, pp. 246–257.
- [27] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg, "Graph Exploration by a Finite Automaton," *Theoretical Computer Science*, vol. 345, no. 2-3, pp. 331–344, Nov. 2005.
- [28] L. Gasieniec, A. Pelc, T. Radzik, and X. Zhang, "Tree exploration with logarithmic memory," in *18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007, pp. 585–594.
- [29] L. Gasieniec and T. Radzik, "Memory efficient anonymous graph exploration," in *34th Workshop Graph-Theoretic Concepts in Computer Science (WG)*, 2008, pp. 14–29.
- [30] R. Gmyr, I. Kostitsyna, F. Kuhn, C. Scheideler, and T. Strothmann, "Forming tile shapes with a single robot," in *33rd European Workshop on Computational Geometry (EuroCG)*, 2017, pp. 9–12.
- [31] 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)*, vol. 117, 2018, pp. 52:1–52:15.
- [32] R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler, and T. Strothmann, "Forming tile shapes with simple robots," in *24th International Conference on DNA Computing and Molecular Programming (DNA)*, 2018, pp. 122–138.
- [33] S. C. Goldstein and T. Mowry, "Claytronics: A scalable basis for future robots," *Robosphere*, Nov. 2004.
- [34] C. E. Gregg, B. Jenett, and K. C. Cheung, "Assembled, modular hardware architectures - what price reconfigurability?" in *2019 IEEE Aerospace Conference*, 2019, pp. 1–10.
- [35] C. E. Gregg, J. H. Kim, and K. C. Cheung, "Ultra-light and scalable composite lattice materials," *Advanced Engineering Materials*, vol. 20, no. 9, p. 1800213, 2018.
- [36] S. Hugo, "Robots on tiles: Recognition of polyomino properties using constant memory," Bachelor's Thesis, TU Braunschweig, 2018.
- [37] F. Hurtado, E. Molina, S. Ramaswami, and V. Sacristán, "Distributed reconfiguration of 2D lattice-based modular robotic systems," *Autonomous Robots*, vol. 38, no. 4, pp. 383–413, Apr 2015.
- [38] B. Jenett and K. Cheung, "Bill-e: Robotic platform for locomotion and manipulation of lightweight space structures," in *25th AIAA/AHS Adaptive Structures Conference*, 2017, p. 1876.
- [39] B. Jenett, A. Abdel-Rahman, K. C. Cheung, and N. Gershenfeld, "Material-robot system for assembly of discrete cellular structures," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4019–4026, 2019.
- [40] B. Jenett, S. Calisch, D. Cellucci, N. Cramer, N. Gershenfeld, S. Swee, and K. C. Cheung, "Digital morphing wing: active wing shaping concept using composite lattice-based cellular structures," *Soft Robotics*, vol. 4, no. 1, pp. 33–48, 2017.
- [41] B. Jenett and D. Cellucci, "A mobile robot for locomotion through a 3d periodic lattice environment," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5474–5479.
- [42] B. Jenett, D. Cellucci, C. Gregg, and K. Cheung, "Meso-scale digital materials: modular, reconfigurable, lattice-based structures," in *ASME 2016 11th International Manufacturing Science and Engineering Conference*, 2016.
- [43] B. Jenett, C. Gregg, D. Cellucci, and K. Cheung, "Design of multifunctional hierarchical space structures," in *IEEE Aerospace Conference*, 2017, pp. 1–10.

- [44] A. Naz, B. Piranda, J. Bourgeois, and S. C. Goldstein, "A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots," in *IEEE 15th International Symposium on Network Computing and Applications (NCA)*, 2016, pp. 254–263.
- [45] P. Panaite and A. Pelc, "Exploring unknown undirected graphs," *Journal of Algorithms*, vol. 33, no. 2, pp. 281 – 295, 1999.
- [46] M. J. Patitz, "An introduction to tile-based self-assembly and a survey of recent results," *Natural Computing*, vol. 13, no. 2, pp. 195–224, 2014.
- [47] A. Schmidt, S. Manzoor, L. Huang, A. T. Becker, and S. P. Fekete, "Efficient parallel self-assembly under uniform control inputs," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3521–3528, 2018.
- [48] T. Tucci, B. Piranda, and J. Bourgeois, "A distributed self-assembly planning algorithm for modular robots," in *17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2018, pp. 550–558.
- [49] E. Winfree, "Algorithmic self-assembly of DNA," Ph.D. dissertation, California Institute of Technology, 1998.
- [50] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman, "Design and self-assembly of two-dimensional DNA crystals," *Nature*, vol. 394, no. 6693, p. 539, 1998.
- [51] D. Woods, H.-L. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin, "Active self-assembly of algorithmic shapes and patterns in polylogarithmic time," in *4th Conference on Innovations in Theoretical Computer Science (IITCS)*, 2013, pp. 353–354.



Parallel Online Algorithms for the Bin Packing Problem

Sándor P. Fekete, Jonas Grosse-Holz, Phillip Keldenich, and Arne Schmidt^(✉)

Department of Computer Science, TU Braunschweig, Braunschweig, Germany
{s.fekete,j.grosse-holz,p.keldenich,arne.schmidt}@tu-bs.de

Abstract. We study *parallel* online algorithms: For some fixed integer k , a collective of k parallel processes that perform online decisions on the same sequence of events forms a *k-copy algorithm*. For any given time and input sequence, the overall performance is determined by the best of the k individual total results. Problems of this type have been considered for online makespan minimization; they are also related to optimization with *advice* on future events, i.e., a number of bits available in advance.

We develop PREDICTIVE HARMONIC₃ (PH3), a relatively simple family of k -copy algorithms for the online Bin Packing Problem, whose joint competitive factor converges to 1.5 for increasing k . In particular, we show that $k = 6$ suffices to guarantee a factor of 1.5714 for PH3, which is better than 1.57829, the performance of the best known 1-copy algorithm ADVANCED HARMONIC, while $k = 11$ suffices to achieve a factor of 1.5406, beating the known lower bound of 1.54278 for a single online algorithm. In the context of online optimization with advice, our approach implies that 4 bits suffice to achieve a factor better than this bound of 1.54278, which is considerably less than the previous bound of 15 bits.

Keywords: Online algorithms · Bin packing · Competitive analysis

1 Introduction

When dealing with unknown future events, optimization with incomplete information typically considers the competitive factor of an online algorithm as its performance measure; the objective becomes to develop a single strategy that performs reasonably well against the worst case. This focus on just *one* option is more restrictive than hedging strategies in a wide variety of other scientific and application fields; these typically make use of *several* parallel choices, thereby increasing the chance that one of them will yield satisfactory results. Examples include scenarios from biology, where a large and diverse progeny increases the odds of surviving offspring; finance and insurance, where a suitable combination of investment strategies is employed to balance a portfolio against extreme losses;

A full version is available on arxiv.org [10].

Phillip Keldenich was partially supported by DFG grant FE407/17-2 as part of the Research Group FOR 1800, “Controlling Concurrent Change”.

© Springer Nature Switzerland AG 2020

E. Bampis and N. Megow (Eds.): WAOA 2019, LNCS 11926, pp. 106–119, 2020.

https://doi.org/10.1007/978-3-030-39479-0_8

and engineering, where redundancy is used to protect against catastrophic failure, either on individual components (such as parts in a machine) or on whole systems (such as automata in a robot swarm or spacecraft in a group of satellites), where it suffices that just one machine delivers a good outcome.

In this paper, we consider such *parallel* online strategies: Instead of making a single sequence of decisions, we consider k parallel processes for some fixed integer k , which we call a *k-copy algorithm*; the objective is to make the best of these k outcomes as good as possible, even in the worst case. We demonstrate the potential of this approach for the well-studied Bin Packing Problem, for which it is known that no single deterministic online algorithm can achieve a competitive factor below 1.5401.

1.1 Our Results

We define a family of k -copy algorithms for the online Bin Packing Problem, called PREDICTIVE HARMONIC₃ (PH3), whose asymptotic competitive ratio converges to 1.5 for large k . We show that $k = 6$ suffices to guarantee a factor of 1.5714, which is better than 1.57829, the performance of the best known 1-copy algorithm ADVANCED HARMONIC [3]. Moreover, $k = 11$ suffices to achieve a competitive ratio of 1.5406 beating the known lower bound of 1.54278 for a 1-copy algorithm [4]. In the context of online optimization with advice, our approach implies that 4 bits suffice to achieve less than 1.5401, which is considerably less than the previous bound of 16 bits of REDBLUE by Angelopoulos et al. [2]; in fact, for $k = 16$ (corresponding to four bits of advice) PH3 achieves a ratio of 1.5305, compared to 3.3750 for REDBLUE, while $k = 65,536$ (corresponding to 16 bits of advice) yields a factor of 1.5001 for PH3, but 1.5293 for REDBLUE.

1.2 Related Work on Online Bin Packing

There is a wide range of online algorithms for bin packing. The Next Fit algorithm [9] achieves a competitive ratio of 2, whereas “Almost Any Fit” algorithms [13] like First Fit or Best Fit achieve competitive ratios of 1.7.

An important online bin packing algorithm is HARMONIC_M, introduced by Lee and Lee [15], which achieves a competitive ratio of less than 1.692 for $M \rightarrow \infty$. Based on HARMONIC_M, SON OF HARMONIC by Heydrich and van Stee [12] achieves a competitive ratio of 1.5816. The currently best known algorithm is ADVANCED HARMONIC, which achieves a competitive ratio of 1.57829 [3].

For lower bounds, Yao [20] established a value of $3/2$ that was later improved to 1.536, independently by Brown [8] and by Liang [16]. Using a generalization of their methods, van Vliet [19] proved a lower bound of 1.5401. Balogh et al. [4] improved the lower bound to 1.54278.

1.3 Related Work on Online Bin Packing with Advice

In the context of online algorithms with advice, Boyar et al. [7] showed that an online algorithm with $n \lceil \log(OPT(I)) \rceil$ bits of advice is sufficient and that

at least $(n - 2OPT(I)) \cdot \log(OPT(I))$ bits of advice are necessary to achieve optimality. In the same paper, they presented an online bin packing algorithm, namely RESERVECRITICAL, with $O(\log(n)) + o(\log(n))$ bits of advice that is 1.5-competitive and an algorithm with $2n + o(n)$ bits of advice that is $\frac{4}{3}$ -competitive. Zhao and Shen [21] developed an algorithm using $3n + o(n)$ bits of advice achieving a competitive ratio of $\frac{5}{4}OPT + 2$. Renault et al. [18] developed an $(1 + \varepsilon)$ -competitive algorithm using $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ bits of advice per request.

Based on RESERVECRITICAL, Angelopoulos et al. [2] developed the algorithm REDBLUE with constant advice that is 1.5-competitive. Their second algorithm achieves a competitive ratio of $1.47012 + \varepsilon$ with finite advice that is exponentially dependent of ε . However, to beat the competitive ratio of 1.5 already an enormous amount of advice is needed, which makes the algorithm impractical.

In terms of lower bounds, Boyar et al. [7] proved that no competitive ratio better than $9/8$ can be reached by any algorithm that uses sub-linear advice. Angelopoulos et al. [2] improved this bound to $7/6$.

1.4 Related Work on Parallel Online Algorithms

Parallel algorithms have already been considered in the field of online algorithms with advice. Boyar et al. [6] presented an algorithm for the online list update problem, making use of 2 bits of advice to choose one out of three algorithms. This algorithm achieves a competitive ratio of $5/3$, beating the lower bound for conventional online algorithms of 2. A practical application of this algorithm was shown by Kamali and Ortiz [14], who applied it in the Burrows-Wheeler transform compression. More work on parallel online algorithms include parallel scheduling [1], finding independent sets [11] and the “multiple-cow” version of the linear search problem [17].

While online algorithms with advice mostly focus on the amount of advice to allow classification of online algorithms and problems, k -copy online algorithms focus on small finite values for k and thus small finite amounts of advice, with more emphasis on practical application. The perspective on different algorithms running in parallel instead of abstract arbitrary information facilitates finer optimization in some cases.

Also, when considering online algorithms with advice, the number of algorithms can only be doubled by increasing the amount of advice by one bit. The perspective of k -copy algorithms allows arbitrary $k \in \mathbb{N}$ for the number of algorithms.

2 Preliminaries

2.1 k -Copy Online Algorithms

In this paper, we consider k online algorithms A_1, \dots, A_k , each of them processing the same input list I in parallel. We call the set $\mathcal{A} := \{A_1, \dots, A_k\}$ a *k -copy online algorithm*.

For an input list I and an online algorithm A , let $A(I)$ denote the number of bins used by A and $\text{OPT}(I)$ denote the number of bins used in an optimal offline solution. The absolute competitive ratio $R_{\mathcal{A}}$ for a k -copy online algorithm \mathcal{A} is defined as

$$R_{\mathcal{A}} = \sup_I \left\{ \frac{\min_{A \in \mathcal{A}} A(I)}{\text{OPT}(I)} \right\}.$$

The asymptotic competitive ratio $R_{\mathcal{A}}^{\infty}$ for algorithm \mathcal{A} is defined as

$$R_{\mathcal{A}}^{\infty} = \lim_{n \rightarrow \infty} \sup_I \left\{ \frac{\min_{A \in \mathcal{A}} A(I)}{\text{OPT}(I)} \mid \text{OPT}(I) = n \right\}$$

As already stated by Boyar et al. [5], any k -copy online algorithm can be converted into an online algorithm with advice, and vice versa.

Lemma 1. *Any k -copy online algorithm can be converted into an online algorithm with $l = \lceil \log_2(k) \rceil$ bits of advice that achieves the same competitive ratio. Conversely, any online algorithm with $l \in \mathbb{N}$ bits of advice can be converted into a k -copy online algorithm without advice with $k = 2^l$ that achieves the same competitive ratio.*

Proof. Let $\mathcal{A} = \{A_1, A_2, \dots, A_k\}$ be a k -copy algorithm. Construct the online algorithm A' that gets a value $i \in \{1, 2, \dots, k\}$ as advice, specifying the index i of the algorithm $A_i \in \mathcal{A}$ that performs best on the given input sequence. The value i can be encoded using $\lceil \log_2(k) \rceil$ bits. A' then behaves like A_i and thus achieves the same competitive ratio as \mathcal{A} .

Let A be an online algorithm that gets $l \in \mathbb{N}$ bits of advice. Construct the online k -copy algorithm \mathcal{A}' with $k = 2^l$ algorithms $A_i, i \in \{1, 2, \dots, k\}$. For each $i \in \{1, 2, \dots, k\}$, the algorithm A_i behaves like A given i encoded in binary as advice. As the values $i \in \{1, 2, \dots, k\}$ cover every possible configuration of the advice bits, for any advice given to A , there is an algorithm $A_i \in \mathcal{A}'$, that assumes this advice. Accordingly, there is an algorithm $A_i \in \mathcal{A}'$, that performs as well as A , i.e., the best algorithm $A_j \in \mathcal{A}$ that performs at least as well as A . Thus, \mathcal{A}' performs at least as well as A .

2.2 Bin Packing

In the online version of bin packing, we are given a list of items $I := \langle a_1, \dots, a_n \rangle$ with $a_i \in (0, 1]$ for $i \in \{1, \dots, n\}$. These items must be packed by an algorithm, one at a time, without any information on subsequent items and without the possibility to change previous decisions. The goal is to pack all items into a minimum number of bins with unit capacity.

Definition 1 (Item size).

Let $S = [0, \frac{1}{3}]$, $M = (\frac{1}{3}, \frac{1}{2}]$, $L = (\frac{1}{2}, \frac{2}{3}]$ and $XL = [\frac{2}{3}, 1]$. We call items in S small, items in M medium, items in L large and items in XL extra large. For a list $I = \langle a_1, a_2, \dots, a_n \rangle$, the set of items $\text{Set}(I) \cap XL$ is noted as I_{XL} for improved readability. The subsets I_L , I_M and I_S are used analogously.

Definition 2 (Size function). Let S be a set (or list) of items. Then, $\text{size}(S) := \sum_{i \in S} i$. For a bin b , we refer to $\text{size}(b)$ as the size of the bin, i.e., the sum of items already packed in b .

Definition 3 (Sub-bins).

Given a bin b , it can be split into two parts b_1 and b_2 , such that the sum of their capacities is equal to the capacity of b . We refer to b_1 and b_2 as sub-bins. We call a sub-bin with capacity C a C -sub-bin.

As sub-bins are not packed with an amount larger than their capacity, each sub-bin can be packed independently from the other.

3 PREDICTIVE HARMONIC₃

Now we introduce the algorithm PREDICTIVE HARMONIC₃ (PH3). Although developed independently, it bears many similarities to RESERVECRITICAL and REDBLUE. PH3 uses the same classifications as the other two algorithms and tries to pack all large items with small items, such that the corresponding bins are packed to a level of at least $2/3$. However, in contrast to REDBLUE, the information needed by PH3 does not depend on the result of RESERVECRITICAL, but only on the number and size of certain item types, and can be calculated in linear time.

The main idea of PH3 is to guess the ratio of how many small items must be packed with large items to obtain a packing density of $2/3$. Having multiple instances of PH3, every instance can guess a different ratio to get close to a competitive ratio of 1.5.

Algorithm 1 PREDICTIVE HARMONIC₃. Given a list $I = \langle a_1, a_2, \dots, a_n \rangle$ of items $a_i \in (0, 1]$, $i \in 1, \dots, n$, and a ratio $r_L \in [0, 1]$, the algorithm packs the items as follows:

- Extra large items are packed into individual bins. These bins are called XL-bins, the set of all XL-bins is called B_{XL} .
- Large items are packed into individual bins. These bins are called L-bins, the set of all L-bins is called B_L . Furthermore, we split each L-bin into a $\frac{2}{3}$ -sub-bin (for large items) and a $\frac{1}{3}$ -sub-bin (for small items).
- Medium items are packed into separate bins together with other medium items (note that at most two of them fit into one bin). These bins are called M-bins, the set of all M-bins is called B_M .
- Small items are packed into a $\frac{1}{3}$ -sub-bin of L-bins in a next fit manner, if the size of small items packed into L-bins is smaller than r_L times the total size of small items packed so far; otherwise we pack the small item into S-bins.

3.1 Competitive Ratio

Using simple bounds for an optimal solution and performing a case analysis, we can prove the following theorem. Due to space constraints, the proof can be found in the full version [10].

Theorem 2. Let $r_L^* = \min \left\{ \frac{|I_L|}{6 \cdot \text{size}(I_S)}, 1 \right\}$ ¹ and $\delta = r_L - r_L^*$. PH3 achieves the asymptotic competitive ratio

$$R_{PH3}^\infty \leq \begin{cases} \frac{3}{2} + \min \left\{ \frac{1}{4r_L^*}, \frac{3}{6r_L^* + 2} \right\} (-\delta) & \text{for } \delta \leq 0 \\ \frac{3}{2} + \min \left\{ \frac{3}{4r_L^*}, \frac{9}{6r_L^* + 2} \right\} \delta & \text{for } \delta \geq 0. \end{cases}$$

3.2 Tightness

Theorem 3. For any $r_L, r_L^* \in [0, 1]$, the asymptotic competitive ratio given in Theorem 2 is tight.

Proof sketch: Let $\langle a_1, a_2, \dots, a_k \rangle \times n$ with $n \in \mathbb{N}$ denote n repetitions of the sequence $\langle a_1, a_2, \dots, a_k \rangle$. Let I be a sequence consisting of concatenated sub-sequences I_S , I_M and I_L , where I_S is a sequence consisting of two interleaved sub-sequences I_{SL} and I_{LL} . With $N \in \mathbb{N}$ and $\varepsilon = 1/(12N + 2)$, we define

$$\begin{aligned} I_L &= \left(\frac{1}{2} + \frac{\varepsilon}{2} \right) \times n_L \text{ with } n_L = \lceil 4r_L^* N \rceil \\ I_M &= \left(\frac{1}{3} + \frac{\varepsilon}{2} \right) \times n_M \text{ with } n_M = \begin{cases} 0 & \text{for } r_L^* \leq 1/3 \\ \lfloor (6r_L^* - 2)N \rfloor & \text{for } r_L^* \geq 1/3 \end{cases} \\ I_{SS} &= \left(\frac{1}{3} - 2\varepsilon, \frac{1}{6} - \varepsilon, \frac{1}{6} - \varepsilon, 12\varepsilon \right) \times n_{SS} \text{ with } n_{SS} = \lceil n'_{SS} \rceil = \lceil (1 - r_L)N \rceil \\ I_{SL} &= \left(\frac{1}{6} - \varepsilon, 3\varepsilon \right) \times n_{SL} \text{ with } n_{SL} = \lceil n'_{SL} \rceil = \lceil 4r_L N \rceil \end{aligned}$$

The proof is based on a case analysis of which item appears next and in which bin this item is packed by PH3. Due to space constraints, a full proof can be found in the full version [10].

4 Parallel PREDICTIVE HARMONIC₃

4.1 Competitive Ratio for PH3 as 1-Copy Online Algorithm

To optimize the performance for PH3 as a 1-copy algorithm, we determine the optimal value for r_L with respect to minimizing the asymptotic competitive ratio over all $r_L^* \in [0, 1]$.

¹ The intuition of this value is that at least $1/2$ of each $1/3$ -sub-bin must be filled to guarantee a packing density of $2/3$. Therefore, for $|I_L|$ bins, we have to fill up a total capacity of $\frac{|I_L|}{6}$ with small items.

Lemma 2 (Monotonicity of competitive ratio of PH3). *For any fixed $r_L \in [0, 1]$, the competitive factor is monotonically decreasing for $r_L^* \in [0, r_L]$ and monotonically increasing for $r_L^* \in [r_L, 1]$.*

Proof. Assume r_L to be fixed. Let $r_{+,<}, r_{-,<} : [0, 1/3] \rightarrow \mathbb{R}$ and $r_{+,>}, r_{-,>} : [1/3, 1] \rightarrow \mathbb{R}$ with

$$\begin{aligned} r_{-,<}(r_L^*) &= \frac{3}{2} + \frac{3}{6r_L^* + 2}(-\delta) &&= R_{PH3}^\infty \text{ for } \delta \leq 0, r_L^* \leq \frac{1}{3} \\ r_{-,>}(r_L^*) &= \frac{3}{2} + \frac{1}{4r_L^*}(-\delta) &&= R_{PH3}^\infty \text{ for } \delta \leq 0, r_L^* \geq \frac{1}{3} \\ r_{+,<}(r_L^*) &= \frac{3}{2} + \frac{9}{6r_L^* + 2}\delta &&= R_{PH3}^\infty \text{ for } \delta \geq 0, r_L^* \leq \frac{1}{3} \\ r_{+,>}(r_L^*) &= \frac{3}{2} + \frac{3}{4r_L^*}\delta &&= R_{PH3}^\infty \text{ for } \delta \geq 0, r_L^* \geq \frac{1}{3} \end{aligned}$$

Consider the derivative of $r_{-,<}$ and $r_{-,>}$.

$$\begin{aligned} \frac{\partial}{\partial r_L^*} r_{-,<}(r_L^*) &= \frac{\partial}{\partial r_L^*} \left(\frac{3}{2} + \frac{3}{6r_L^* + 2}(-\delta) \right) \\ &= \frac{\partial}{\partial r_L^*} \left(\frac{3(r_L^* - r_L)}{6r_L^* + 2} \right) \\ &= \frac{18r_L + 6}{(6r_L^* + 2)^2} \geq 0 \text{ for } 0 \leq r_L \leq r_L^* \leq \frac{1}{3} \\ \frac{\partial}{\partial r_L^*} r_{-,>}(r_L^*) &= \frac{\partial}{\partial r_L^*} \left(\frac{3}{2} + \frac{1}{4r_L^*}(-\delta) \right) \\ &= \frac{\partial}{\partial r_L^*} \left(\frac{r_L^* - r_L}{4r_L^*} \right) \\ &= \frac{r_L}{4(r_L^*)^2} \geq 0 \text{ for } 0 \leq r_L \leq r_L^* \text{ and } \frac{1}{3} \leq r_L^* \leq 1 \end{aligned}$$

As the derivatives of $r_{-,<}$ and $r_{-,>}$ are both non-negative in their respective domains, they are both monotonically increasing. Because $r_{-,<}(\frac{1}{3}) = r_{-,>}(\frac{1}{3})$, we conclude that the competitive ratio is monotonically increasing for $r_L^* \in [r_L, 1]$.

Now consider the derivative of $r_{+,<}$ and $r_{+,>}$.

$$\begin{aligned}
\frac{\partial}{\partial r_L^*} r_{+,<}(r_L^*) &= \frac{\partial}{\partial r_L^*} \left(\frac{3}{2} + \frac{9}{6r_L^* + 2} \delta \right) \\
&= \frac{\partial}{\partial r_L^*} \left(\frac{9(r_L - r_L^*)}{6r_L^* + 2} \right) \\
&= \frac{-54r_L - 18}{(6r_L^* + 2)^2} \leq 0 \text{ for } r_L^* \leq r_L \leq 1 \text{ and } 0 \leq r_L^* \leq \frac{1}{3} \\
\frac{\partial}{\partial r_L^*} r_{+,>}(r_L^*) &= \frac{\partial}{\partial r_L^*} \left(\frac{3}{2} + \frac{3}{4r_L^*} \delta \right) \\
&= \frac{\partial}{\partial r_L^*} \left(\frac{3(r_L - r_L^*)}{4r_L^*} \right) \\
&= \frac{-3r_L}{4(r_L^*)^2} \leq 0 \text{ for } \frac{1}{3} \leq r_L^* \leq r_L \leq 1
\end{aligned}$$

As the derivatives of $r_{+,<}$ and $r_{+,>}$ are both non-positive in their respective domains, they are both monotonically decreasing. Because $r_{+,<}(\frac{1}{3}) = r_{+,>}(\frac{1}{3})$, we conclude that the competitive ratio is monotonically decreasing for $r_L^* \in [0, r_L]$.

Because of Lemma 2, the competitive ratio does not decrease with r_L^* increasing for $\delta \leq 0$. Thus, as an upper bound on the competitive ratio for $\delta \leq 0$, only the competitive ratio for $r_L^* = 1$ has to be considered.

$$\begin{aligned}
R_{PH3}^\infty &\leq \frac{3}{2} + \frac{1}{4}(-\delta) \text{ for } \delta \leq 0 \\
&= \frac{3}{2} + \frac{1}{4}(1 - r_L) \\
&= \frac{7}{4} - \frac{r_L}{4}
\end{aligned}$$

For $\delta \geq 0$, the competitive ratio does not decrease with r_L^* decreasing. In this case, the competitive ratio for $r_L^* = 0$ is an upper bound on the competitive ratio.

$$\begin{aligned}
R_{PH3}^\infty &\leq \frac{3}{2} + \frac{9}{2}\delta \text{ for } \delta \geq 0 \\
&= \frac{3}{2} + \frac{9}{2}(r_L - 0) \\
&= \frac{3}{2} + \frac{9}{2}r_L
\end{aligned}$$

At the same time, these values are lower bounds on the overall competitive ratio. Given these bounds, this linear program can be formulated to minimize the competitive ratio:

$$\begin{aligned}
& \text{Minimize } R_{PH3}^\infty \\
& \text{Subject to } R_{PH3}^\infty \geq \frac{7}{4} - \frac{r_L}{4} \\
& \quad R_{PH3}^\infty \geq \frac{3}{2} + \frac{9}{2}r_L \\
& \quad r_L \geq 0 \\
& \quad r_L \leq 1
\end{aligned}$$

The optimal solution for this linear program is $r_L = 1/19$ and $R_{PH3}^\infty = 33/19 < 1.7369$. Figure 1 shows the asymptotic competitive ratio of PH3 over r_L^* for $r_L = 1/19$.

Compared to other known algorithms for online bin packing, PH3 is not a good choice for worst-case behavior. Among the classical algorithms, only NF and WF, both of which are 2-competitive, are worse than PH3. Any AAF algorithm achieves an asymptotic competitive ratio $R_{AAF}^\infty = 1.7$ [9] and thus performs slightly better than PH3. The best-performing online algorithm for bin packing currently known, SON OF HARMONIC, is 1.5816-competitive and thus clearly superior to PH3 [12].

However, if we know in advance that r_L^* is restricted to some interval $I_r = [a, b] \subset [0, 1]$, the above argument can be used to prove a better competitive ratio.

4.2 Competitive Ratio for PH3 as k -Copy Online Algorithm

PH3's property of achieving a better competitive ratio for r_L^* being further restricted can be used to create a set of $k \in \mathbb{N}$ algorithms achieving a better competitive ratio. For this purpose, the interval $[0, 1]$ is split into k sub-intervals $I_1, \dots, I_k \subset [0, 1]$ with $\cup_{i \in \{1, \dots, k\}} I_i = [0, 1]$. Each interval I_i is covered by one instance of the algorithm PH3 A_i , such that A_i achieves a targeted competitive ratio $R \in (3/2, 33/19)$ for $r_L^* \in I_i$.

R is restricted to $(3/2, 33/19)$, because any competitive ratio above or equal to $33/19$ can be achieved with the instance of PH3 shown above, and k -copy PH3 cannot achieve a competitive ratio of $3/2$ or less with finitely many algorithms.

To calculate the number k of algorithms needed to achieve a given competitive ratio R , the following iterative approach can be used.

Let \mathcal{A} be a set of algorithms. Initially, $\mathcal{A} := \emptyset$. We initialize our iterative approach with $i = 0$ and set $r_{max}^0 = 0$. Then, while $r_{max}^i < 1$, we increase i by one and we compute three values r_{min}^i , r_L^i and r_{max}^i . With these three values we can define algorithm A_i for which r_L^i denotes the value of r_L , r_{min}^i denotes the minimal and r_{max}^i denotes the maximal value for r_L^* for which A_i is still R -competitive. By Lemma 2, A_i will be R -competitive for the interval $[r_{min}^i, r_{max}^i]$. All three values are computed as follows. We set $r_{min}^i = r_{max}^{i-1}$. Given r_{min}^i , r_L^i can be computed:

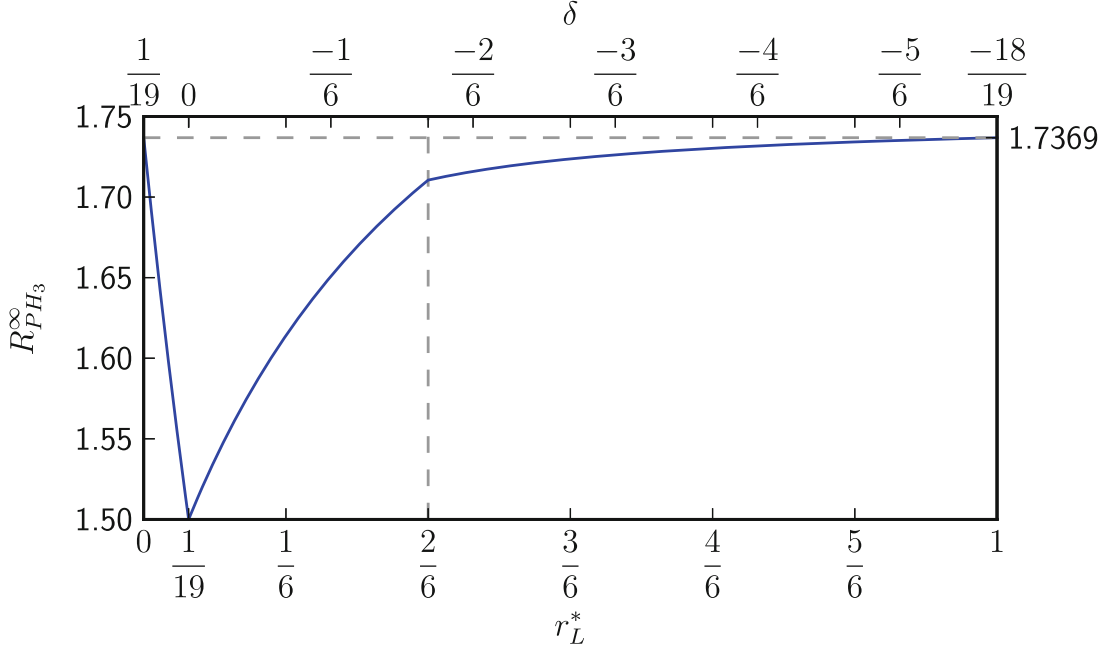


Fig. 1. Competitive ratio of the optimal 1-copy PH3 algorithm dependent on r_L^* for a fixed r_L .

If $r_{min}^i \leq 1/3$, we have $R = \frac{3}{2} + \frac{9}{2+6r_{min}^i}(r_L^i - r_{min}^i)$. Solving this equation for r_L^i we get $r_L^i = r_{min}^i + (R - \frac{3}{2}) \left(\frac{2+6r_{min}^i}{9} \right)$. If $r_{min}^i \geq 1/3$, we have $R = \frac{3}{2} + \frac{3}{4r_{min}^i}(r_L^i - r_{min}^i)$. Solving this equation for r_L^i yields $r_L^i = r_{min}^i + (R - \frac{3}{2}) \left(\frac{4r_{min}^i}{3} \right)$.

Having r_L^i , we can compute r_{max}^i . Because the competitive ratio is the minimum of two values, we get two candidates $r_{max,1}^i$ and $r_{max,2}^i$ for r_{max}^i . We can take the maximum of those two candidates, i.e., $r_{max}^i = \max(r_{max,1}^i, r_{max,2}^i)$, because it is sufficient to be R -competitive in one case. In the first case ($\frac{3}{6r_L^*+2} < \frac{1}{4r_L^*}$) we obtain $r_{max,1}^i = \frac{3r_L^i-3+2R}{12-6R}$ and in the second case we get $r_{max,2}^i = \frac{r_L^i}{7-4R}$.

Now consider the case when $r_{max}^i \geq 1$. Because each algorithm A_ℓ with $1 \leq \ell \leq i$ is R -competitive for the interval $[r_{min}^\ell, r_{max}^\ell] = [r_{max}^{\ell-1}, r_{max}^\ell]$ with $r_{min}^0 = 0$, there is an algorithm A_m for any $r_L^* \in [0, 1]$ that is R -competitive. Therefore, we have a i -copy online algorithm for bin packing achieving the competitive factor R .

Following this method, we see that $k = 6$ algorithms are sufficient to guarantee a competitive ratio $R = 1.5815$. This beats the currently best 1-copy online algorithm SON OF HARMONIC with a competitive ratio of 1.5816. Figure 2 shows the competitive ratio achieved by the individual algorithms over $r_L^* \in [0, 1]$ for $R = 1.5815$. Note that 1.5815 is not the best competitive ratio achievable by 6-copy PH3, as shown below in Fig. 3. Using $k = 12$ algorithms, a competitive

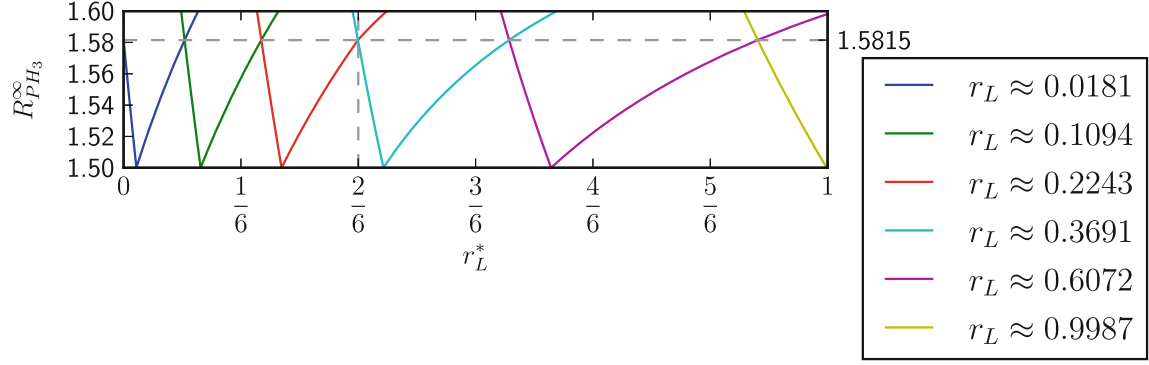


Fig. 2. 6-copy PH3 beats the best 1-copy online algorithm known to date, achieving an asymptotic competitive ratio $R_{PH3}^{\infty} < 1.5815$.

ratio $R = 1.5402 < 1.5403$ can be achieved, beating the highest known lower bound for 1-copy online algorithms.

To compute the best competitive ratio achievable by $k \in \mathbb{N}$ algorithms, we use binary search on R starting in the interval $[3/2, 33/19]$ and test in each iteration if we can guarantee R -competitiveness with at most k algorithms. Figure 3 shows the best competitive ratios achievable by k -copy PH3.

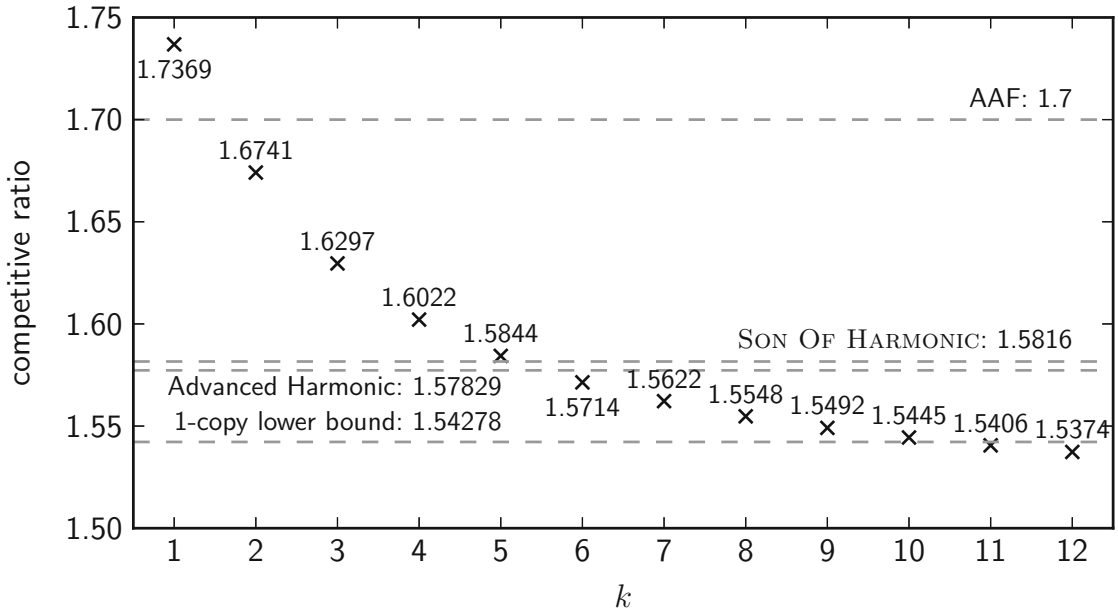


Fig. 3. k -copy PH3 performance dependent on k .

4.3 Comparison to Related Algorithms

Because k -copy online algorithms can be translated to an online algorithm with advice and vice versa (see Lemma 1), it seems natural to compare these two

variants, even though k -copy allows a more precise analysis on the competitive ratio. In this subsection we compare our algorithm to the best known online algorithm with constant advice, namely REDBLUE introduced by Angelopoulos et al. [2]. Their second algorithm is 1.47012-competitive (and thus beats our algorithm), but the amount of advice needed by this algorithm is too large. As the focus of k -copy algorithms is to provide good solutions for small k , it is reasonable to only compare k -copy PH3 to REDBLUE.

Table 1 shows a comparison between REDBLUE and k -copy PH3 for small amounts of advice. The competitive ratios given are rounded up to the fourth decimal place. The competitive ratios for REDBLUE are computed using the upper bound on the competitive ratio $1.5 + 15/(2^{\ell/2+1})$. The competitive ratios for k -copy PH3 are calculated using binary search as described above.

Table 1. Comparison of the performance of k -copy PH3 and REDBLUE.

| Advice in bits | k | $R_{\text{REDBLUE}}^{\infty}$ | R_{PH3}^{∞} |
|----------------|-------|-------------------------------|---------------------------|
| 4 | 16 | 3.3750 | 1.5305 |
| 5 | 32 | 2.8258 | 1.5155 |
| 6 | 64 | 2.4375 | 1.5078 |
| 7 | 128 | 2.1629 | 1.5040 |
| 8 | 256 | 1.9688 | 1.5020 |
| 9 | 512 | 1.8315 | 1.5010 |
| 10 | 1024 | 1.7344 | 1.5005 |
| 11 | 2048 | 1.6657 | 1.5003 |
| 12 | 4096 | 1.6172 | 1.5002 |
| 13 | 8192 | 1.5829 | 1.5001 |
| 14 | 16384 | 1.5586 | 1.5001 |
| 15 | 32768 | 1.5414 | 1.5001 |
| 16 | 65536 | 1.5293 | 1.5001 |

Table 1 clearly shows the advantage of k -copy PH3 over REDBLUE for few bits of advice. With as few as 5 bits of advice, or $k = 32$, k -copy PH3 achieves a better competitive ratio than REDBLUE with 16 bits of advice, which corresponds to $k = 65536$ algorithms when used as k -copy algorithm.

Although REDBLUE and k -copy PH3 work in a similar way, k -copy PH3 achieves a better competitive ratio due to the more precise analysis of the intervals for r_L^* , in which each algorithm achieves the competitive ratio. By avoiding overlaps in these intervals, fewer algorithms are needed.

On the other hand, REDBLUE simply splits an interval for its parameter β evenly into $2^{\ell/2}$ intervals; translated into a k -copy setting, this leads to overlaps in the intervals covered by each algorithm.

5 Conclusion

We studied the concept of parallel online algorithms for the Bin Packing Problem. We developed a k -copy online algorithm named PH3 and showed that PH3 has an asymptotic competitive ratio of 1.5 for large k ; in particular, $k = 11$ suffices to break through the lower bound of a single online algorithm. We also considered the relationship to online algorithms with advice and achieved a considerable improvement compared to a previous algorithm.

There are various directions for future work. We saw that PH3 is $(1.5 + \varepsilon)$ -competitive if $\frac{|I_L|}{6 \text{size}(I_S)} \leq 1$, i.e., when there is a surplus of small items. If there are too few small items, PH3 is asymptotically $(1.5 + \varepsilon)$ -competitive. Can we make better use of the second case for an improvement? Can we guarantee an absolute competitive ratio of $1.5(+\varepsilon)$?

How does the asymptotic competitive ratio of PH3 depend on k ? It seems to be something like $\frac{3}{2} + O\left(\frac{1}{k + \log_2(k+1)}\right)$. Translated to an online algorithm with ℓ bits of advice, this would yield an asymptotic competitive ratio of $\frac{3}{2} + O\left(\frac{1}{2^\ell + \ell}\right)$.

We also believe that the concept of k -copy algorithms is useful for a wide range of other problems.

References

1. Albers, S., Hellwig, M.: Online makespan minimization with parallel schedules. *Algorithmica* **78**(2), 492–520 (2017)
2. Angelopoulos, S., Dürr, C., Kamali, S., Renault, M., Rosén, A.: Online bin packing with advice of small size. In: Dehne, F., Sack, J.-R., Stege, U. (eds.) WADS 2015. LNCS, vol. 9214, pp. 40–53. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21840-3_4
3. Balogh, J., Békési, J., Dósa, G., Epstein, L., Levin, A.: A new and improved algorithm for online bin packing. In: 26th Annual European Symposium on Algorithms (ESA 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
4. Balogh, J., Békési, J., Dósa, G., Epstein, L., Levin, A.: A new lower bound for classic online bin packing. arXiv preprint [arXiv:1807.05554](https://arxiv.org/abs/1807.05554) (2018). (To appear at 17th Workshop on Approximation and Online Algorithms (WAOA))
5. Boyar, J., Favrholt, L.M., Kudahl, C., Larsen, K.S., Mikkelsen, J.W.: Online algorithms with advice: a survey. *ACM Comput. Surv. (CSUR)* **50**(2), 19 (2017)
6. Boyar, J., Kamali, S., Larsen, K.S., López-Ortiz, A.: On the list update problem with advice. In: 8th Conference on Language and Automata Theory and Applications (LATA), pp. 210–221 (2014)
7. Boyar, J., Kamali, S., Larsen, K.S., López-Ortiz, A.: Online bin packing with advice. *Algorithmica* **74**(1), 507–527 (2016)
8. Brown, D.M.: A lower bound for on-line one-dimensional bin packing algorithms. Technical report (1979)
9. Csirik, J., Woeginger, G.J.: On-line packing and covering problems. In: Fiat, A., Woeginger, G.J. (eds.) *Online Algorithms: The State of the Art*. LNCS, vol. 1442, pp. 147–177. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0029568>
10. Fekete, S.P., Grosse-Holz, J., Keldenich, P., Schmidt, A.: Parallel online algorithms for the bin packing problem (2019). arXiv preprint (1910.03249)

11. Halldórsson, M.M., Iwama, K., Miyazaki, S., Taketomi, S.: Online independent sets. *Theor. Comput. Sci.* **289**(2), 953–962 (2002)
12. Heydrich, S., van Stee, R.: Beating the harmonic lower bound for online bin packing. In: *The 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pp. 41:1–41:14 (2016)
13. Johnson, D.S.: Fast algorithms for bin packing. *J. Comput. Syst. Sci.* **8**(3), 272–314 (1974)
14. Kamali, S., Ortiz, A.L.: Better compression through better list update algorithms. In: *2014 Data Compression Conference*, pp. 372–381 (2014)
15. Lee, C.C., Lee, D.T.: A simple on-line bin-packing algorithm. *J. ACM* **32**, 562–572 (1985)
16. Liang, F.M.: A lower bound for on-line bin packing. *Inf. Process. Lett.* **10**(2), 76–79 (1980)
17. López-Ortiz, A., Schuierer, S.: On-line parallel heuristics, processor scheduling and robot searching under the competitive framework. *Theor. Comput. Sci.* **310**(1–3), 527–537 (2004)
18. Renault, M.P., Rosén, A., van Stee, R.: Online algorithms with advice for bin packing and scheduling problems. *Theor. Comput. Sci.* **600**, 155–170 (2015)
19. van Vliet, A.: An improved lower bound for on-line bin packing algorithms. *Inf. Proc. Lett.* **43**(5), 277–284 (1992)
20. Yao, A.C.-C.: New algorithms for bin packing. *J. ACM* **27**(2), 207–227 (1980)
21. Zhao, X., Shen, H.: On the advice complexity of one-dimensional online bin packing. In: Chen, J., Hopcroft, J.E., Wang, J. (eds.) *FAW 2014. LNCS*, vol. 8497, pp. 320–329. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08016-1_29

Parallel Online Algorithms for the Bin Packing Problem

(Preprint — Submitted for journal publication to Transaction on Algorithms ACM)

SÁNDOR P. FEKETE, JONAS GROSSE-HOLZ, PHILLIP KELDENICH, and ARNE SCHMIDT,
Department of Computer Science, TU Braunschweig, Germany

We study *parallel* online algorithms: For some fixed integer k , a collective of k parallel processes that perform online decisions on the same sequence of events forms a k -copy algorithm. For any given time and input sequence, the overall performance is determined by the best of the k individual total results. Problems of this type have been considered for online makespan minimization; they are also related to optimization with *advice* on future events, i.e., a number of bits available in advance. Parallel online algorithms are also of interest in practical scenarios in whichin which redundancy is used for hedging against undesired outcomes.

We develop PREDICTIVE HARMONIC₃ (PH3), a relatively simple family of k -copy algorithms for the online Bin Packing Problem, whose joint competitive factor converges to 1.5 for increasing k . In particular, we show that $k = 6$ suffices to guarantee a factor of 1.5714 for PH3, which is better than 1.57829, the performance of the best known 1-copy algorithm ADVANCED HARMONIC, while $k = 11$ suffices to achieve a factor of 1.5406, beating the known lower bound of 1.54278 for a single online algorithm. In the context of online optimization with advice, our approach implies that 4 bits suffice to achieve a factor better than this bound of 1.54278, which is considerably less than the previous bound of 15 bits.

CCS Concepts: • **Theory of computation** → **Design and analysis of algorithms**; *Online algorithms*.

Additional Key Words and Phrases: Online algorithms, bin packing, redundancy, competitive analysis, k -copy online algorithms.

ACM Reference Format:

Sándor P. Fekete, Jonas Grosse-Holz, Phillip Keldenich, and Arne Schmidt. 2020. Parallel Online Algorithms for the Bin Packing Problem (Preprint — Submitted for journal publication to Transaction on Algorithms ACM). *ACM Trans. Algor.* 99, 99, Article 99 (December 2020), 23 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

1 INTRODUCTION

When dealing with unknown future events, optimization with incomplete information typically considers the competitive factor of an online algorithm as its performance measure; the objective becomes to develop a single strategy that performs reasonably well against the worst case. This focus on just *one* option is more restrictive than hedging strategies in a wide variety of other scientific and application fields; these typically make use of *several* parallel choices, thereby increasing the chance that one of them will yield satisfactory results. Examples include scenarios from biology, where a large and diverse progeny increases the odds of surviving offspring; finance and insurance, where a suitable combination of investment strategies is employed to balance a portfolio against extreme losses; and engineering, where redundancy is used to protect against catastrophic failure, either on individual components (such as parts in a machine) or on whole systems (such as automata in

Authors' address: Sándor P. Fekete, s.fekete@tu-bs.de; Jonas Grosse-Holz, j.grosse-holz@tu-bs.de; Phillip Keldenich, p.keldenich@tu-bs.de; Arne Schmidt, arne.schmidt@tu-bs.de, Department of Computer Science, TU Braunschweig, Braunschweig, Germany.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2020 Association for Computing Machinery.

1549-6325/2020/12-ART99 \$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

a robot swarm or spacecraft in a group of satellites), for which it suffices that just one machine delivers a good outcome.

In this paper, we consider such *parallel* online strategies: Instead of making a single sequence of decisions, we consider k parallel processes for some fixed integer k , which we call a *k-copy algorithm*; the objective is to make the best of these k outcomes as good as possible, even in the worst case. We demonstrate the potential of this approach for the well-studied Bin Packing Problem, for which it is known that no single deterministic online algorithm can achieve a competitive factor below 1.5401.

1.1 Our Results

We define a family of k -copy algorithms for the online Bin Packing Problem, called PREDICTIVE HARMONIC₃ (PH3), whose asymptotic competitive ratio converges to 1.5 for large k . We show that $k = 6$ suffices to guarantee a factor of 1.5714, which is better than 1.57829, the performance of the best known 1-copy algorithm ADVANCED HARMONIC [3]. Moreover, $k = 11$ suffices to achieve a competitive ratio of 1.5406 beating the known lower bound of 1.54278 for a 1-copy algorithm [4]. In the context of online optimization with advice, our approach implies that 4 bits suffice to achieve less than 1.5401, which is considerably less than the previous bound of 16 bits of REDBLUE by Angelopoulos et al. [2]; in fact, for $k = 16$ (corresponding to four bits of advice) PH3 achieves a ratio of 1.5305, compared to 3.3750 for REDBLUE, while $k = 65,536$ (corresponding to 16 bits of advice) yields a factor of 1.5001 for PH3, but 1.5293 for REDBLUE. As a last step, give an analytic proof that the competitive ratio of our algorithm can be bounded from above by $\frac{3}{2} + \frac{3 \cdot 4^{1/k} - 3}{6 \cdot 4^{1/k} + 2}$, which converges to $\frac{3}{2}$ for large k .

1.2 Related Work on Online Bin Packing

There is a wide range of online algorithms for bin packing. The Next Fit algorithm [9] achieves a competitive ratio of 2, whereas “Almost Any Fit” algorithms [12] like First Fit or Best Fit achieve competitive ratios of 1.7.

An important online bin packing algorithm is HARMONIC_M, introduced by Lee and Lee [15], which achieves a competitive ratio of less than 1.692 for $M \rightarrow \infty$. Based on HARMONIC_M, SON OF HARMONIC by Heydrich and van Stee [11] achieves a competitive ratio of 1.5816. The currently best known algorithm is ADVANCED HARMONIC, which achieves a competitive ratio of 1.57829 [3].

For lower bounds, Yao [20] established a value of $3/2$ that was later improved to 1.536, independently by Brown [8] and by Liang [16]. Using a generalization of their methods, van Vliet [19] proved a lower bound of 1.5401. Balogh et al. [4] improved the lower bound to 1.54278.

1.3 Related Work on Online Bin Packing with Advice

In the context of online algorithms with advice, Boyar et al. [7] showed that an online algorithm with $n \lceil \log(OPT(I)) \rceil$ bits of advice is sufficient and that at least $(n - 2OPT(I)) \cdot \log(OPT(I))$ bits of advice are necessary to achieve optimality. In the same paper, they presented an online bin packing algorithm, namely RESERVECRITICAL, with $O(\log(n)) + o(\log(n))$ bits of advice that is 1.5-competitive and an algorithm with $2n + o(n)$ bits of advice that is $\frac{4}{3}$ -competitive. Zhao and Shen [21] developed an algorithm using $3n + o(n)$ bits of advice achieving a competitive ratio of $\frac{5}{4}OPT + 2$. Renault et al. [18] developed an $(1 + \epsilon)$ -competitive algorithm using $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ bits of advice per request.

Based on RESERVECRITICAL, Angelopoulos et al. [2] developed the algorithm REDBLUE with constant advice that is 1.5-competitive. Their second algorithm achieves a competitive ratio of

$1.47012 + \varepsilon$ with finite advice that is exponentially dependent of ε . However, to beat the competitive ratio of 1.5 already an enormous amount of advice is needed, which makes the algorithm impractical.

In terms of lower bounds, Boyar et al. [7] proved that no competitive ratio better than $9/8$ can be reached by any algorithm that uses sub-linear advice. Angelopoulos et al. [2] improved this bound to $7/6$.

1.4 Related Work on Parallel Online Algorithms

Parallel algorithms have already been considered in the field of online algorithms with advice. Boyar et al. [6] presented an algorithm for the online list update problem, making use of 2 bits of advice to choose one out of three algorithms. This algorithm achieves a competitive ratio of $5/3$, beating the lower bound for conventional online algorithms of 2. A practical application of this algorithm was shown by Kamali and Ortiz [14], who applied it in the Burrows-Wheeler transform compression. More work on parallel online algorithms include parallel scheduling [1], finding independent sets [10] and the “multiple-cow” version of the linear search problem [17].

While online algorithms with advice mostly focus on the amount of advice to allow classification of online algorithms and problems, k -copy online algorithms focus on small finite values for k and thus small finite amounts of advice, with more emphasis on practical application. The perspective on different algorithms running in parallel instead of abstract arbitrary information facilitates finer optimization in some cases.

Also, when considering online algorithms with advice, the number of algorithm can only be doubled by increasing the amount of advice by one bit. The perspective of k -copy algorithms allows arbitrary $k \in \mathbb{N}$ for the number of algorithms.

2 PRELIMINARIES

2.1 k -Copy Online Algorithms

In this paper, we consider k online algorithms A_1, \dots, A_k , each of them processing the same input list I in parallel. We call the set $\mathcal{A} := \{A_1, \dots, A_k\}$ a k -copy online algorithm.

For an input list I and an online algorithm A , let $A(I)$ denote the number of bins used by A and $\text{OPT}(I)$ denote the number of bins used in an optimal offline solution. The absolute competitive ratio $R_{\mathcal{A}}$ for a k -copy online algorithm \mathcal{A} is defined as

$$R_{\mathcal{A}} = \sup_I \left\{ \frac{\min_{A \in \mathcal{A}} A(I)}{\text{OPT}(I)} \right\}.$$

The asymptotic competitive ratio $R_{\mathcal{A}}^{\infty}$ for algorithm \mathcal{A} is defined as

$$R_{\mathcal{A}}^{\infty} = \lim_{n \rightarrow \infty} \sup_I \left\{ \frac{\min_{A \in \mathcal{A}} A(I)}{\text{OPT}(I)} \mid \text{OPT}(I) = n \right\}$$

As already stated by Boyar et al. [5], any k -copy online algorithm can be converted into an online algorithm with advice, and vice versa.

LEMMA 1. *Any k -copy online algorithm can be converted into an online algorithm with $l = \lceil \log_2(k) \rceil$ bits of advice that achieves the same competitive ratio. Conversely, any online algorithm with $l \in \mathbb{N}$ bits of advice can be converted into a k -copy online algorithm without advice with $k = 2^l$ that achieves the same competitive ratio.*

PROOF. Let $\mathcal{A} = \{A_1, A_2, \dots, A_k\}$ be a k -copy algorithm. Construct the online algorithm A' that gets a value $i \in \{1, 2, \dots, k\}$ as advice, specifying the index i of the algorithm $A_i \in \mathcal{A}$ that performs best on the given input sequence. The value i can be encoded using $\lceil \log_2(k) \rceil$ bits. A' then behaves like A_i and thus achieves the same competitive ratio as \mathcal{A} .

Let A be an online algorithm that gets $l \in \mathbb{N}$ bits of advice. Construct the online k -copy algorithm \mathcal{A}' with $k = 2^l$ algorithms $A_i, i \in \{1, 2, \dots, k\}$. For each $i \in \{1, 2, \dots, k\}$, the algorithm A_i behaves like A given i encoded in binary as advice. As the values $i \in \{1, 2, \dots, k\}$ cover every possible configuration of the advice bits, for any advice given to A , there is an algorithm $A_i \in \mathcal{A}'$, that assumes this advice. Accordingly, there is an algorithm $A_i \in \mathcal{A}'$, that performs as well as A , i.e., the best algorithm $A_j \in \mathcal{A}$ that performs at least as well as A . Thus, \mathcal{A}' performs at least as well as A . \square

2.2 Bin Packing

In the online version of bin packing, we are given a list of items $I := \langle a_1, \dots, a_n \rangle$ with $a_i \in (0, 1]$ for $i \in \{1, \dots, n\}$. These items must be packed by an algorithm, one at a time, without any information on subsequent items and without the possibility to change previous decisions. The goal is to pack all items into a minimum number of bins with unit capacity.

DEFINITION 2 (ITEM SIZE). Let $S = [0, \frac{1}{3}]$, $M = (\frac{1}{3}, \frac{1}{2}]$, $L = (\frac{1}{2}, \frac{2}{3}]$ and $XL = [\frac{2}{3}, 1]$. We call items in S small, items in M medium, items in L large and items in XL extra large. For a list $I = \langle a_1, a_2, \dots, a_n \rangle$, the set of items $\text{Set}(I) \cap XL$ is noted as I_{XL} for improved readability. The subsets I_L, I_M and I_S are used analogously.

DEFINITION 3 (SIZE FUNCTION). Let S be a set (or list) of items. Then, $\text{size}(S) := \sum_{i \in S} i$. For a bin b , we refer to $\text{size}(b)$ as the size of the bin, i.e., the sum of items already packed in b .

DEFINITION 4 (SUB-BINS). Given a bin b , it can be split into two parts b_1 and b_2 , such that the sum of their capacities is equal to the capacity of b . We refer to b_1 and b_2 as sub-bins. We call a sub-bin with capacity C a C -sub-bin.

As sub-bins are not packed with an amount larger than their capacity, each sub-bin can be packed independently from the other.

3 PREDICTIVE HARMONIC₃

Now we introduce the algorithm PREDICTIVE HARMONIC₃ (PH3). Although developed independently, it bears many similarities to RESERVECRITICAL and REDBLUE. PH3 uses the same classifications as the other two algorithms and tries to pack all large items with small items, such that the corresponding bins are packed to a level of at least $2/3$. However, in contrast to REDBLUE, the information needed by PH3 does not depend on the result of RESERVECRITICAL, but only on the number and size of certain item types, and can be calculated in linear time.

The main idea of PH3 is to guess the ratio of how many small items must be packed with large items to obtain a packing density of $2/3$. Having multiple instances of PH3, every instance can guess a different ratio to get close to a competitive ratio of 1.5.

ALGORITHM 5. PREDICTIVE HARMONIC₃

Given a list $I = \langle a_1, a_2, \dots, a_n \rangle$ of items $a_i \in (0, 1], i \in 1, \dots, n$, and a ratio $r_L \in [0, 1]$, the algorithm packs the items as follows:

- Extra large items are packed into individual bins. These bins are called XL-bins, the set of all XL-bins is called B_{XL} .
- Large items are packed into individual bins. These bins are called L-bins, the set of all L-bins is called B_L . Furthermore, we split each L-bin into a $\frac{2}{3}$ -sub-bin (for large items) and a $\frac{1}{3}$ -sub-bin (for small items).
- Medium items are packed into separate bins together with other medium items (note that at most two of them fit into one bin). These bins are called M-bins, the set of all M-bins is called B_M .

- Small items are packed into a $\frac{1}{3}$ -sub-bin of L-bins in a next fit manner, if the size of small items packed into L-bins is smaller than r_L times the total size of small items packed so far; otherwise we pack the small item into S-bins.

3.1 Competitive Ratio

Before we proceed to prove the asymptotic competitive ratio, we recall some lower bounds for an optimal offline solution.

LEMMA 6. (Lower bounds on OPT) For any input sequence I in online bin packing, the following lower bounds on OPT hold:

- (1) $OPT \geq |I_{XL}| + |I_L| \geq |I_L|$
- (2) $OPT \geq |I_{XL}| + \frac{|I_M| + |I_L|}{2}$
- (3) $OPT \geq \text{size}(I)$

PROOF.

- (1) As any two items $a, a' \in XL \cup L$ with $a \neq a'$ fulfill $a + a' > 1$, OPT has to put each of these items in a different bin. Thus, $OPT \geq |I_{XL}| + |I_L|$.
- (2) First, note that no item $a \in M \cup L$ can be packed with an item $a' \in XL$. Also, no more than two items $a, a' \in M \cup L$ can be packed in the same bin, as each of these items is greater than $1/3$. Thus,

$$OPT \geq |I_{XL}| + \frac{|I_M| + |I_L|}{2}$$

- (3) As OPT has to pack the items into unit-sized bins, the number of bins must be at least the items total size $\sum_{a \in I} a = \text{size}(I)$. \square

Using this bounds and performing a case analysis, we can prove the following theorem.

THEOREM 7. Let $r_L^* = \min \left\{ \frac{|I_L|}{6 \text{ size}(I_S)}, 1 \right\}^1$ and $\delta = r_L - r_L^*$. PH3 achieves the asymptotic competitive ratio

$$R_{PH3}^\infty \leq \begin{cases} \frac{3}{2} + \min \left\{ \frac{1}{4r_L^*}, \frac{3}{6r_L^* + 2} \right\} (-\delta) & \text{for } \delta \leq 0 \\ \frac{3}{2} + \min \left\{ \frac{3}{4r_L^*}, \frac{9}{6r_L^* + 2} \right\} \delta & \text{for } \delta \geq 0. \end{cases}$$

PROOF. First, consider $r_L^* = \frac{|I_L|}{6 \text{ size}(I_S)}$.

$$PH_3 = |B_{XL}| + |B_M| + |B_S| + |B_L|$$

As each extra large item $a \in I_{XL}$ is packed into a separate XL-bin and each such item satisfies $a \geq 2/3$, we get

$$|B_{XL}| = |I_{XL}| \leq \frac{3}{2} \text{ size}(I_{XL}).$$

Two medium items are packed into each M-bin, except for the last one if the number of M-items is odd. Each M-item $a \in I_M$ fulfills $a > 1/3$, thus

$$|B_M| = \left\lceil \frac{|I_M|}{2} \right\rceil \leq \frac{|I_M| + 1}{2} < \frac{3}{2} \text{ size}(I_M) + \frac{1}{2}.$$

¹The intuition of this value is that at least $1/2$ of each $1/3$ -sub-bin must be filled to guarantee a packing density of $2/3$. Therefore, for $|I_L|$ bins, we have to fill up a total capacity of $\frac{|I_L|}{6}$ with small items.

PH3 will only open a new S-bin, if the current item $a \in I_S$ is supposed to be packed into an S-bin and does not fit into the currently open bin. As $\forall a \in I_S : i \leq 1/3$, any S-bin $b \in B_S$ except the last one has to fulfill $\text{size}(b) > 2/3$, otherwise another item $a \in I_S$ would fit. A fraction of $(1 - r_L)$ of the size of small items plus at most one item is packed into S-bins. If a bin $b \in B_S$ with $\text{size}(b) < 2/3$ exists, this additional item can be packed there, otherwise it is packed into an additional bin. In either case, there is at most one bin $b \in B_S$ with $\text{size}(b) < 2/3$.

$$\begin{aligned}
 |B_S| &\leq \frac{3}{2} (1 - r_L) \text{size}(I_S) + 1 \\
 &= \frac{3}{2} (1 - \delta) \text{size}(I_S) - \frac{3}{2} r_L^* \text{size}(I_S) + 1 \\
 &= \frac{3}{2} (1 - \delta) \text{size}(I_S) - \frac{3}{2} \frac{|I_L|}{6 \text{size}(I_S)} \text{size}(I_S) + 1 \\
 &= \frac{3}{2} (1 - \delta) \text{size}(I_S) - \frac{1}{4} |I_L| + 1
 \end{aligned}$$

For B_L , consider the subsets $B_{LL} = \{b \in B_L \mid b \text{ contains a large item}\}$ and $B_{LS} = \{b \in B_L \mid b \text{ contains a small item}\}$. As both small and large items are packed into L-bins by increasing index, either $B_{LS} \subseteq B_{LL} = B_L$ or $B_{LL} \subseteq B_{LS} = B_L$ holds. Thus, $|B_L| = \max\{|B_{LL}|, |B_{LS}|\}$.

No two large items are packed into the same bin, so $|B_{LL}| = |I_L|$.

Let I'_S the set of small items packed into L-bins. As small items are smaller than $1/3$ and are only packed into L-bins if less than a part r_L of the small items packed so far have been packed into L-bins, $\text{size}(I'_S) < r_L \text{size}(I_S) + 1/3$. The set I'_S is packed into the $1/3$ -sub-bins by a NF-algorithm. For an optimal packing OPT' of small items in these sub-bins, $NF \leq 2 OPT' + 1$. Each sub-bin has capacity $1/3$, so OPT needs at least $3 \text{size}(I'_S)$ bins to pack all items.

$$\begin{aligned}
 |B_{LS}| &\leq 2 OPT' + 1 \\
 &\leq 6 \text{size}(I'_S) + 1 \\
 &< 6 r_L \text{size}(I_S) + 3 \\
 &= 6 (r_L^* + \delta) \text{size}(I_S) + 3 \\
 &= 6 \left(\frac{|I_L|}{6 \text{size}(I_S)} + \delta \right) \text{size}(I_S) + 3 \\
 &= |I_L| + 6 \delta \text{size}(I_S) + 3
 \end{aligned}$$

The maximum of the bounds on $|B_{LL}|$ and $|B_{LS}|$ is an upper bound on $|B_L|$.

$$\begin{aligned}
 |B_L| &\leq \max\{|I_L|, |I_L| + 6 \delta \text{size}(I_S) + 3\} \\
 &= |I_L| + \max\{0, 6 \delta \text{size}(I_S) + 3\}
 \end{aligned}$$

Combining the bounds for $|B_S|$ and $|B_L|$ yields

$$\begin{aligned}
 |B_S| + |B_L| &\leq \underbrace{\frac{3}{2} (1 - \delta) \text{size}(I_S) - \frac{1}{4} |I_L| + 1}_{\geq |B_S|} + \underbrace{|I_L| + \max\{0, 6 \delta \text{size}(I_S) + 3\}}_{\geq |B_L|} \\
 &= \frac{3}{2} (1 - \delta) \text{size}(I_S) + \frac{3}{4} |I_L| + 1 + \max\{0, 6 \delta \text{size}(I_S) + 3\}
 \end{aligned}$$

Each large item $a \in I_L$ is greater than $1/2$. Therefore $|I_L| < 2 \cdot \text{size}(I_L)$.

$$|B_S| + |B_L| < \frac{3}{2}(\text{size}(I_S) + \text{size}(I_L)) - \frac{3}{2}\delta \text{size}(I_S) + 1 + \max\{0, 6\delta \text{size}(I_S) + 3\}$$

Let

$$\begin{aligned} \Delta &= \frac{3}{2} + \max\{0, 6\delta \text{size}(I_S) + 3\} - \frac{3}{2}\delta \text{size}(I_S) \\ &= \max\left\{\frac{3}{2} - \frac{3}{2}\delta \text{size}(I_S), \frac{9}{2} + \frac{9}{2}\delta \text{size}(I_S)\right\} \\ \Rightarrow |B_S| + |B_L| &< \frac{3}{2}(\text{size}(I_S) + \text{size}(I_L)) + \Delta - \frac{1}{2} \end{aligned}$$

Summing up,

$$\begin{aligned} PH_3 &= |B_{XL}| + |B_M| + |B_S| + |B_L| \\ &< \underbrace{\frac{3}{2} \text{size}(I_{XL})}_{\geq |B_{XL}|} + \underbrace{\frac{3}{2} \text{size}(I_M)}_{> |B_M|} + \frac{1}{2} + \underbrace{\frac{3}{2}(\text{size}(I_S) + \text{size}(I_L)) + \Delta - \frac{1}{2}}_{\geq |B_S| + |B_L|} \\ &= \frac{3}{2}(\text{size}(I_{XL}) + \text{size}(I_M) + \text{size}(I_S) + \text{size}(I_L)) + \Delta \\ &= \frac{3}{2} \text{size}(I) + \Delta. \end{aligned}$$

Because of Lemma 6, $OPT \geq \text{size}(I)$.

$$\Rightarrow PH_3 < \frac{3}{2}OPT + \Delta$$

This yields the asymptotic competitive ratio:

$$\begin{aligned} \lim_{OPT \rightarrow \infty} \frac{PH_3}{OPT} &\leq \lim_{OPT \rightarrow \infty} \frac{3}{2} + \frac{\Delta}{OPT} \\ &= \lim_{OPT \rightarrow \infty} \frac{3}{2} + \frac{\max\left\{\frac{3}{2} - \frac{3}{2}\delta \text{size}(I_S), \frac{9}{2} + \frac{9}{2}\delta \text{size}(I_S)\right\}}{OPT} \\ &= \lim_{OPT \rightarrow \infty} \frac{3}{2} + \max\left\{-\frac{3}{2}\delta, \frac{9}{2}\delta\right\} \frac{\text{size}(I_S)}{OPT} \end{aligned}$$

Clearly, OPT is an upper bound on $\text{size}(I_S)$. However, better bounds can be found using r_L^* .

$$r_L^* = \frac{|I_L|}{6 \text{size}(I_S)} \quad \Leftrightarrow \quad \text{size}(I_S) = \frac{|I_L|}{6r_L^*}$$

Because of Lemma 6, $|I_L|$ is a lower bound on OPT .

$$\Rightarrow \text{size}(I_S) = \frac{|I_L|}{6r_L^*} \leq \frac{OPT}{6r_L^*}$$

As each item $i \in I_L$ is larger than $1/2$, $|I_L| < 2 \text{size}(I_L)$. According to Lemma 6, $OPT \geq \text{size}(I)$, i.e. $OPT \geq \text{size}(I_S) + \text{size}(I_L)$.

$$\begin{aligned} \Rightarrow \text{size}(I_S) &= \frac{|I_L|}{6r_L^*} \leq \frac{\text{size}(I_L)}{3r_L^*} \\ \Leftrightarrow \left(1 + \frac{1}{3r_L^*}\right) \text{size}(I_S) &\leq \frac{\text{size}(I_L) + \text{size}(I_S)}{3r_L^*} \\ \Leftrightarrow \text{size}(I_S) &\leq \frac{\text{size}(I_L) + \text{size}(I_S)}{3r_L^* + 1} \\ &\leq \frac{OPT}{3r_L^* + 1} \end{aligned}$$

As these are both upper bounds on $\text{size}(I_S)$, their minimum is also an upper bound on $\text{size}(I_S)$. This results in an asymptotic upper bound on Δ in comparison to OPT :

$$\begin{aligned} \lim_{OPT \rightarrow \infty} \frac{\Delta}{OPT} &= \max \left\{ -\frac{3}{2}\delta, \frac{9}{2}\delta \right\} \min \left\{ \frac{1}{6r_L^*}, \frac{1}{3r_L^* + 1} \right\} \\ \Leftrightarrow \lim_{OPT \rightarrow \infty} \frac{\Delta}{OPT} &= \begin{cases} \min \left\{ \frac{1}{4r_L^*}, \frac{3}{6r_L^* + 2} \right\} (-\delta) & \text{for } \delta \leq 0 \\ \min \left\{ \frac{3}{4r_L^*}, \frac{9}{6r_L^* + 2} \right\} \delta & \text{for } \delta \geq 0 \end{cases} \end{aligned}$$

This results in a competitive ratio dependent on δ , proving Theorem 7 for $r_L^* = \frac{|I_L|}{6 \text{size}(I_S)}$:

$$\begin{aligned} R_{PH3}^\infty &= \lim_{OPT \rightarrow \infty} \frac{PH_3}{OPT} \\ &\leq \lim_{OPT \rightarrow \infty} \frac{3}{2} + \frac{\Delta}{OPT} \\ &= \begin{cases} \frac{3}{2} + \min \left\{ \frac{1}{4r_L^*}, \frac{3}{6r_L^* + 2} \right\} (-\delta) & \text{for } \delta \leq 0 \\ \frac{3}{2} + \min \left\{ \frac{3}{4r_L^*}, \frac{9}{6r_L^* + 2} \right\} \delta & \text{for } \delta \geq 0. \end{cases} \end{aligned}$$

Now consider $r_L^* = 1$.

As $\delta = r_L - r_L^* = r_L - 1$ and $r_L \in [0, 1]$, δ must be less or equal to 0. Furthermore, $\text{size}(I_S)$ must be less or equal to $|I_L|/6$, as otherwise r_L^* would be less than 1.

$$PH_3 = |B_{XL}| + |B_L| + |B_M| + |B_S|$$

As above, each large item $a \in I_{XL}$ is packed into a separate bin and medium items are packed by twos.

$$\begin{aligned} |B_{XL}| &= |I_{XL}| \\ |B_M| &= \left\lceil \frac{|I_M|}{2} \right\rceil \leq \frac{|I_M| + 1}{2} \end{aligned}$$

As pointed out above, $|B_L| \leq |I_L| + \max\{0, 6\delta \text{size}(I_S) + 3\}$. As $\delta \leq 0$, $|B_L| \leq |I_L| + 3$.

The argumentat used above to get a bound on $|B_S|$ applies here as well.

$$\begin{aligned}
 |B_S| &\leq \frac{3}{2}(1 - r_L) \text{size}(I_S) + 1 \\
 &= \frac{3}{2}(-\delta) \text{size}(I_S) + 1 \\
 &\leq \frac{3}{2}(-\delta) \frac{|I_L|}{6} + 1 \\
 &= \frac{|I_L|}{4}(-\delta) + 1 \\
 &\leq \frac{OPT}{4}(-\delta) + 1
 \end{aligned}$$

In summary, this yields an upper bound on PH_3 .

$$\begin{aligned}
 PH_3 &\leq \underbrace{|I_{XL}|}_{=|B_{XL}|} + \underbrace{\frac{|I_M| + 1}{2}}_{\geq |B_M|} + \underbrace{|I_L| + 3}_{\geq |B_L|} + \underbrace{\frac{OPT}{4}(-\delta) + 1}_{\geq |B_S|} \\
 &= |I_{XL}| + |I_L| + \frac{|I_M|}{2} + \frac{OPT}{4}(-\delta) + \frac{9}{2}
 \end{aligned}$$

For $|I_L| \geq |I_M|$ an estimate against OPT can be formulated as follows using Lemma 6.

$$\begin{aligned}
 PH_3 &\leq \underbrace{|I_{XL}| + |I_L|}_{\leq OPT} + \underbrace{\frac{|I_L|}{2}}_{\leq OPT/2} + \frac{OPT}{4}(-\delta) + \frac{9}{2} \\
 &\leq \left(\frac{3}{2} + \frac{1}{4}(-\delta) \right) OPT + \frac{9}{2}
 \end{aligned}$$

For $|I_L| < |I_M|$, PH_3 can be estimated against OPT using another bound from Lemma 6.

$$\begin{aligned}
 PH_3 &\leq |I_{XL}| + \frac{4|I_L| + 2|I_M|}{4} + \frac{OPT}{4}(-\delta) + \frac{9}{2} \\
 &< \underbrace{|I_{XL}| + \frac{3|I_L| + 3|I_M|}{4}}_{\leq 3/2 OPT} + \frac{OPT}{4}(-\delta) + \frac{9}{2} \\
 &\leq \left(\frac{3}{2} + \frac{1}{4}(-\delta) \right) OPT + \frac{9}{2}
 \end{aligned}$$

This bound leads straight to the competitive ratio.

$$\begin{aligned}
 \Rightarrow \lim_{OPT \rightarrow \infty} \frac{PH_3}{OPT} &\leq \lim_{OPT \rightarrow \infty} \left(\frac{3}{2} + \frac{1}{4}(-\delta) \right) \frac{OPT}{OPT} + \frac{9}{2 OPT} \\
 &= \frac{3}{2} + \frac{1}{4}(-\delta) \\
 &= \frac{3}{2} + \min \left\{ \frac{1}{4r_L^*}, \frac{3}{6r_L^* + 2} \right\} (-\delta), \quad \delta \leq 0
 \end{aligned}$$

□

3.2 Tightness

THEOREM 8. *For any $r_L, r_L^* \in [0, 1]$, the asymptotic competitive ratio given in Theorem 7 is tight.*

PROOF. Let $\langle a_1, a_2, \dots, a_k \rangle \times n$ with $n \in \mathbb{N}$ denote n repetitions of the sequence $\langle a_1, a_2, \dots, a_k \rangle$.

Let $N \in \mathbb{N}$ and $\varepsilon = 1/(12N + 2)$.

Let I be a sequence consisting of the concatenated subsequences I_S, I_M and I_L . Let I_S be a sequence consisting of the two interleaved subsequences I_{SL} and I_{LL} .

Let

$$\begin{aligned} I_L &= \left(\frac{1}{2} + \frac{\varepsilon}{2} \right) \times n_L \text{ with } n_L = \lceil 4r_L^* N \rceil \\ I_M &= \left(\frac{1}{3} + \frac{\varepsilon}{2} \right) \times n_M \text{ with } n_M = \begin{cases} 0 & \text{for } r_L^* \leq 1/3 \\ \lfloor (6r_L^* - 2)N \rfloor & \text{for } r_L^* \geq 1/3 \end{cases} \\ I_{SS} &= \left(\frac{1}{3} - 2\varepsilon, \frac{1}{6} - \varepsilon, \frac{1}{6} - \varepsilon, 12\varepsilon \right) \times n_{SS} \text{ with } n_{SS} = \lceil n'_{SS} \rceil = \lceil (1 - r_L)N \rceil \\ I_{SL} &= \left(\frac{1}{6} - \varepsilon, 3\varepsilon \right) \times n_{SL} \text{ with } n_{SL} = \lceil n'_{SL} \rceil = \lceil 4r_L N \rceil \end{aligned}$$

Let I_S be interleaved in such a way that whenever PH3 would pack the next item in an L-bin, the next item in I_S is the next item in I_{SL} , otherwise it is the next item in I_{SS} . If the corresponding subsequence contains no more items, the next item is taken from the other one.

Note that the size of small items packed into each S-bin is exactly four times the size of small items packed into each L-bin. As $n_{SL} \geq n'_{SL}$ and $n_{SS} \geq n'_{SS}$ and

$$\frac{\text{size}(I_{SL})}{\text{size}(I_S)} \approx \frac{n'_{SL}}{n'_{SL} + 4n'_{SS}} = r_L.$$

PH3 will pack at least n_{SS} S-bins and n_{SL} L-bins. The last S-bin packed by PH3 may contain items from I_{SS} and the last L-bin might contain items from I_{SL} , depending on the differences $n_{SS} - n'_{SS}$ and $n_{SL} - n'_{SL}$. For the sake of simplicity, the up to two possible additional bins packed with small items are ignored, as they do not have impact on the competitive ratio due to the infinite length of the input sequences considered.

When packing S-bins, PH3 packs the items $1/3 - 2\varepsilon, 1/6 - \varepsilon, 1/6 - \varepsilon, 12\varepsilon$. The next item with size $1/3 - 2\varepsilon$ does no longer fit and a new bin is opened. As the subsequence packed into each S-bin is repeated n_{SS} times, $|B_S| = n_{SS}$.

As items of size $1/3 + \varepsilon/2$ are packed by twos in M-bins, $|B_M| = \lceil n_M/2 \rceil$.

When packing the $1/3$ -sub-bins of the L-bins, PH3 packs one item of size $1/6 - \varepsilon$, then one item of size 3ε . The next item has size $1/6 - \varepsilon$. As $1/6 - \varepsilon + 3\varepsilon + 1/6 - \varepsilon > 1/3$, a new bin will be opened. Accordingly, each L-bin $b \in B_L$ that is packed with small items will contain exactly the items $1/6 - \varepsilon$ and 3ε . As these two items occur n_{SL} times each in I_{SL} , the number of these bins is n_{SL} .

The items of size $1/2 + \varepsilon/2$ are large and thus packed into the $2/3$ -sub-bins of L-bins individually. As there are n_L such items, the number of L-bins that contain a large item is n_L as well.

The total number of L-bins is the maximum of the number of L-bins that contain small items and L-bins that contain large items: $|B_L| = \max\{n_{SL}, n_L\}$.

As the input sequence contains no extra large items, $|B_{XL}| = 0$.

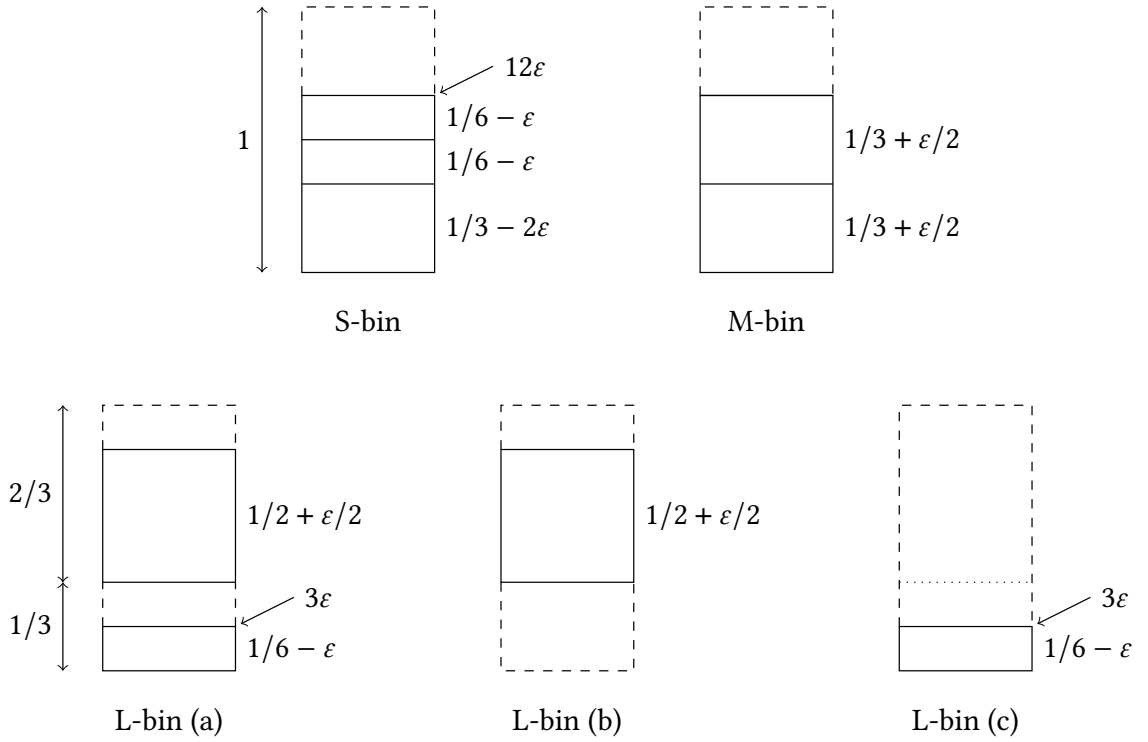


Fig. 1. The main types of bins packed by PH3. Dependent on r_L and r_L^* , type (a) L-bins and type (b) or type (c) L-bins are packed.

The packing of the bins is illustrated in Figure 1. Overall, PH3 needs the following number of bins.

$$\begin{aligned}
 PH_3 &= |B_{XL}| + |B_L| + |B_M| + |B_L| \\
 &= 0 + \max\{n_{SL}, n_L\} + \left\lceil \frac{n_M}{2} \right\rceil + n_{SS} \\
 &\geq \max\{4r_L N, 4r_L^* N\} + \frac{n_M}{2} + n_{SS} \\
 &= \max\{r_L, r_L^*\} 4N + \frac{n_M}{2} + (1 - r_L)N \\
 &= 3r_L^* N + \max\{\delta, 0\} 4N + \frac{n_M}{2} + N - \delta N
 \end{aligned}$$

To get an upper bound on OPT, consider the algorithm FFD. FFD is an offline bin packing algorithm that sorts the items in decreasing order and puts each item in the first bin that has enough space left to fit the item [13]. Note that although the order of the items is fixed, FFD can simulate an ordered list by assigning a bin to each item beforehand.

FFD will first pack each large item in a separate bin, opening n_L bins.

The medium items are packed one each in the first n_M bins alongside the large items. Note that there are $n_L = 4r_L^* N$ large items and $n_M = \lfloor (6r_L^* - 2)N \rfloor$ medium items. As $r_L^* \leq 1$, $n_L \geq n_M$ and thus each medium item can be packed with a large item. Let B_L the set of bins containing only a large item and B_M the set of bins containing a large and a medium item. Note that $|B_M| = n_M$ and $|B_L| = n_L - n_M$.

Items of size $1/3 - 2\varepsilon$ are packed into the bins B_L one each. If there are more such items than there are bins in B_L , the remaining items are packed by threes.

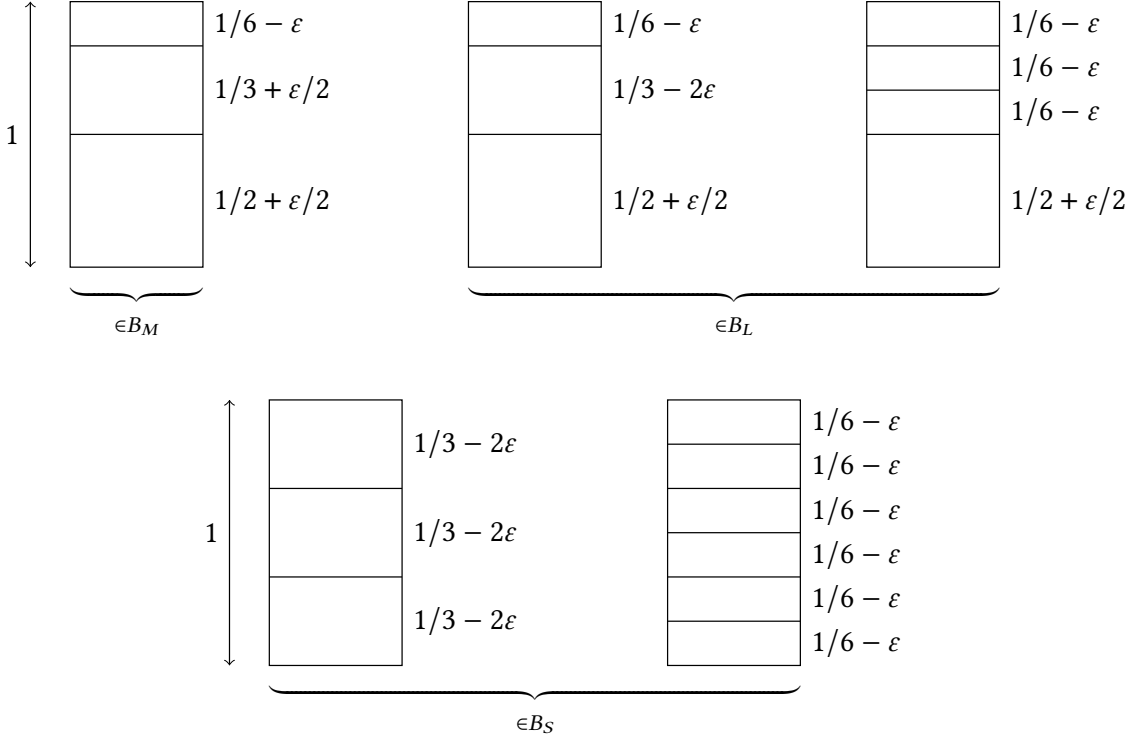


Fig. 2. The main types of bins packed by FFD.

Afterwards, the items of size $1/6 - \varepsilon$ are packed into the bins B_M and B_L , items that do not fit are packed into additional bins.

Each bin in B_M then contains, besides to the large and the medium item, exactly one item of size $1/6 - \varepsilon$.

Each bin $b \in B_L$ is packed either with the items $\{1/2 + \varepsilon/2, 1/3 - 2\varepsilon, 1/6 - \varepsilon\}$ or the items $\{1/2 + \varepsilon/2, 1/6 - \varepsilon, 1/6 - \varepsilon, 1/6 - \varepsilon\}$. In either case, the size of the packed small items is $\text{size}(b \cap S) = 1/2 - 3\varepsilon$.

Let size_S denote the total size of all items of size $1/3 - 2\varepsilon$ and $1/6 - \varepsilon$. Let size'_S denote the total size those items of size $1/3 - 2\varepsilon$ and $1/6 - \varepsilon$ that are packed into additional bins.

$$\begin{aligned}
 \text{size}'_S &= \text{size}_S - |B_L| \left(\frac{1}{2} - 3\varepsilon \right) - |B_M| \left(\frac{1}{6} - \varepsilon \right) \\
 &= n_{SS} \left(\frac{1}{3} - 2\varepsilon \right) + (n_{SL} + 2n_{SS} - n_M) \left(\frac{1}{6} - \varepsilon \right) - (n_L - n_M) \left(\frac{1}{2} - 3\varepsilon \right) \\
 &= (n_{SL} + 4n_{SS} - 3n_L + 2n_M) \left(\frac{1}{6} - \varepsilon \right)
 \end{aligned}$$

When packing the items of size $1/3 - 2\varepsilon$ and $1/6 - \varepsilon$ in additional bins, FFD packs three items of size $1/3 - 2\varepsilon$ or six items of size $1/6 - \varepsilon$ in each bin except one bin, which may contain items of both sizes, and the last bin, which may contain fewer items. Each of these bins, except for the last

has size $1 - 6\epsilon$. Let B_S denote the set of these bins.

$$\begin{aligned} B_S &= \left\lceil \frac{\text{size}'_S}{1 - 6\epsilon} \right\rceil \\ &= \left\lceil \frac{1}{1 - 6\epsilon} \left(\frac{1}{6} - \epsilon \right) (n_{SL} + 4n_{SS} - 3n_L + 2n_M) \right\rceil \\ &\leq \frac{n_{SL} + 4n_{SS} - 3n_L + 2n_M + 6}{6} \end{aligned}$$

For packing the items of size 12ϵ and the items of size 3ϵ , at most one additional bin is used, as

$$\begin{aligned} n_{SS}(12\epsilon) + n_{SL}(3\epsilon) &\leq (12n'_{SS} + 1)\epsilon + (3n_{SL} + 1)\epsilon \\ &= (12 - 12r_L)N\epsilon + 12r_LN\epsilon + 2\epsilon \\ &= (12N + 2)\epsilon \\ &= 1 \end{aligned}$$

Figure 2 shows a visualization of the main types of bins packed by FFD before items of size 12ϵ or smaller are packed. As FFD provides a feasible solution to the offline bin packing problem, this also yields a lower bound on the optimal solution OPT .

$$\begin{aligned} OPT &\leq FFD \leq |B_L| + |B_M| + |B_S| + 1 \\ &\leq n_L + \frac{n_{SL} + 4n_{SS} - 3n_L + 2n_M + 6}{6} + 1 \\ &= \frac{3n_L + n_{SL} + 4n_{SS} + 2n_M + 12}{6} \\ &= \frac{3\lceil 4r_L^*N \rceil + \lceil 4r_LN \rceil + 4\lceil (1 - r_L)N \rceil + 2n_M + 12}{6} \\ &\leq \frac{12r_L^*N + 4r_LN + 4(1 - r_L)N + 2n_M + 20}{6} \\ &= \frac{(12r_L^* + 4)N + 2n_M + 20}{6} \end{aligned}$$

Consider $r_L^* \geq 1/3$.

$$\begin{aligned} n_M &= \lfloor (6r_L^* - 2)N \rfloor \\ OPT &\leq \frac{(12r_L^* + 4)N + 2\lfloor (6r_L^* - 2)N \rfloor + 20}{6} \\ &\leq \frac{24r_L^*N + 20}{6} < 4r_L^*N + 4 \\ PH_3 &\geq 3r_L^*N + \max\{\delta, 0\}4N + \frac{\lfloor (6r_L^* - 2)N \rfloor}{2} + N - \delta N \\ &\geq 3r_L^*N + \max\{\delta, 0\}4N + (3r_L^* - 1)N + N - \delta N - 1 \\ &= 6r_L^*N + \max\{\delta, 0\}4N - \delta N - 1 \end{aligned}$$

Note that for $r_L^* \geq 1/3$, the following two equations hold.

$$\min \left\{ \frac{1}{4r_L^*}, \frac{3}{6r_L^* + 2} \right\} = \frac{1}{4r_L^*}$$

$$\min \left\{ \frac{3}{4r_L^*}, \frac{9}{6r_L^* + 2} \right\} = \frac{3}{4r_L^*}$$

Note that for $OPT \rightarrow \infty, N \rightarrow \infty$ also holds. Thus, for $N \rightarrow \infty$, a lower bound on the competitive ratio for $r_L^* \geq 1/3$ can be given.

$$\begin{aligned} \lim_{OPT \rightarrow \infty} \frac{PH_3}{OPT} &\geq \lim_{N \rightarrow \infty} \frac{6r_L^*N + \max\{\delta, 0\}4N - \delta N - 1}{4r_L^*N + 4} \\ &= \frac{3}{2} + \frac{4 \max\{\delta, 0\} - \delta}{4r_L^*} \\ &= \begin{cases} \frac{3}{2} + \frac{3}{4r_L^*} \delta & \text{for } \delta \geq 0 \\ \frac{3}{2} + \frac{1}{4r_L^*} (-\delta) & \text{for } \delta \leq 0 \end{cases} \\ &= R_{PH_3}^\infty \text{ for } r_L^* \geq 1/3 \end{aligned}$$

Now consider $r_L^* \leq 1/3$.

$$\begin{aligned} n_M &= 0 \\ OPT &\leq \frac{(12r_L^* + 4)N + 20}{6} \\ PH_3 &\geq 3r_L^*N + \max\{\delta, 0\}4N + N - \delta N \end{aligned}$$

Note that for $r_L^* \leq 1/3$, the following two equations hold.

$$\min \left\{ \frac{1}{4r_L^*}, \frac{3}{6r_L^* + 2} \right\} = \frac{3}{6r_L^* + 2}$$

$$\min \left\{ \frac{3}{4r_L^*}, \frac{9}{6r_L^* + 2} \right\} = \frac{9}{6r_L^* + 2}$$

As above, for $N \rightarrow \infty$, a lower bound on the competitive ratio for $r_L^* \leq 1/3$ can be given .

$$\begin{aligned}
\lim_{OPT \rightarrow \infty} \frac{PH_3}{OPT} &\geq \lim_{N \rightarrow \infty} 6 \frac{3r_L^* N + \max\{\delta, 0\} 4N + N - \delta N}{(12r_L^* + 4)N + 20} \\
&= \frac{9r_L^* + 12 \max\{\delta, 0\} + 3 - 3\delta}{6r_L^* + 2} \\
&= \frac{3}{2} + \frac{12 \max\{\delta, 0\} - 3\delta}{6r_L^* + 2} \\
&= \begin{cases} \frac{3}{2} + \frac{9}{6r_L^* + 2} \delta & \text{for } \delta \geq 0 \\ \frac{3}{2} + \frac{3}{6r_L^* + 2} (-\delta) & \text{for } \delta \leq 0 \end{cases} \\
&= R_{PH_3}^\infty \text{ for } r_L^* \leq 1/3
\end{aligned}$$

□

4 PARALLEL PREDICTIVE HARMONIC₃

Now we consider PH3 in the context of k -copy algorithms.

4.1 Competitive Ratio for PH3 as 1-Copy Online Algorithm

To optimize the performance for PH3 as a 1-copy algorithm, we determine the optimal value for r_L with respect to minimizing the asymptotic competitive ratio over all $r_L^* \in [0, 1]$.

LEMMA 9 (MONOTONICITY OF COMPETITIVE RATIO OF PH3). *For any fixed $r_L \in [0, 1]$, the competitive factor is monotonically decreasing for $r_L^* \in [0, r_L]$ and monotonically increasing for $r_L^* \in [r_L, 1]$.*

PROOF. Assume r_L to be fixed. Let $r_{+,<}, r_{-,<} : [0, 1/3] \rightarrow \mathbb{R}$ and $r_{+,>}, r_{-,>} : [1/3, 1] \rightarrow \mathbb{R}$ with

$$\begin{aligned}
r_{-,<}(r_L^*) &= \frac{3}{2} + \frac{3}{6r_L^* + 2} (-\delta) &&= R_{PH_3}^\infty \text{ for } \delta \leq 0, r_L^* \leq \frac{1}{3} \\
r_{-,>}(r_L^*) &= \frac{3}{2} + \frac{1}{4r_L^*} (-\delta) &&= R_{PH_3}^\infty \text{ for } \delta \leq 0, r_L^* \geq \frac{1}{3} \\
r_{+,<}(r_L^*) &= \frac{3}{2} + \frac{9}{6r_L^* + 2} \delta &&= R_{PH_3}^\infty \text{ for } \delta \geq 0, r_L^* \leq \frac{1}{3} \\
r_{+,>}(r_L^*) &= \frac{3}{2} + \frac{3}{4r_L^*} \delta &&= R_{PH_3}^\infty \text{ for } \delta \geq 0, r_L^* \geq \frac{1}{3}
\end{aligned}$$

Consider the derivative of $r_{-,<}$ and $r_{-,>}$.

$$\begin{aligned}
\frac{\partial}{\partial r_L^*} r_{-,<}(r_L^*) &= \frac{\partial}{\partial r_L^*} \left(\frac{3}{2} + \frac{3}{6r_L^* + 2} (-\delta) \right) \\
&= \frac{\partial}{\partial r_L^*} \left(\frac{3(r_L^* - r_L)}{6r_L^* + 2} \right) \\
&= \frac{18r_L + 6}{(6r_L^* + 2)^2} \geq 0 \text{ for } 0 \leq r_L \leq r_L^* \leq \frac{1}{3} \\
\frac{\partial}{\partial r_L^*} r_{-,>}(r_L^*) &= \frac{\partial}{\partial r_L^*} \left(\frac{3}{2} + \frac{1}{4r_L^*} (-\delta) \right) \\
&= \frac{\partial}{\partial r_L^*} \left(\frac{r_L^* - r_L}{4r_L^*} \right) \\
&= \frac{r_L}{4(r_L^*)^2} \geq 0 \text{ for } 0 \leq r_L \leq r_L^* \text{ and } \frac{1}{3} \leq r_L^* \leq 1
\end{aligned}$$

As the derivatives of $r_{-,<}$ and $r_{-,>}$ are both non-negative in their respective domains, they are both monotonically increasing. Because $r_{-,<}(\frac{1}{3}) = r_{-,>}(\frac{1}{3})$, we conclude that the competitive ratio is monotonically increasing for $r_L^* \in [r_L, 1]$.

Now consider the derivative of $r_{+,<}$ and $r_{+,>}$.

$$\begin{aligned}
\frac{\partial}{\partial r_L^*} r_{+,<}(r_L^*) &= \frac{\partial}{\partial r_L^*} \left(\frac{3}{2} + \frac{9}{6r_L^* + 2} \delta \right) \\
&= \frac{\partial}{\partial r_L^*} \left(\frac{9(r_L - r_L^*)}{6r_L^* + 2} \right) \\
&= \frac{-54r_L - 18}{(6r_L^* + 2)^2} \leq 0 \text{ for } r_L^* \leq r_L \leq 1 \text{ and } 0 \leq r_L^* \leq \frac{1}{3} \\
\frac{\partial}{\partial r_L^*} r_{+,>}(r_L^*) &= \frac{\partial}{\partial r_L^*} \left(\frac{3}{2} + \frac{3}{4r_L^*} \delta \right) \\
&= \frac{\partial}{\partial r_L^*} \left(\frac{3(r_L - r_L^*)}{4r_L^*} \right) \\
&= \frac{-3r_L}{4(r_L^*)^2} \leq 0 \text{ for } \frac{1}{3} \leq r_L^* \leq r_L \leq 1
\end{aligned}$$

As the derivatives of $r_{+,<}$ and $r_{+,>}$ are both non-positive in their respective domains, they are both monotonically decreasing. Because $r_{+,<}(\frac{1}{3}) = r_{+,>}(\frac{1}{3})$, we conclude that the competitive ratio is monotonically decreasing for $r_L^* \in [0, r_L]$. \square

Because of Lemma 9, the competitive ratio does not decrease with r_L^* increasing for $\delta \leq 0$. Thus, as an upper bound on the competitive ratio for $\delta \leq 0$, only the competitive ratio for $r_L^* = 1$ has to be considered.

$$\begin{aligned}
R_{PH3}^\infty &\leq \frac{3}{2} + \frac{1}{4}(-\delta) \text{ for } \delta \leq 0 \\
&= \frac{3}{2} + \frac{1}{4}(1 - r_L) \\
&= \frac{7}{4} - \frac{r_L}{4}
\end{aligned}$$

For $\delta \geq 0$, the competitive ratio does not decrease with r_L^* decreasing. In this case, the competitive ratio for $r_L^* = 0$ is an upper bound on the competitive ratio.

$$\begin{aligned} R_{PH3}^\infty &\leq \frac{3}{2} + \frac{9}{2}\delta \text{ for } \delta \geq 0 \\ &= \frac{3}{2} + \frac{9}{2}(r_L - 0) \\ &= \frac{3}{2} + \frac{9}{2}r_L \end{aligned}$$

At the same time, these values are lower bounds on the overall competitive ratio. Given these bounds, this linear program can be formulated to minimize the competitive ratio:

$$\begin{aligned} &\text{Minimize } R_{PH3}^\infty \\ &\text{Subject to } R_{PH3}^\infty \geq \frac{7}{4} - \frac{r_L}{4} \\ &\quad R_{PH3}^\infty \geq \frac{3}{2} + \frac{9}{2}r_L \\ &\quad r_L \geq 0 \\ &\quad r_L \leq 1 \end{aligned}$$

The optimal solution for this linear program is $r_L = 1/19$ and $R_{PH3}^\infty = 33/19 < 1.7369$. Figure 3 shows the asymptotic competitive ratio of PH3 over r_L^* for $r_L = 1/19$.

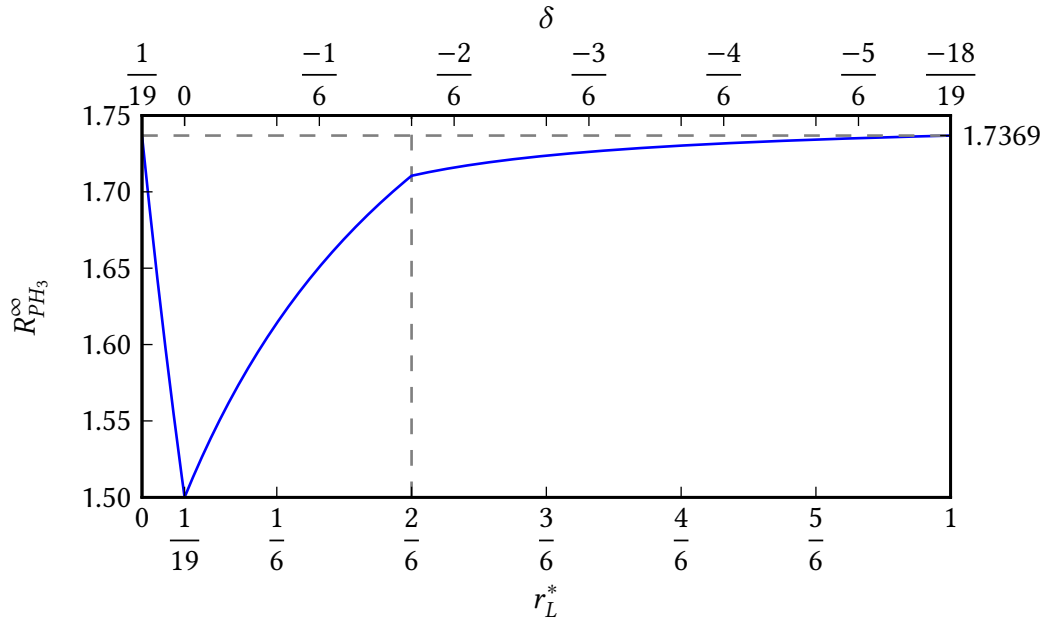


Fig. 3. Competitive ratio of the optimal 1-copy PH3 algorithm dependent on r_L^* for a fixed r_L .

Compared to other known algorithms for online bin packing, PH3 is not a good choice for worst-case behavior. Among the classical algorithms, only NF and WF, both of which are 2-competitive, are worse than PH3. Any AAF algorithm achieves an asymptotic competitive ratio $R_{AAF}^\infty = 1.7$ [9] and thus performs slightly better than PH3. The best-performing online algorithm for bin packing currently known, SON OF HARMONIC, is 1.5816-competitive and thus clearly superior to PH3 [11].

However, if we know in advance that r_L^* is restricted to some interval $I_r = [a, b] \subset [0, 1]$, the above argument can be used to prove a better competitive ratio.

4.2 Competitive Ratio for PH3 as k -Copy Online Algorithm

PH3's property of achieving a better competitive ratio for r_L^* being further restricted can be used to create a set of $k \in \mathbb{N}$ algorithms achieving a better competitive ratio. For this purpose, the interval $[0, 1]$ is split into k sub-intervals $I_1, \dots, I_k \subset [0, 1]$ with $\cup_{i \in \{1, \dots, k\}} I_i = [0, 1]$. Each interval I_i is covered by one instance of the algorithm PH3 A_i , such that A_i achieves a targeted competitive ratio $R \in (3/2, 33/19)$ for $r_L^* \in I_i$.

R is restricted to $(3/2, 33/19)$, because any competitive ratio above or equal to $33/19$ can be achieved with the instance of PH3 shown above, and k -copy PH3 cannot achieve a competitive ratio of $3/2$ or less with finitely many algorithms.

To calculate the number k of algorithms needed to achieve a given competitive ratio R , the following iterative approach can be used.

Let \mathcal{A} be a set of algorithms. Initially, $\mathcal{A} := \emptyset$. We initialize our iterative approach with $i = 0$ and set $r_{\max}^0 = 0$. Then, while $r_{\max}^i < 1$, we increase i by one and we compute three values r_{\min}^i , r_L^i and r_{\max}^i . With these three values we can define algorithm A_i for which r_L^i denotes the value of r_L , r_{\min}^i denotes the minimal and r_{\max}^i denotes the maximal value for r_L^* for which A_i is still R -competitive. By Lemma 9, A_i will be R -competitive for the interval $[r_{\min}^i, r_{\max}^i]$. All three values are computed as follows. We set $r_{\min}^i = r_{\max}^{i-1}$. Given r_{\min}^i , r_L^i can be computed:

If $r_{\min}^i \leq 1/3$, we have $R = \frac{3}{2} + \frac{9}{2+6r_{\min}^i}(r_L^i - r_{\min}^i)$. Solving this equation for r_L^i we get $r_L^i = r_{\min}^i + (R - \frac{3}{2}) \left(\frac{2+6r_{\min}^i}{9} \right)$. If $r_{\min}^i \geq 1/3$, we have $R = \frac{3}{2} + \frac{3}{4r_{\min}^i}(r_L^i - r_{\min}^i)$. Solving this equation for r_L^i yields $r_L^i = r_{\min}^i + (R - \frac{3}{2}) \left(\frac{4r_{\min}^i}{3} \right)$.

Having r_L^i , we can compute r_{\max}^i . Because the competitive ratio is the minimum of two values, we get two candidates $r_{\max,1}^i$ and $r_{\max,2}^i$ for r_{\max}^i . We can take the maximum of those two candidates, i.e., $r_{\max}^i = \max(r_{\max,1}^i, r_{\max,2}^i)$, because it is sufficient to be R -competitive in one case. In the first case ($\frac{3}{6r_L^i+2} < \frac{1}{4r_L^i}$) we obtain $r_{\max,1}^i = \frac{3r_L^i-3+2R}{12-6R}$ and in the second case we get $r_{\max,2}^i = \frac{r_L^i}{7-4R}$.

Now consider the case when $r_{\max}^i \geq 1$. Because each algorithm A_ℓ with $1 \leq \ell \leq i$ is R -competitive for the interval $[r_{\min}^\ell, r_{\max}^\ell] = [r_{\max}^{\ell-1}, r_{\max}^\ell]$ with $r_{\min}^0 = 0$, there is an algorithm A_m for any $r_L^* \in [0, 1]$ that is R -competitive. Therefore, we have a i -copy online algorithm for bin packing achieving the competitive factor R .

Following this method, we see that $k = 6$ algorithms are sufficient to guarantee a competitive ratio $R = 1.5815$. This beats the currently best 1-copy online algorithm SON OF HARMONIC with a competitive ratio of 1.5816. Figure 4 shows the competitive ratio achieved by the individual algorithms over $r_L^* \in [0, 1]$ for $R = 1.5815$. Note that 1.5815 is not the best competitive ratio achievable by 6-copy PH3, as shown below in Figure 5. Using $k = 12$ algorithms, a competitive ratio $R = 1.5402 < 1.5403$ can be achieved, beating the highest known lower bound for 1-copy online algorithms.

To compute the best competitive ratio achievable by $k \in \mathbb{N}$ algorithms, we use binary search on R starting in the interval $[3/2, 33/19]$ and test in each iteration if we can guarantee R -competitiveness with at most k algorithms. Figure 5 shows the best competitive ratios achievable by k -copy PH3.

4.3 Comparison to Related Algorithms

Because k -copy online algorithms can be translated to an online algorithm with advice and vice versa (see Lemma 1), it seems natural to compare these two variants, even though k -copy allows a

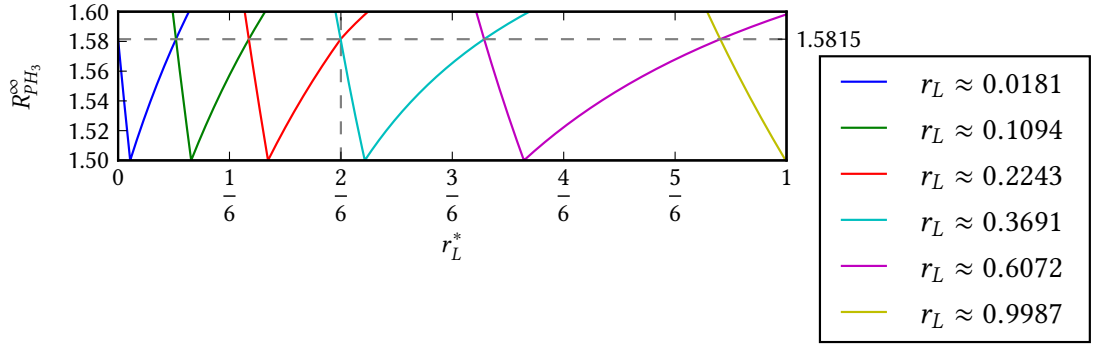


Fig. 4. 6-copy PH3 beats the best 1-copy online algorithm known to date, achieving an asymptotic competitive ratio $R_{PH3}^{\infty} < 1.5815$.

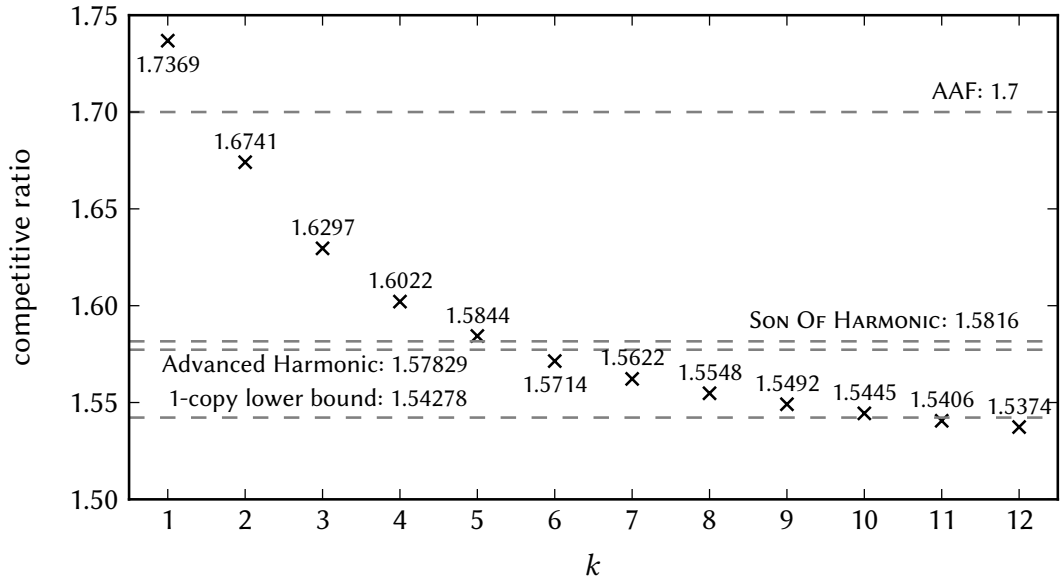


Fig. 5. k -copy PH3 performance dependent on k .

more precise analysis on the competitive ratio. In this subsection we compare our algorithm to the best known online algorithm with constant advice, namely REDBLUE introduced by Angelopoulos et al. [2]. Their second algorithm is 1.47012-competitive (and thus beats our algorithm), but the amount of advice needed by this algorithm is too large. As the focus of k -copy algorithms is to provide good solutions for small k , it is reasonable to only compare k -copy PH3 to REDBLUE.

Table 1 shows a comparison between REDBLUE and k -copy PH3 for small amounts of advice. The competitive ratios given are rounded up to the fourth decimal place. The competitive ratios for REDBLUE are computed using the upper bound on the competitive ratio $1.5 + 15/(2^{\ell/2+1})$. The competitive ratios for k -copy PH3 are calculated using binary search as described above.

Table 1 clearly shows the advantage of k -copy PH3 over REDBLUE for few bits of advice. With as few as 5 bits of advice, or $k = 32$, k -copy PH3 achieves a better competitive ratio than REDBLUE with 16 bits of advice, which corresponds to $k = 65536$ algorithms when used as k -copy algorithm.

| Advice in bits | k | $R_{\text{REDBLUE}}^{\infty}$ | R_{PH3}^{∞} |
|----------------|-------|-------------------------------|---------------------------|
| 4 | 16 | 3.3750 | 1.5305 |
| 5 | 32 | 2.8258 | 1.5155 |
| 6 | 64 | 2.4375 | 1.5078 |
| 7 | 128 | 2.1629 | 1.5040 |
| 8 | 256 | 1.9688 | 1.5020 |
| 9 | 512 | 1.8315 | 1.5010 |
| 10 | 1024 | 1.7344 | 1.5005 |
| 11 | 2048 | 1.6657 | 1.5003 |
| 12 | 4096 | 1.6172 | 1.5002 |
| 13 | 8192 | 1.5829 | 1.5001 |
| 14 | 16384 | 1.5586 | 1.5001 |
| 15 | 32768 | 1.5414 | 1.5001 |
| 16 | 65536 | 1.5293 | 1.5001 |

Table 1. Comparison of the performance of k -copy PH3 and REDBLUE.

Although REDBLUE and k -copy PH3 work in a similar way, k -copy PH3 achieves a better competitive ratio due to the more precise analysis of the intervals for r_L^* , in which each algorithm achieves the competitive ratio. By avoiding overlaps in these intervals, fewer algorithms are needed.

On the other hand, REDBLUE simply splits an interval for its parameter β evenly into $2^{\ell/2}$ intervals; translated into a k -copy setting, this leads to overlaps in the intervals covered by each algorithm.

5 NON-RECURSIVE UPPER BOUND FOR k -COPY PH3

In this section we provide an analytic proof that the competitive ratio of k -copy PH3 can be upper bounded by $\frac{3}{2} + \frac{3 \cdot 4^{1/k} - 3}{6 \cdot 4^{1/k} + 2}$ for k algorithms. To this end, we start by calculating the number of algorithms needed to cover the interval $[0, \frac{1}{3}]$ and $[\frac{1}{3}, 1]$. We call these $k_{\leq \frac{1}{3}}$ and $k_{\geq \frac{1}{3}}$, respectively.

LEMMA 10. *Let $R \in (\frac{3}{2}, \frac{33}{19}]$. Then, $k_{\leq \frac{1}{3}} \geq \frac{1}{\log_2(\frac{R}{6-3R})}$.*

PROOF. With the construction of the algorithms above we have

$$\begin{aligned}
 r_{\max}^i &= \frac{3r_L^i - 3 + 2R}{12 - 6R} = \frac{3 \cdot \left(r_{\max}^{i-1} + (R - \frac{3}{2}) \left(\frac{2+6r_{\max}^{i-1}}{9} \right) \right)}{12 - 6R} \\
 &= \frac{2R + 6Rr_{\max}^{i-1} - 12 + 6R}{6 \cdot (6 - 3R)} \\
 &= \frac{1}{3} \left(\frac{1 + 3r_{\max}^{i-1}}{6 - 3R} R - 1 \right)
 \end{aligned}$$

From this, we can proof that $r_{\max}^n = \frac{1}{3} \left(\left(\frac{R}{6-3R} \right)^n - 1 \right)$. For $n = 0$ this is true, as we start with $r_{\max}^0 = 0$. Now, suppose this equation is true for some $n \in \mathbb{N}$. We show that it is still true for $n + 1$.

$$\begin{aligned}
r_{\max}^{n+1} &= \frac{1}{3} \left(\frac{1 + 3r_{\max}^n}{6 - 3R} R - 1 \right) \\
&= \frac{1}{3} \left(\left(\frac{R}{6 - 3R} \right)^n R - 1 \right) \\
&= \frac{1}{3} \left(\left(\frac{R}{6 - 3R} \right)^{n+1} - 1 \right)
\end{aligned}$$

To cover the $[0, \frac{1}{3}]$ interval with $n := k_{\leq \frac{1}{3}}$ algorithms, r_{\max}^n must be at least $\frac{1}{3}$. Therefore,

$$\begin{aligned}
r_{\max}^n \geq \frac{1}{3} &\Leftrightarrow \frac{1}{3} \left(\left(\frac{R}{6 - 3R} \right)^n - 1 \right) \geq \frac{1}{3} \\
&\Leftrightarrow \left(\frac{R}{6 - 3R} \right)^n - 1 \geq 1 \\
&\Leftrightarrow n \geq \log_{\frac{R}{6 - 3R}} (2) \\
&\Leftrightarrow n \geq \frac{1}{\log_2 \left(\frac{R}{6 - 3R} \right)}
\end{aligned}$$

□

LEMMA 11. Let $R \in (\frac{3}{2}, \frac{33}{19}]$. Then, $k_{\geq \frac{1}{3}} \geq \frac{1}{\log_3 \left(\frac{4}{7 - 4R} - 1 \right) - 1}$.

PROOF. Similar to Lemma 10, we have

$$\begin{aligned}
r_{\max}^i &= \frac{r_L^i}{7 - 4R} = \frac{r_{\max}^{i-1} + (R - \frac{3}{2}) \left(\frac{4r_{\max}^{i-1}}{3} \right)}{7 - 4R} \\
&= \frac{1 + \frac{4R}{3} - 2}{7 - 4R} r_{\max}^{i-1} \\
&= \frac{4R - 3}{3(7 - 4R)} r_{\max}^{i-1} \\
&= \frac{1}{3} \left(\frac{4}{7 - 4R} - 1 \right) r_{\max}^{i-1}
\end{aligned}$$

With $r_{\max}^0 = \frac{1}{3}$ we have $r_{\max}^n = \left(\frac{1}{3} \right)^{n+1} \left(\frac{4}{7 - 4R} - 1 \right)^n$. To cover the $[\frac{1}{3}, 1]$ interval with $n := k_{\geq \frac{1}{3}}$ algorithms, r_{\max}^n must be at least 1.

$$\begin{aligned}
r_{\max}^n \geq 1 &\Leftrightarrow \left(\frac{1}{3} \right)^{n+1} \left(\frac{4}{7 - 4R} - 1 \right)^n \geq 1 \\
&\Leftrightarrow (n + 1) \log_3 \left(\frac{1}{3} \right) + n \log_3 \left(\frac{4}{7 - 4R} - 1 \right) \geq 0 \\
&\Leftrightarrow n \left(\log_3 \left(\frac{4}{7 - 4R} - 1 \right) - 1 \right) \geq 1 \\
&\Leftrightarrow n \geq \frac{1}{\log_3 \left(\frac{4}{7 - 4R} - 1 \right) - 1}
\end{aligned}$$

□

LEMMA 12. For $R \in (\frac{3}{2}, \frac{33}{19}]$ we have $k_{\leq \frac{1}{3}} \geq k_{\geq \frac{1}{3}}$.

PROOF. Let $A(R) := \log_2 \left(\frac{R}{6-3R} \right)$ and $B(R) := \log_3 \left(\frac{4}{7-4R} - 1 \right) - 1$. Then, $k_{\leq \frac{1}{3}} \geq k_{\geq \frac{1}{3}}$ is true if and only if $A(R) \leq B(R)$. We show in the following that both functions are monotonically increasing for $R \in [\frac{1}{3}, \frac{33}{19}]$ and that B increases faster than A in this interval.

Because $A(R) = \log_2 \left(\frac{2}{6-3R} - \frac{1}{3} \right)$, we can directly see that both, A and B , are monotonically increasing for $R \in [\frac{1}{3}, \frac{33}{19}]$. Now, consider $A'(R) = \frac{2}{2R \log 2 - R^2 \log 2}$ and $B'(R) = \frac{16}{(7-4R)(4R-3) \log 3}$. Then,

$$A'(R) \leq B'(R) \Leftrightarrow \frac{8 \log 2}{\log 3} \frac{2R - R^2}{(7-4R)(4R-3)} \geq 1$$

Consider the derivative of $\frac{2R-R^2}{(7-4R)(4R-3)}$, i.e., $\frac{42R-8R^2-42}{((7-4R)(4R-3))^2}$. This function is positive whenever the numerator is positive. Because the numerator is a negative parabola with roots at approximately 1.34 and 3.9, the function is positive in the interval $[\frac{3}{2}, \frac{33}{19}]$. Therefore, $\frac{A'(R)}{B'(R)}$ is monotonically increasing. For $R = \frac{3}{2}$, this quotient is equal to $\frac{\log 4}{\log 3} > 1$, i.e., B grows faster than A .

Because $A(\frac{3}{2}) = 0 = B(\frac{3}{2})$ and $A'(R) \leq B'(R)$ for all $R \in [\frac{3}{2}, \frac{33}{19}]$, it follows that $A(R) \leq B(R)$ for all $R \in [\frac{3}{2}, \frac{33}{19}]$ and therefore $k_{\leq \frac{1}{3}} \geq k_{\geq \frac{1}{3}}$. \square

THEOREM 13. *PH3 with k algorithms has a competitive factor of at most $\frac{3}{2} + \frac{3 \cdot 4^{1/k} - 3}{6 \cdot 4^{1/k} + 2}$.*

PROOF. From previous lemmas we conclude:

$$k \leq k_{\leq \frac{1}{3}} + k_{\geq \frac{1}{3}} \leq 2k_{\leq \frac{1}{3}} = \frac{2}{\log_2 \left(\frac{R}{6-3R} \right)}$$

Solving this for R yields

$$R \leq 2 - \frac{2}{1 + 3 \cdot 4^{1/k}} = \frac{3}{2} + \underbrace{\frac{3 \cdot 4^{1/k} - 3}{6 \cdot 4^{1/k} + 2}}_{\rightarrow 0, k \rightarrow \infty} \quad \square$$

6 CONCLUSION

We have studied the concept of parallel online algorithms for the Bin Packing Problem. We developed a k -copy online algorithm named PH3 and showed that PH3 has an asymptotic competitive ratio of at most $\frac{3}{2} + \frac{3 \cdot 4^{1/k} - 3}{6 \cdot 4^{1/k} + 2}$, which converges to $\frac{3}{2}$ for large k ; in particular, $k = 11$ suffices to break through the lower bound for a single online algorithm. We also considered the relationship to online algorithms with advice and achieved a considerable improvement compared to a previous algorithm.

There are various directions for future work. We saw that PH3 is $(1.5 + \varepsilon)$ -competitive if $\frac{|I_L|}{6 \text{ size}(I_S)} \leq 1$, i.e., when there is a surplus of small items. If there are too few small items, PH3 is asymptotically $(1.5 + \varepsilon)$ -competitive. Can we make better use of the second case for an improvement? Can we guarantee an absolute competitive ratio of $1.5(+\varepsilon)$? We also believe that the concept of k -copy algorithms is useful for a wide range of other problems.

REFERENCES

- [1] Susanne Albers and Matthias Hellwig. 2017. Online makespan minimization with parallel schedules. *Algorithmica* 78, 2 (2017), 492–520.
- [2] Spyros Angelopoulos, Christoph Dürr, Shahin Kamali, Marc Renault, and Adi Rosén. 2015. Online Bin Packing with Advice of Small Size. In *14th Symp. Alg. Dat. Struct. (WADS)*. 40–53.
- [3] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. 2018. A New and Improved Algorithm for Online Bin Packing. In *26th Annual European Symposium on Algorithms (ESA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

- [4] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. 2018. A new lower bound for classic online bin packing. *arXiv preprint arXiv:1807.05554* (2018). To appear at 17th Workshop on Approximation and Online Algorithms (WAOA).
- [5] Joan Boyar, Lene M Favrholdt, Christian Kudahl, Kim S Larsen, and Jesper W Mikkelsen. 2017. Online algorithms with advice: A survey. *ACM Computing Surveys (CSUR)* 50, 2 (2017), 19.
- [6] Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. 2014. On the List Update Problem with Advice. In *8th Conf. Lang. Autom. Th. Appl. (LATA)*. 210–221.
- [7] Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. 2016. Online Bin Packing with Advice. *Algorithmica* 74, 1 (2016), 507–527. <https://doi.org/10.1007/s00453-014-9955-8>
- [8] Donna M. Brown. 1979. *A Lower Bound for On-Line One-Dimensional Bin Packing Algorithms*. Technical Report.
- [9] János Csirik and Gerhard J. Woeginger. 1998. On-line packing and covering problems. In *Online Algorithms: The State of the Art*. 147–177. <https://doi.org/10.1007/BFb0029568>
- [10] Magnús M Halldórsson, Kazuo Iwama, Shuichi Miyazaki, and Shiro Taketomi. 2002. Online independent sets. *Th. C.S.* 289, 2 (2002), 953–962.
- [11] Sandy Heydrich and Rob van Stee. 2016. Beating the Harmonic Lower Bound for Online Bin Packing. In *43rd Int. Coll. Autom., Lang., Prog. (ICALP)*. 41:1–41:14.
- [12] David S. Johnson. 1974. Fast algorithms for bin packing. *J. Comput. System Sci.* 8, 3 (1974), 272 – 314. [https://doi.org/10.1016/S0022-0000\(74\)80026-7](https://doi.org/10.1016/S0022-0000(74)80026-7)
- [13] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. 1974. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. *SIAM J. Comput.* 3, 4 (1974), 299–325. <https://doi.org/10.1137/0203025> [arXiv:https://doi.org/10.1137/0203025](https://arxiv.org/abs/https://doi.org/10.1137/0203025)
- [14] S. Kamali and A. L. Ortiz. 2014. Better Compression through Better List Update Algorithms. In *2014 Data Compression Conference*. 372–381.
- [15] C. C. Lee and D. T. Lee. 1985. A Simple On-line Bin-packing Algorithm. *J. ACM* (1985), 562–572. <https://doi.org/10.1145/3828.3833>
- [16] Frank M. Liang. 1980. A lower bound for on-line bin packing. *Inform. Process. Lett.* 10, 2 (1980), 76 – 79. [https://doi.org/10.1016/S0020-0190\(80\)90077-0](https://doi.org/10.1016/S0020-0190(80)90077-0)
- [17] Alejandro López-Ortiz and Sven Schuierer. 2004. On-line parallel heuristics, processor scheduling and robot searching under the competitive framework. *Th. C.S.* 310, 1-3 (2004), 527–537.
- [18] Marc P Renault, Adi Rosén, and Rob van Stee. 2015. Online algorithms with advice for bin packing and scheduling problems. *Th. C.S.* 600 (2015), 155–170.
- [19] André van Vliet. 1992. An improved lower bound for on-line bin packing algorithms. *Inf. Proc. Letters* 43, 5 (1992), 277 – 284. [https://doi.org/10.1016/0020-0190\(92\)90223-I](https://doi.org/10.1016/0020-0190(92)90223-I)
- [20] Andrew Chi-Chih Yao. 1980. New Algorithms for Bin Packing. *J. ACM* 27, 2 (1980), 207–227. <https://doi.org/10.1145/322186.322187>
- [21] Xiaofan Zhao and Hong Shen. 2014. On the Advice Complexity of One-Dimensional Online Bin Packing. In *Frontiers in Algorithmics*. 320–329.

