

Local Reconfiguration Algorithms for Geometric Modular Systems

3D Hybrid Programmable Matter and Distributed Sliding Squares

Dissertation

In partial fulfillment of the requirements for the academic degree
Doctor rerum naturalium (Dr. rer. nat.)

Faculty of Computer Science,
Electrical Engineering and Mathematics
Department of Computer Science
Research Group Theory of Distributed Systems

David Liedtke



Advisor:

Prof. Dr. Christian Scheideler

Paderborn University

Reviewers:

Prof. Dr. Christian Scheideler

Paderborn University

Prof. Dr. Sevag Gharibian

Paderborn University

Prof. Dr. Peter Kling

Darmstadt University
of Applied Sciences

Members of the Doctoral Panel:

Prof. Dr. Christian Scheideler (Chairperson)

Paderborn University

Prof. Dr. Sevag Gharibian

Paderborn University

Prof. Dr. Marcel Campen

Paderborn University

Dr. Matthias Fischer

Paderborn University

Dr. Thorsten Götte

University of Hamburg

Contact:

David Liedtke (liedtke@mail.upb.de)

Abstract

Programmable matter and modular self-reconfigurable robots pursue similar approaches. They give rise to systems of many simple units that collectively form, modify, or transform geometric structures. A central challenge is to achieve global objectives despite severe local restrictions on memory, sensing, and movement. This thesis studies local reconfiguration algorithms for two closely related settings with different underlying geometries and control mechanisms.

The first part considers the three-dimensional hybrid model for programmable matter. In this model, a single constant-memory agent manipulates a connected configuration of passive tiles on the face-centered cubic lattice. We first prove a fundamental limitation of this model: in three dimensions, universal reconfiguration is impossible if the agent does not initially carry a tile. Using such a tile, we then present an algorithm that transforms any connected configuration into a compact canonical shape. In addition, we study the problem of coating the surface of a three-dimensional object with tiles.

The second part studies the sliding square model, a model for modular self-reconfigurable robots that is typically considered from a centralized perspective. In this model, square modules occupy cells of the square grid and reconfigure by sliding along neighboring modules or rotating around them. We depart from this centralized perspective and present a distributed algorithm that transforms any side-connected configuration into a rhombus, the most compact form in the square grid.

Zusammenfassung

Programmierbare Materie und modulare selbstrekonfigurierbare Roboter verfolgen ähnliche Ansätze. Aus vielen einfachen Einheiten entstehen Systeme, die gemeinsam geometrische Strukturen bilden, verändern oder in andere Formen überführen können. Eine zentrale Herausforderung besteht darin, globale Ziele trotz stark eingeschränkter lokaler Fähigkeiten in Bezug auf Speicher, Wahrnehmung und Bewegung zu erreichen. Diese Arbeit untersucht lokale Rekonfigurationsalgorithmen in zwei eng verwandten Szenarien, die sich jedoch in ihrer Geometrie und in ihren Steuerungsmechanismen unterscheiden.

Der erste Teil behandelt das dreidimensionale hybride Modell für programmierbare Materie. Darin manipuliert ein einzelner Agent mit konstantem Speicher eine zusammenhängende Konfiguration passiver Kacheln auf dem kubisch flächenzentrierten Gitter. Wir zeigen zunächst, dass universelle Rekonfiguration in drei Dimensionen unmöglich ist, wenn der Agent zu Beginn keine Kachel trägt. Mit einer solchen Kachel stellen wir anschließend einen Algorithmus vor, der die Kacheln in eine kompakte kanonische Form überführt. Darüber hinaus untersuchen wir das Problem, die Oberfläche eines Objekts mit Kacheln zu beschichten.

Der zweite Teil befasst sich mit dem Sliding-Square-Modell für modulare selbstrekonfigurierbare Roboter, das üblicherweise zentralisiert betrachtet wird. Darin belegen quadratische Module Zellen des Quadratgitters und rekonfigurieren sich, indem sie entlang benachbarter Module gleiten oder um diese rotieren. Wir verlassen diese zentralisierte Sichtweise und stellen einen verteilten Algorithmus vor, der die Module in die kompakteste Form im Quadratgitter überführt, einen Rhombus.

Acknowledgments

First and foremost, I would like to thank my advisor, Christian Scheideler, for his guidance and support over the past five years. He helped me discover interesting research directions, discussed many ideas with me, and spent a great deal of time reading and improving my papers. I am especially grateful for the freedom and trust he gave me, and for the way he supports his group by shielding his assistants from many non-research tasks.

I am also deeply grateful to Irina Kostitsyna, one of my coauthors, for our many discussions. Her feedback was always thoughtful and constructive, and she remained patient and supportive even when my ideas were still rough. I would also like to thank Kristian Hinnenthal, who introduced me to this field and with whom I had the pleasure of collaborating on the work that followed.

My thanks also go to all current and former members of the Theory of Distributed Systems group. I am grateful for the friendly atmosphere, the many discussions, and the support over the years. In particular, I would like to thank Julian Werthmann, who was a wonderful office partner and with whom I shared many delightful conversations, even if they were rarely about research. I also thank Andreas Padalkin, who was a great travel companion at various conferences.

Finally, I would like to thank my family and friends. Without their support, I would not have made it this far. Special thanks go to my parents, Irntraud and Reiner, and to my girlfriend Marion, who supported me especially during stressful periods and helped me keep going when things became difficult.

Contents

1. Introduction	1
I. Hybrid Programmable Matter	7
2. The Hybrid Model for Programmable Matter	9
2.1. Preliminaries	9
2.2. Model Definition	10
2.2.1. Underlying Graph	10
2.2.2. Agent Model	12
2.2.3. Constraints and Terminology	13
2.3. Problem Statements	14
2.4. Related Work	15
3. The Necessity of an Initial Tile for 3D Reconfiguration	17
3.1. Safely Removable and Safely Shiftable Tiles	17
3.2. Impossibility Proof	19
4. Reconfiguration into Compact Low-Diameter Structures	31
4.1. Related Work	33
4.2. Additional Terminology	35
4.3. The Algorithm	37
4.3.1. Parallelogram Formation in Two Dimensions	38
4.3.2. Projection	40
4.3.3. Icicle Formation	42
4.4. Analysis	46
4.5. Termination Criterion and Runtime	56
4.6. Experimental Analysis	63
4.7. Conclusion	66
5. Coating Surfaces with Tiles	67
5.1. Related Work	69
5.2. Coating Algorithm for Surface Graphs	70
5.2.1. High-Level Description	70
5.2.2. Additional Terminology	72
5.2.3. The Algorithm	74
5.3. Analysis	76
5.3.1. Runtime	86
5.4. Coating in the FCC Lattice Graph	87
5.4.1. Emulating Coatable Surface Graphs	90
5.5. Conclusion	94

II. Modular Self-Reconfigurable Robots	95
6. The Sliding Square Model	97
6.1. Model Definition	98
6.2. Problem Statement	100
6.2.1. Additional Terminology	101
6.3. Related Work	102
7. Distributed Rhombus Formation	105
7.1. A Distributed Rhombus Formation Algorithm	106
7.1.1. Activating Bridged Removable Modules	109
7.1.2. Moving Past Critical Pairs	110
7.2. Analysis	111
7.2.1. The Case of a Size-One Target Hole	124
7.2.2. Correctness and Runtime	125
7.3. Utilizing Parallelism	125
7.3.1. Maintaining the Spanning Tree	127
7.3.2. Moving Through Leaves	129
7.3.3. Ignoring Unattached Leaves	129
7.3.4. Theoretical Guarantees	130
7.4. Parallel Variants and Experimental Evaluation	132
7.5. Conclusion	135
Bibliography	137

Introduction

Programmable matter refers to materials composed of many simple units that collectively alter their shape or physical properties in a programmable manner. Such systems promise technologies in which physical structures can autonomously assemble, reconfigure, and repair themselves in response to changing conditions or tasks. Recent advances in molecular engineering, swarm robotics, and modular robotic systems have brought this vision closer to reality and have inspired a growing body of theoretical models that investigate the algorithmic foundations of such systems.

Potential applications of programmable matter span a wide range of physical scales. At the nanoscale, molecular robots constructed from DNA strands have already been designed to transport cargo, communicate with one another, navigate membrane surfaces, and perform simple pathfinding operations. Future developments may enable large numbers of such units to collaboratively assemble structures inside the human body, for example to support targeted drug delivery, tissue repair, or the isolation of pathogens. At larger scales, programmable matter could be used for environmental remediation by assembling filtration systems that remove pollutants from water or air, or for autonomous construction in space, where robotic systems assemble lightweight structures such as antennas, solar arrays, or habitats from modular building blocks.

Across these settings, the individual units are typically subject to severe resource limitations. At very small scales, physical size restricts the computational capabilities, memory, sensing, and communication available to each unit. Even in macroscopic robotic systems, practical considerations such as scalability, robustness, and energy consumption often require the individual units to remain simple. As a result, programmable matter systems must be designed so that many simple units can collectively achieve complex tasks. This raises fundamental algorithmic questions: How can large structures be assembled or reconfigured when each unit has only limited memory and access to local information? How can global objectives be achieved without centralized control and using only local communication?

Two closely related research traditions investigate these questions. In the distributed computing and self-assembly communities, they are typically studied in the area of *programmable matter*. In contrast, the robotics community considers similar questions in the context of *modular self-reconfigurable robots*, where the robot consists of modules that physically connect to one another and collectively rearrange their shape. These traditions also differ in their algorithmic assumptions. Programmable matter models usually consider fully distributed systems in which each individual unit operates autonomously. By contrast, models for modular self-reconfigurable robots are typically studied in centralized settings, where a global controller computes a sequence of moves that transforms the robot into a desired shape.

This thesis studies algorithmic questions arising from both of these research traditions. The first part considers *hybrid programmable matter*, a model in which a small number of active agents manipulate a large collection of passive building blocks. The second part considers the *sliding square model*, a well-established model for modular self-reconfigurable robots in which modules move relative to their neighbors by sliding along adjacent modules or rotating around them. While this model has traditionally been studied in centralized settings, this thesis examines it from a distributed perspective in which modules operate autonomously based only on local information. In both parts, we assume minimal capabilities of each individual unit with respect to memory, sensing, and communication.

Hybrid Programmable Matter. Over the past decades, numerous models of programmable matter have been proposed, which primarily differ in the capabilities of the individual units. A common distinction is between passive and active systems. In passive systems, the units do not perform computation or deliberate motion; instead, structures emerge from local interactions determined by external stimuli or structural properties. Prominent examples of passive programmable matter are DNA tile assembly models such as the abstract Tile Assembly Model and its variants [Pat14]. In these models, tiles carry glues on their sides that determine how they bind to one another, causing larger structures to assemble through local attachment rules. In practice, such tiles can be constructed by folding DNA strands into nanoscale building blocks. Other examples of passive systems include population protocols [Ang+06] and certain biological systems such as slime molds [Che+23].

In contrast, active systems consist of units that autonomously perform computation and movement. A prominent example is the Amoebot model [Der+14], in which mobile units, called particles, have the capabilities of finite automata and move by expansions and contractions. Other examples include the nubot model [Woo+13], metamorphic robots [Chi94], and swarm robotic systems [WPN14]. While active systems offer considerable flexibility, manufacturing large numbers of autonomous computational units remains technologically challenging, particularly at very small scales. In contrast, passive building blocks such as DNA tiles can be produced efficiently in large quantities [HL19].

The hybrid model for programmable matter [Gmy+18] combines passive and active approaches by allowing a small number of active agents to manipulate a large collection of passive building blocks, called tiles. Unlike the DNA tile assembly models discussed above, the tiles in the hybrid model are assumed to be uniform and stateless and cannot move on their own. Instead, the structure is manipulated by active agents that have only the computational power of finite automata. These agents can move on top of the tile structure, pick up tiles, and place them at other locations. Throughout this process, the tiles and agents are required to remain connected in order to prevent parts of the structure from drifting apart, for example in liquid or low-gravity environments.

This hybrid approach offers several advantages compared to purely passive or purely active systems. Because the tiles are simple passive building blocks, they can be produced efficiently in large quantities. Moreover, since the agents remain separate from the constructed structure, they can be reused once a task is completed.

In many scenarios, complex tasks can already be accomplished with only a small number of active agents. In this thesis we study the extreme case in which the system is controlled by a single active agent.

While the hybrid model is primarily motivated by molecular self-assembly, similar concepts also arise at larger scales. For example, robotic construction systems have been developed in which robots move across a structure while manipulating modular building blocks in order to assemble lightweight structures for space applications [Gre+24]. In such systems, a small number of robots operate on a large set of passive building blocks, closely resembling the hybrid model.

In this thesis we consider the three-dimensional variant of the hybrid model, in which tiles have the shape of rhombic dodecahedra and are positioned at points of the face-centered cubic lattice. In contrast to rectangular tiles, this geometry allows an agent to move around a tile while remaining connected to it. Within this model, we study algorithmic problems related to the manipulation of tile structures by a single active agent. In particular, we investigate the problems of shape reconfiguration and coating, where coating refers to applying a thin layer of tiles to an object. In addition, we prove a fundamental limitation of the model by showing that shape reconfiguration is impossible if the agent does not initially carry a tile.

Modular Self-Reconfigurable Robots. The second part of this thesis studies modular self-reconfigurable robots through the sliding square model, a classical model introduced by Fitch et al. [FBR03]. In this model, identical square modules occupy cells of a two-dimensional grid and form a single connected configuration through shared edges. Modules can change the overall shape of the system by moving relative to their neighbors either by sliding along adjacent modules or by rotating around them. A sliding move shifts a module to a neighboring empty cell while remaining adjacent to a supporting module, whereas a rotation moves a module around the corner of an adjacent module.

Compared to the hybrid model, the sliding square model more directly captures the physical constraints of modular robotic systems. In particular, during any movement the set of nonmoving modules must remain connected so that the robotic structure does not break apart. Moreover, rotational moves require additional free space: besides the target cell, an additional neighboring cell must be empty in order for a module to rotate around a corner.

Several variants of this model have been studied. Some allow additional movements beyond sliding and rotation, while others restrict the move-set further, for example by requiring additional empty cells around rotating modules. In this thesis we consider the original sliding square model with only sliding and rotation moves.

Most previous work on the sliding square model considers reconfiguration planning in a centralized setting, where a global controller computes a sequence of moves that transforms an initial configuration into a desired target shape. In contrast, this thesis investigates the model in a distributed setting in which modules operate autonomously based only on local information. As in the hybrid model, we assume minimal capabilities with respect to memory, sensing, and communication. Within this setting, we study the problem of transforming arbitrary connected configurations into a rhombus shape.

Thesis Overview

The first part of this thesis studies the hybrid model for programmable matter. Chapter 2 introduces the model formally, reviews related work, and summarizes several concepts that recur throughout this part of the thesis. The following two chapters investigate reconfiguration problems in this model. In Chapter 3, we prove a fundamental limitation of the three-dimensional hybrid model: it is impossible to reconfigure all initial tile configurations if the agent does not initially carry a tile. Chapter 4 studies the problem of efficiently transforming an arbitrary initial configuration into one with properties that facilitate further reconfiguration. In particular, the resulting structure has small diameter, is easy to navigate, and contains tiles that can be removed without violating connectivity. Finally, Chapter 5 addresses the coating problem. Given a static object and a depot of tiles at some location on its surface, the goal is to cover the entire surface with a thin layer of tiles.

The second part of this thesis studies the sliding square model in a distributed setting. Similar to the first part, Chapter 6 introduces the model and surveys related work. Chapter 7 considers the problem of transforming an arbitrary connected configuration of square modules into a rhombus shape, which serves as a compact canonical configuration with minimum diameter and no holes. It presents a fully distributed algorithm for this problem, whose performance is limited by modules moving sequentially. The chapter then explores how parallel movements can accelerate the reconfiguration process by introducing two parallel variants and evaluating their performance through simulations.

In the following, we briefly summarize the results presented in the research chapters and list the publications on which they are based.

Chapter 3: The Necessity of an Initial Tile for 3D Reconfiguration. We begin by investigating which assumptions are necessary for reconfiguration in the three-dimensional hybrid model. Previous work showed that in two dimensions reconfiguration remains possible even though a finite automaton cannot always determine whether a tile is safely removable, that is, whether removing it preserves connectivity of the tile configuration. Instead, certain tiles can always be shifted by one step to a neighboring empty position without breaking connectivity. This property allows two-dimensional algorithms to proceed without requiring the agent to carry a tile initially.

In the three-dimensional setting, however, the situation is different. Previous work assumed that the agent initially carries a tile and suggested that this assumption may be necessary, but no formal proof had been given. We show that there exist connected tile configurations that cannot be reconfigured without such an initial tile. Thus, in contrast to the two-dimensional case, this assumption is indeed necessary for universal reconfiguration in three dimensions.

The results of this chapter have not appeared in prior publications.

Chapter 4: Reconfiguration into Compact Low-Diameter Structures. We next consider how to transform an arbitrary tile configuration into a structure that supports further reconfiguration. As discussed in Chapter 3, a finite-state agent

cannot in general identify tiles whose removal preserves connectivity. To address this challenge, we introduce an intermediate structure called an *icicle*, consisting of a parallelogram-shaped platform with downward-extending lines of tiles. Compared to the line structure used in previous work, the diameter of an icicle is smaller on average, and it contains multiple removable tiles that can be identified locally. This enables a finite-state agent to easily navigate the configuration and locate such tiles. Moreover, during its construction the tiles and the agent stay within a region that tightly bounds the configuration in two dimensions, which makes the approach suitable for space-critical applications. We present a single-agent algorithm that transforms any connected tile configuration into an icicle in $\mathcal{O}(n^3)$ steps.

The results of this chapter are based on the following publications.

Kristian Hinnenthal, David Liedtke, and Christian Scheideler. *Efficient Shape Formation by 3D Hybrid Programmable Matter: An Algorithm for Low Diameter Intermediate Structures*. In: *Theoretical Computer Science*, 1057:115552, 2025 [HLS25].

An earlier conference version appeared in: *Proceedings of the 3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, pp. 15:1–15:20, 2024 [HLS24].

Chapter 5: Coating Surfaces with Tiles. After studying reconfiguration problems in the previous chapters, we now consider the coating problem, where the goal is to cover the surface of some object with a thin layer of tiles. We first formulate this problem independently of the underlying geometry by considering an abstract surface graph that is a triangulation of a closed 3D surface. Given such a graph and a depot node containing the agent and tiles, the goal is to move a tile from the depot to every node. For simplicity, we assume that the depot contains a sufficient supply of tiles. In practice, this depot could correspond to the attachment point of a tile configuration from which the agent can easily retrieve tiles, such as the icicle structure introduced in Chapter 4.

We present a single-agent algorithm that solves this problem for surface graphs whose degree is bounded by a small constant. The algorithm requires $\mathcal{O}(n^2)$ steps, which is worst-case optimal even compared to an agent with global knowledge and unrestricted memory.

We then show how this algorithm can be applied to the geometry of the three-dimensional hybrid model. While surface graphs arising in this model may have larger degree, the algorithm applies directly to a restricted class of objects whose surface graphs satisfy the required degree bound. To enable the algorithm on all surfaces in the 3D hybrid model, we slightly relax the model by allowing multiple distinguishable types of tiles. This additional information allows us to emulate a suitable virtual surface graph on which the algorithm can be executed.

The results of this chapter are based on the following publication.

Irina Kostitsyna, David Liedtke, and Christian Scheideler. *Universal Coating by 3D Hybrid Programmable Matter*. In: *Structural Information and Communication Complexity (SIROCCO)*, pp. 384–401, 2024 [KLS24].

Chapter 7: Distributed Rhombus Formation. We study the problem of transforming an arbitrary connected configuration of square modules into a rhombus shape, which serves as a compact canonical configuration with minimum diameter and no holes. We present a fully distributed algorithm that solves this problem under minimal assumptions: the modules have constant memory, communicate only with immediate neighbors, and initially share a designated leader and a common compass. To the best of our knowledge, this is the only distributed reconfiguration algorithm for the classical sliding square model. The only other distributed result considers a relaxed variant in which rotating modules are allowed to pass through other modules. Without this relaxation, additional difficulties arise, since there exist configurations in which no module has a connectivity-preserving path to its position in the rhombus. We address this by temporarily relocating modules to suitable intermediate positions.

The algorithm is sequential in nature and solves the problem in $\mathcal{O}(n^2)$ rounds, which is optimal for sequential reconfiguration algorithms. To accelerate the process, we introduce two parallel variants based on a spanning-tree structure that allows modules to determine certain connectivity-preserving moves using only local information. Simulation results show a substantial speedup for the first variant and linear average runtime for the second variant.

The results of this chapter are based on the following publications.

Irina Kostitsyna, David Liedtke, and Christian Scheideler. *Distributed Rhombus Formation of Sliding Squares*. Submitted to *Theoretical Computer Science* (special issue) [KLS26a].

An earlier conference version appeared in: *Proceedings of the 28th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2026)*, pp. 325–342 [KLS26b].

Part I.

Hybrid Programmable Matter

The Hybrid Model for Programmable Matter

This chapter introduces the three-dimensional hybrid model for programmable matter used throughout the first part of this thesis. The model captures systems in which a small number of active agents manipulate a large collection of passive building blocks, called tiles. In this thesis, we focus on the case of a single active agent. The agent moves across the tiles, can pick up a tile, transport it, and place it at another location while preserving connectivity among the tiles.

We assume that tiles are hollow, meaning that the agent can move both on top of tiles and through them. In addition, tiles are assumed to be foldable in the sense that an agent carrying a tile can move through other tiles while transporting it.

We begin by introducing basic graph-theoretic notation and terminology that will be used throughout this thesis. We then present the formal definition of the hybrid model, including the underlying graph, the capabilities of the agent, and its constraints. Next, we define the problems studied in the following chapters. Finally, we review related work on programmable matter systems and previous work using the hybrid model.

2.1. Preliminaries

We start by introducing recurring definitions from basic graph theory and the terminology used throughout this thesis.

Let $G = (V, E)$ be an undirected graph with node set V and edge set E . A *path* P between two nodes $v, w \in V$ is a sequence of nodes

$$P = (v = u_0, \dots, u_k = w)$$

such that for each i with $0 \leq i \leq k - 1$, there is an edge $\{u_i, u_{i+1}\} \in E$. We call P *simple* if all nodes in P are pairwise distinct, and we call P a *cycle* if $k \geq 3$ and $\{u_k, u_0\} \in E$. Its *length* is k , that is, the number of edges traversed by the path. A *shortest path* between v and w is a path between v and w of minimum length. For indices $0 \leq i \leq j \leq k$, we call $(u_i, \dots, u_j)$ a *subpath* of P . If $i = 0$, then this subpath is a *prefix* of P . If $j = k$, then it is a *suffix* of P .

Remark 2.1. For simplicity, we slightly abuse notation and treat tuples as sets when referring to their contents, e.g., we write $u \in P$ if node u appears in path P .

The *distance* between v and w , denoted by $d_G(v, w)$, is the length of a shortest path between v and w in G . The *diameter* $D(G)$ of graph G is the maximum distance between any two nodes $v, w \in V$, that is,

$$D(G) = \max_{v, w \in V} d_G(v, w).$$

For a node $v \in V$, the *neighborhood* is defined as

$$N_G(v) = \{u \in V \mid \{v, u\} \in E\},$$

Note that the neighborhood of v does not include v itself.

The *degree* of v is $\deg_G(v) = |N_G(v)|$, and the *degree* $\Delta(G)$ of graph G is the maximum degree of any node in G , that is,

$$\Delta(G) = \max_{v \in V} \deg_G(v).$$

We denote by $G|_U$ the subgraph of G induced by a subset of nodes $U \subseteq V$, that is,

$$G|_U = (U, \{\{v, w\} \in E \mid v, w \in U\}).$$

Remark 2.2. For a subset $U \subseteq V$, we write $d_U(v, w)$, $N_U(v)$, and $\deg_U(v)$ as shorthand for $d_{G|_U}(v, w)$, $N_{G|_U}(v)$, and $\deg_{G|_U}(v)$. For ease of notation, we may omit subscripts if the underlying graph is clear from context and no ambiguity arises.

2.2. Model Definition

In the *3D hybrid model*, we consider a single active agent r with limited sensing and computational power that operates on a finite set of passive *tiles* positioned at nodes of some underlying graph G with a fixed embedding in three-dimensional space. We define the graph and its embedding in Section 2.2.1, the agent model in Section 2.2.2, and model constraints and additional terminology in Section 2.2.3.

2.2.1. Underlying Graph

The face-centered cubic (FCC) lattice is a three-dimensional lattice whose unit cell is the cube. Within each unit cell, lattice points are located at the eight corners of the cube and at the center of each of its six faces (see Figure 2.1a).

The full lattice is obtained by infinitely repeating this unit cell in all directions. When the cubes are arranged in this way, lattice points are shared between neighboring unit cells. Each face centered lattice point is shared by two neighboring cubes and therefore contributes one half to each unit cell, while each corner lattice point is shared by eight adjacent cubes and contributes one eighth (see Figures 2.1c and 2.1d).

Choosing cubes of side length two and placing them such that one lattice point coincides with the origin yields lattice points with integer coordinates. Moreover, these points are precisely those integer grid points whose coordinate sum is even.

Definition 2.3 (FCC lattice graph). *The FCC lattice is the set of points $L_{\text{FCC}} = \{(x, y, z) \in \mathbb{Z}^3 \mid x + y + z \text{ is even}\}$. The FCC lattice graph is the infinite graph $G = (V, E)$ defined as follows. The node set is $V = L_{\text{FCC}}$. Two nodes $u, v \in V$ are connected by an edge $\{u, v\} \in E$ if and only if their Euclidean distance is $\sqrt{2}$.*

Since the nodes of G are points in $\mathbb{R}^3$, the coordinate representation of L_{FCC} directly induces an embedding of G in three-dimensional Euclidean space.

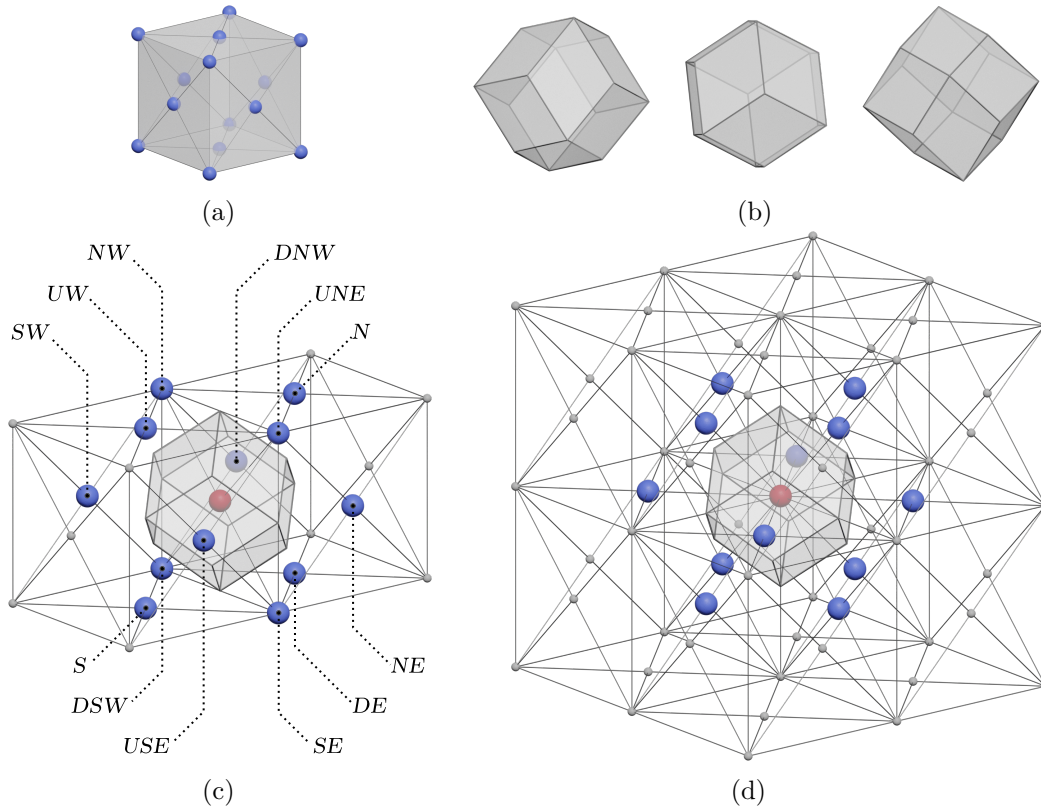


Figure 2.1: (a) The unit cell of the FCC lattice. Lattice points occur at the eight cube corners and at the centers of its six faces. (b) The Voronoi cell of the FCC lattice, a rhombic dodecahedron, shown in different orientations to illustrate its geometry. (c) A face-centered lattice point together with its twelve nearest neighbors. In the FCC lattice graph, these neighbors are adjacent to the central point, and the labels indicate the corresponding compass directions. The Voronoi cell is shown for geometric context. (d) The same neighborhood structure for a corner lattice point.

Each lattice point in L_{FCC} has twelve nearest neighbors at Euclidean distance exactly $\sqrt{2}$. Consequently, each node in G is adjacent to twelve other nodes.

We describe the relative positions of these neighbors by the twelve compass directions UNE , UW , USE , N , NW , SW , S , SE , NE , DNW , DSW , and DE ; see Figure 2.1c.

Observation 2.4. *In the embedding of G induced by the coordinate representation of L_{FCC} , the twelve compass directions correspond to the following vectors:*

$$\begin{array}{lll}
 \overrightarrow{UNE} = (1, 1, 0) & \overrightarrow{UW} = (0, 1, 1) & \overrightarrow{USE} = (1, 0, 1) \\
 \overrightarrow{N} = (0, 1, -1) & \overrightarrow{NW} = (-1, 1, 0) & \overrightarrow{SW} = (-1, 0, 1) \\
 \overrightarrow{S} = (0, -1, 1) & \overrightarrow{SE} = (1, -1, 0) & \overrightarrow{NE} = (1, 0, -1) \\
 \overrightarrow{DNW} = (-1, 0, -1) & \overrightarrow{DSW} = (-1, -1, 0) & \overrightarrow{DE} = (0, -1, -1)
 \end{array}$$

For a node $u \in V$ and a compass direction $\vec{d}$, we denote by $u + \vec{d}$ the node obtained by adding the corresponding direction vector $\vec{d}$ to the coordinates of u . Furthermore, we write $-\vec{d}$ for the opposite compass direction, for example $-\text{UNE} = \text{DSW}$.

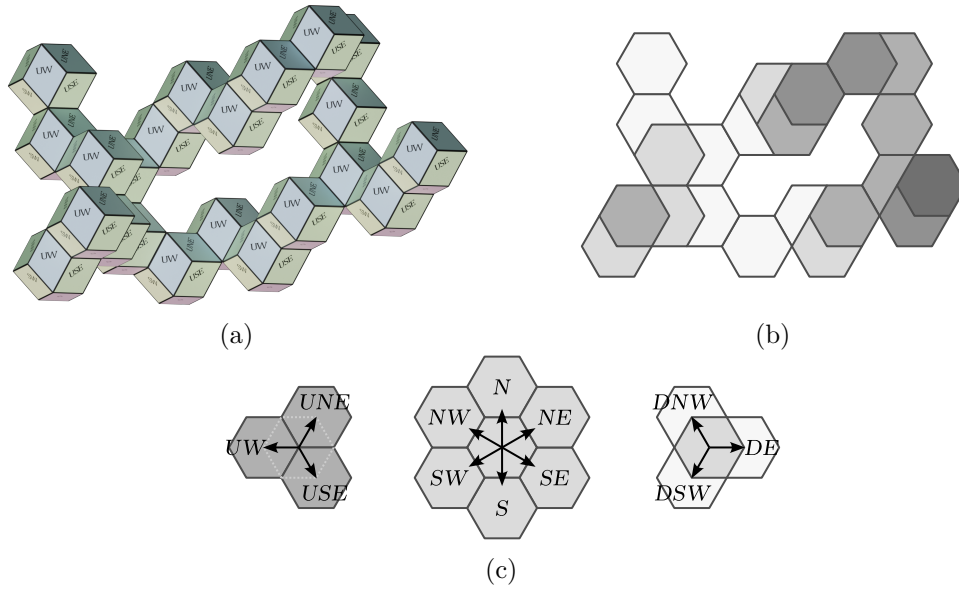


Figure 2.2: (a) An example set of tiles in three-dimensional view. The faces of tiles are labeled by their compass directions. (b) The same set of tiles as a stack of two-dimensional hexagonal layers. Tiles that are closer to the viewer, that is, further upward, are shown in a darker shade of grey, while tiles below are shown in a lighter shade. (c) Illustration of the compass directions in the stacked representation.

In the three-dimensional hybrid model, we consider a finite set of tiles that have the shape of rhombic dodecahedra¹, that is, polyhedra with 12 congruent rhombic faces (see Figure 2.1b). This shape arises naturally as the cells in the Voronoi tessellation of the FCC lattice. Consequently, rhombic dodecahedra fully tessellate three-dimensional space without gaps or overlaps, and two tiles are adjacent in the lattice graph if and only if they share a common face.

Tiles are passive in the sense that they cannot perform any computation or movement on their own. A node $v \in V$ is *tiled* if a tile is positioned at v ; otherwise v is *empty*. Each node can hold at most one tile, and each tile occupies at most one node at any time.

In the two-dimensional variant of the model, tiles are hexagons positioned at the nodes of the infinite triangular lattice. The FCC lattice graph can be decomposed into layers that each form the infinite triangular lattice. This allows us to depict sets of tiles as stacks of two-dimensional hexagonal layers. Figure 2.2 shows an example set of tiles in three-dimensional view and in the corresponding stacked representation. We use both representations interchangeably, whichever is more suitable for visualization.

2.2.2. Agent Model

The agent r is the only active entity in this model. It has strictly limited sensing and computational capabilities and can manipulate passive tiles by picking up a tile,

¹Readers inclined to tinker can find a cut-out template for this shape at the end of this thesis.

transporting it, and placing it at another location. In particular, we assume an agent with the computational capabilities of a deterministic finite automaton that performs discrete *Look-Compute-Move* cycles.

In the *look*-phase, the agent observes whether its current position $p \in V$ and the twelve nodes adjacent to p are tiled or empty. The agent is equipped with a compass that allows it to distinguish the relative positions of its neighbors using the twelve compass directions introduced in Section 2.2.1. Its initial orientation and chirality may be arbitrary, but they remain fixed throughout the execution. For ease of presentation, our algorithms and their analysis are described according to the agent's local view; that is, we do not distinguish between local and global compass directions.

Based on the information gathered in the look phase, the agent determines its next state transition according to the finite automaton in the *compute*-phase.

In the *move*-phase, the agent performs an *action* corresponding to the previously determined state transition. It either (i) moves to an empty or tiled node adjacent to p , (ii) places a tile at p if $p \notin \mathcal{T}$ and r carries a tile, (iii) picks up a tile from p if $p \in \mathcal{T}$ and r carries no tile, or (iv) terminates. The agent can carry at most one tile at a time and during actions (ii) and (iii) the agent loses and gains a tile, respectively. The agent may occupy the same node as a tile; that is, nodes may simultaneously contain a tile and the agent. It is worth noting that we allow the agent to move through tiles while carrying one.

Formally, the agent is a deterministic finite automaton with transition function

$$Q \times \{0, 1\}^{14} \rightarrow Q \times \{\text{move}_{UNE}, \dots, \text{move}_{DE}, \text{place}, \text{lift}, \text{terminate}\},$$

where Q denotes the set of states. An element of $\{0, 1\}^{14}$ encodes the presence of tiles at the agent's current position and its twelve adjacent nodes, as well as whether the agent currently carries a tile. The elements of $\{\text{move}_{UNE}, \dots, \text{move}_{DE}, \text{place}, \text{lift}, \text{terminate}\}$ correspond to the actions (i)–(iv) performed by the agent during the move phase.

This formalization is useful in the impossibility proof presented in Chapter 3. For ease of presentation, however, the algorithms in later chapters are described at a higher level of abstraction using textual descriptions and pseudocode. Any algorithm whose branching decisions depend only on the information captured by $\{0, 1\}^{14}$ can be translated into a finite automaton whose states encode the current instruction of the pseudocode. Additionally, a constant number of variables with constant-size domains can be incorporated into the set of states Q . Therefore, any algorithm satisfying these conditions can be realized by a finite automaton.

2.2.3. Constraints and Terminology

Let $\mathcal{T} \subseteq V$ denote the set of nodes currently occupied by tiles. A *configuration* $C = (\mathcal{T}, p)$ consists of the set $\mathcal{T}$ together with the position $p \in V$ of the agent.

Definition 2.5. A configuration $C = (\mathcal{T}, p)$ is connected if $p \in \mathcal{T}$ and the subgraph $G|_{\mathcal{T}}$ is connected, or if the agent carries a tile and the subgraph $G|_{\mathcal{T} \cup \{p\}}$ is connected.

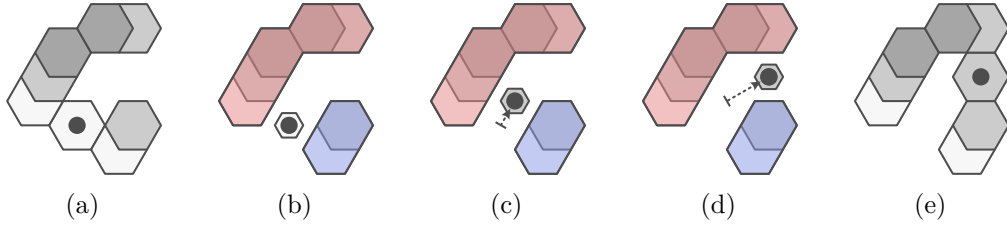


Figure 2.3: Example of a connectivity-preserving sequence of configurations. The agent (grey disk) lifts a tile (b) and then moves in directions *UNE* (c) and *NE* (d). In (b)–(d), the tiled nodes form two connected components, but connectivity is preserved since the agent carries a tile and remains adjacent to both components. After placing the tile (e), the tiled nodes form a single connected component again.

All algorithms considered in this thesis must maintain connectivity throughout their execution; that is, we assume that the initial configuration is connected and every intermediate configuration must remain connected according to Definition 2.5. Intuitively, the agent may only move on or adjacent to tiles. The set of tiled nodes may temporarily become disconnected, provided that a tile carried by the agent maintains connectivity between the components (see Figure 2.3 for an example). From a practical standpoint, this constraint prevents the agent and tiles from drifting apart, for example in liquid or low-gravity environments.

We assume that the agent is initially positioned at a tiled node, since otherwise no valid action may be available. Additionally, we assume that the agent initially carries a tile. As shown in Chapter 3, this assumption is necessary: without an initial tile, the agent may not find any tile that can be moved without breaking connectivity.

We measure time in discrete *steps*, where each step corresponds to a complete *Look–Compute–Move* cycle of the agent. The running time of an algorithm is measured by the number of steps until the agent terminates.

2.3. Problem Statements

In the first part of this thesis, we study the following three problems in the three dimensional hybrid model.

The first problem is the *initial tile problem*. It concerns the necessity of the assumption that the agent initially carries a tile for shape reconfiguration. More specifically, we ask whether a finite automaton agent that starts in an arbitrary connected configuration and does not initially carry a tile can always terminate on a tile that can be moved to a neighboring empty node without violating connectivity. Tiles with this property are called *safely shiftable*. We show that no algorithm can always terminate on a safely shiftable tile for every connected initial configuration.

The second problem is the *icicle formation problem*. The goal is to transform an arbitrary connected tile configuration into a configuration that forms an icicle. Informally, an icicle consists of a parallelogram of tiles that is uppermost in the sense that none of its tiles has a neighboring tile in direction *UNE*, *UW*, or *USE*, and from which lines of tiles extend in direction *DSW*. Equivalently, all tiles without a

neighboring tile in direction UNE are contained in a (partially filled) parallelogram. We present an algorithm that solves this problem in $\mathcal{O}(n^3)$ steps.

The third problem is the *coating problem*. In this setting, we are given a connected set of nodes, called the object, that is static and cannot be traversed by the agent. The goal is to cover the surface of the object with tiles. More precisely, tiles must be placed on all nodes adjacent to the object that are reachable from the agent’s initial position without passing through the object. For simplicity, we assume that the agent has access to a sufficient supply of tiles at its starting position. In practice, this supply of tiles may correspond to the attachment point of a configuration from which the agent can easily retrieve tiles, such as an icicle. We present an algorithm that solves this problem in $\mathcal{O}(n^2)$ steps, which is worst case optimal.

2.4. Related Work

We briefly review prior work related to the hybrid model for programmable matter and to closely related models. More detailed discussions of problem-specific results appear in the corresponding chapters.

The three-dimensional hybrid model was introduced in [HRS20], where a single active agent transforms an arbitrary connected configuration of tiles into a straight line. While the three-dimensional variant has received relatively little attention so far, a substantial body of work exists for the two-dimensional version of the hybrid model. Gmyr et al. present algorithms that transform connected tile configurations into canonical hole-free structures such as lines, blocks, and trees [Gmy+20]. More recently, Friemel et al. study universal shape reconfiguration under the assumption that the agent can distinguish between target and non-target nodes [FLS26]. They provide a worst-case optimal algorithm for hole-free target shapes and an algorithm for a larger class of target shapes with holes. Other work considers recognition problems for specific geometric structures [Gmy+18] and maintenance problems under dynamic tile failures and multiple agents [NFS23]. Together, these results cover shape reconfiguration, recognition, and maintenance in the hybrid model.

A closely related family of two-dimensional models represents tile configurations as polyominoes, that is, as finite connected sets of square grid cells. In the CADbot model, a finite automaton operates on such a configuration and uses the configuration itself as a form of external memory. This model is stronger than the hybrid model considered here: the agent does not have to carry tiles, but may change the state of grid cells directly, and its movement is not constrained to preserve connectivity. Within this framework, Fekete et al. present procedures for fundamental operations such as bounding, counting, copying, reflecting, rotating, and scaling shapes [Fek+21a]. Subsequent work adapts these procedures to preserve connectivity and complements the theory with simulation and hardware implementations [Nie+20; Fek+22; Abd+20].

There is also related work on polyomino reconfiguration by one or more robots that physically carry and move tiles while maintaining connectivity. In contrast to the hybrid model, these works study the centralized computation of global reconfiguration schedules for both single-agent and multi-agent systems. They include obstacle-aware reconfiguration, cooperative reconfiguration by multiple robots, hardness

results, approximation algorithms, and simulation and hardware evaluations [Gar+22; Gar+24; Bec+25; Gar+25].

Other related work on square-grid tile systems studies construction tasks for deterministic finite automata without connectivity constraints. Czyzowicz et al. consider the problem of building compact nest structures [CDP21], while Kant et al. study the construction of forts, that is, shapes that maximize covered area while minimizing the farthest distance between two tiles [KPM21].

Related ideas have also been explored in hardware. Early implementations include robots such as BILL-E and MOJO, which move on or through lattice structures, as well as material-robot systems that assemble discrete cellular voxels [JC17a; JC17b; Jen+19]. More recent systems include SOLL-E, an inchworm-like robot that traverses the exterior of a lattice structure while carrying and placing modules, and MMIC-I, which moves through the interior of such structures and is equipped with fastening tools [Par+23; For+23]. Gregg et al. combine these ideas in a robotic structural system based on strong and lightweight truss-like tiles, where SOLL-E and MMIC-I jointly reconfigure lattice structures [Gre+24].

The hybrid model arose as a variant of the Amoebot model, introduced by Derakhshandeh et al. [Der+14]. Both models are based on the same underlying graph, but they differ substantially in how motion is realized. In the Amoebot model, the mobile entities, called particles, are all uniform and active. Moreover, particles do not move directly from one node to another, but by sequences of expansions and contractions. As in the hybrid model, connectivity must be preserved throughout the execution, where an expanded particle is treated as occupying two adjacent nodes at the same time.

A wide range of distributed algorithms has been studied in the Amoebot model, including leader election [Der+15b; Day+17], shape reconfiguration [Der+15a; Der+16; DiL+20a; DiL+20b; KPS23], convex hull formation [Day+20], and coating problems [Day+18; Der+17; Tra20]. Related work also studied auxiliary communication structures, recovery, and fault tolerance [KPS22; DiL+18; NS20; KSW22].

Several restricted or extended variants of the Amoebot model have also been considered. On the restrictive side, the SILBOT model assumes oblivious particles without direct communication and has been used to study problems such as leader election, compaction, line formation, scattering, and coating [DAn+20; NPP25; NP24; NP25]. On the other hand, stronger extensions provide additional communication or movement capabilities, including reconfigurable circuits for fast communication or improved coordination mechanisms such as joint movement [Fel+22; PKS25].

A more distant line of work concerns active self-assembly models such as the nubot model. In this model, units on the triangular grid can change state, create or delete neighbors, and move, thereby supporting growth as well as motion. The nubot model has been used to study the assembly of lines, patterns, and geometric shapes [Woo+13; Che+14].

Finally, closely related questions also arise in modular self-reconfigurable robotics. Since this line of work forms the second part of the thesis, we discuss it separately and in more detail in Chapter 6.

The Necessity of an Initial Tile for 3D Reconfiguration

In 2020, Gmyr et al. showed that a single finite automaton cannot, in general, decide whether a given tile is *safely removable*, that is, whether its removal preserves connectivity of the tile set [Gmy+20]. Nevertheless, this limitation does not prevent reconfiguration in two dimensions. Even if a tile cannot be safely removed, there always exists a locally identifiable tile that can still be *safely shifted* by one step. For example, a locally northernmost tile can always be moved one step south, provided that the target position is empty. During this movement, connectivity is preserved since the agent remains adjacent to all neighbors of the moved tile. As a consequence, two-dimensional reconfiguration does not require the agent to initially carry a tile.

In the three-dimensional setting, the situation is different. Assuming that the agent initially carries a tile is useful, since it allows the agent to simulate moving a tile from a node v to a node w : it first places its carried tile at w , then picks up the tile at v , and finally moves to w . Since tiles are uniform and stateless, the resulting configuration is the same as if the tile at v had been moved directly to w . Accordingly, previous work commonly assumes that the agent initially carries a tile, including [HRS20; HLS25]. In particular, [HRS20] claims that reconfiguration without such a tile is impossible. However, no formal proof of this claim has been given so far. In this chapter, we close this gap by proving that an initial tile is indeed necessary for universal reconfiguration in three dimensions.

Formal definitions of safely removable and safely shiftable tiles are given in Section 3.1. In that section, we also show that neither property can, in general, be identified using only local information. The impossibility proof is presented in Section 3.2.

3.1. Safely Removable and Safely Shiftable Tiles

We now formally define the notions of safely removable and safely shiftable tiles.

Definition 3.1. Let $G = (V, E)$ be the FCC lattice graph, let $\mathcal{T} \subseteq V$ be a finite set of tiled nodes, and assume that the induced subgraph $G|_{\mathcal{T}}$ is connected.

- (a) A tile at node $v \in \mathcal{T}$ is *safely removable* if the subgraph $G|_{\mathcal{T} \setminus \{v\}}$ is connected.
- (b) A tile at node $v \in \mathcal{T}$ is *safely shiftable* if there exists a node w adjacent to v such that the subgraph $G|_{(\mathcal{T} \cup \{w\}) \setminus \{v\}}$ is connected.

The properties defined above depend only on the set of occupied nodes and are therefore independent of any tile that may be carried by the agent. In addition, tiles remain fixed on their initial nodes throughout the impossibility proof.

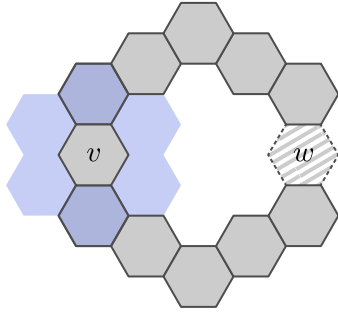
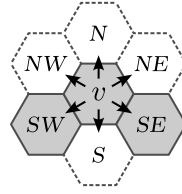
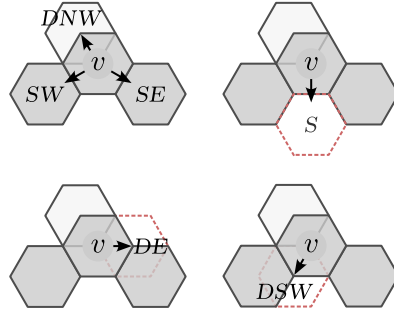


Figure 3.1: Tile v is safely removable if and only if node w is tiled. In both configurations, the local neighborhood of v (highlighted in blue) is identical.



(a)



(b)

Figure 3.2: (a) In two dimensions, a locally northernmost tile v is always safely shiftable. (b) In three dimensions, locally extremal tiles are not necessarily safely shiftable. Tile v is locally UNE -most, and the common neighborhood of v and any two of its tiled neighbors contains a single node (dashed outline) that is not adjacent to the third tiled neighbor of v .

Remark 3.2. Throughout this chapter, we use the terms *tile* and *node* of G interchangeably. For example, we may say that a node $v \in \mathcal{T}$ is safely removable.

We now discuss basic properties of safely removable and safely shiftable tiles. It follows directly from the definition that every safely removable tile is also safely shiftable. Moreover, every configuration in which the set of tiled nodes is connected contains at least one safely removable tile. Indeed, since $G|_{\mathcal{T}}$ is finite and connected, it has a spanning tree. Every tree with at least one node contains a leaf, and removing a leaf does not disconnect the tree. Hence, if v is a leaf of some spanning tree of $G|_{\mathcal{T}}$, then any two nodes in $\mathcal{T} \setminus \{v\}$ remain connected by a path in the spanning tree that does not contain v . Therefore, the subgraph $G|_{\mathcal{T} \setminus \{v\}}$ is connected, and v is safely removable.

However, neither safely removable nor safely shiftable tiles can, in general, be identified using only local information. This limitation already appears in two dimensions. Consider a configuration forming a hexagon of side length at least three, and let $v \in \mathcal{T}$ be a tile that is not a corner of the hexagon; see Figure 3.1. In this configuration, all tiles lie on a simple cycle, so v is safely removable. Now remove a tile w that does not lie on the same side of the hexagon as v . The resulting configuration contains no cycle, so the only safely removable tiles are the two tiles adjacent to w . Hence, v is not safely removable. However, the local neighborhood of v is identical in both configurations.

Despite this limitation, in two dimensions there always exists a safely shiftable tile that can be identified locally, for instance a locally northernmost tile; see Figure 3.2a. Let $v \in \mathcal{T}$ be such a tile. Since it is locally northernmost, v has no neighbors in directions NW, N, or NE. If v is safely removable, there is nothing to show. Otherwise, there are tiles at $v + SW$ and $v + SE$, while the node at $v + S$ is empty. Since $v + S$ is adjacent to both $v + SW$ and $v + SE$, the subgraph $G|_{(\mathcal{T} \cup \{v+s\}) \setminus \{v\}}$ is connected. Hence, v is safely shiftable.

In contrast, in three dimensions, locally extremal tiles are not necessarily safely shiftable. Consider the configuration depicted in Figure 3.2b, consisting of four tiles: a tile at some node v , and one tile in each of the directions DNW , SW , and SE of v . The tile v is locally extremal, as it has no neighbors in any upward direction UNE , UW , or USE , nor in any northern direction NW , N , or NE . Moreover, v and its tiled neighbors have no common neighbor, that is,

$$N(v) \cap N(v + SW) \cap N(v + SE) \cap N(v + DNW) = \emptyset.$$

To see this, consider the common neighborhood of v and any two tiled neighbors of v and observe that it contains exactly one node that is not adjacent to the third tiled neighbor; see Figure 3.2b. For example,

$$N(v) \cap N(v + SE) \cap N(v + DNW) = \{v + DE\},$$

and $v + DE$ is not adjacent to the tiled neighbor $v + SW$. Hence, there is no node $w \in N(v)$ such that $G|_{(\mathcal{T} \cup \{w\}) \setminus \{v\}}$ is connected, and thus v is not safely shiftable.

Since safe removability and shiftable cannot, in general, be identified locally, they are primarily used to argue correctness. In Chapter 4, we therefore introduce stronger properties that can be identified locally and suffice to enable shape reconfiguration in three dimensions.

3.2. Impossibility Proof

In this section, we prove that tile reconfiguration in three dimensions is impossible for a single finite automaton agent that does not initially carry a tile. The proof follows a structure similar to that of Gmyr et al. [Gmy+20], where they show that an automaton which terminates on a safely removable tile in a hexagonal shape must also terminate on some inner tile of a hexagonal spiral, that is, not on one of its two endpoints, thereby contradicting the existence of such an automaton. The hexagon, however, is not suitable for our proof, since its spiral contains a safely shiftable tile at each corner. Instead, we consider tiles arranged in the shape of a truncated octahedron; see Figure 3.3a. Starting from a single corner of the truncated octahedron, we iteratively realize incident edges while contracting the edge lengths by a factor of at least $\frac{1}{2}$ in each iteration, yielding a self-similar polyhedral spiral; see Figure 3.5. After i iterations, there are 2^i endpoints, and only the tiles at these positions are safely shiftable. The main difficulties lie in the construction: we must choose edge lengths that admit a discretization using tiles, while ensuring that tiles belonging to different polyhedral edges are placed sufficiently far apart so that an automaton agent can observe tiles belonging to only one polyhedral edge at a time. We now formalize this intuition and prove that no such automaton can exist.

Theorem 3.3. *There exists no finite automaton A that, when executed by an agent starting on a tiled node of an arbitrary three-dimensional configuration and without initially carrying a tile, can always terminate on a safely shiftable tile.*

Proof. Assume for contradiction that there exists such a finite automaton A . We let an agent execute A on a discretized truncated octahedron and use the resulting execution sequence to construct a self-similar polyhedral spiral in which A terminates on a tile that is not safely shiftable, contradicting the assumption.

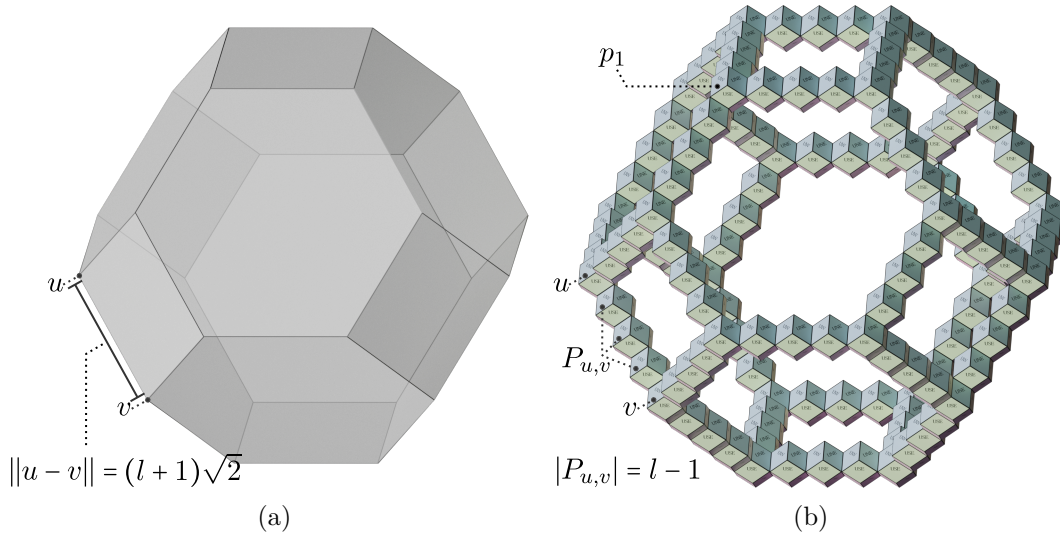


Figure 3.3: (a) A truncated octahedron. The Euclidean distance between adjacent corners u and v is $(l + 1)\sqrt{2}$. (b) The corresponding configuration $(\mathcal{T}^{(l)}, p_1)$. Tiles are placed on all corners and along all edges of the polyhedron. The tiles along the edge $\{u, v\} \in E_{\text{poly}}$ form the discrete edge $P_{u,v}$ of size $l - 1$ (here $l = 4$).

The truncated octahedron. A *truncated octahedron* is the Archimedean solid obtained by truncating all six corners of a regular octahedron; see Figure 3.3a. It has 8 hexagonal faces and 6 square faces, hence 14 faces in total, as well as 24 corners and 36 edges. Each corner is incident to exactly one square face and two hexagonal faces. Consequently, every edge is incident either to two hexagonal faces or to one hexagonal and one square face. We refer to edges of the former type as *hexagonal edges* and to edges of the latter type as *square edges*. This distinction will be used later when arguing that the spiral construction does not self-intersect.

Geometric embedding. Let V_{poly} and E_{poly} denote the corner set and edge set of the truncated octahedron, respectively. Fix a rotation of the polyhedron such that each edge in E_{poly} is parallel to some edge of the embedded FCC lattice graph $G = (V, E)$; see Definition 2.3. Recall that in the embedding of G , each edge has Euclidean length $\sqrt{2}$. Let l be an integer with $l \geq 4$.

We scale and translate the truncated octahedron so that all corners in V_{poly} coincide with nodes in V and every polyhedral edge has Euclidean length $(l + 1)\sqrt{2}$. In particular, for each edge $\{u, v\} \in E_{\text{poly}}$, we have $v = u + (l + 1)\vec{x}$ for some compass direction x ; as an example, $v = u + (l + 1)\vec{SE}$ in Figure 3.3.

Initial configuration. We now define the initial configuration on which the agent executes A . See Figure 3.3 for an illustration.

After scaling and translation, each corner of the truncated octahedron coincides with a node of G . We place a tile on each such node and refer to them as *corner tiles*.

For every polyhedral edge $\{u, v\} \in E_{\text{poly}}$, let $P_{u,v}$ denote the set of internal nodes on the unique shortest path from u to v in G . This path is unique since, by construction,

v can be reached from u by repeatedly taking the same compass direction. Moreover, the path has length l and therefore contains exactly $|P_{u,v}| = l - 1$ internal nodes. Note that $P_{u,v}$ is not empty by our choice $l \geq 4$. We refer to $P_{u,v}$ as the *discrete edge* corresponding to the polyhedral edge $\{u, v\}$, and say that $P_{u,v}$ is *incident* to the corner tiles u and v .

We define the set of tiled nodes of the initial configuration as

$$\mathcal{T}^{(l)} := V_{\text{poly}} \cup \bigcup_{\{u,v\} \in E_{\text{poly}}} P_{u,v}.$$

Among all corner tiles in $\mathcal{T}^{(l)}$, let p_1 be the unique tile that is uppermost and, among those, westernmost, and among those, northernmost. We start the agent at p_1 without carrying a tile and consider the resulting initial configuration $(\mathcal{T}^{(l)}, p_1)$.

Border nodes. We next formalize the nodes at which the agent can switch between discrete edges. Recall that the agent may move through tiled nodes as well as through empty nodes adjacent to tiles. We focus on nodes that are relevant for such transitions. Empty nodes that are not adjacent to any tile cannot be traversed by the agent and will therefore be ignored. Assume that the agent is currently located on a tile of some discrete edge, or at an empty node adjacent to exactly one discrete edge. We define the set of *border nodes* such that, whenever the agent moves to a tile of a different discrete edge, or to an empty node adjacent to it, it must pass through at least one border node.

Definition 3.4. A node $v \in V$ is a border node if one of the following holds:

- (a) v is adjacent to tiles of at least two distinct discrete edges, or
- (b) v is an empty node that is adjacent to a corner tile and to a tile of exactly one discrete edge.

We now determine the location and number of border nodes in the configuration. Each corner tile is a border node, as it is adjacent to exactly three distinct discrete edges. By our choice $l \geq 4$ and the geometry of the truncated octahedron, the distance between any two tiles that lie on distinct discrete edges and are not adjacent to a corner tile is at least 4. In particular, there is no empty node adjacent to two distinct discrete edges that is not within distance 2 of a corner tile. Hence, each border node is adjacent to a corner tile or to some tile adjacent to a corner tile. In particular, each border node can be uniquely associated with a corner tile.

We now determine the set of border nodes associated with a fixed corner tile. Consider a corner tile v with incident discrete edges in directions S , NE , and DNW ; see Figure 3.4. All other cases are analogous up to rotation.

The border nodes adjacent to at least two distinct discrete edges (as defined in Definition 3.4a) are

$$v + N, v + SE, v + DSW, v + DE, \text{ and } v + NE + DNW.$$

Note that $v + NE + DNW$ is the only such node not adjacent to v ; see Figure 3.4c. This is because the discrete edges in directions NE and DNW correspond to a square

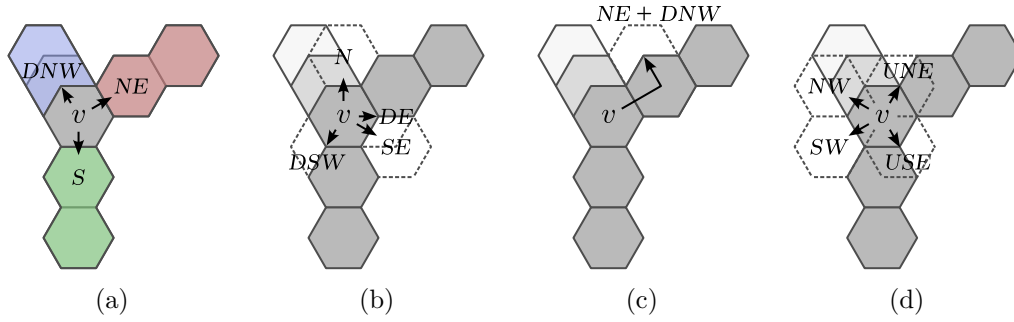


Figure 3.4: Border nodes associated with a corner tile v . (a) The corner tile v and its incident discrete edges; v itself is a border node. (b) Border nodes adjacent to v and to at least two distinct discrete edges. (c) The border node $v + NE + DNW$, the only such node not adjacent to v . (d) Empty border nodes adjacent to v and to exactly one discrete edge. All other cases are analogous up to rotation.

face of the truncated octahedron, whereas the other pairs correspond to hexagonal faces. The border nodes that are empty and adjacent to v and exactly one discrete edge (as defined in Definition 3.4b) are

$$v + NW, v + SW, v + NE, \text{ and } v + SE.$$

All other empty nodes within distance 2 of v are not adjacent to any tile and hence cannot be traversed by the agent.

Including v , the total number of border nodes associated with v is 10. Since each discrete edge contains $l - 1 \geq 3$ tiles, the sets of border nodes associated with distinct corner tiles are pairwise disjoint. As $\mathcal{T}^{(l)}$ contains 24 corner tiles, there are 240 border nodes in total.

Finally, we argue that the agent cannot bypass border nodes. Consider two tiles on distinct discrete edges that are not adjacent to a corner tile. As argued above, their distance is at least 4. Hence, any path between them that does not pass through a border node must contain an internal empty node that is not adjacent to any tile. Such a node cannot be traversed by the agent without violating the connectivity constraint.

As an example, consider two discrete edges corresponding to opposite sides of a square face of the truncated octahedron. Let u and v be tiles at the midpoints of these edges. The shortest path between u and v has length exactly $l \geq 4$ and contains three empty nodes. While the first and third are adjacent to tiles, the second is not, so the agent cannot traverse this path.

Execution of A on $(\mathcal{T}^{(l)}, p_1)$. By assumption, A terminates on every configuration. In particular, when executed on the initial configuration $(\mathcal{T}^{(l)}, p_1)$, it terminates after a finite number T of rounds.

Let Q be the state set of A , and let $s := |Q|$. We describe the execution as a finite sequence of *system states*

$$(p_1, q_1), (p_2, q_2), \dots, (p_T, q_T),$$

where, for each $i \in \{1, \dots, T\}$, the pair (p_i, q_i) denotes the agent's position $p_i \in V$ and automaton state $q_i \in Q$ at the start of round i . Equivalently, round i starts with the agent at position p_i in state q_i and ends with the agent at some position p_{i+1} in some state q_{i+1} , unless $i = T$, in which case the agent terminates during round T .

We partition the sequence of system states into phases that start with the agent at a border node. Let $B \subset V$ denote the set of all border nodes, and let $t_1 < \dots < t_m$ be exactly those indices $i \in \{1, \dots, T\}$ for which $p_i \in B$. Since the agent starts at a corner tile, there exists at least one phase, and the first phase starts in round $t_1 = 1$. For each $j \in \{1, \dots, m-1\}$, the j -th phase consists of the system states of rounds $t_j, t_j + 1, \dots, t_{j+1} - 1$, and the final phase ($j = m$) consists of the system states of rounds $t_m, \dots, T$. In particular, phases may have length 1, e.g., when the agent moves directly between two border nodes or when it changes state without changing its position. Except for the final phase, the last position of a phase must be adjacent to a border node, namely to the first position of the subsequent phase.

Suppose that the execution consists of more than $240s$ phases. Since there are exactly 240 border nodes and the automaton has s states, it follows by the pigeonhole principle that there exist two rounds $i < j$ such that $(p_i, q_i) = (p_j, q_j)$ and p_i is a border node. Since the agent is in the same state and observes the same local neighborhood at the start of rounds i and j , its behavior from these rounds onward is identical. Consequently, $(p_{i+k}, q_{i+k}) = (p_{j+k}, q_{j+k})$ for all $k \geq 0$. In particular, the execution is periodic from round i onward and therefore infinite. This contradicts the assumption that the execution terminates after a finite number of rounds. Hence, the execution consists of at most $240s$ phases.

Let $(S_1, \dots, S_k)$ denote the sequence of system states at the starts of the phases, that is, $S_j := (p_{t_j}, q_{t_j})$ for $j \in \{1, \dots, k\}$. We call $(S_1, \dots, S_k)$ the *border sequence* of the execution of A on $(\mathcal{T}^{(l)}, p_1)$. By the previous argument, we have $k \leq 240s$. Moreover, each S_j consists of a border node and an automaton state, and hence there are at most $240s$ possible values for each S_j . It follows that there are at most $240s^{240s}$, that is, finitely many, distinct border sequences overall. Since there are infinitely many integers $l \geq 4$ but only finitely many possible border sequences, it follows by the pigeonhole principle that there exists an infinite set $L \subseteq \mathbb{N}$ such that the execution of A yields the same border sequence for all $l \in L$.

Self-similar polyhedral spiral construction. In the following, we construct a self-similar polyhedral spiral by starting from a single corner tile and iteratively realizing discrete edges and new corner tiles with edge lengths drawn from the set L .

Let l_1 be the smallest value in L satisfying $l_1 \geq 4$. Let $l_1, l_2, \dots, l_{240s+1}$ be a sequence of integers drawn from L , that is, $l_i \in L$ for all $i \in \{1, \dots, 240s+1\}$, such that $l_{i+1} > 2l_i + 1$ for all $i \in \{1, \dots, 240s\}$. Such a sequence exists since L is infinite.

By definition of L , the execution of A on each of the configurations

$$(\mathcal{T}^{(l_1)}, p_1^{(l_1)}), (\mathcal{T}^{(l_2)}, p_1^{(l_2)}), \dots, (\mathcal{T}^{(l_{240s+1})}, p_1^{(l_{240s+1})})$$

yields the same border sequence. Here, for each $i \in \{1, \dots, 240s+1\}$, the starting position $p_1^{(l_i)}$ denotes the unique corner tile of $\mathcal{T}^{(l_i)}$ that is uppermost and, among those, westernmost, and among those, northernmost.

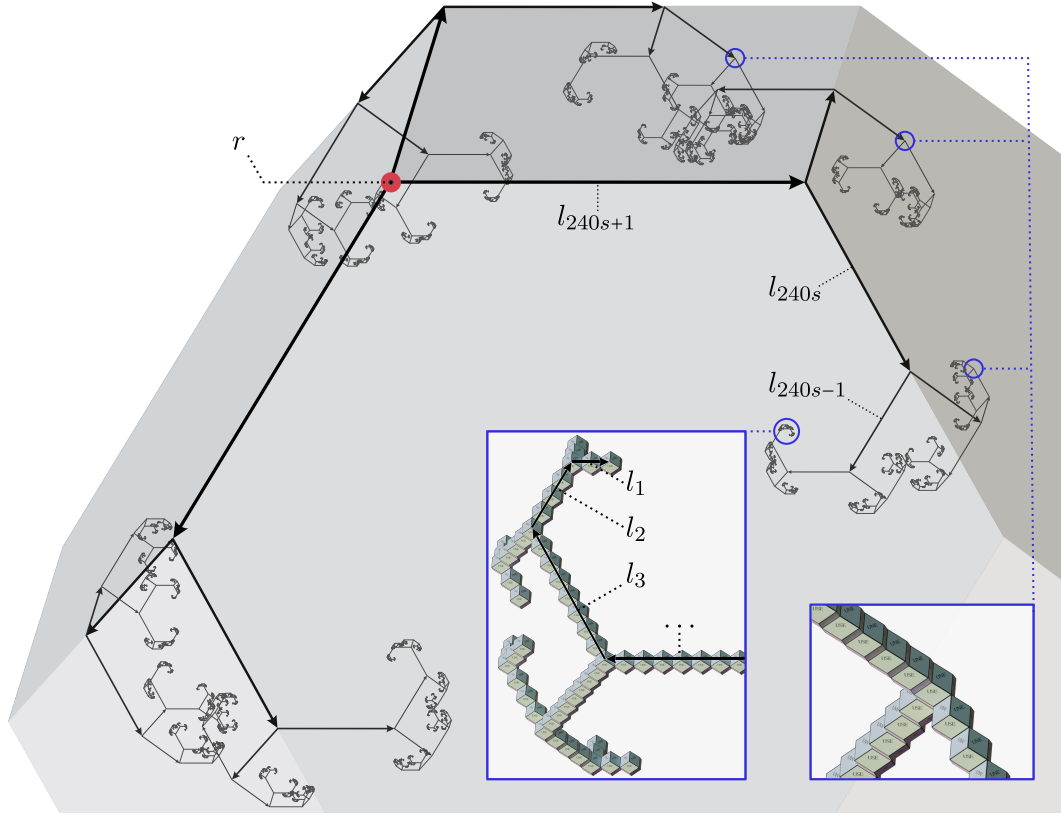


Figure 3.5: The self-similar polyhedral spiral. Lines represent discrete edges, and branching points correspond to corner tiles. Arrows indicate the parent-child relationship between corner tiles and are labeled by the distance between them (only shown for selected pairs). Corner types repeat; for example, the three circled corner tiles on the right have the same type as shown in the closeup on the bottom right.

We define a *corner type* to be a set of three compass directions. For a corner tile u in one of the configurations $\mathcal{T}^{(l_i)}$, let $D(u)$ denote the set of compass directions of its three incident discrete edges. For instance, by our choice of the starting position $p_1^{(l_i)}$, we have $D(p_1^{(l_i)}) = \{NE, DNW, S\}$ for all $i \in \{1, \dots, 240s + 1\}$. Note that within a single configuration $\mathcal{T}^{(l_i)}$, no two distinct corner tiles share the same corner type.

We now construct the self-similar polyhedral spiral. See Figure 3.5 for an illustration. For simplicity, we continue to use the notions of discrete edges for straight-line tile paths and corner tiles for their endpoints, even though the configuration no longer resembles a polyhedron.

We start with a single corner tile r (the *root*) and set its corner type to $D(r) := D(p_1^{(l_i)}) = \{NE, DNW, S\}$. For $240s + 1$ iterations, indexed by $i = 240s + 1, 240s, \dots, 1$, we perform the following steps. In iteration i , we consider all corner tiles u that currently have no incident discrete edge if $i = 240s + 1$, or exactly one incident discrete edge if $i < 240s + 1$. For every such corner tile u and for each direction $x \in D(u)$ not yet used by an incident discrete edge, we attach a new discrete edge in direction x consisting of $l_i - 1$ tiles and place a new corner tile v at its far endpoint.

The corner type of v is chosen so that the local continuation in direction x matches that in $\mathcal{T}^{(l_i)}$. To be precise, let w be the unique corner tile in $\mathcal{T}^{(l_i)}$ incident to the discrete edge in direction x of the corner tile with type $D(u)$. We set $D(v) := D(w)$. The newly placed corner tile v is called a *child* of u , and u is called its *parent*. This operation yields a straight line of $l_i + 1$ tiles in total, and the distance between u and v is l_i .

Let $\mathcal{T}_{\text{spiral}}$ denote the resulting set of tiles.

Properties of $\mathcal{T}_{\text{spiral}}$. With the following claims, we show that distinct discrete edges in $\mathcal{T}_{\text{spiral}}$ are sufficiently far apart in G . Consequently, every discrete edge in $\mathcal{T}_{\text{spiral}}$ has a corresponding discrete edge in some configuration $\mathcal{T}^{(l_i)}$ with the same local structure, in the sense that their two-neighborhoods are identical. Therefore, an agent cannot distinguish between such corresponding discrete edges, whether it is located on a tile of the edge or on an adjacent empty node. Hence, the self-similar polyhedral spiral $\mathcal{T}_{\text{spiral}}$ is locally indistinguishable from the truncated octahedron configurations $\mathcal{T}^{(l_i)}$.

We call a corner tile v a *descendant* of a corner tile u if v can be reached from u by repeatedly following child relations; in this case, u is an *ancestor* of v .

Claim 1. *Let $u \in \mathcal{T}_{\text{spiral}}$ be a corner tile that is added in iteration i of the construction. Then every descendant v of u satisfies $d(u, v) < l_i - 1$.*

Proof. Let v be a descendant of u . Then there exists a sequence of corner tiles $u = v_0, v_1, \dots, v_t = v$ such that v_{j+1} is a child of v_j for all $j \in \{0, \dots, t-1\}$. Since u is added in iteration i , its children are placed at distance l_{i-1} from u , the children of those tiles are placed at distance l_{i-2} , and so forth. Thus, $d(v_j, v_{j+1}) = l_{i-1-j}$ for all $j \in \{0, \dots, t-1\}$, and hence

$$d(u, v) \leq \sum_{j=0}^{t-1} d(v_j, v_{j+1}) = \sum_{j=0}^{t-1} l_{i-1-j}.$$

By the choice of the sequence $l_1, \dots, l_{240s+1}$, we have $l_{k+1} > 2l_k + 1$ for all $k \in \{1, \dots, 240s\}$, and in particular $l_k < l_{k+1}/2$ for all such k . It follows that $l_{i-1-j} \leq 2^{-j}l_{i-1}$ for all $j \in \{0, \dots, t-1\}$. Hence,

$$d(u, v) \leq l_{i-1} \sum_{j=0}^{t-1} 2^{-j} \leq l_{i-1} \sum_{j=0}^{\infty} 2^{-j} = 2l_{i-1} < l_i - 1,$$

which proves the claim. $\triangleleft$

Claim 2. *Let u and v be tiles that lie on two distinct discrete edges of $\mathcal{T}_{\text{spiral}}$. Then either $d(u, v) > 2$, or u and v are adjacent to the same corner tile.*

Proof. Let u_p be the corner tile incident to the discrete edge containing u whose parent is not incident to that discrete edge, and define v_p analogously for v ; see Figures 3.6 and 3.7. If u and v are adjacent to the same corner tile, we are done, so assume this is not the case.

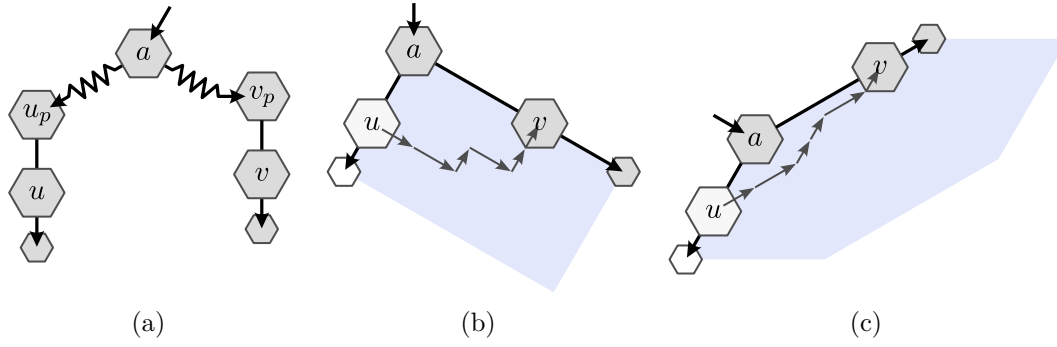


Figure 3.6: Notation used in the proof of Claim 2 and illustration of Case 1. (a) Tiles u and v lie on distinct discrete edges with corner tiles u_p and v_p ; straight arrows indicate parent-child relations, and wiggly arrows indicate ancestor relations. (b,c) Case 1 ($a = u_p = v_p$): the two discrete edges correspond to two sides of a square (b) or hexagon (c) with common corner a ; grey arrows indicate an example of a shortest path from u to v .

Let a be the least common ancestor of u_p and v_p , that is, the common ancestor of u_p and v_p that is placed last in the construction of $\mathcal{T}_{\text{spiral}}$, and let i be the iteration in which a is placed.

Case 1: $a = u_p = v_p$. Then the two discrete edges containing u and v are distinct, contain the same number of tiles, and are incident to the same corner tile a ; see Figures 3.6b and 3.6c. Hence, they correspond to two sides of a square or a hexagon with common corner a . Any shortest path from u to v consists only of steps parallel to these two sides, and the number of steps in each direction equals the distance to a . Hence,

$$d(u, v) = d(v, a) + d(a, u).$$

By assumption, at least one of u and v is not adjacent to a , and thus its distance to a is at least 2. The distance of the other tile to a is at least 1. Hence, $d(u, v) > 2$. We will repeatedly apply this additivity of distances via a common corner tile in the following cases; note that, although we argued it for tiles on the discrete edges, the same argument applies when u and v are corner tiles, in particular the children of a .

Case 2: exactly one of u_p and v_p equals a . Assume $a = u_p$ and $a \neq v_p$; the other case is symmetric. Let c be the child of a incident to the discrete edge containing u , and let d be the unique child of a that is an ancestor of or equal to v_p ; see Figure 3.7.

Case 2a: $c \neq d$. See Figure 3.7a. By Case 1 applied to the corner tiles c and d with common parent a , we obtain

$$d(c, d) = d(c, a) + d(a, d) = l_{i-1} + l_{i-1} = 2l_{i-1}.$$

Since u lies on the discrete edge between a and c , we have $d(c, u) \leq l_{i-1} - 1$. Since v lies on a straight line between two descendants of d , Claim 1, applied to the corner tile d added in iteration $i - 1$, implies that $d(v, d) \leq l_{i-1} - 2 < l_{i-1} - 1$.

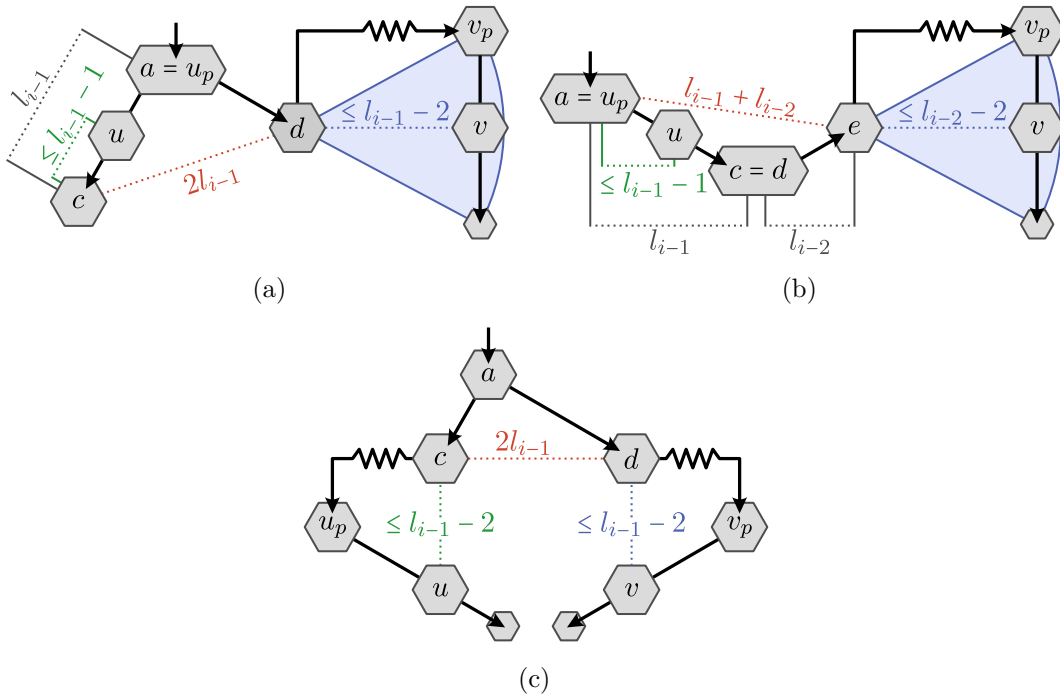


Figure 3.7: Illustration of Cases 2 and 3 in the proof of Claim 2. Dashed lines indicate distances between tiles. (a) Case 2a ($a = u_p$, $c \neq d$): c and d are distinct children of a , where c is incident to the discrete edge containing u , and d is an ancestor of or equal to v_p . (b) Case 2b ($a = u_p$, $c = d$): the child c of a incident to the discrete edge containing u is also an ancestor of or equal to v_p ; distances are measured via the child e of c that is an ancestor of or equal to v_p . (c) Case 3 ($a \neq u_p$ and $a \neq v_p$): c and d are distinct children of a such that c is an ancestor of or equal to u_p and d is an ancestor of or equal to v_p .

Using the triangle inequality, we obtain

$$\begin{aligned}
 d(c, u) + d(u, v) + d(v, d) &\geq d(c, d) \\
 d(u, v) &\geq d(c, d) - d(c, u) - d(v, d) \\
 &\geq 2l_{i-1} - (l_{i-1} - 1) - (l_{i-1} - 2) \\
 &= 3 > 2.
 \end{aligned}$$

Case 2b: $c = d$. Then c is equal to v_p or an ancestor of v_p ; see Figure 3.7b. If $c = v_p$, then u lies on the discrete edge between a and c whereas v lies on a distinct discrete edge incident to c . By assumption, u and v are not adjacent to the same corner tile, so at least one of them is not adjacent to c . Thus, the case is analogous to Case 1.

Otherwise, c is an ancestor of v_p . Let e be the child of c that is an ancestor of or equal to v_p . Then e is added in iteration $i - 2$. Since v lies on a straight line between two descendants of e , Claim 1 applied to e yields $d(v, e) \leq l_{i-2} - 2$. Moreover, u lies on the discrete edge between a and c , and hence $d(a, u) \leq l_{i-1} - 1$. By Case 1

applied to the corner tiles a and e with intermediate corner tile c , we obtain

$$d(a, e) = d(a, c) + d(c, e) = l_{i-1} + l_{i-2}.$$

Using the triangle inequality, we obtain

$$\begin{aligned} d(a, u) + d(u, v) + d(v, e) &\geq d(a, e) \\ d(u, v) &\geq d(a, e) - d(a, u) - d(v, e) \\ &\geq (l_{i-1} + l_{i-2}) - (l_{i-1} - 1) - (l_{i-2} - 2) \\ &= 3 > 2. \end{aligned}$$

Case 3: $a \neq u_p$ and $a \neq v_p$. Let c be the unique child of a that is an ancestor of or equal to u_p , and let d be the unique child of a that is an ancestor of or equal to v_p ; see Figure 3.7c. Then $c \neq d$, and as in Case 2a we have $d(c, d) = 2l_{i-1}$. Applying Claim 1 to c and d (both added in iteration $i - 1$) yields $d(c, u) \leq l_{i-1} - 2$ and $d(v, d) \leq l_{i-1} - 2$. Using the triangle inequality, we obtain

$$\begin{aligned} d(c, u) + d(u, v) + d(v, d) &\geq d(c, d) \\ d(u, v) &\geq d(c, d) - d(c, u) - d(v, d) \\ &\geq 2l_{i-1} - 2(l_{i-1} - 2) \\ &= 4 > 2. \end{aligned}$$

In all cases we have $d(u, v) > 2$, which concludes the proof of the claim. $\triangleleft$

As a consequence of Claim 2, distinct discrete edges meet only at corner tiles. Hence, $\mathcal{T}_{\text{spiral}}$ induces a tree in G .

Note that in the analysis of the truncated octahedron, we require that the distance between any two tiles that lie on distinct discrete edges and are not adjacent to any corner tile is at least 4. This ensures that an agent cannot bypass border nodes without violating the connectivity constraint. In the self-similar polyhedral spiral, the weaker bound established in Claim 2 suffices. Although shorter paths between distinct discrete edges may exist, the agent cannot distinguish locally identical configurations and therefore follows the same execution as in the truncated octahedron.

Execution of A on $(\mathcal{T}_{\text{spiral}}, r)$. We first observe that, analogous to the truncated octahedron configurations, every border node in $\mathcal{T}_{\text{spiral}}$ can be uniquely associated with a corner tile that has at least one child. Indeed, corner tiles without children are incident to only one discrete edge and thus have no associated border nodes. By Claim 2, there is no empty node adjacent to two distinct discrete edges that is not within distance 2 of a corner tile. Hence, each border node lies in the neighborhood of a corner tile or of a tile adjacent to it, and therefore can be uniquely associated with this corner tile. Since $\mathcal{T}_{\text{spiral}}$ does not necessarily contain the same number of corner tiles as any of the $\mathcal{T}^{(l_i)}$, we define a mapping φ from the border nodes of $\mathcal{T}_{\text{spiral}}$ to the border nodes of $\mathcal{T}^{(l_i)}$ as follows.

Fix some $i \in \{1, \dots, 240s + 1\}$. For every corner tile $u \in \mathcal{T}_{\text{spiral}}$ with at least one child, let $v \in \mathcal{T}^{(l_i)}$ be the unique corner tile with $D(v) = D(u)$, which exists by construction since no two corner tiles in $\mathcal{T}^{(l_i)}$ share the same type. We set $\varphi(u) := v$.

Now let w be a border node associated with u and distinct from u . We define $\varphi(w)$ to be the unique border node associated with v that lies in the same relative position with respect to v as w does with respect to u .

Finally, we extend φ to system states by defining $\varphi(p, q) := (\varphi(p), q)$ for every border node p and every automaton state $q \in Q$.

Let $S = (S_1, \dots, S_k)$ be the common border sequence of the executions of A on $(\mathcal{T}^{(l_i)}, p_1^{(l_i)})$ for $i \in \{1, \dots, 240s + 1\}$. Analogously, let $S' = (S'_1, \dots, S'_{k'})$ be the border sequence of the execution of A on $(\mathcal{T}_{\text{spiral}}, r)$.

Claim 3. $\varphi(S'_i) = S_i$ for all $i \in \{1, \dots, k'\}$. In particular, $k' = k$.

Proof. We prove $\varphi(S'_i) = S_i$ for all $i \in \{1, \dots, k'\}$ by induction over i .

Base case. We have $S'_1 = (r, q_1)$, where q_1 is the initial state of A . By construction, $D(r) = D(p_1^{(l_i)})$ and therefore $\varphi(r) = p_1^{(l_i)}$. Hence, $\varphi(S'_1) = (p_1^{(l_i)}, q_1) = S_1$.

Induction step. Let $i \in \{1, \dots, k' - 1\}$ and assume that $\varphi(S'_i) = S_i$. Write $S'_i = (p', q)$ and $S_i = (p, q)$.

The agent is in the same state q at $p' \in \mathcal{T}_{\text{spiral}}$ and at $p = \varphi(p') \in \mathcal{T}^{(l_i)}$. By construction of φ , the local neighborhoods at p' and p are identical. Hence, the first transition of phase i is identical in both executions.

Let $(p'_{\text{next}}, q_{\text{next}})$ and $(p_{\text{next}}, q_{\text{next}})$ be the system states obtained after this first transition in $\mathcal{T}_{\text{spiral}}$ and $\mathcal{T}^{(l_i)}$, respectively.

If p'_{next} is a border node, then so is p_{next} , and the next phase starts at these nodes. Thus $\varphi(S'_{i+1}) = S_{i+1}$.

Otherwise, let u be the corner tile associated with the border node p' . Then p'_{next} is either a tile on a unique discrete edge incident to u , or an empty node adjacent to such a discrete edge. Let j be the iteration in which u was added in the construction of $\mathcal{T}_{\text{spiral}}$. If the chosen discrete edge leads to a child of u , then it consists of $l_{j-1} - 1$ tiles; otherwise, it leads to the parent of u and consists of $l_j - 1$ tiles. In either case, by construction this edge coincides locally with the corresponding discrete edge in $\mathcal{T}^{(l_{j-1})}$ or $\mathcal{T}^{(l_j)}$, respectively. By Claim 2, the agent cannot observe tiles from other discrete edges before reaching the next border node. Hence, starting from corresponding nodes in the same state, A performs exactly the same sequence of transitions until the agent reaches the next border node or A terminates. In particular, the i -th phase in the execution of A on $\mathcal{T}_{\text{spiral}}$ is identical to the i -th phase in the execution on $\mathcal{T}^{(l_j)}$ for some $j \in \{1, \dots, 240s + 1\}$. Therefore, $\varphi(S'_{i+1}) = S_{i+1}$ whenever phase $i + 1$ exists. Since phase transitions include termination, both executions terminate in the same phase, and thus $k' = k$. $\triangleleft$

By Claim 3, the border sequence of the execution of A on $(\mathcal{T}_{\text{spiral}}, r)$ coincides with the common border sequence of the executions on $(\mathcal{T}^{(l_i)}, p_1^{(l_i)})$. In particular, the execution on $(\mathcal{T}_{\text{spiral}}, r)$ consists of exactly $k \leq 240s$ phases and thus visits at most $240s$ border nodes. Since $\mathcal{T}_{\text{spiral}}$ induces a tree in G and leaves are added only in the last iteration, the shortest path from the root r to any leaf contains exactly $240s + 2$ corner tiles. All corner tiles on this path except for the leaf are border nodes. Hence, reaching a leaf requires visiting at least $240s + 1$ border nodes, including r .

Since the execution on $(\mathcal{T}_{\text{spiral}}, r)$ visits at most $240s$ border nodes, the agent never reaches any leaf. Because only leaves are safely shiftable in $\mathcal{T}_{\text{spiral}}$, the automaton must terminate on a tile that is not safely shiftable. This contradicts the assumption that A always terminates on a safely shiftable tile. $\square$

Reconfiguration into Compact Low-Diameter Structures

Reconfiguration into canonical shapes is a natural intermediate objective in programmable matter. Once a structure has been transformed into a regular form, subsequent tasks can rely on predictable geometric properties rather than having to account for the arbitrary features of the initial configuration. Canonical shapes therefore provide a useful foundation for more advanced reconfiguration tasks.

The previous chapter showed why such intermediate shapes are particularly relevant in our setting. Although every connected configuration contains safely removable and safely shiftable tiles, these tiles cannot, in general, be identified using only local information. Arbitrary connected configurations are therefore difficult to manipulate directly. This motivates the search for canonical shapes with a simpler local structure. In particular, we are interested in shapes that are both compact, so that distances within the structure remain small, and sufficiently regular for safely removable tiles to be identified locally.

The canonical shape studied in this chapter is an *icicle*, which consists of a parallelogram-shaped platform with downward-extending lines of tiles (see Figure 4.1). We present a single-agent algorithm that transforms any initially connected configuration into an icicle in $\mathcal{O}(n^3)$ steps, matching the asymptotic runtime of the line-formation algorithm from prior work [HRS20]. The regular structure of an icicle, much like that of a line, allows an agent without an initial tile to find safely removable tiles. However, an icicle offers several additional advantages. In favorable cases, its diameter is as low as $\mathcal{O}(n^{1/3})$, whereas a line always has diameter n . Moreover, an icicle contains multiple safely removable tiles, so the agent can often find one without traversing the entire structure. In a line, by contrast, only the two endpoints are safely removable. Our algorithm has another useful property: throughout the reconfiguration, all movement is confined to a region that tightly bounds the configuration in two of the three dimensions. This is particularly useful in space-critical applications, where the available workspace is limited and large intermediate expansions of the configuration may be infeasible.

We complement the theoretical analysis with simulation results. These indicate that, on average, the algorithm decreases the diameter of the configuration and that its observed runtime remains consistently below the worst-case upper bound, staying close to n^2 throughout our experiments. Although the simulations do not exhibit a large increase in diameter, such an increase is possible for specific families of configurations. We describe one such family, for which the diameter increases by a factor of $\mathcal{O}(n^{1/3})$, and we conjecture that this is the largest possible increase.

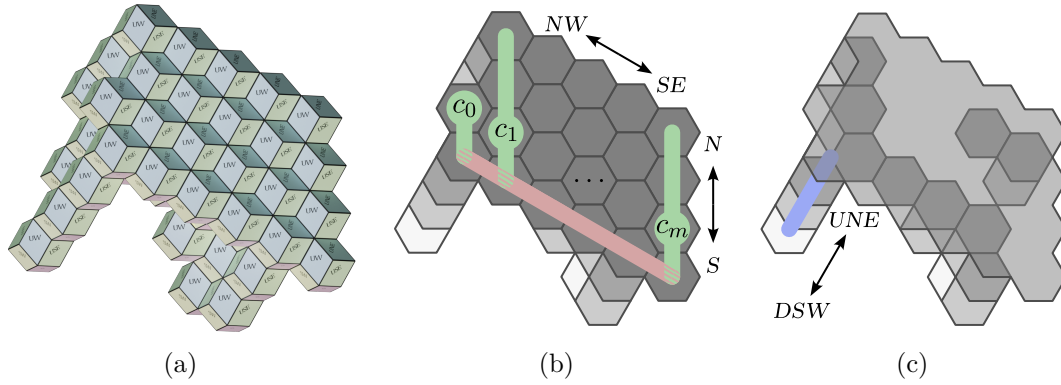


Figure 4.1: Example of an icicle configuration with a partially filled parallelogram, shown in three-dimensional view (a) and as a stack of two-dimensional hexagonal layers (b,c). In (b), the southernmost row of the parallelogram and several columns are highlighted. In (c), the south-westernmost tower is highlighted. For better visibility, the parallelogram is shown transparently.

Problem Definition. We now formalize the corresponding reconfiguration task. Consider an arbitrary initially connected configuration $C^0 = (\mathcal{T}^0, p^0)$ with $p^0 \in \mathcal{T}^0$.

Remark 4.1. *Superscripts generally denote step numbers and may be omitted when they are clear from the context.*

An algorithm solves the *icicle formation problem* if its execution yields a sequence

$$C^0 = (\mathcal{T}^0, p^0), \dots, C^T = (\mathcal{T}^T, p^T)$$

of connected configurations such that $\mathcal{T}^T$ forms an icicle, each configuration C^t for $1 \leq t \leq T$ is obtained from C^{t-1} by a single action of the agent, and the agent terminates in step T .

Remark 4.2. *In this chapter, we often identify a tile with the node of G it occupies whenever this causes no ambiguity. When the distinction matters, in particular when describing movement, we distinguish explicitly between tiles and nodes.*

We now formally define the icicle; see Figure 4.1. A maximal consecutive sequence of tiles is called a *column* in directions N and S , a *row* in directions NW and SE , and a *tower* in directions UNE and DSW . A *parallelogram* consists of a maximal consecutive sequence of equally sized columns $c_0, \dots, c_m$, ordered from west to east, whose southernmost tiles lie in a common row, that is, $v_i + SE = v_{i+1}$ for all $0 \leq i < m$, where v_i is the southernmost tile of column c_i . In a *partially filled* parallelogram, the westernmost column c_0 may be smaller than the remaining equally sized columns.

An *icicle* is a connected set of towers whose uppermost tiles form a parallelogram, possibly partially filled. Equivalently, an icicle consists of such a parallelogram from which towers extend in direction DSW . This structure has the key property that safely removable tiles can be identified locally: every tile with a neighboring tile in direction UNE but no neighboring tile in direction DSW is safely removable. If no such tile exists, then all towers have size one, and the northernmost tile of the westernmost column is safely removable.

Chapter Outline. We begin by discussing problem-specific related work in Section 4.1. In Section 4.2, we introduce the additional terminology used throughout this chapter. We then present a non-halting icicle-formation algorithm in Section 4.3 and prove in Section 4.4 that any initially connected configuration converges to an icicle under this algorithm. In Section 4.5, we establish a termination criterion and analyze the runtime. Section 4.6 presents and discusses the simulation results, and Section 4.7 concludes the chapter.

4.1. Related Work

In this section, we discuss related work with a focus on shape reconfiguration. Table 4.1 summarizes the known results in the hybrid model and includes our result for comparison.

For the three-dimensional hybrid model, Hinnenthal et al. present an algorithm that transforms any connected configuration of n tiles into a line in $\mathcal{O}(n^3)$ steps [HRS20]. This is the only previous result for the three-dimensional hybrid model outside the results presented in this thesis. It therefore provides the main point of comparison for our result, both in terms of the chosen target shape and the runtime. Whereas their algorithm reconfigures into a line, the present chapter considers reconfiguration into an icicle, that is, into a canonical shape with substantially smaller diameter.

For the two-dimensional hybrid model, Gmyr et al. present algorithms that transform connected configurations into several canonical hole-free shapes, namely lines, blocks, and trees, in worst-case optimal $\mathcal{O}(n^2)$ steps [Gmy+20]. This result is closely related to the present chapter because it shows that the choice of the intermediate shape matters: different canonical target shapes provide different structural properties that may be useful for subsequent reconfiguration tasks. In particular, the block considered there consists of a consecutive sequence of rows whose westernmost tiles all lie in a common column. In this sense, the icicle studied in this chapter can be viewed as a three-dimensional generalization of that block shape.

A more general reconfiguration problem is studied in [FLS26]. There, the active agent is assumed to be able to distinguish whether a node lies within a given target shape. For simply connected targets, the authors give an algorithm with worst-case optimal runtime $\mathcal{O}(mn)$, where m denotes the number of initially unoccupied target nodes. They also give an algorithm with runtime $\mathcal{O}(n^4)$ for a larger class of shapes with holes. In contrast to the present chapter, that work studies reconfiguration into arbitrary given target shapes rather than into a fixed canonical shape.

Other work in the hybrid model has considered neighboring problems concerned with specific geometric structures. Gmyr et al. study recognition rather than reconfiguration and show how a single active agent can decide whether a given configuration is a parallelogram with a given height-to-length ratio [Gmy+18]. This is relevant here because parallelogram-shaped structures also appear in the definition of the icicle. Nokhanji et al. study the problem of maintaining a line in the presence of dynamic tile failures using multiple finite-state agents [NFS23]. Although this differs from the reconfiguration problems considered here, it is related through its focus on maintaining a canonical shape under limited computational capabilities.

Dim.	Target shape	Runtime	Reference
2D	line, block, tree	$\mathcal{O}(n^2)$	[Gmy+20]
2D	any	$\mathcal{O}(mn)$ simply connected target, $\mathcal{O}(n^4)$ targets with holes	[FLS26]
3D	line	$\mathcal{O}(n^3)$	[HRS20]
3D	icicle	$\mathcal{O}(n^3)$	[HLS25]

Table 4.1: Shape reconfiguration results in the hybrid model. Here, n denotes the number of tiles and m denotes the number of target nodes without a tile.

Amoebots. A large body of work studies shape formation and reconfiguration in the Amoebot model. Early work establishes the feasibility of leader election and line formation [Der+15b]. Derakhshandeh et al. subsequently present a general framework for shape formation based on two primitives, the spanning forest primitive and the snake formation primitive, and use it to obtain optimal algorithms for hexagon and triangle formation [Der+15a]. A first universal shape formation result appears in [Der+16], where the particles form a target shape composed of a constant number of unit triangles, scaled according to the number of available particles. Later work considerably extends this line of research. Di Luna et al. give a universal shape formation algorithm for a broader class of target shapes and characterize which shapes can be formed deterministically from simply connected initial configurations [DiL+20a]. Related work shows that a constant number of finite-state particles can simulate a Turing-complete mobile agent and uses this to obtain more general shape formation results, including circles, spirals, and fractal objects [DiL+20b].

Most of these shape formation algorithms proceed via canonical intermediate structures. In contrast, Peters et al. present the first Amoebot reconfiguration algorithm that does not rely on a canonical intermediate shape [KPS23]. Instead, they introduce geometric primitives and shortest-path communication structures that allow particle systems to exploit the symmetric difference between source and target shape and to route particles along many parallel shortest paths. A key ingredient in this context is the feather tree communication structure introduced in [KPS22].

A related line of work studies the recovery of canonical shapes from faults. Di Luna et al. consider line recovery after crashes [DiL+18], and Nokhanji and Santoro study connected line recovery in the presence of faulty particles while maintaining connectivity throughout the reconfiguration [NS20]. Kostitsyna et al. study fault-tolerant hexagon formation under memory-reset faults and develop a fault-tolerant spanning forest primitive for that purpose [KSW22].

Restricted and extended variants of the Amoebot model have also been studied. On the restrictive side, the SILBOT model, in which particles are oblivious and cannot communicate, has been used to study leader election, compaction, line formation, and scattering [DAn+20; NPP25]. At the other end of the spectrum, Padalkin et al. study a stronger extension of the Amoebot model in which particles may jointly push and pull entire substructures [PKS25]. Another extension, studied by Feldmann et al., introduces reconfigurable circuits for fast signal transmission through the particle system and uses them to speed up coordination and shape recognition tasks, including the recognition of parallelograms [Fel+22].

Other Models. A related two-dimensional model considers agents that manipulate connected tile structures on the square grid, known as polyominoes. Shape reconfiguration in this model is studied in a centralized setting, where the goal is to compute a global reconfiguration schedule for one or more robots. Garcia et al. consider connected reconfiguration in the presence of obstacles [Gar+22] and later extend this line of work to cooperative reconfiguration by multiple agents [Gar+24]. Becker et al. study a generalized model with weighted movement costs and show that computing a connectivity-preserving schedule of minimum makespan for arbitrary input and target configurations is NP-hard. They also provide a polynomial-time constant-factor approximation algorithm for a restricted class of configurations [Bec+25]. Garcia et al. complement these results with simulation and hardware experiments for single-robot tile reconfiguration [Gar+25].

This emphasis on centralized computation is also common in shape reconfiguration for modular self-reconfigurable robots. There, using canonical intermediate shapes often yields a universal strategy for arbitrary input and target shapes: one computes a schedule from the input shape to the canonical shape and another from the target shape to the same canonical shape, and then obtains a schedule from source to target by reversing the latter sequence. This line of work belongs to the second part of the thesis and will be discussed in detail in Chapter 6.

A different square-grid model considers a deterministic finite automaton that manipulates passive tiles without preserving connectivity. Czyzowicz et al. consider the problem of rearranging n tiles (called bricks in their terminology) into a most compact structure, called a nest [CDP21]. They give an optimal $O(sn)$ -time algorithm, where s denotes the span, that is, the maximum graph distance between any two tiles in the initial shape. Kant et al. study the formation of a fort, that is, a hollow rectangle of minimum span and maximum covered area, and give a tight $\Theta(n^2)$ bound for that problem [KPM21].

A more distant model is the nubot model, an active self-assembly model. In this model, units on a triangular grid can change state, create or delete neighbors, and move, thereby supporting growth as well as motion. Woods et al. introduce the nubot model and show how to assemble lines, patterns, and connected two-dimensional shapes in expected polylogarithmic time [Woo+13]. Chen et al. further study fast shape construction in a variant with nondeterministic agitation, modeling effects such as Brownian motion [Che+14]. Unlike the hybrid model, this model focuses on growth and self-assembly rather than reconfiguration of a fixed set of tiles.

4.2. Additional Terminology

For this chapter, it is convenient to use a different coordinate system than the Cartesian coordinates introduced earlier. Specifically, we express all coordinates with respect to the basis $(\overrightarrow{NW}, \overrightarrow{N}, \overrightarrow{UNE})$; see Figure 4.2. For a node $v \in V$, we write

$$c(v) = (x(v), y(v), z(v))$$

for its coordinates in this basis. In particular, the x -, y -, and z -coordinates correspond to the directions NW , N , and UNE , respectively. In this representation, moving in

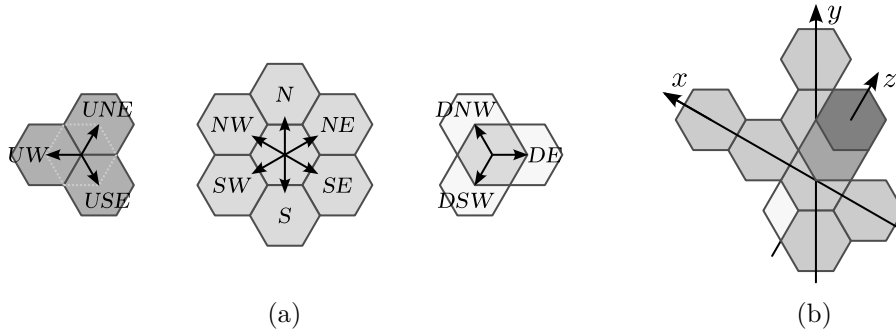


Figure 4.2: (a) Compass directions, repeated for ease of reference. (b) The coordinate axes used throughout this chapter: x increases in direction NW, y increases in direction N, and z increases in direction UNE.

direction UNE or DSW changes only the z -coordinate. Hence, all nodes on the same tower, that is, on the same sequence of nodes in directions UNE and DSW , have the same x - and y -coordinates, which will simplify the analysis of our algorithm.

Observation 4.3. *The direction vectors from Observation 2.4, expressed in the basis $(\overrightarrow{NW}, \overrightarrow{N}, \overrightarrow{UNE})$, are*

$$\begin{array}{lll} \overrightarrow{NW} = (1, 0, 0) & \overrightarrow{N} = (0, 1, 0) & \overrightarrow{UNE} = (0, 0, 1) \\ \overrightarrow{UW} = (1, -1, 1) & \overrightarrow{USE} = (0, -1, 1) & \overrightarrow{SW} = (1, -1, 0) \\ \overrightarrow{SE} = (-1, 0, 0) & \overrightarrow{NE} = (-1, 1, 0) & \overrightarrow{S} = (0, -1, 0) \\ \overrightarrow{DSW} = (0, 0, -1) & \overrightarrow{DE} = (-1, 1, -1) & \overrightarrow{DNW} = (0, 1, -1) \end{array}$$

For a node set $S \subseteq V$, let $x_{\min}^S$ and $x_{\max}^S$ denote the minimum and maximum x -coordinate of any node within S . We define $y_{\min}^S$, $y_{\max}^S$, $z_{\min}^S$, and $z_{\max}^S$ analogously. Throughout this chapter, we normalize coordinates with respect to the initial set of tiled nodes $\mathcal{T}^0$, that is,

$$x_{\min}^{\mathcal{T}^0} = y_{\min}^{\mathcal{T}^0} = z_{\min}^{\mathcal{T}^0} = 0.$$

The *bounding cylinder* $\mathfrak{C}(S)$ is the set of all nodes, both empty and tiled, whose coordinates are bounded by the minimum and maximum x - and y -coordinates of any node within S (see Figure 4.3a). Formally,

$$\mathfrak{C}(S) = \{v \in V \mid x_{\min}^S \leq x(v) \leq x_{\max}^S, y_{\min}^S \leq y(v) \leq y_{\max}^S\}.$$

The *bounding box* $\mathfrak{B}(S)$ is a subset of the bounding cylinder that additionally bounds the z -coordinate (see Figure 4.3b). Formally,

$$\mathfrak{B}(S) = \{v \in \mathfrak{C}(S) \mid z_{\min}^S \leq z(v) \leq z_{\max}^S\}.$$

We refer to the extents of the bounding cylinder and the bounding box along the x -, y -, and z -axes as their *width*, *height*, and *depth*, respectively. Note that the bounding cylinder has infinite depth. By our choice of coordinates, every bounding box is a filled, possibly degenerate, parallelepiped (see Figure 4.3b). A node v is said to be *inside* the bounding cylinder or bounding box of S if $v \in \mathfrak{C}(S)$ or $v \in \mathfrak{B}(S)$, respectively, and *outside* otherwise.

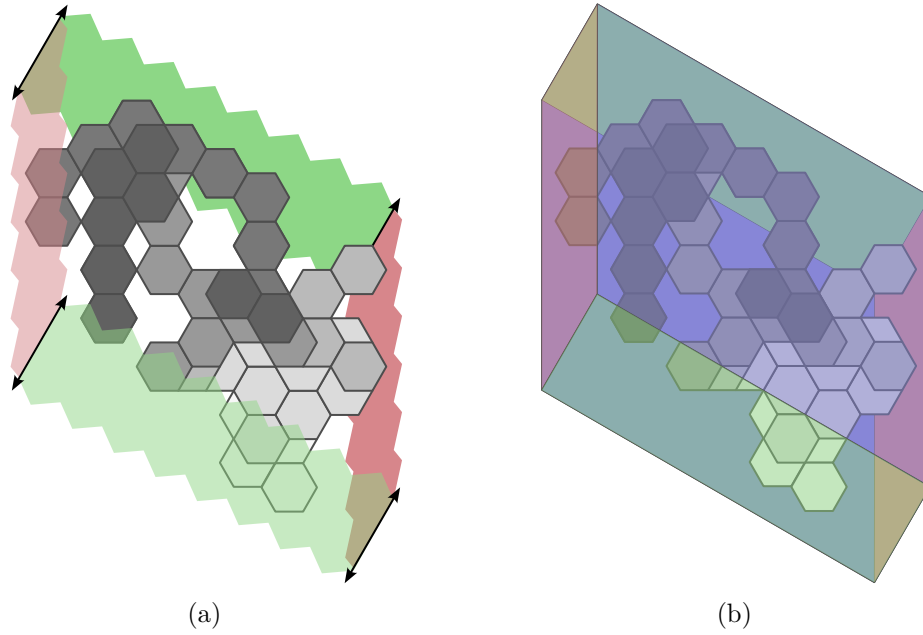


Figure 4.3: The bounding cylinder (a) and bounding box (b) of an example configuration. In (a), the arrows indicate that the bounding cylinder extends infinitely in directions *UNE* and *DSW*.

Remark 4.4. *If not specified otherwise, the bounding cylinder and bounding box are taken with respect to the set of tiled nodes of the current configuration.*

For each $i \in \mathbb{Z}$, the *layer* L_i is the set of all nodes in the bounding cylinder of all tiled nodes whose z -coordinate is i , that is,

$$L_i = \{v \in \mathfrak{C}(\mathcal{T}) \mid z(v) = i\}.$$

Nodes with z -coordinate greater than i are said to lie *above* L_i , and nodes with smaller z -coordinate are said to lie *below* L_i . A connected component of the induced subgraph $G|_{L_i \cap \mathcal{T}}$ is called a *fragment* of L_i . For example, the configuration shown in Figure 4.3 contains six fragments. The two uppermost fragments lie in the same layer, while each of the remaining four fragments occupies its own layer.

4.3. The Algorithm

We now describe the algorithm that transforms an arbitrary connected configuration into an icicle. At a high level, the agent repeatedly attempts to transform locally uppermost fragments into partially filled parallelograms. During this process, it rearranges tiles within a single layer and, when necessary to preserve connectivity, may also place tiles in the layer below. If the agent encounters a tile in a higher layer, then the current fragment was not locally uppermost, so it discontinues the current transformation, moves upward, and starts again there. Once a fragment has been transformed into a parallelogram, the agent projects it, that is, it shifts each tile of the fragment to the first empty node in direction *DSW*.

The algorithm is composed of three parts. We first describe a two-dimensional algorithm that transforms a fragment into a partially filled parallelogram. Its central routine is *BUILDPAR*, which traverses the fragment to identify a local violation of the parallelogram shape. If such a violation is found, the routine moves one tile to repair it; otherwise, the fragment already forms a partially filled parallelogram. We then describe the projection routine *PROJECT*, which projects the parallelogram in direction *DSW* and thereby creates the towers of the resulting icicle. Finally, we combine the routines *BUILDPAR* and *PROJECT* into the icicle-formation algorithm *BUILDICICLE*. The following textual descriptions fully specify these algorithms and routines. For reference, we provide pseudocode as a supplementary formal description.

4.3.1. Parallelogram Formation in Two Dimensions

We begin with the two-dimensional subroutine that transforms a fragment into a partially filled parallelogram. The main steps of this subroutine are illustrated by an example execution on the next page; see Figure 4.4.

Search phase. The procedure begins with a *search phase* in which the agent attempts to locate a locally westernmost column and, among such columns with the same x -coordinate, the northernmost one. To do so, it repeatedly moves in directions *NW*, *SW*, and *N*, prioritized in that order, until none of these moves leads to a tiled node. At that point, the agent is located at the northernmost tiled node of some column, and this node is locally westernmost. The search phase is then complete.

Parallelogram subroutine. Starting from the northernmost tiled node v of some column c , where v is locally westernmost, the agent executes the *BUILDPAR* subroutine. Its goal is to traverse the fragment in search of local violations of the shape of a partially filled parallelogram with westernmost column c . To this end, the agent traverses the columns from west to east. Within each column, it first moves north as far as possible and then traverses the column in direction *S*. If, while traversing the first column c , the agent encounters a column farther west, as in Figure 4.4b, it discontinues the current traversal and returns to the search phase. Now let c' denote the column directly east of c . If, while moving north in c' , the agent encounters a column to the west that is distinct from c , it first continues to the northernmost tiled node of c' and then moves south until some column lies in direction *NW*. It then discontinues the current traversal and returns to the search phase. This ensures that, if several such columns exist, the agent reaches the northernmost column directly west of c' .

While moving south in a column, the agent checks whether certain adjacent empty nodes violate the shape of a partially filled parallelogram with westernmost column c . More precisely, it inspects the empty nodes in directions *NE* and *SE* of each column, the empty node immediately south of each column, and, for every column other than c , the empty node immediately north of that column.

We now formalize the notion of a violating node. For orientation, we indicate in parentheses which subfigure of the example execution in Figure 4.4 corresponds to each condition.

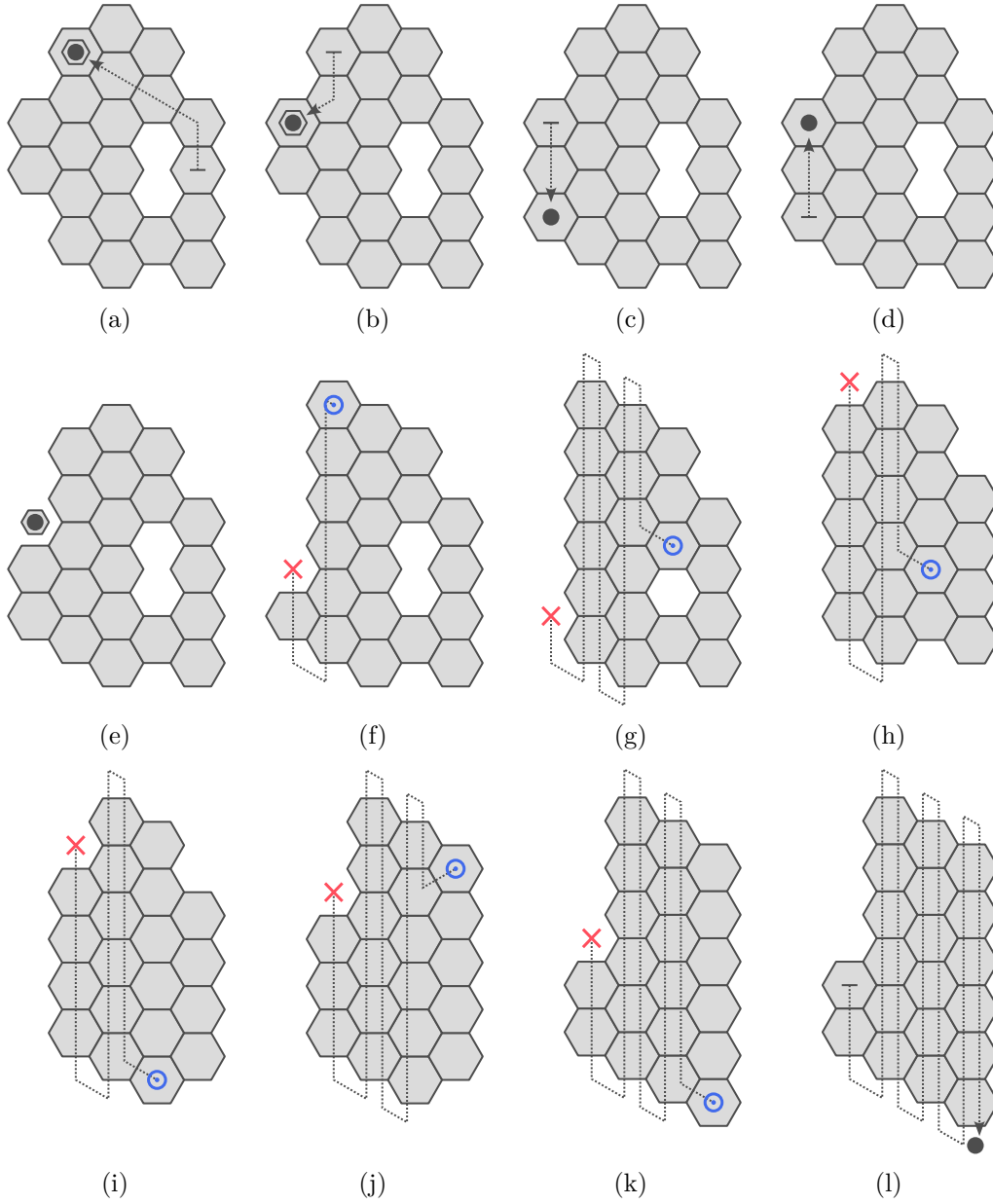


Figure 4.4: Example execution of the parallelogram formation algorithm on a two-dimensional configuration. The agent performs several steps between consecutive configurations shown. In (a) and (b), the agent identifies a westernmost column. In (c), it places a tile, then returns to the tile where it entered the *BUILDPAR* routine in (d), and picks it up in (e). Subsequent iterations omit these intermediate steps and indicate the resulting tile shift from the cross to the circle, with the dashed line showing the path traversed before placing the tile. In (l), the agent terminates.

Definition 4.5. An empty node w is violating if

- (a) w has tiled neighbors in directions N, NE, and SE (4.4c),
- (b) w has tiled neighbors in directions S and SE (4.4d),
- (c) w has tiled neighbors in directions NW and N (4.4e–4.4g and 4.4i), or
- (d) w has tiled neighbors in directions NW, SW, and S (4.4h).

If the agent finds such a violating node w , it places its carried tile at w , returns to column c , and picks up the tile from v in exchange. Note that the tile placed at w may be the tile that the agent initially carried. We refer to this exchange as a *tile shift* from v to w . Afterwards, the agent moves to a tiled neighbor and resumes with a new search phase.

The agent terminates once it has traversed the entire fragment without finding a violating node. At that point, it is located at the empty node immediately south of the easternmost column.

Pseudocode. For reference, the formal pseudocode of the parallelogram-formation algorithm is given in Algorithm 1 on the following page. Throughout the pseudocode, p denotes the current position of the agent. The search phase is implemented in lines 2–3, and the subroutine *BUILDPAR* in lines 6–39.

Within *BUILDPAR*, the variable *firstColumn* keeps track of whether the agent is positioned in the column at which the routine was entered. The main loop in lines 7–31 is executed as long as the agent carries a tile. It implements the traversal of the current column in direction S in lines 8–19, including the checks for violating nodes east and south of the column. The movement to the northernmost tiled node of the next column to the east is implemented in lines 20–29, including the check for a violating node north of that column. Finally, if no such column exists, the procedure terminates on the empty node below the easternmost column in lines 30–31.

There are several situations in which *BUILDPAR* returns without placing a tile. This happens whenever the agent encounters a fragment above in lines 8–9 (which can be ignored in the 2D setting), a column more western than the column at which *BUILDPAR* was entered in lines 10–11, or a column directly north of that column in lines 23–26. After placing a tile and thereby exiting the main loop, the agent retrieves the tile from the node at which *BUILDPAR* was entered in lines 32–38.

4.3.2. Projection

Whenever a locally uppermost fragment is a partially filled parallelogram, the agent applies the projection routine *PROJECT*. During this routine, each tile of the fragment is projected in direction *DSW*, that is, shifted to the first empty node in direction *DSW*. The projection is performed column by column, starting with the easternmost column and proceeding from east to west. Within each column, the tiles are projected from north to south; see Figures 4.5a to 4.5e.

Let $v_0, \dots, v_k$ be the nodes of a column that is being projected, ordered from north to south. For each $i \in \{0, \dots, k\}$, the agent performs a tile shift from v_i to the first

Algorithm 1: 2DPARALLELOGRAMFORMATION

```

1 while true do
2   while  $p + X \in \mathcal{T}$  for some  $X \in \{NW, SW, N\}$  do
3      $\lfloor$  move to the first available tile among NW, SW and N
4     run BUILDPAR
5     if  $p \notin \mathcal{T}$  then return  $\triangleright$  terminate S of easternmost column

  procedure BUILDPAR
6    firstColumn  $\leftarrow$  true
7    while agent carries tile do
8      if  $p + UW \in \mathcal{T}$  or  $p + USE \in \mathcal{T}$  or  $p + UNE \in \mathcal{T}$  then
9        return  $\triangleright$  found fragment above (irrelevant in 2D)
10     else if firstColumn and  $p + SW \in \mathcal{T}$  then
11       return  $\triangleright$  found more western column
12     else if  $p + NE \in \mathcal{T}$  and  $p + SE \notin \mathcal{T}$  then
13       move SE; place tile; move NW  $\triangleright$  placed tile below eastern column
14     else if  $p + N, p + SE \in \mathcal{T}$  and  $p + NE \notin \mathcal{T}$  then
15       move NE; place tile; move SW  $\triangleright$  placed tile above eastern column
16     else
17       move S
18       if  $p \notin \mathcal{T}$  and  $p + N, p + NE, p + SE \in \mathcal{T}$  then
19         place tile; move N  $\triangleright$  place tile below current column
20       else if  $p \notin \mathcal{T}$  and  $p + NE \in \mathcal{T}$  then
21         move NE; move N; firstColumn  $\leftarrow$  false  $\triangleright$  enter next column
22       while  $p \in \mathcal{T}$  do
23         if  $p + SW \notin \mathcal{T}$  and  $p + NW \in \mathcal{T}$  then
24           while  $p + N \in \mathcal{T}$  do move N
25           while  $p + NW \notin \mathcal{T}$  do move S
26           return  $\triangleright$  found more northern column
27         move N
28       if  $p + S, p + SE \in \mathcal{T}$  then
29         place tile; move S  $\triangleright$  place tile above current column
30       else if  $p \notin \mathcal{T}$  and  $p + NE \notin \mathcal{T}$  then
31         return  $\triangleright$  no next column  $\rightarrow$  terminate

32 if not firstColumn then
33   while  $p + X \in \mathcal{T}$  for some  $X \in \{SW, S, NW\}$  do
34      $\lfloor$  move to the first available tile among SW, S and NW
35 while  $p + N \in \mathcal{T}$  do
36    $\lfloor$  move N
37 pick up tile; move to the first available tile among S, SE and NE
38 return

```

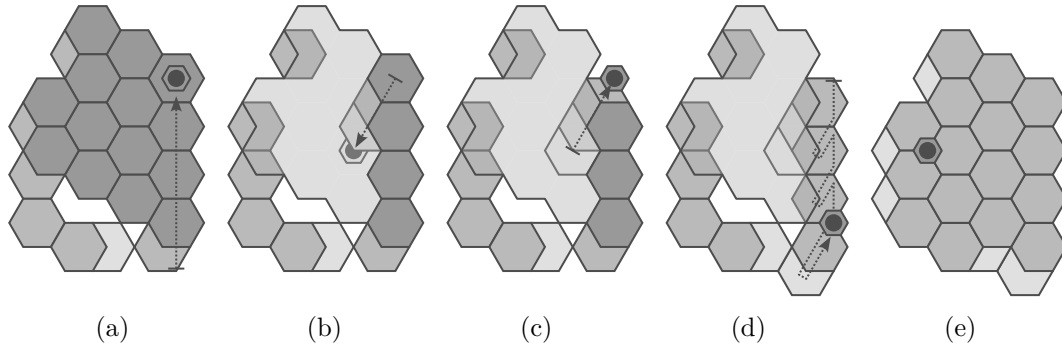


Figure 4.5: Projection of a parallelogram in direction DSW . In (a)–(d), the agent projects a single column; (e) shows the configuration after the projection. Tiles of the same fragment that belong to other columns are shown transparently in (b)–(d).

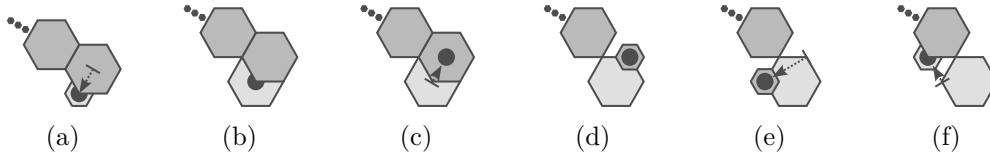


Figure 4.6: Special case of projecting a parallelogram of height one. After picking up the tile in (d), the agent preserves connectivity by moving SW in (e) and DNW in (f), thereby entering the node in direction DSW of the next column.

empty node w_i in direction DSW of v_i . That is, it first moves to w_i , places its carried tile there, and then returns to pick up the tile from v_i . If $i < k$, it subsequently moves S to enter v_{i+1} . After picking up the last tile of the column at v_k , the agent moves NW and continues the projection in the neighboring column to the west.

A special case arises when the parallelogram has height one. After picking up a tile, the agent moves SW and then DNW ; see Figure 4.6. This places it at the node DSW of the unique tile in the next column, so the first step of projecting that tile is skipped. In this way, connectivity is preserved during the projection.

After projecting the last tile of the fragment, the agent moves one step in direction DSW to enter the layer below.

Pseudocode. For reference, the formal pseudocode of the projection routine is given in Algorithm 2. The variable *heightOne* is initialized in line 1 and indicates whether the current parallelogram has height one. The projection of a single tile is implemented in lines 5–8. The remaining lines of the main loop handle the traversal of the current column, the transition into or below the next column to the west, and termination, which may depend on the value of *heightOne*.

4.3.3. Icicle Formation

At a high level, the icicle-formation algorithm repeatedly transforms a locally upper-most fragment into a partially filled parallelogram and then projects this fragment in direction DSW . The first step uses the routine *BUILDPAR* introduced above. In

Algorithm 2: PROJECT

```

procedure PROJECT
1  $heightOne \leftarrow (p + N \notin \mathcal{T} \text{ and } p + S \notin \mathcal{T})$ 
2 while true do
3   if not  $heightOne$  then
4     while  $p + N \in \mathcal{T}$  do move  $N$ 
5   while  $p \in \mathcal{T}$  do move  $DSW$ 
6   place tile
7   while  $p + UNE \in \mathcal{T}$  do move  $UNE$ 
8   pick up tile
9   if  $p + S \in \mathcal{T}$  then
10    move  $S$ 
11  else if  $p + NW \in \mathcal{T}$  and  $heightOne$  then
12    move  $SW$ ; move  $DNW$ 
13  else if  $p + NW \in \mathcal{T}$  and not  $heightOne$  then
14    move  $NW$ 
15  else
16    move  $DSW$ ; return

```

the two-dimensional setting, when the agent starts at the northernmost node v of a locally westernmost column, the tile at v is always safely shiftable in the sense of Definition 3.1b. Moreover, Figure 4.7a illustrates the only critical case that arises in two dimensions; there, connectivity is preserved because the shifted tile does not leave the neighborhood of its tiled neighbors.

In three dimensions, the situation is more subtle. As discussed in Chapter 3, the notions of safely removable and safely shiftable tiles are useful for reasoning about connectivity, but they cannot in general be identified using only local information. For the present algorithm, we therefore introduce stronger local notions, namely simply removable and simply shiftable tiles.

Recall that $N_{\mathcal{T}}(v)$ denotes the neighborhood of v in the induced subgraph $G|_{\mathcal{T}}$, that is, the set of all tiled neighbors of v .

Definition 4.6. *Let $v \in \mathcal{T}$ be a tiled node.*

- (a) *The tile at node v is simply removable if $G|_{N_{\mathcal{T}}(v)}$ is connected.*
- (b) *The tile at node v is simply shiftable if $G|_{N_{\mathcal{T}}(v)}$ is disconnected and there exists a node $w \in N_G(v)$ such that $G|_{N_{\mathcal{T}}(v) \cup \{w\}}$ is connected. Such a node w is called a bridge node of v .*
- (c) *The tile at node v is called not simply movable if it is neither simply removable nor simply shiftable.*

Remark 4.7. *Unlike the notions of safely removable and safely shiftable from Definition 3.1, the notions of simply removable and simply shiftable are disjoint by definition. This is convenient for the algorithmic description and the analysis, since each tile falls into at most one of the two classes.*

We now describe the full icicle-formation algorithm *BUILDICICLE*.

Search phase. The search phase extends the one used for parallelogram formation to three dimensions. The agent attempts to locate a locally uppermost fragment and, within that fragment, a locally westernmost column. Again, if multiple such columns have the same x -coordinate, it attempts to reach the northernmost among them. To do so, it repeatedly moves in directions UW , USE , UNE , NW , SW , and N until none of these moves leads to a tiled node. At that point, the agent is located at the northernmost tiled node v of some column, and this node is locally westernmost and locally uppermost. The search phase is then complete.

Simply removable tile. If the tile at v is simply removable, the agent enters the routine *BUILDPAR*. There are three possible outcomes. First, the routine may return after detecting a more western column or a tile in a higher layer. In this case, the agent returns to the search phase. Second, the routine may return after placing a tile. Then the agent moves back to pick up the tile at v , moves to the next tile in direction S or SE , and returns to the search phase. Third, the routine may return at the empty node south of the easternmost column of the fragment. In this case, the fragment forms a partially filled parallelogram, and the agent executes the projection routine *PROJECT*. After the projection, it moves one step in direction DSW to enter the layer below and returns to the search phase.

Case distinction. If the tile at v is not simply removable, then it is either simply shiftable or not simply movable. In both cases, the agent must perform additional steps to preserve connectivity. It therefore acts according to the case distinction described below, in the given order; see Figure 4.7 for a graphical overview. Afterwards, it returns to the search phase.

Case A. If v is simply shiftable and $v + SE$ is a bridge node of v , then the agent shifts the tile from v to $v + SE$ and afterwards moves to $v + S$; see Figures 4.7a to 4.7g.

Case B. If the only tiled neighbors of v are at DSW and SE , then the agent moves to $v + SE$ in order to observe the node $w = v + SE + DSW$; see Figure 4.7h. If w is empty, it shifts the tile from v to w and afterwards moves to $v + SE$. Otherwise, if w is tiled, then the tile at v is safely removable, so the agent moves back to v , enters *BUILDPAR*, and proceeds afterwards as in the simply removable case.

Case C. If v is simply shiftable, $v + DE$ is a bridge node of v , and v has a tiled neighbor other than possibly those at DSW and SE , then the agent shifts the tile from v to $v + DE$ and then moves to the first available tile at $v + S$, $v + SE$, or $v + NE$; see Figures 4.7i to 4.7r.

Case D. If the only tiled neighbors of v are at DNW , S , and NE , then the agent proceeds as follows; see Figure 4.7s. First, it moves in direction S until it reaches a node w that has a tiled neighbor at UW , USE , SW , SE , DSW , or DE , or no tiled neighbor at S . If w has a tiled neighbor at UW , USE , or SW , then the agent immediately returns to the search phase. Otherwise, the column is split at w into a southern part and a northern part. If $w + DE$ is empty, then w belongs to the northern part; otherwise, it

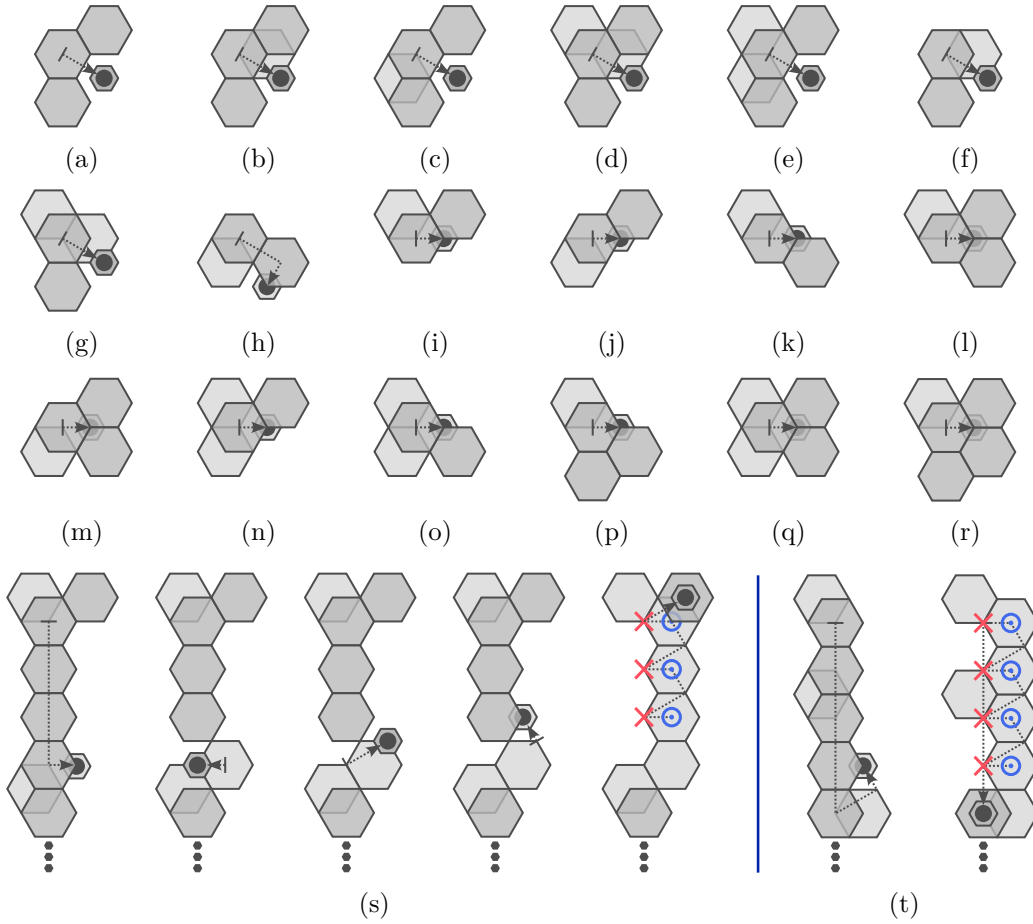


Figure 4.7: All cases in which the northernmost tiled node of a locally westernmost column is not simply removable. For brevity, (a)–(r) show only the movement of the agent to the node to which the tile is shifted. In (h), the agent may instead enter *BUILDPAR* if that node is tiled. In (s), only the first tile shift is shown in detail; all remaining shifts in (s) and (t) are abbreviated. Figures (s) and (t) only show the cases where a tile is encountered in direction *DSW* and *DE*, respectively.

belongs to the southern part. The agent then shifts each tile of the northern part one step in direction *DE*.

More precisely, let $v_1, \dots, v_k$ be the nodes of the northern part of the column, ordered from south to north, where $v_k = v$. For each $i = 1, \dots, k$, the agent performs a tile shift from v_i to $v_i + DE$. After picking up a tile from node v_i with $i < k$, it moves *NE* and then *DNW* to position itself at $v_{i+1} + DE$ while preserving connectivity. After picking up the last tile from $v_k = v$, it moves to $v + NE$.

Case E. If the only tiled neighbors of v are at *DNW* and *S*, then the agent proceeds as follows; see Figure 4.7t. It moves in direction *S* until it reaches a node w that has a tiled neighbor at *UW*, *USE*, *SW*, *SE*, or *DE*, or no tiled neighbor at *S*. In contrast to Case D, nodes with a tiled neighbor at *DSW* are ignored during this traversal. If no such tiled neighbor is encountered, then the entire column forms a parallelogram of

width one, and the agent projects it in direction DSW . Otherwise, the column is split at w , and each tile in its northern part is shifted one step in direction DE , exactly as in Case D. After picking up the tile at v , the agent repeatedly moves in direction S (through empty nodes) and enters the first tiled node at s or SE .

Remarks on the case distinction. The choices made in Cases B and D deserve some explanation. In Case B, the node $v + DE$ is a bridge node of v , but the agent instead attempts to shift the tile to $v + SE + DSW$. The reason is that $v + DE$ does not lie in the bounding cylinder of the tiles observable from v . Similarly, in Case D, the node $v + DSW$ is a bridge node of v . Although this node lies in the bounding cylinder of the observable tiles, it has the same x - and y -coordinates as v . For the analysis, it is important that tiles are never placed outside the bounding cylinder and that, except during projections, they consistently progress eastward or southward.

Pseudocode. For reference, the formal pseudocode of the icicle-formation algorithm is given in Algorithm 3. The search phase is implemented in lines 2–3.

The case in which the search phase ends at a simply removable tile is implemented in lines 4–5. This part enters the *BUILDPAR* routine; see Algorithm 1. If *BUILDPAR* returns with the agent on an empty node, then the fragment is a partially filled parallelogram, and the agent executes the *PROJECT* subroutine; see Algorithm 2.

The remaining branches implement Case A–Case E in the order of the case distinction. Lines 6–7 implement Case A, lines 8–13 implement Case B, lines 14–16 implement Case C, lines 17–25 implement Case D, and lines 26–38 implement Case E. In Case B, the agent may again enter *BUILDPAR* and afterwards either project the resulting parallelogram or continue analogously to the simply removable case.

4.4. Analysis

In this section, we analyze the execution of *BUILDICICLE*. At this point, the algorithm is not yet terminating. Our goal is therefore to show that, for every initially connected configuration, the resulting sequence of configurations converges to an icicle. The next section introduces the termination criterion and analyzes the resulting runtime.

Proof structure. The convergence proof is divided into two stages. In the first stage, we show that every initially connected configuration eventually reaches a configuration that contains a fragment with certain structural properties, which we introduce below. Intuitively, these properties ensure that the fragment is locally uppermost, covers all relevant positions in the bounding cylinder, and forms a parallelogram that is flush with the southern, eastern, and northern sides of the bounding cylinder. In the second stage, we show that, once such a fragment has been reached, the subsequent execution converges to an icicle.

Before turning to the two stages of the convergence proof, we verify that the algorithm respects the connectivity constraint of the hybrid model. In the following, let $C^i = (\mathcal{T}^i, p^i)$ denote the configuration after i steps of the execution.

Algorithm 3: *BUILDICICLE*

```

1 while true do
2   while  $p + X \in \mathcal{T}$  for some  $X \in \{UW, USE, UNE, NW, SW, N\}$  do
3      $\lfloor$  move to the first available tile among  $UW, USE, UNE, NW, SW$ , and  $N$ 
4   if  $G|_{N_{\mathcal{T}}(p)}$  is connected then  $\triangleright$  simply removable tile
5      $\lfloor$  run BUILDPAR; if  $p \notin \mathcal{T}$  then move  $N$ ; run PROJECT
6   else if  $G|_{N_{\mathcal{T}}(p) \cup \{p+SE\}}$  is connected then  $\triangleright$  Case A
7      $\lfloor$  move  $SE$ ; place tile; move  $NW$ , pick up tile; move  $S$ 
8   else if  $N_{\mathcal{T}}(p) = \{p + DSW, p + SE\}$  then  $\triangleright$  Case B
9     if  $p + SE + DSW \in \mathcal{T}$  then
10       $\lfloor$  run BUILDPAR; if  $p \notin \mathcal{T}$  then move  $N$ ; run PROJECT
11    else
12       $\lfloor$  move  $SE$ ; move  $DSW$ ; place tile
13       $\lfloor$  move  $UNE$ ; move  $NW$ ; pick up tile; move  $SE$ 
14  else if  $G|_{N_{\mathcal{T}}(p) \cup \{p+DE\}}$  is connected then  $\triangleright$  Case C
15     $\lfloor$  move  $DE$ ; place tile; move  $UW$ ; pick up tile
16     $\lfloor$  move to the first available tile among  $S, SE$  and  $NE$ 
17  else if  $N_{\mathcal{T}}(p) = \{p + DNW, p + S, p + NE\}$  then  $\triangleright$  Case D
18    while  $p + S \in \mathcal{T}$  and  $p + X \notin \mathcal{T}$  for all  $X \in \{UW, USE, SW, SE, DSW, DE\}$  do
19       $\lfloor$  move  $S$ 
20    if  $p + X \notin \mathcal{T}$  for all  $X \in \{UW, USE, SW\}$  then
21      if  $p + DE \in \mathcal{T}$  then move  $N$ 
22       $\lfloor$  move  $DE$ ; place tile; move  $UW$ ; pick up tile
23      while  $p + N \in \mathcal{T}$  do
24         $\lfloor$  move  $SE$ ; move  $DNW$ ; place tile; move  $UW$ ; pick up tile
25       $\lfloor$  move  $NE$ 
26  else if  $N_{\mathcal{T}}(p) = \{p + DNW, p + S\}$  then  $\triangleright$  Case E
27    while  $p + S \in \mathcal{T}$  and  $p + X \notin \mathcal{T}$  for all  $X \in \{UW, USE, SW, SE, DE\}$  do
28       $\lfloor$  move  $S$ 
29    if  $p + X \notin \mathcal{T}$  for all  $X \in \{UW, USE, SW\}$  then
30      if  $p + SE \notin \mathcal{T}$  and  $p + DE \notin \mathcal{T}$  then
31         $\lfloor$  move  $N$ ; run PROJECT
32      else
33        if  $p + DE \in \mathcal{T}$  then move  $N$ 
34         $\lfloor$  move  $DE$ ; place tile; move  $UW$ ; pick up tile
35        while  $p + N \in \mathcal{T}$  do
36           $\lfloor$  move  $SE$ ; move  $DNW$ ; place tile; move  $UW$ ; pick up tile
37        while  $p \notin \mathcal{T}$  do
38           $\lfloor$  move  $S$ ; if  $p + SE \in \mathcal{T}$  then move  $SE$ 

```

Lemma 4.8. *If the agent disconnects $G|_{\mathcal{T}^i}$ in step i , then $G|_{\mathcal{T}^{i+4}}$ is connected. Moreover, for every j with $i < j < i + 4$, the graph $G|_{\mathcal{T}^j \cup \{p^j\}}$ is connected and the agent carries a tile.*

Proof. If the agent disconnects $G|_{\mathcal{T}^i}$ in step i , then in this step it picks up a tile from some node v . For $j = i + 1$, the second claim follows immediately: the graph $G|_{\mathcal{T}^i}$ is connected by assumption, and after the tile at v is picked up, the agent is still positioned at v in step $i + 1$. Hence, $G|_{\mathcal{T}^{i+1} \cup \{p^{i+1}\}}$ is connected.

Since removing the tile at v disconnects $G|_{\mathcal{T}^i}$, the node v is a cut node of $G|_{\mathcal{T}^i}$. In particular, v is also a cut node of $G|_{N_{\mathcal{T}}(v)}$, and therefore v is not simply removable. Hence, the agent cannot enter the *BUILD PAR* procedure, which is only executed when the current tile is simply removable.

In Cases A–C, the agent first places a tile at a bridge node. By Definition 4.6, this preserves connectivity among the nodes in $N_{\mathcal{T}}(v)$. Hence, it remains to consider the *PROJECT* procedure and Cases D and E.

First, consider the *PROJECT* procedure. After the tile at v is projected, the node $v + DSW$ is tiled. Moreover, $v + DSW$ is adjacent to $v + SW$, and $y(v + SW) = y(v) - 1$. Thus, if the parallelogram has height greater than one, then connectivity is maintained throughout the projection. Since tiles are projected from N to S , this also covers the case in which the westernmost column is only partially filled.

It remains to consider the case in which the *PROJECT* procedure is applied to a parallelogram of height one; see Figure 4.6. After shifting the tile from v to $v + DSW$, the agent moves below the next column via the nodes $v + SW$ and $v + SW + DNW$. Both of these nodes are adjacent to the tiled node $v + DSW$ and to $v + NW$, which is the only tiled node in the next column. Hence, $G|_{\mathcal{T}^{i+2} \cup \{p^{i+2}\}}$ and $G|_{\mathcal{T}^{i+3} \cup \{p^{i+3}\}}$ are connected. After the tile is placed at $v + SW + DNW$ in step $i + 3$, the graph $G|_{\mathcal{T}^{i+4}}$ is connected again.

Finally, consider Cases D and E, depicted in Figures 4.7s and 4.7t. In these cases, the agent performs a tile shift from v to $v + DE$. Hence, when the tile at v is picked up, the node $v + DE$ is already tiled. The agent then moves below the next tile at $v + N$ via the nodes $v + NE$ and $v + NE + DNW$. Both of these nodes are adjacent to the tiled node $v + DE$ and to the node $v + N$, whose tile is shifted next. Therefore, the argument is analogous: $G|_{\mathcal{T}^{i+2} \cup \{p^{i+2}\}}$ and $G|_{\mathcal{T}^{i+3} \cup \{p^{i+3}\}}$ are connected, and after the final placement step the graph $G|_{\mathcal{T}^{i+4}}$ is connected again. $\square$

Auxiliary lemmas. We begin with two auxiliary lemmas about the bounding cylinder. The first shows that tiles are never placed outside the current bounding cylinder. The second implies that, even if the bounding cylinder shrinks during the execution, it remains aligned with the eastern side of the initial bounding cylinder.

Lemma 4.9. *If, during the execution of BUILD ICICLE, a tile is shifted from some node v to some node w , then there exist tiled nodes $u_x, u_y \in \mathcal{T}$ such that $x(w) = x(u_x)$ and $y(w) = y(u_y)$.*

Proof. We prove the claim by explicitly specifying tiled nodes u_x and u_y whose x - and y -coordinates coincide with those of w , respectively. Note that choosing $u_x = v$ or $u_y = v$ is valid, since the tile is placed at w before the tile at v is picked up. By

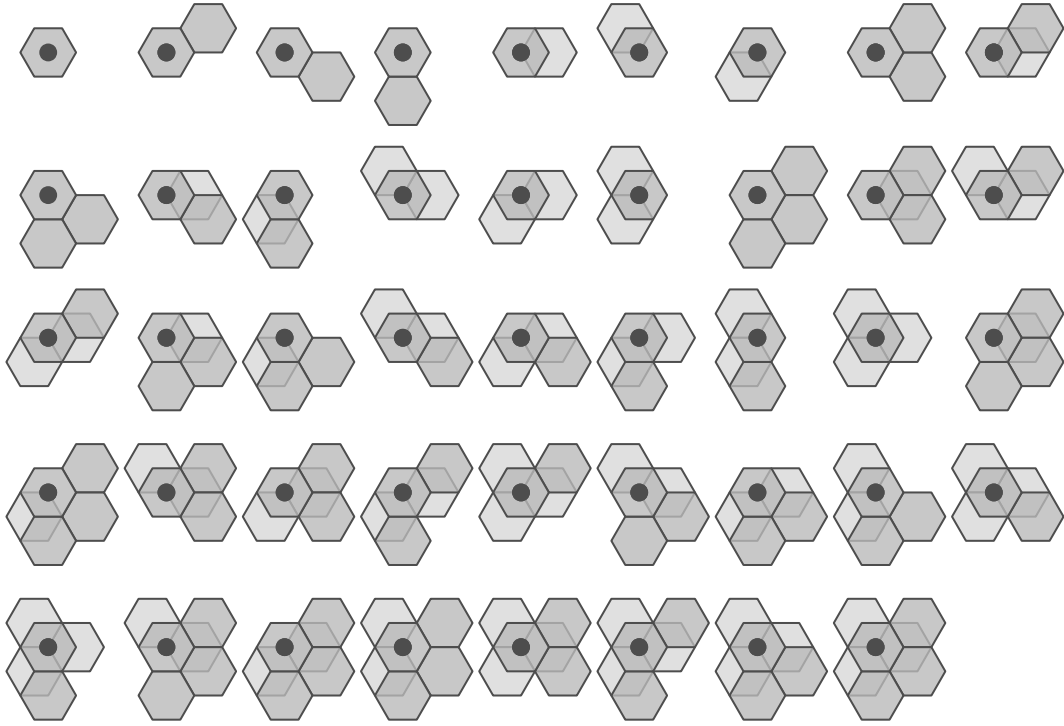


Figure 4.8: The 44 neighborhoods $N_T(v)$ not covered by Cases A–E. In each neighborhood, the agent can pick up the simply removable tile at v .

Observation 4.3, moving in direction N or S preserves the x -coordinate, moving in direction NW or SE preserves the y -coordinate, and moving in direction UNE or DSW preserves both coordinates.

First, suppose that the tile shift is performed during the *PROJECT* procedure. Then $w + UNE$ is tiled, so choosing $u_x = u_y = w + UNE$ proves the claim.

Next, suppose that the tile shift is performed during the *BUILDPAR* procedure. Then the tile is placed at a violating node in the sense of Definition 4.5. Hence, either w has tiled neighbors at N , NE , and SE , at S and SE , at NW and N , or at NW , SW , and S . In each case, we can choose a tiled node at N or S as u_x , and a tiled node at NW or SW as u_y . Hence, the claim follows.

It remains to consider tile shifts outside these two procedures. Every such shift occurs in one of Cases A–E. Let v be the northernmost node of the column considered there. Since the agent leaves the search phase at v , the node v has no tiled neighbor in directions UW , USE , UNE , NW , SW , or N . Thus, the only possible tiled neighbors of v are in directions DE , DNW , DSW , SE , NE , and S . Hence, there are $2^6 = 64$ possible neighborhoods. The 20 cases in which v is not simply removable are shown in Figure 4.7. For completeness, the remaining 44 cases, where v is simply removable, are shown in Figure 4.8. Thus, the case distinction is complete.

We now specify suitable choices of u_x and u_y for the 20 non-simply-removable cases, enumerated as in Figure 4.7. For (a)–(e), choose $u_x = w + N$ and $u_y = v$. For (f) and (g), choose $u_x = w + DNW$ and $u_y = v$. For (h)–(j), (l)–(n), and (q)–(r), choose $u_x = u_y = w + UNE$. For (k) and (o)–(p), choose $u_x = w + USE$ and $u_y = w + NW$.

Finally, consider cases (s) and (t). Here, the algorithm performs several tile shifts, namely from the nodes $v_1, \dots, v_k$ to the nodes $w_1 = v_1 + DE, \dots, w_k = v_k + DE$, in this order, where v_1 is the southernmost node from which a tile is shifted and $v_k = v$ is the northernmost node of the column. For each $i < k$, we may choose $u_y = v_{i+1}$, and for $i = k$, we choose $u_y = v_k + DNW$. In case (s), we may choose $u_x = v_k + NE$ for all i . In case (t), either $v_1 + S + DE$ or $v_k + SE$ is a valid choice for u_x . Indeed, one of these two nodes must be tiled for the first tile shift to occur at node v_1 in case (t).

Therefore, every tile shift performed by *BUILDICICLE* satisfies the claim. $\square$

Lemma 4.10. *For every $i \geq 0$, there exists a tiled node $v \in \mathcal{T}^i$ with $x(v) = 0$.*

Proof. Assume for contradiction that the claim is false, and let i be the first step such that no tiled node $u \in \mathcal{T}^i$ satisfies $x(u) = 0$. Recall that coordinates are normalized so that the minimum initial x -coordinate is 0. Hence, $i > 0$, and in step $i - 1$ the agent picks up the tile at the unique node $v \in \mathcal{T}^{i-1}$ with $x(v) = 0$. Let w be the node at which the agent places a tile before picking up the tile at v . We distinguish whether this tile shift occurs during the *PROJECT* procedure, during the *BUILDPAR* procedure, or in Cases A–E, and derive a contradiction in each case.

First, consider the *PROJECT* procedure. Between placing a tile at w and picking up the tile at v , the agent moves only in direction *UNE*. By Observation 4.3, this implies that $x(w) = x(v)$. Since the tile is placed at w before the tile at v is picked up, the node w is tiled at that moment. This contradicts that v is the unique tiled node with x -coordinate 0.

Next, consider the *BUILDPAR* procedure. Then either w is contained in the same column as v , or w lies strictly east of that column. In the first case, we again obtain $x(w) = x(v) = 0$, contradicting that v is the unique tiled node with x -coordinate 0. In the second case, some tiled node has negative x -coordinate, contradicting Lemma 4.9.

Finally, consider Cases A–E. In each of these cases, there is a tiled node u in direction *NE*, *SE*, or *DE* from v . Hence, $x(u) = -1$, again contradicting Lemma 4.9. $\square$

Structural properties of fragments. To measure progress within the bounding cylinder toward the east and south, we combine the x - and y -coordinates into a single value. During the procedure *BUILDPAR* and in Cases C–E, the y -coordinate of a tile may increase while its x -coordinate decreases. Although Lemma 4.9 implies that the bounding cylinder can never grow, it may shrink. In that case, Lemma 4.10 ensures that the new bounding cylinder remains aligned with the eastern side of the initial bounding cylinder $\mathfrak{C}(\mathcal{T}^0)$. This motivates the following linearization of the x - and y -coordinates with respect to the current bounding cylinder $\mathfrak{C}(\mathcal{T})$.

Let $y_{\max}^{\mathcal{T}}$ and $y_{\min}^{\mathcal{T}}$ be the maximum and minimum y -coordinates of nodes in $\mathfrak{C}(\mathcal{T})$, and let

$$h = y_{\max}^{\mathcal{T}} - y_{\min}^{\mathcal{T}} + 1$$

be the *height* of $\mathfrak{C}(\mathcal{T})$, that is, its extent in y -direction. For a node $v \in \mathfrak{C}(\mathcal{T})$, we define its *cylinder index* by

$$\gamma(v) = x(v) \cdot h + y(v) - y_{\min}^{\mathcal{T}}.$$

Thus, among all nodes of a fixed z -coordinate, the value $\gamma(v)$ enumerates the nodes of the bounding cylinder column by column from east to west, and within each column from south to north.

We next introduce three structural properties of fragments, referred to as P1–P3. Roughly speaking, P1 describes a fragment that is locally uppermost, P2 requires that the fragment covers all cylinder indices of tiled nodes, and P3 requires that the fragment forms a parallelogram that is flush with the southern, eastern, and northern sides of the bounding cylinder.

Definition 4.11. *Let $F \subseteq \mathcal{T}$ be a fragment.*

P1: *F is a platform if $\{v + X \mid v \in F, X \in \{UW, USE, UNE\}\} \cap \mathcal{T}$ is empty.*

P2: *F is covering if, for every node $v \in \mathcal{T}$, there exists a node $w \in F$ with $\gamma(w) = \gamma(v)$.*

P3: *F is an aligned parallelogram if, for every node $v \in F$, it holds that for all i with $\gamma(v) \geq i \geq 0$, there exists a node $w \in F$ with $\gamma(w) = i$.*

Alternative characterization of icicles. We can now use P1–P3 to give an alternative characterization of the icicle shape.

Definition 4.12. *A connected configuration $C = (\mathcal{T}, p)$ is an icicle if it contains a fragment F that satisfies P1–P3 and, for every node $v \in \mathcal{T} \setminus F$, the node $v + UNE$ contains a tile.*

This definition is equivalent to the earlier definition of an icicle from the introduction of this chapter. Indeed, since F is covering, every tiled node outside F lies somewhere in direction DSW or UNE from a tile of F , and since F is a platform, no tiled node can lie in direction UNE from it. Thus, F is precisely the uppermost parallelogram-shaped fragment of the icicle, and the condition on nodes in $\mathcal{T} \setminus F$ ensures that all remaining tiled nodes form towers in direction DSW from F .

First stage of the convergence proof. From this point onward, we no longer index configurations by individual steps of the execution. Instead, we consider only configurations in which the agent leaves the search phase. Such a configuration must eventually be reached, since moving upward increases the z -coordinate, while moving in directions SW , NW , or N strictly increases the cylinder index, and both quantities are bounded in any finite configuration. To simplify notation, we write $C^i = (\mathcal{T}^i, p^i)$ for the configuration in which the agent leaves the search phase for the i -th time.

Note that the sets of tiled nodes $\mathcal{T}^i$ and $\mathcal{T}^{i+1}$ need not be distinct for all i . Indeed, the agent may leave the search phase again after repositioning itself farther west or in a higher layer, without shifting a tile. Likewise, C^1 may coincide with the initial configuration if the agent initially starts on a tile from which it cannot move farther upward or in directions SW , NW , or N .

Let $\mathcal{P}^i$ denote the set of all platforms in configuration C^i , that is, all fragments satisfying P1. Consider the potential

$$\Phi^i = \sum_{v \in \mathcal{T}^i} \gamma(v) + |\mathcal{P}^i|.$$

We first show that this potential is monotone and bounded from below.

Lemma 4.13. *For every $i \geq 0$, it holds that $\Phi^i \geq \Phi^{i+1} \geq 0$. Moreover, if $\Phi^i = \Phi^{i+1}$, then either*

- (1) *no tile was shifted between steps i and $i + 1$, or*
- (2) *a fragment was projected between steps i and $i + 1$.*

Proof. Recall that we normalized the coordinates so that the minimum x -coordinate in the initial configuration is 0. Hence, by definition of the cylinder index, $\gamma(v) \geq 0$ for every initially tiled node $v \in \mathcal{T}^0$. By Lemma 4.9, this remains true for every tiled node $v \in \mathcal{T}^i$ and every $i \geq 0$. Since the number $|\mathcal{P}^i|$ of platforms is also non-negative, it follows that $\Phi^i \geq 0$ for all $i \geq 0$.

We now prove monotonicity. If $\mathcal{T}^i = \mathcal{T}^{i+1}$, then trivially $\Phi^i = \Phi^{i+1}$, and no tile was shifted between the i -th and $(i+1)$ -st time that the agent leaves the search phase. Hence, assume that $\mathcal{T}^i \neq \mathcal{T}^{i+1}$. Then at least one tile is shifted before the agent leaves the search phase for the $(i+1)$ -st time. We distinguish whether such a shift occurs during the *PROJECT* procedure, the *BUILDPAR* procedure, or in Cases A–E.

First, consider the *PROJECT* procedure, in which a fragment F is projected. For every $v \in F$, the tile at v is shifted to the first empty node somewhere in direction *DSW*. After the projection, every node in the set $\{v + \text{DSW} \mid v \in F\}$ is tiled, and all these nodes belong to the same fragment in configuration C^{i+1} . This fragment may or may not be a platform. Hence, the number of platforms can increase by at most one. On the other hand, it decreases by exactly one, since all nodes of F are empty in C^{i+1} and only platforms are projected. Therefore, $|\mathcal{P}^i| \geq |\mathcal{P}^{i+1}|$. Moreover, Observation 4.3 implies that each tile shift during the projection preserves the cylinder index, and thus

$$\sum_{v \in \mathcal{T}^i} \gamma(v) = \sum_{v \in \mathcal{T}^{i+1}} \gamma(v).$$

Consequently, $\Phi^i \geq \Phi^{i+1}$.

Next, consider the *BUILDPAR* procedure. Here, a tile is shifted from a simply removable node v of some fragment F to some node w . Since v is simply removable, picking up the tile at v does not disconnect F . Moreover, if v has tiled neighbors below, then these remain connected to F within $N_{\mathcal{T}}(v)$. Hence, picking up the tile at v cannot create a new platform. Placing a tile at w cannot create a new platform either, since w is adjacent to a node of F , and thus does not create a new fragment. Hence, the number of platforms does not increase. Moreover, following the column-wise traversal, w is either the node s of the column containing v , or a node strictly east of v . In both cases, $\gamma(w) < \gamma(v)$. Therefore, $\Phi^i > \Phi^{i+1}$.

Finally, consider Cases A–E. In Case A, a tile is shifted from v to $v + \text{SE}$. In Case B, it is shifted from v to $v + \text{SE} + \text{DSW}$. In Case C, it is shifted from v to $v + \text{DE}$. These shifts change the cylinder index by

$$\gamma(v + \text{SE}) = \gamma(v + \text{SE} + \text{DSW}) = \gamma(v) - h, \text{ and}$$

$$\gamma(v + \text{DE}) = \gamma(v) - h + 1.$$

Moreover, Figure 4.7 shows that $h > 1$ in each of these cases. Hence, each of these shifts strictly decreases the sum of cylinder indices.

The only case in which the number of platforms can increase is Case D, and then by at most one. However, in this case at least two tiles are shifted in direction DE , so the sum of cylinder indices decreases by at least $2h - 2 > 0$. Thus, this decrease more than compensates for at most one additional platform.

In Case E, the agent either enters the *PROJECT* procedure or shifts at least two tiles in direction DE . We have already shown that the former case yields $\Phi^i \geq \Phi^{i+1}$, while the latter yields $\Phi^i > \Phi^{i+1}$.

Thus, Φ decreases strictly in every case in which a tile is shifted, except during a projection, where it is non-increasing. Together with the case in which no tile is shifted, this proves the lemma. $\square$

The next lemma shows that once the agent is positioned in a fragment satisfying P1–P3, it remains in such a fragment thereafter. The following lemma shows that, starting from any configuration, the execution eventually either decreases the potential Φ or reaches such a fragment. In combination with Lemma 4.13, this implies that the agent eventually reaches a fragment satisfying P1–P3.

Lemma 4.14. *If $p^i \in F^i$ for a fragment F^i that satisfies P1–P3, then there exists a later step $j > i$ with $p^j \in F^j$ for some fragment F^j that also satisfies P1–P3. Moreover, between steps i and j , the agent starts in F^i and shifts only tiles of F^i .*

Proof. First, consider the case where F^i has height greater than one. Since F^i is a platform (P1), the agent cannot leave F^i to a higher layer. Moreover, since F^i is covering (P2), none of the Cases A–E can apply: each of these cases require a neighboring tile below that is not directly DSW of some node in F^i ; see Figure 4.7. Hence, the agent must enter the *BUILDPAR* procedure. If p^i is not westernmost within F^i , then the agent may re-enter the search phase when observing the westernmost column, and then enter the *BUILDPAR* starting from its westernmost column. Since F^i is an aligned parallelogram, it cannot encounter a violating node there and therefore proceeds to the *PROJECT* procedure. Thus, there exists a step $j > i$ such that between steps i and j , the agent always starts in F^i , only tiles of F^i are shifted, and in step j all tiles of F^i have been shifted out of F^i .

Now consider the case where F^i has height one. Then, at the westernmost node of F^i , the agent may enter Case B instead of the *BUILDPAR* procedure and shift the tile to the node in direction $SE + DSW$. The remaining tiles in F^i still form a parallelogram of height one, so the same case repeats. Eventually, the node in direction $SE + DSW$ is already tiled or only a single tile remains within F^i . In both cases, the agent enters the *BUILDPAR* procedure and subsequently proceeds to the *PROJECT* procedure, exactly as above. Hence, the same conclusion holds in this case.

Let

$$F = \{v + DSW \mid v \in F^i\},$$

and order the nodes of F by decreasing cylinder index as $v_1, \dots, v_k$, where v_1 has largest cylinder index. We show that every node of F is tiled in step j .

If between steps i and j tiles are shifted only as part of a projection, then this is immediate: each node of F is either already tiled or is the first empty node in

direction *DSW* of some node in F^i . Otherwise, the agent enters Case B repeatedly before the final projection. Then v_1 must already be tiled when Case B is applied for the first time. Subsequently, the agent fills the empty nodes of F in order $v_2, v_3, \dots$ until it reaches a node v_ℓ that is already tiled. At that point, the agent enters the *PROJECT* procedure and projects precisely the tiles in direction *UNE* of $v_\ell, \dots, v_k$. Hence, every node of F is tiled in step j .

Since F^i satisfies P2, the fragment containing F cannot contain a node whose cylinder index does not already occur in F^i . On the other hand, every cylinder index occurring in F^i also occurs in F . Hence, the fragment containing F is exactly F , and therefore F satisfies P2 and P3. Moreover, all nodes of F^i are empty in step j , so F also satisfies P1. Finally, since F is a platform, the agent cannot leave F to a higher layer. Therefore, $p^j \in F$, and with $F^j := F$ the claim follows. $\square$

Lemma 4.15. *For every $i \geq 0$, there exists a step $j > i$ such that*

- (1) $\Phi^i > \Phi^j$, or
- (2) $p^j \in F^j$ for some fragment F^j of C^j satisfying P1–P3.

Proof. Let F^i be the fragment containing the agent in configuration C^i , that is, $p^i \in F^i$. If F^i satisfies P1–P3, then the claim follows directly from Lemma 4.14. Hence, assume that F^i does not satisfy P1–P3.

If $\Phi^i > \Phi^{i+1}$, then the claim holds with $j = i + 1$. Thus, assume that $\Phi^i = \Phi^{i+1}$. By Lemma 4.13, no tile is shifted between steps i and $i + 1$, or the fragment containing the agent is projected.

If no tile is shifted, then the agent must have found a more western column or a fragment in a higher layer. In the former case, its cylinder index increases, and in the latter case, its z -coordinate increases. Since both quantities are bounded, this cannot happen indefinitely. Hence, eventually a tile is shifted. If any subsequent tile shift is performed outside of the *PROJECT* procedure, then Lemma 4.13 implies that the potential decreases strictly, and we are done. We may therefore assume for the remainder of the proof that every subsequent tile shift is performed as part of the *PROJECT* procedure.

Consider any fragment that is projected during this process. Every projection is preceded by an execution of *BUILDPAR* in which the agent does not encounter a neighboring tile above and does not encounter a violating node. Hence, the projected fragment is a platform and a parallelogram. We claim that it does not satisfy P2. Indeed, assume for contradiction that it does satisfy P2. By Lemma 4.10, there is always a tiled node with x -coordinate 0. Hence, by P2, the fragment contains a node with x -coordinate 0. Likewise, by P2, it contains a node with minimum y -coordinate and a node with maximum y -coordinate over all nodes in the current configuration. Since the fragment is a parallelogram, it therefore follows that it also satisfies P3, and thus all of P1–P3, contradicting our assumption. Therefore, every projected fragment during this process violates P2.

Let F be such a projected fragment. Since F violates P2, there exists a tiled node w whose cylinder index is not covered by F . By Lemma 4.8, connectivity is maintained throughout the execution. Hence, there is a path from a node of F to w that contains only tiled nodes. Since F is a platform, this path must leave F through

a node in a lower layer. Let u be the first node on such a path whose cylinder index is not covered by F . Then u lies below F and is adjacent to a node whose cylinder index is covered by F .

After the projection of F , every node directly DSW of a node in F is tiled. Hence, the next fragment F' in which the agent leaves the search phase contains all nodes directly DSW of F , and therefore $|F'| \geq |F|$. Moreover, since u is adjacent to a node whose cylinder index is covered by F , after finitely many further projections the agent reaches a fragment that contains at least one tiled node whose cylinder index was not covered by F . Thus, as long as the current fragment still violates P2, the size of the fragment in which the agent leaves the search phase increases strictly after finitely many projections.

Since every fragment has size at most n , this cannot continue indefinitely. Hence, eventually the fragment containing the agent satisfies P2. As argued above, every fragment projected during this process is a platform and a parallelogram. Any such fragment satisfying P2 also satisfies P3. Therefore, eventually the agent is positioned in a fragment satisfying P1–P3, which proves the claim. $\square$

Initially, there are at most n platforms, and the cylinder index of every tiled node is at most n^2 . Hence, $\Phi^0 = \mathcal{O}(n^3)$. Since Lemma 4.13 shows that Φ is non-increasing and bounded from below, it can decrease only finitely many times. Together with Lemmas 4.14 and 4.15, this implies that the execution eventually reaches a configuration in which the agent is positioned in a fragment satisfying P1–P3.

Second stage of the convergence proof. We now show that, once the agent is positioned in a fragment satisfying P1–P3, the execution converges to an icicle. For this purpose, we introduce a second potential Ψ^i . This potential counts the empty nodes inside the bounding box $\mathfrak{B}(\mathcal{T}^i)$ that have a tiled node somewhere in direction DSW . Formally, let

$$U^i = \{v \in \mathfrak{B}(\mathcal{T}^i) \setminus \mathcal{T}^i \mid v + k \cdot DSW \in \mathcal{T}^i \text{ for some } k > 0\},$$

and define $\Psi^i = |U^i|$.

Lemma 4.16. *Let $p^i \in F^i$, where F^i satisfies P1–P3. If $\Psi^i > 0$, then $\Psi^{i+1} < \Psi^i$.*

Proof. Since $\Psi^i > 0$, the set U^i is non-empty. We show that there exists a node $u^i \in U^i$ with $u^i \notin U^{i+1}$, and that every node in U^{i+1} already belongs to U^i . This implies $U^{i+1} \subsetneq U^i$, and therefore $\Psi^{i+1} < \Psi^i$.

Let $u^i \in U^i$ be a node of maximum z -coordinate. Since F^i satisfies P1 and P2, no node in the bounding box $\mathfrak{B}(\mathcal{T}^i)$ has z -coordinate larger than the common z -coordinate $z(F^i)$ of the fragment F^i . Hence, $z(u^i) \leq z(F^i)$.

We claim that every node of U^i lies strictly below F^i . Consider an empty node u with $z(u) = z(F^i)$. If there were a tiled node somewhere in direction DSW of u , then no node of F^i could have the same cylinder index as this node. This would contradict P2. Thus, every node of U^i lies strictly below F^i , that is, $z(u) < z(F^i)$ for all $u \in U^i$.

Let $w = u^i + k \cdot \text{UNE}$ be the node in F^i with $\gamma(w) = \gamma(u^i)$. Since u^i has maximum z -coordinate in U^i , every node $u^i + j \cdot \text{UNE}$ with $0 < j < k$ is tiled. By Lemma 4.14, the fragment F^i is projected, and in particular the tile at w is shifted to the first empty node in direction DSW , namely to u^i . Hence, u^i contains a tile in step $i + 1$, and therefore $u^i \notin U^{i+1}$.

It remains to show that no new node enters U^{i+1} . Let $u^{i+1} \in U^{i+1}$, and assume for contradiction that $u^{i+1} \notin U^i$. Since $u^{i+1} \in U^{i+1}$, there exists a node $v = u^{i+1} + k \cdot \text{DSW}$ that contains a tile in step $i + 1$ for some $k > 0$. Since $u^{i+1} \notin U^i$, one of the following two cases must hold.

In the first case, u^{i+1} contains a tile in step i . Hence, the tile at u^{i+1} is picked up between step i and $i + 1$. Let F^{i+1} be the fragment that the agent enters after projecting F^i and in which it subsequently leaves the search phase. By Lemma 4.14, the fragment F^{i+1} consists exactly of the nodes directly DSW of F^i , and all nodes in F^i are empty in step $i + 1$. Hence, it also satisfies P1–P3. In particular, F^{i+1} is projected as well, and therefore u^{i+1} lies strictly below F^{i+1} . Since F^{i+1} lies strictly below F^i , we obtain

$$z(u^{i+1}) < z(F^{i+1}) < z(F^i).$$

On the other hand, $u^{i+1} \in F^i$ implies $z(u^{i+1}) = z(F^i)$, a contradiction.

In the second case, v contains no tile in step i . Then v must have been tiled by the projection, so some tile was shifted from a node $w \in F^i$ to v , where v is the first empty node in direction DSW of w . Hence, $\gamma(w) = \gamma(v) = \gamma(u^{i+1})$. Since $u^{i+1} \notin U^i$, the node u^{i+1} is empty in step i and there is no tiled node somewhere in direction DSW of it. Therefore, u^{i+1} lies strictly between w and v on the line of constant cylinder index, contradicting that v is the first empty node in direction DSW of w .

Both cases yield a contradiction. Hence, every node of U^{i+1} already belongs to U^i . Therefore, $U^{i+1} \subsetneq U^i$, and so $\Psi^{i+1} < \Psi^i$. $\square$

By Lemma 4.14, once the agent reaches a configuration in which it is positioned in a fragment satisfying P1–P3, it remains in such a fragment thereafter. By the conclusion of the first stage, such a configuration is eventually reached. From that point onward, Lemma 4.16 shows that the potential Ψ^i decreases strictly until it becomes zero.

Hence, eventually there is no empty node in the bounding box that has a tiled node somewhere in direction DSW . Consequently, every tiled node that does not belong to the uppermost fragment satisfying P1–P3 has a tiled neighbor in direction UNE . By Definition 4.12, the resulting configuration is therefore an icicle, which yields the main convergence theorem.

Theorem 4.17. *For every initially connected configuration $C^0 = (\mathcal{T}^0, p^0)$ with $p^0 \in \mathcal{T}^0$, the execution of BUILDICICLE converges to an icicle.*

4.5. Termination Criterion and Runtime

Once the agent is positioned in a fragment satisfying P1–P3, it remains in such a fragment and thereafter performs only projections. If the configuration is already an icicle in the sense of Definition 4.12, then during each projection the agent encounters

every tiled node of the configuration. This observation is the basis for the following termination criterion.

Termination criterion. Whenever a projection is initiated, the agent assumes that the projected fragment is an icicle unless it observes a violation of Definition 4.12 during the projection. A violation is a tiled node v such that $v + \text{UNE} \notin F$ and $v + \text{UNE} \notin \mathcal{T}$, where F denotes the fragment in which the projection was initiated. If no violation is observed during the entire projection, then the agent terminates.

Note that an observed violation is not reconsidered, even if it disappears later during the same projection. For example, suppose that a tile is shifted from some node $w \in F$ to $v + \text{UNE}$ as part of the projection. When the agent observes the tiled node v , the node $v + \text{UNE}$ is still empty, so this is a violation of Definition 4.12. Afterwards, the tile is placed at $v + \text{UNE}$, and this violation disappears. However, the agent simply continues the projection routine and moves back to w in direction UNE to pick up the tile there. One could modify the projection routine to continue searching for further violations or to re-check whether the previously observed violation has disappeared. We avoid this additional bookkeeping and keep the termination criterion simple. The price is that the agent performs one additional projection after the configuration has already converged to an icicle.

Since the agent terminates only after a projection during which it can observe all tiled nodes, and only if Definition 4.12 is satisfied throughout that projection, correctness of the terminating algorithm follows.

Runtime. We analyze the runtime by partitioning the execution into three types of segments: projection segments, decreasing- x segments, and search segments. The first type consists of the traversal that identifies a fragment to be projected, followed by the execution of the projection routine.

There are two ways in which such a segment can arise. First, the agent may enter *BUILDPAR* and traverse a fragment that has the shape of a partially filled parallelogram. If this traversal reaches the empty node south of the easternmost column of the fragment, the agent initiates a projection. Second, the agent may reach Case E, where it traverses a single column whose northernmost tile is not simply shiftable. If this column forms a parallelogram of width one, the agent initiates a projection. In both cases, we group the preceding traversal and the subsequent projection routine into one segment.

We now define this segment formally. Consider an execution of *BUILDICICLE*, and suppose that the agent enters *PROJECT* in step j . Let $i \leq j$ be the last step before j in which the agent leaves the search phase. Let $k \geq j$ be the first step after j in which the agent enters the search phase again or terminates. Then the interval of steps $[i, k)$ is called a *projection segment*.

The second type of segment captures tile shifts that decrease the x -coordinate of a tile. Such segments do not overlap with projection segments, since a projection never changes the x -coordinate of a tile. Except for Case D and Case E, each such tile shift forms its own segment. In these two cases, multiple consecutive tile shifts are performed, and we group all of them into a single segment.

Formally, consider a tile shift from a node v to a node w with $x(w) < x(v)$, and suppose that the tile is placed at w in step j . Let $i \leq j$ be the last step before j in which the agent leaves the search phase, and let $k > j$ be the first step after j in which the agent enters the search phase again. Then the interval of steps $[i, k)$ is called a *decreasing- x segment*.

To obtain the third type of segment, we remove all projection segments and decreasing- x segments from the execution. We then group the remaining steps into maximal consecutive intervals. Each such interval starts and ends with a search phase. Moreover, by construction, none of the tile shifts in such an interval decreases the x -coordinate of a tile. The only tile shifts outside projections with this property are those performed in *BUILDPAR*, where the tile is placed at the empty node south of the column in which *BUILDPAR* was entered. Thus, every tile shift in such an interval decreases only the y -coordinate. We call these intervals *search segments*.

Formally, a maximal non-empty interval of steps $[i, k)$ is called a *search segment* if no step in this interval is contained in a projection segment or a decreasing- x segment.

Bounding the segments. We now bound the total length of the three types of segments. First, we show that the total number of decreasing- x segments and projection segments is $\mathcal{O}(n^2)$. Since search segments are maximal intervals between projection segments and decreasing- x segments, this also implies that there are only $\mathcal{O}(n^2)$ search segments. We then show that projection segments and decreasing- x segments together contain $\mathcal{O}(n^3)$ steps.

It remains to bound the search segments. For this, we separate tile shifts from all other steps. Each search segment contains $\mathcal{O}(n)$ steps outside of tile shifts, while the total number of steps spent shifting tiles over all search segments is $\mathcal{O}(n^3)$. Together with the bound on the number of search segments, this implies that all search segments together contain $\mathcal{O}(n^3)$ steps.

Lemma 4.18. *There are $\mathcal{O}(n^2)$ decreasing- x segments and projection segments.*

Proof. We first bound the number of decreasing- x segments. By Lemma 4.9 and the normalization of coordinates, the x -coordinate of every tile is bounded below by zero and above by n . As no tile shift ever increases the x -coordinate of a tile, each individual tile is shifted in at most n distinct decreasing- x segments. Since every decreasing- x segment contains at least one such tile shift and there are n tiles, the total number of decreasing- x segments is $\mathcal{O}(n^2)$.

Next, we bound the number of projection segments. We first consider projections that occur before the agent reaches a fragment satisfying P1–P3.

Every fragment that is projected is a platform, and such platforms can arise in three ways. They are either already present in the initial configuration, created when an execution of Case D disconnects a fragment, or obtained by projecting all tiles above an existing platform. We call platforms of the first kind *initial platforms* and platforms of the second kind *split platforms*. In the third case, one platform is replaced by another, which inherits its type.

We begin with initial platforms. At any time, there are at most n initial platforms. However, reducing the number of initial platforms by one may require $\mathcal{O}(n)$ projec-

tions. In the worst case, a fragment F forms the bottom of a tower, in the sense that for every node $v \in F$ all nodes $v + i \cdot \text{UNE}$ with $1 \leq i \leq j$ are tiled for some $j = \mathcal{O}(n)$. Then F is not a platform yet, but it inherits the type of an initial platform only after at least j fragments above F in direction UNE have been projected. Thus, the total number of projections of initial platforms is $\mathcal{O}(n^2)$.

We now consider split platforms. Let $v_1, \dots, v_k$ be the nodes whose tiles are shifted to nodes $w_1, \dots, w_k$ during an execution of Case D; see Figure 4.7s. Let F be the fragment containing $v_1, \dots, v_k$ before these shifts, and let F_1 and F_2 be the resulting fragments afterwards, that is,

$$F \setminus \{v_1, \dots, v_k\} = F_1 \cup F_2.$$

The number of platforms can increase only in two cases. Either F is already a platform, or all nodes of F that have a tiled neighbor above belong entirely to F_1 or entirely to F_2 . Therefore, one execution of Case D creates at most one split platform.

Without loss of generality, suppose that F_1 is projected before F_2 . After the projection of F_1 , all nodes in $\{v + \text{DSW} \mid v \in F_1\}$ belong to the same fragment as the nodes $w_1, \dots, w_k$. Moreover, this fragment is adjacent to F_2 and lies below it. Hence, if F_1 or F_2 is a split platform, then one additional projection suffices to reduce the number of platforms by one.

Each execution of Case D is contained in a decreasing- x segment. As shown above, there are $\mathcal{O}(n^2)$ decreasing- x segments, and thus $\mathcal{O}(n^2)$ split platforms are created. Together with the bound for initial platforms, this gives $\mathcal{O}(n^2)$ projections before the agent reaches a fragment satisfying P1–P3.

Once a fragment satisfying P1–P3 exists, at most n further projections are required until the configuration becomes an icicle, and one additional projection causes the agent to terminate. Indeed, the bounding boxes of the tiled nodes before and after a sequence of $n + 1 = \mathcal{O}(n)$ projections are disjoint, and, by the proof of Lemma 4.16, no new violation of the icicle shape can be created during these projections.

We have thus shown that the number of projection segments is $\mathcal{O}(n^2)$ as well, which concludes the proof. $\square$

Lemma 4.19. *Projection and decreasing- x segments contain $\mathcal{O}(n^3)$ steps in total.*

Proof. We first consider projection segments. Each such segment begins with an execution of *BUILDPAR* or with a column traversal in Case E. In both cases, the fragment that is projected is traversed fully. During an execution of *BUILDPAR*, each tile of this fragment is visited at most twice. During the column traversal in Case E, each tile is visited exactly once. Hence, this initial traversal takes $\mathcal{O}(n)$ steps.

During the subsequent projection, each node v of the projected fragment is visited at most three times. It is visited at most once while the agent moves to the northernmost tile of a column, once while it traverses the column in direction S , and once when it picks up the tile at v . Any tiled node outside the projected fragment is visited at most twice: at most once while the agent moves in direction DSW and at most once while it returns in direction UNE . Thus, each tiled node is visited at most a constant number of times during the projection. Consequently, each projection segment contains $\mathcal{O}(n)$ steps.

By Lemma 4.18, there are $\mathcal{O}(n^2)$ projection segments. Therefore, the total number of steps contained in projection segments is $\mathcal{O}(n^3)$.

It remains to consider decreasing- x segments. If such a segment corresponds to a tile shift in Cases A–C, then a tile is shifted in direction SE , DE , or $SE + DSW$, respectively. Each of these tile shifts takes $\mathcal{O}(1)$ steps. If it corresponds to an execution of Case D or Case E, then the agent first traverses k tiles of a column in direction s and then shifts all k tiles in direction DE . Since $k \leq n$, the total length of the segment is $\mathcal{O}(n)$.

Finally, consider a segment that corresponds to an execution of *BUILDPAR*. Suppose that, during this execution, the tile at v is shifted to a node w . Each tile of the fragment is visited at most twice before the tile is placed at w and at most once when it picks up the tile at v . Hence, this segment contains $\mathcal{O}(n)$ steps.

Thus, every decreasing- x segment contains $\mathcal{O}(n)$ steps. Since there are $\mathcal{O}(n^2)$ such segments by Lemma 4.18, their total length is $\mathcal{O}(n^3)$. Together with the bound for projection segments, this proves the lemma. $\square$

Lemma 4.20. *Each search segment contains $\mathcal{O}(n)$ steps outside of tile shifts.*

Proof. Let $[i, k)$ be an arbitrary search segment. We show that the total number of steps outside of tile shifts in the interval $[i, k)$ is $\mathcal{O}(n)$.

Consider a maximal subinterval $[i', k') \subseteq [i, k)$ during which the agent remains in a fixed layer L . Let

$$n_L = |L \cap \mathcal{T}|$$

be the number of tiled nodes in this layer. Since a search segment contains no projection, once the agent enters a higher layer, it cannot return to a lower layer within the same search segment. It is therefore enough to show that the number of steps outside of tile shifts in $[i', k')$ is $\mathcal{O}(n_L)$.

We prove this in two parts. First, we show that the total number of search-phase steps in $[i', k')$ is $\mathcal{O}(n_L)$. Second, we show that the total number of steps spent in *BUILDPAR* outside of tile shifts in $[i', k')$ is also $\mathcal{O}(n_L)$.

For the first part, consider a search phase within $[i', k')$. While the agent remains in the layer L , every search-phase step is in direction NW , SW , or N . Each of these moves strictly increases the cylinder index of the agent's position. Thus, within one search phase, the agent visits no tiled node twice.

Between two search phases in the same layer, the agent executes *BUILDPAR*. Since $[i, k)$ is a search segment, no tile shift in this interval decreases the x -coordinate. Thus, if such an execution of *BUILDPAR* shifts a tile, it places the tile at the empty node south of the column in which *BUILDPAR* was entered. In this case, the agent returns to the search phase one step farther south in the same column, and hence its cylinder index has decreased by exactly one. Such a shift is performed only if the target node has a neighboring tile in direction SE . Equivalently, the shift requires an empty node northwest of a tile in the eastern neighboring column. Therefore, over the whole subinterval $[i', k')$, the number of such decreases is bounded by the number of tiled nodes in L whose column is not locally westernmost, and hence by n_L .

All other executions of *BUILDPAR* that return to the search phase in the same layer move the agent to a node with larger cylinder index. Indeed, this happens when

the agent finds a more western column, or when it finds a more northern column with the same x -coordinate as the current column. The tiled nodes in L have exactly n_L distinct cylinder indices. Since the total number of unit decreases is $\mathcal{O}(n_L)$, the total number of search-phase steps in $[i', k']$ is $\mathcal{O}(n_L)$.

For the second part, we bound the steps in *BUILDPAR* outside of tile shifts. Consider an execution of *BUILDPAR* between steps i' and k' , and let c be the column in which this execution was entered.

Traversing c in direction s to find a more western column takes $\mathcal{O}(|c|)$ steps. Afterward, the agent enters the search phase at a node with larger x -coordinate. Within the same subinterval, it never enters the search phase at a node with smaller x -coordinate again. Hence, for a fixed column c , this happens at most once. Consequently, the additional steps caused by finding more western columns sum to $\mathcal{O}(n_L)$.

Finding a more northern column with the same x -coordinate as c takes $\mathcal{O}(|c'|)$ steps, where c' is the eastern neighboring column of c . Indeed, the agent traverses c once and c' at most twice, and in this case c' is larger than c . The agent then enters the northernmost column among all columns neighboring c' to the west. Within the same subinterval, its cylinder index decreases only if a tile is shifted to the empty node south of the column in which *BUILDPAR* was entered. Thus, if the agent enters c' again in the same subinterval, then at that point c' has only one western neighboring column. Hence, for a fixed column c' , this happens at most once. Consequently, the additional steps caused by finding more northern columns sum to $\mathcal{O}(n_L)$.

Finally, the subinterval $[i', k']$ may end with an execution of *BUILDPAR* during which the agent enters a higher layer. In such an execution, each tile in L is visited at most twice. Therefore, it contributes $\mathcal{O}(n_L)$ steps.

Altogether, the steps spent in *BUILDPAR* outside of tile shifts during $[i', k']$ sum to $\mathcal{O}(n_L)$. Together with the bound on search-phase steps, this proves that the number of steps outside of tile shifts in $[i', k']$ is $\mathcal{O}(n_L)$.

Since the agent never returns to a lower layer within the same search segment, the subintervals for different layers are disjoint. Moreover, no tile changes its z -coordinate in a search segment, so no tile is counted in two different layers. It follows that summing over the set $\mathcal{L}$ of all layers visited during $[i, k]$ gives

$$\sum_{L \in \mathcal{L}} \mathcal{O}(n_L) = \mathcal{O}(n).$$

Therefore, the search segment $[i, k]$ contains $\mathcal{O}(n)$ steps outside of tile shifts. $\square$

Lemma 4.21. *Tile shifts during search segments take $\mathcal{O}(n^3)$ steps in total.*

Proof. By definition, every search segment is disjoint from all projection segments and decreasing- x segments. Thus, the only tile shifts in search segments are those performed in *BUILDPAR* where a tile is placed at the empty node south of the column in which *BUILDPAR* was entered.

We first bound the number of steps of one such tile shift. Let c be the column in which the corresponding execution of *BUILDPAR* was entered, let v be the northernmost node of c , and let w be the empty node south of c . During the tile shift, the agent only visits nodes u with

$$\gamma(w) \leq \gamma(u) \leq \gamma(v).$$

Each such node is visited at most twice. The first visit occurs while the agent moves in direction s within the column c before placing the tile at w . The second occurs while the agent moves in direction N within c after placing the tile at w and before picking up the tile from v . Thus, this tile shift takes $\mathcal{O}(\gamma(v) - \gamma(w))$ steps.

By Lemma 4.9, the cylinder index of every tile is bounded below by zero. Moreover, initially every tile has cylinder index at most n^2 . Since every tile shift considered here moves a tile to a node with smaller cylinder index, each tile can contribute at most $\mathcal{O}(n^2)$ steps to such tile shifts over the entire execution. Summing over all n tiles gives a total of $\mathcal{O}(n^3)$ steps. $\square$

By Lemma 4.18, the number of search segments is $\mathcal{O}(n^2)$. Together with Lemma 4.20, this gives $\mathcal{O}(n^3)$ steps outside of tile shifts over all search segments. The tile shifts within search segments contribute another $\mathcal{O}(n^3)$ steps by Lemma 4.21. Hence, all search segments together contain $\mathcal{O}(n^3)$ steps. Combining this with the bound for projection and decreasing- x segments from Lemma 4.19, we obtain the main runtime bound.

Theorem 4.22. *BUILDICICLE has a runtime of $\mathcal{O}(n^3)$ steps.*

Tightness of the runtime bound. We conclude by showing that the runtime bound from Theorem 4.22 is asymptotically tight. To this end, we define a family of configurations whose runtime is $\Omega(n^3)$.

For simplicity, assume that n is divisible by 3. Consider a configuration consisting of a parallelogram formed by two columns of height $\frac{n}{3}$. From the northernmost tile of its eastern column, attach a line of $\frac{n}{3}$ tiles in direction SE . Let the agent initially start at the northwesternmost tile of the configuration. See Figure 4.9 for an illustration.

For the following argument, we fix the bounding box of the initial configuration. Accordingly, unlike in the rest of this chapter, the term *column* refers here to a maximal set of nodes of equal x -coordinate in this initial bounding box. These columns are ordered from west to east and denoted by $c_1, c_2, \dots, c_{\frac{n}{3}+2}$. Note that some of these columns may become empty during the execution.

We show that this configuration requires $\Omega(n^3)$ steps. Throughout the execution, the northernmost tile of the westernmost non-empty column is safely shiftable. Hence, whenever the agent leaves the search phase, it enters the *BUILDPAR* procedure.

Moreover, for every $k \in \{0, \dots, \frac{n}{3} - 1\}$, immediately after the columns $c_1, \dots, c_k$ become empty, the columns c_{k+1} and c_{k+2} each contain exactly $\frac{n}{3}$ tiles, while the column c_{k+3} contains exactly $k + 1$ tiles. For $k = 0$, this means that initially c_1 and c_2 each contain exactly $\frac{n}{3}$ tiles, while c_3 contains exactly one tile. Now assume that it holds for some $k \geq 0$. Then each of the next $\frac{n}{3} - (k + 1)$ tile shifts moves one tile from c_{k+1} to c_{k+3} . Afterwards, the column c_{k+3} contains exactly $\frac{n}{3}$ tiles, while c_{k+1} contains exactly $k + 1$ tiles. The remaining $k + 1$ tiles of c_{k+1} are then shifted into c_{k+4} , which contained only a single tile before. Consequently, the column c_{k+1} becomes empty, and the same statement holds with $k + 1$ in place of k .

It follows that $\frac{n}{3} - 1 = \Omega(n)$ columns must become empty before the configuration becomes a parallelogram. Each of these columns contains exactly $\frac{n}{3} = \Omega(n)$ tiles.

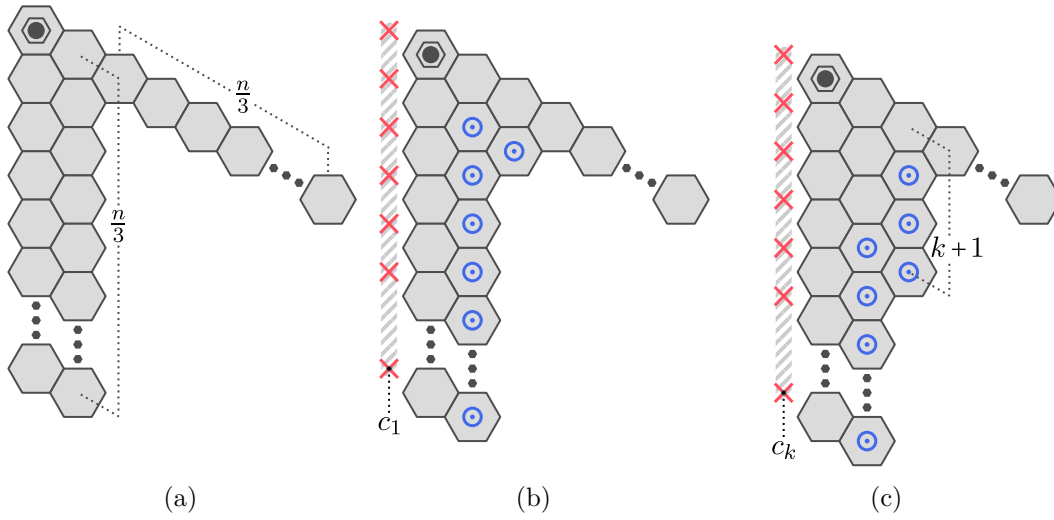


Figure 4.9: (a) A configuration that requires $\Omega(n^3)$ steps for reconfiguration, consisting of two columns of height $\frac{n}{3}$ and $\frac{n}{3}$ additional tiles in the northernmost row. (b) The configuration after column c_1 becomes empty. (c) The configuration after column c_k becomes empty; at this point, column c_{k+3} contains $k+1$ tiles. Crosses and circles indicate the last $\frac{n}{3}$ tiles that were picked up and placed, respectively.

Furthermore, each individual tile shift takes $\Omega(n)$ steps. Indeed, if c_{k+1} is the first non-empty column, then during a tile shift, the agent traverses at least column c_{k+2} , which contains exactly $\frac{n}{3}$ tiles.

Altogether, $\Omega(n)$ columns become empty throughout the execution, each of these columns requires $\Omega(n)$ tile shifts, and each tile shift takes $\Omega(n)$ steps. Therefore, the total runtime is $\Omega(n^3)$. Together with the upper bound from Theorem 4.22, this shows that the runtime bound is asymptotically tight.

4.6. Experimental Analysis

While our algorithm matches the runtime bound of the 3D line-formation algorithm [HRS20], the icicle offers clear structural advantages over a line. Its diameter can be as small as $\mathcal{O}(n^{1/3})$, whereas a line always has diameter n . Of course, no improvement is possible when the initial configuration already resembles a line. On the other hand, we conjecture that if the initial diameter is as small as $\mathcal{O}(n^{1/3})$, which is optimal in 3D, then our algorithm increases the diameter by at most a factor of $\mathcal{O}(n^{1/3})$.

To support this conjecture, we conducted simulations on random configurations whose tiles were initially contained in a sphere of radius $\mathcal{O}(n^{1/3})$. In total, we ran 12,250 simulations of the icicle-formation algorithm. For each value of n with $10 \leq n < 500$, we sampled 25 random configurations as follows. We repeatedly selected empty nodes uniformly at random from a sphere of radius $4n^{1/3}$ and placed tiles on them until a connected component of size at least n was formed. Afterwards, all tiles outside that component were removed, the agent was placed on a tile chosen uniformly at random from the component, and the algorithm was simulated until

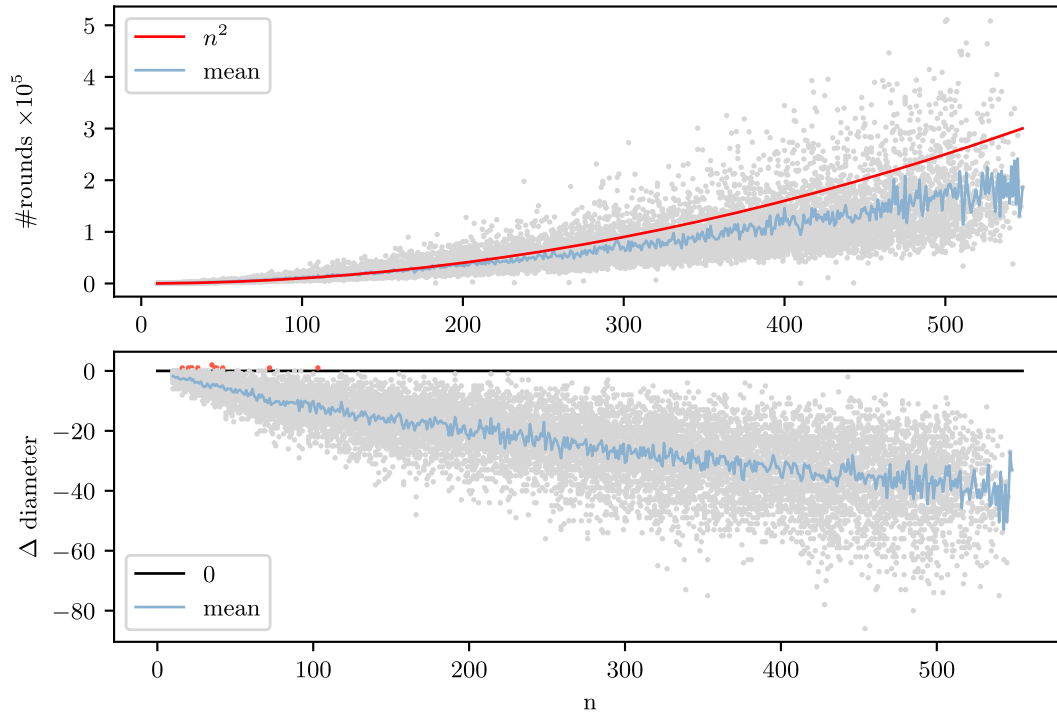


Figure 4.10: Results of 12,250 simulations on random configurations of size between 10 and 550. The upper plot shows the number of steps until termination. The lower plot shows the difference in diameter between the input and output configurations.

termination. The radius $4n^{1/3}$ was chosen to generate configurations that remain relatively compact, while being noticeably sparser and more varied than almost completely filled balls. Smaller radii tended to produce very dense configurations with few gaps and little structural diversity.

For each run, we recorded the runtime and the difference in diameter between the input and output configurations; the results are shown in Figure 4.10. Because of the random generation process, configurations of size larger than 500 also occurred, although less frequently. More precisely, the average number of sampled configurations was approximately 24.4 for sizes at most 450, but only approximately 15.4 for sizes larger than 450. This discrepancy explains the noticeable increase in variance near the threshold of 500.

We have shown an asymptotic upper bound of $\mathcal{O}(n^3)$ rounds and a matching lower bound. Over the sampled range, however, the runtime measurements appear much closer to n^2 than to n^3 . This is likely due to the fact that the sampled configurations are initially rather compact. Instances in which the diameter increases, shown as red points above the x -axis in the lower plot, are rare, and disappear as the configuration size grows. Overall, the simulations therefore suggest that, on average, the algorithm tends to decrease rather than increase the diameter. A constant multiplicative increase can already be explained by the geometric difference between a ball-shaped input configuration and a parallelepiped-shaped output configuration. However, there are configurations for which the diameter increase is asymptotically larger.

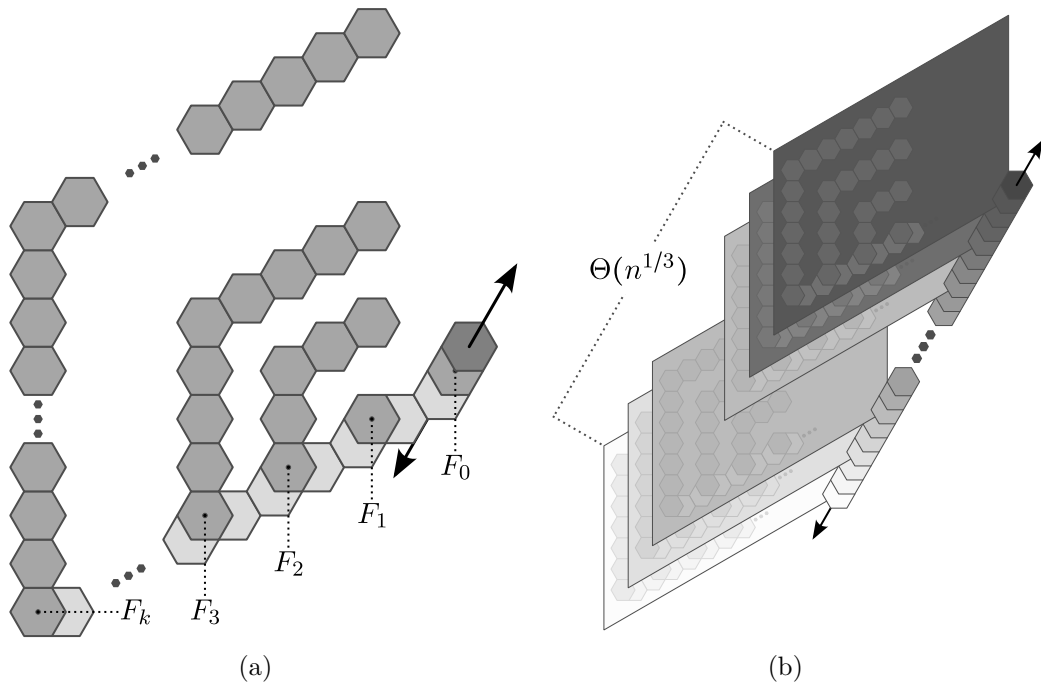


Figure 4.11: Configuration used for the conjectured worst-case increase in diameter. (a) The three-layer configuration C , whose middle layer contains the fragments $F_0, \dots, F_k$. (b) A configuration obtained by stacking $\Theta(n^{1/3})$ copies of C in direction UNE . Consecutive copies are connected by a tower at F_0 , as indicated by the arrows.

A conjectured worst case for the diameter increase. We now describe a family of configurations that we conjecture to be worst-case in this respect. The construction is illustrated in Figure 4.11.

We first consider the three-layer configuration C shown in Figure 4.11a. Its middle layer contains $k = \Theta(n^{1/3})$ fragments $F_1, \dots, F_k$, ordered from east to west, where each fragment F_i has size $\mathcal{O}(i)$ and the agent initially starts at a node $p^0 \in F_1$. In addition, there is a fragment F_0 of size one east of the initial agent position. For every i with $0 \leq i < k$, the bounding box of F_i contains no node of F_{i+1} . Hence, the agent builds and projects parallelograms in the order $F_1, \dots, F_k$.

Moreover, for every $i \geq 1$, the bounding box of F_i contains the node p^0 . Therefore, whenever one of the fragments $F_1, \dots, F_k$ is projected, a tile is projected from p^0 in direction DSW . Altogether, this happens k times before the agent traverses the lower layer and eventually reaches the fragment F_0 , where it can move upward again.

Now consider the configuration obtained by stacking $\Theta(n^{1/3})$ copies of C in direction UNE , as shown in Figure 4.11b. Its initial diameter is $\mathcal{O}(n^{1/3})$. Indeed, for any two nodes v and w , there is a path from v to w consisting of at most seven straight segments of tiles and two single steps, and each such segment has length $\mathcal{O}(n^{1/3})$.

During the execution of the icicle-formation algorithm, however, some tile at node p^0 is projected once in each of the $\Theta(n^{1/3})$ copies and, within each copy, it is projected $\Theta(n^{1/3})$ times. Consequently, some tile at p^0 is projected a total of $\Theta(n^{2/3})$ times. It follows that the resulting icicle has depth $\Theta(n^{2/3})$, and therefore diameter $\Theta(n^{2/3})$.

4.7. Conclusion

In this chapter, we introduced the icicle as a compact canonical target shape in the 3D hybrid model and presented a single-agent algorithm that transforms any initially connected configuration into an icicle within $\mathcal{O}(n^3)$ steps. We proved correctness of the algorithm, established termination, and showed that the runtime bound is asymptotically tight. Moreover, throughout the execution, the agent and all tiles remain within the bounding cylinder of the initial configuration, which makes the algorithm particularly appealing in space-constrained environments.

The icicle is useful not only because of its compactness, but also because its regular local structure admits a local algorithm and, in particular, locally detectable safely removable tiles. As pointed out in the previous chapter, safely removable tiles cannot in general be identified using only local information in arbitrary connected configurations. This severely hinders shape-reconfiguration tasks.

While the depth of the bounding box, and hence also the diameter, may increase in some cases, our experiments indicate that the diameter tends to decrease on average. At the same time, the worst-case increase in diameter is not yet fully understood. We conjecture that it is bounded by a factor of $\mathcal{O}(n^{1/3})$, and we presented a family of configurations that achieves exactly this increase and that we believe to be worst-case in this respect.

Several directions for future work remain open. Most notably, it would be interesting to determine the exact worst-case increase in diameter and to design variants of the algorithm with stronger diameter guarantees. Another natural direction is to extend the construction to a multi-agent setting. Since the algorithm proceeds through several tightly coupled phases, coordinating multiple agents appears to be a challenging and non-trivial problem.

Coating Surfaces with Tiles

A potential application of programmable matter is the coating of an object's surface with an additional functional layer. Such a layer may protect the underlying object from its environment, equip it with sensing or communication capabilities, or alter physical properties such as friction, conductivity, or reflectivity. At very small scales, one may also envision biomedical applications in which such a coating isolates harmful biological structures.

The coating problem is closely tied to the geometry of the object's surface. If a programmable matter system reaches the object through a single point of contact, connectivity constraints force the particles to spread from that point. Thus, the surface geometry determines whether computationally simple particles can navigate the surface, which parts are reachable, and whether many particles must pass through narrow bottlenecks. These challenges become even more pronounced in the hybrid model considered here, where a single agent must traverse the surface repeatedly while carrying tiles one by one.

In this chapter, we study the coating of a surface with tiles in the hybrid model. As in the previous chapter, our goal is to design an efficient algorithm under severe local restrictions. Instead of reconfiguring a tile configuration into a target shape, we incrementally remove its tiles and use them to cover the surface of a given object. The tile configuration therefore acts as a depot and should have a regular structure that is easy for the agent to traverse and retrieve tiles from. For example, the icicle introduced in the previous chapter can serve as such a depot.

The main difficulty is that the agent has to explore the surface to find empty nodes where it can place a tile, while preserving a route back to the depot, the only source of additional tiles. This is particularly challenging under the memory restrictions of the hybrid model. Indeed, arbitrary graph exploration is impossible for a single agent with constant memory alone. In related settings, overcoming this limitation requires additional placeable markers. This motivates separating the combinatorial aspects of surface traversal from the geometric details of the hybrid model.

We therefore formulate the problem on an abstract *surface graph*. Its nodes represent positions on the surface of an object, and its edges represent adjacencies along that surface. We assume that the surface graph is a triangulation of a closed three-dimensional surface; see Figure 5.1. Moreover, it is equipped with a fixed embedding in $\mathbb{R}^3$ in which the edges have only constantly many possible orientations, and the neighborhood of each node forms a chordless cycle.

The graph has a designated depot node, which initially contains the agent and a sufficient supply of tiles. This node is an abstraction; in the hybrid model, it may correspond to the attachment point of a regular tile structure. The goal is to transport one tile from the depot to every other node of the surface graph.

For surface graphs of degree at most six, we present a single-agent algorithm that solves the coating problem in $\mathcal{O}(n^2)$ steps. We also show that this runtime is asymptotically optimal, even compared with agents that have global knowledge and unrestricted memory.

We then return to the FCC lattice graph and study how this algorithm can be applied there. In that setting, the object is represented by a connected subset of nodes, and the surface graph is induced by the nodes adjacent to this subset. Although this subgraph is not necessarily a triangulation with the properties required above, we show that the algorithm can be emulated directly for a restricted class of objects.

More generally, we show that for a surface graph of arbitrary degree Δ satisfying the assumptions above, the algorithm can be emulated in $\mathcal{O}(\Delta^2 n^2)$ steps if the model is strengthened slightly. In particular, we assume $2^{2\Delta}$ distinguishable types of passive tiles. These types allow us to encode a constant amount of information at a node. Since Δ is constant in the three-dimensional hybrid model, this yields a quadratic-time coating algorithm.

Problem Definition. For the first part of this chapter, we consider the hybrid model on a graph $H = (V_H, E_H)$, called the *surface graph*, instead of on the FCC lattice graph G . Thus, the agent has the same capabilities and is subject to the same constraints as in Chapter 2, except that adjacency is defined by H rather than by G . In particular, the agent may move only along edges of H , each node of H can hold at most one tile, and each tile occupies at most one node at a time. The only exception is a designated depot node, which initially holds the agent and a sufficient supply of tiles.

We assume that H is a triangulation of a closed three-dimensional surface, equipped with a fixed embedding in $\mathbb{R}^3$ in which edges have only constantly many possible orientations. This ensures that a constant-memory agent can distinguish incident edges by their local orientations. We further assume that the neighborhood of each node is a *chordless simple cycle*, that is, the subgraph $H|_{N(v)}$ contains exactly one simple cycle for every $v \in V_H$. This assumption arises naturally when the triangles of the triangulation are chosen sufficiently small. For the algorithm in this part of the chapter, we additionally assume that every node of H has degree at most six. Since the agent is restricted to nodes of H , which represent positions on the surface of an object, the set consisting of the tiles, the agent, and the object remains connected throughout the execution. Thus, connectivity is always preserved in this setting and need not be handled explicitly.

In the FCC lattice graph, V_H should be understood as the set of all nodes adjacent to a connected subset of nodes representing the object to be coated. For now, however, we work in the more general surface-graph setting and return to this connection later in the chapter.

A *configuration* is defined analogously to the hybrid model on G . It is a pair $C = (\mathcal{T}, p)$, where $\mathcal{T} \subseteq V_H$ denotes the set of all tiled nodes and $p \in V_H$ denotes the agent's position. Let $C^0 = (\mathcal{T}^0, p^0)$ denote the initial configuration, where $\mathcal{T}^0 = \{p^0\}$ and the depot is located at the agent's initial position p^0 .

Remark 5.1. *As in the previous chapter, superscripts generally refer to step numbers and may be omitted when they are clear from the context.*

An algorithm solves the *coating problem* on H if its execution yields a sequence of configurations

$$C^0 = (\mathcal{T}^0, p^0), \dots, C^T = (\mathcal{T}^T, p^T)$$

such that $\mathcal{T}^T = V_H$, each configuration C^t for $1 \leq t \leq T$ is obtained from C^{t-1} by a single action of the agent, and the agent terminates in step T .

Chapter Outline. We begin with related work on coating problems and closely related tasks in Section 5.1. We then present our algorithm for coating general surface graphs in Section 5.2. Afterwards, we analyze the algorithm in Section 5.3. In Section 5.4, we make the connection to the FCC lattice and show how the algorithm can be applied there, including the construction for surface graphs of arbitrary constant degree. Finally, we conclude the chapter in Section 5.5.

5.1. Related Work

In this section, we discuss related work with a focus on coating and related geometric operations on a given configuration. The coating problem asks for the construction of a tight enclosing layer around a fixed object. This also makes other enclosing structures, such as bounding boxes or convex hulls, relevant.

In the Amoebot model, the coating layer is formed by the moving particles themselves. Derakhshandeh et al. present a universal coating algorithm that uniformly coats objects of arbitrary shape using anonymous particles with constant-size memory and only local interactions [Der+17]. Daymude et al. analyze the runtime of this algorithm, proving a linear upper bound in the number of asynchronous rounds together with a matching lower bound [Day+18]. Traversat proposes a coating algorithm for the three-dimensional Amoebot model that runs in $\mathcal{O}(nR)$ asynchronous rounds, where R is the length of the longest chain of connected particles [Tra20].

In the SILBOT model, Navarra and Piselli give an optimal deterministic algorithm for coating in $\Theta(n^2)$ rounds under the assumption of common chirality [NP24]. They later improve this result by considering coating with only one axis agreement [NP25].

Related coating problems have also been considered in modular robotics and swarm robotics. Thalamy et al. study scaffold-based reconfiguration in the 3D Catom model, where an internal scaffold is constructed first and then coated with an outer layer of modules [Tha+19; Tha+20; TPB20]. Cheraghi et al. study decentralized coating algorithms for robot swarms that move as close to an object as possible using only local communication [CWG20]. Another paper considers a leader-based approach in which a designated robot first scans the object and then coordinates the placement of the remaining robots, with particular emphasis on cave-shaped objects [CG20].

Bounding boxes have been studied in the CADbot model, where a finite automaton moves on the square grid and may freely create or remove tiles at its current position. Fekete et al. develop a toolbox of operations including bounding, counting, copying, reflecting, rotating, and scaling shapes, without preserving connectivity of the configuration [Fek+21a]. In particular, they give an algorithm for constructing a bounding box in time $\mathcal{O}(wh \max\{w, h\})$, where w and h denote the width and height of the configuration. Niehs et al. study recognition and reconfiguration of

arbitrary cellular structures by simple robots and demonstrate the corresponding procedures in simulation and on hardware [Nie+20]. Fekete et al. then provide the corresponding theoretical treatment of connected reconfiguration, including procedures for determining bounding boxes and scaling shapes [Fek+22]. In particular, they match the earlier runtime for bounding-box construction, but assume that the agent has access to one placeable marker.

A related problem in the Amoebot model is convex hull formation. Daymude et al. study how a particle system can form the convex hull of an object in $\mathcal{O}(B)$ asynchronous rounds, where B denotes the length of the object's boundary [Day+20]. Within the same asymptotic runtime, they also form a restricted-orientation hull that has a smaller enclosed area.

5.2. Coating Algorithm for Surface Graphs

In this section, we present our coating algorithm for the surface graph H . Throughout the section, the agent operates only on H , and all notation is with respect to H .

As for the FCC lattice graph, for a subset $U \subseteq V_H$ we use $d_U(v, w)$, $N_U(v)$, and $\deg_U(v)$ as shorthand for $d_{H|U}(v, w)$, $N_{H|U}(v)$, and $\deg_{H|U}(v)$. We further write $\mathcal{E} := V_H \setminus \mathcal{T}$ for the set of empty nodes.

For a node set $W \subseteq V_H$, we define the *boundary* of W as

$$B(W) := \left(\bigcup_{w \in W} N(w) \right) \setminus W.$$

Thus, $B(W)$ is the set of nodes outside W that are adjacent to at least one node in W . For $W = \{w\}$, we simply write $B(w)$.

We first give a high-level description of the algorithm and introduce the terminology needed for its formal specification. Finally, we present the algorithm itself.

5.2.1. High-Level Description

The main challenge is to explore H to find an empty node where a tile can be placed, while maintaining a route back to the depot, the only source of additional tiles. Exploring arbitrary graphs of diameter D and degree Δ requires $\Omega(D \log \Delta)$ memory bits [Fra+05]. Even in the more restricted setting of plane labyrinths, that is, grid graphs in the two-dimensional integer lattice, constant-memory exploration requires two placeable markers [Hof81; BK78].

Our agent has only constant memory and it cannot place markers. Tiles themselves cannot simply serve as markers, since every node of H must eventually be tiled and the agent cannot distinguish individual tiles. Instead, our approach is to place tiles strategically so that the remaining empty nodes form a narrow tunnel leading back to the depot. These empty nodes serve as implicit markers.

The agent traverses this tunnel by following the left-hand and right-hand rules, as commonly done in labyrinth traversal. That is, it moves while consistently keeping contact with tiled nodes either on its left or on its right. The tunnel allows the agent to explore H while preserving a route back to the depot. The key idea is that an

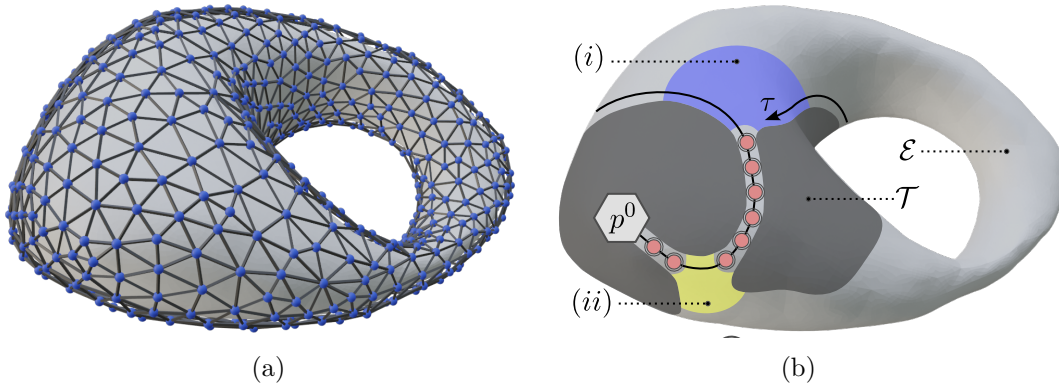


Figure 5.1: (a) A triangulation H . (b) The simple prefix τ starts at the depot p^0 and follows the boundary of the tiled nodes $\mathcal{T}$. Marked nodes are shown as circles. The region labeled (i) is the *search region* explored by the agent. In the region labeled (ii), τ is adjacent to empty nodes that are not marked. The agent explores the search region to find an overlap with this exposed part of τ . If such an overlap exists, the agent can place a tile at a marked node without disconnecting $H|_{\mathcal{E}}$.

empty node becomes part of the tunnel once the tiled nodes in its neighborhood are disconnected. This motivates the following definition.

Definition 5.2. *An empty node $v \in \mathcal{E}$ is marked if $H|_{N_{\mathcal{T}}(v)}$ is disconnected, and unmarked otherwise. A node v marks a node w if placing a tile at v turns w into a marked node. Similarly, v unmarks a node w if placing a tile at v turns w into an unmarked node.*

At a high level, the agent follows the left-hand rule along the boundary of the tiled nodes until it finds a node where it can place its carried tile. It then returns to the depot by following the right-hand rule, picks up a tile, and repeats.

When following the left-hand or right-hand rules, the agent cannot leave the connected component of $H|_{\mathcal{E}}$ that contains its current position. It is therefore crucial that every tile placement preserves the connectivity of $H|_{\mathcal{E}}$. Otherwise, the agent either ends up in a connected component without access to the depot, or it retains access to the depot but can no longer reach all empty nodes.

A naive way to preserve connectivity is to never place a tile at a marked node. However, this strategy works only if the surface represented by H is simply connected, that is, if it contains no holes. Indeed, a marked node is not necessarily a cut node of $H|_{\mathcal{E}}$. If the surface contains holes, then the naive strategy can reduce $H|_{\mathcal{E}}$ to a graph in which every node is marked. Since we allow surfaces with arbitrarily many holes, this graph can contain arbitrarily many cycles and may therefore be impossible to explore. Hence, any coating algorithm must eventually place a tile at a marked node.

In our algorithm, we maintain a simple path τ of empty nodes, called the *simple prefix*, that contains all marked nodes and is completely contained in the boundary $B(\mathcal{T})$ of the tiled nodes; see Figure 5.1b. The marked nodes on τ need not be connected, that is, there may be several sections where τ is exposed to empty nodes that are not marked. However, we ensure that consecutive marked parts of τ are sufficiently close to each other. We formalize this below using the notion of *segments*.

During each traversal of the simple prefix, the agent visits all marked nodes exactly once. It does so by following the left-hand rule until there are no more marked nodes in the segment at its current position. It then explores a constant-size neighborhood, called the *search region*. We define search regions so that they contain all nearby marked nodes, or more precisely, all segments in a local neighborhood.

If the agent finds a marked node in this search region, then it has found one of the exposures of τ mentioned above that was already visited before. Equivalently, there is a marked node that lies on a cycle of $H|_{\mathcal{E}}$. Placing a tile at such a node does not disconnect $H|_{\mathcal{E}}$.

Although this traversal visits all marked nodes, the agent does not necessarily traverse the entire simple prefix. For example, in Figure 5.1b, the subpath of the simple prefix following the topmost marked node is not traversed, since the next segment after that node contains no marked node.

5.2.2. Additional Terminology

We first introduce the terminology needed to describe the agent's boundary traversal and its local search before each tile placement.

Anchors. The agent maintains a direction pointer to a tiled neighbor. When the agent is at an empty node p , this pointer refers to a tiled node $a(p) \in N(p)$, called the *anchor* of p . In the analogy of wall-following in a labyrinth, $a(p)$ is the node on which the agent's hand is currently placed.

Initially, the agent is positioned at the depot p^0 . It moves to an arbitrary neighbor and sets its anchor to p^0 . Afterwards, it can follow the left-hand rule by moving to the first empty node v in the clockwise order of $N(p)$, starting at $a(p)$. The new anchor is the last tiled node encountered between $a(p)$ and v in that order; see Figure 5.2a. The right-hand rule is defined analogously, using counterclockwise order. Note that the anchor does not necessarily change after every move. Since we assume $N(p)$ to be chordless, the cyclic order of the nodes in $N(p)$ is uniquely defined.

We write L and R for the directions to the next node according to the left-hand rule and the right-hand rule, respectively. Thus, if the agent is at p , it moves to $p + L$ when following the left-hand rule and to $p + R$ when following the right-hand rule.

Boundary traversal. Let $s \in N(p^0)$ be a dedicated starting node chosen arbitrarily during the initialization. We define the *boundary path* $\sigma = (\sigma_0, \dots, \sigma_m)$ as the path obtained by a left-hand-rule traversal along the boundary $B(\mathcal{T})$ that starts and ends at $s = \sigma_0 = \sigma_m$.

The path σ is not necessarily a simple cycle. Indeed, during a full traversal of σ , the agent may visit the same marked node multiple times, with different anchors on different visits; see, for example, the node σ_j in Figure 5.2b. To avoid ambiguity, $a(\sigma_i)$ refers to the anchor after moving from σ_{i-1} to σ_i . For a node v that occurs on σ , $a(v)$ refers to the anchor at the first occurrence of v on σ .

The *segment* $S(\sigma_i)$ of a node σ_i contains all nodes that can be reached from σ_i by following the left-hand and right-hand rules while keeping the anchor fixed at $a(\sigma_i)$;

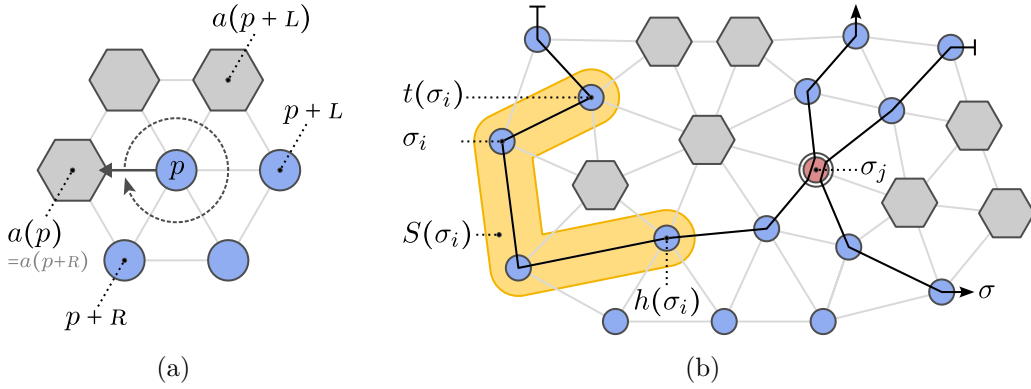


Figure 5.2: (a) Illustration of the anchor update while following the left-hand and right-hand rules. Empty nodes are depicted as disks, while tiled nodes are depicted as hexagons. (b) The segment $S(\sigma_i)$, tail $t(\sigma_i)$, and head $h(\sigma_i)$ of a node $\sigma_i \in \sigma$ in an example configuration. The node σ_j is marked, which we depict as a red disk with a double outline. Thus, the boundary path σ contains σ_j more than once.

see Figure 5.2b. Intuitively, $S(\sigma_i)$ consists of the nodes that touch the anchor $a(\sigma_i)$ from the same side as σ_i .

A node $\sigma_j \in S(\sigma_i)$ is a *successor* of σ_i if $j > i$, and a *predecessor* of σ_i if $j < i$. For example, the node σ_i in Figure 5.2b has two successors and one predecessor. We call the node without a successor in $S(\sigma_i)$ the *head* of σ_i and denote it by $h(\sigma_i)$. Similarly, we call the node without a predecessor in $S(\sigma_i)$ the *tail* of σ_i and denote it by $t(\sigma_i)$. Thus, $h(\sigma_i)$ and $t(\sigma_i)$ are the endpoints of the maximal part of the boundary traversal during which the anchor remains fixed at $a(\sigma_i)$.

We can now formally introduce the path mentioned in the high-level description. Let $\tau = (\tau_0, \dots, \tau_l)$ be the longest prefix of σ that is simple. That is, $\tau_i = \sigma_i$ for every $0 \leq i \leq l$, the nodes $\tau_0, \dots, \tau_l$ are pairwise distinct, and either $l = m$, or σ_{l+1} already occurs in τ . We call τ the *simple prefix*. This is the path on which the agent moves back and forth after each tile placement. Since τ is a prefix of σ , the definitions of segments, heads, and tails also apply to its nodes.

Remark 5.3. *Following the convention introduced in Chapter 2, we slightly abuse notation and treat paths as sets when referring to their contents. For example, for a node v and a set of nodes W , we write $v \in \tau$ if v occurs on τ , and $W \subseteq \tau$ if every node in W occurs on τ .*

Search region. Finally, we introduce the *search region* explored by the agent before each tile placement. Let p and q be adjacent empty nodes. The search region $R(p, q)$ is a local neighborhood of empty nodes, defined as if q were tiled; see Figure 5.3. It consists of all segments that contain some empty node reachable from p in at most three steps without passing through q or through a tiled node. Since q is treated as tiled, any segment that would otherwise contain q is split at q , and only the parts that contain such a reachable node are included.

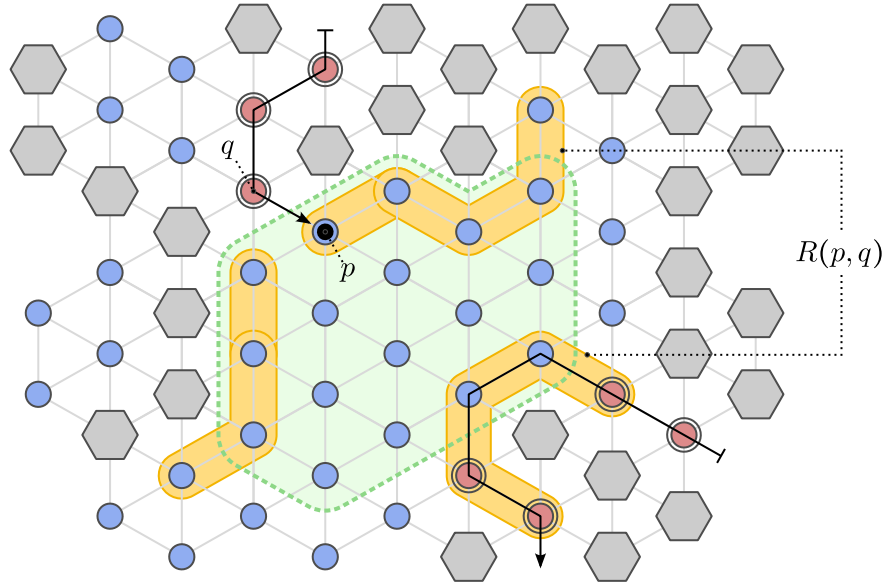


Figure 5.3: The search region $R(p, q)$ consists of the depicted segments. The agent is at node p , depicted as a black dot, and the arrows show the prefix of τ up to p . The green outline shows the nodes at distance at most three from p in $H|_{\mathcal{E} \setminus \{q\}}$. For visual clarity, the graph is drawn with degree exactly six at every node.

Definition 5.4. Let $p \in \mathcal{E}$ and let $q \in N_{\mathcal{E}}(p)$ be an empty neighbor of p . Let $\mathcal{T}_q := \mathcal{T} \cup \{q\}$ and $\mathcal{E}_q := \mathcal{E} \setminus \{q\}$. For $v \in \mathcal{E}_q$ and $w \in N_{\mathcal{T}_q}(v)$, let $S_q(v, w)$ denote the segment of v with respect to anchor w , after treating q as tiled.

The search region $R(p, q)$ is defined as

$$R(p, q) := \bigcup_{\substack{v \in \mathcal{E}_q \\ d_{\mathcal{E}_q}(p, v) \leq 3}} \bigcup_{w \in N_{\mathcal{T}_q}(v)} S_q(v, w).$$

5.2.3. The Algorithm

We now describe the algorithm *COATSURFACE* in detail. It consists of an initialization followed by two alternating phases, called the *search phase* and the *fetch phase*. The initialization chooses the starting node of the boundary traversal and initializes the agent's anchor. In the search phase, the agent determines where to place the tile it currently carries. In the fetch phase, the agent returns to the depot, picks up a new tile, and moves back along the simple prefix to the next position at which the search phase starts. After the initialization, the agent alternates between the search phase and the fetch phase until every node contains a tile.

Initialization. During the initialization, the agent picks up a tile at the depot p^0 and moves to an arbitrary node $s \in N(p^0)$. It stores the direction of p^0 relative to s , so that it can recognize s later by the adjacent depot. It then initializes $a(s) \leftarrow p^0$ and $skipSearch \leftarrow \text{false}$. Recall that s is the first node of both the boundary path

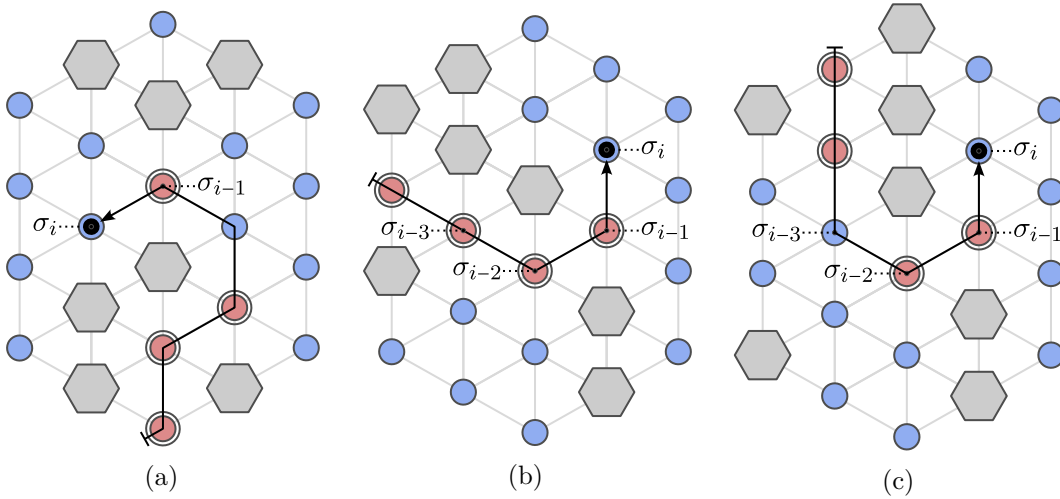


Figure 5.4: Examples in which the agent places a tile at a marked node in the search phase. The tile is placed at σ_{i-1} in (a), and at σ_{i-2} in (b) and (c). The flag *skipSearch* is set to true only in case (c), since σ_{i-3} is neither marked nor equal to s . As a result, the agent places a tile at σ_{i-3} in the next search phase.

and the simple prefix. The variable *skipSearch* is a flag indicating whether the exploration of the search region is skipped in the next search phase. This flag is needed to maintain a crucial invariant, which we prove in the analysis. After the initialization, the agent performs one step according to the left-hand rule and enters the search phase.

Search phase. The search phase is always entered at a node p that is neither marked nor equal to s . Moreover, $p + R$ is the last node of the simple prefix that is marked or equal to s . In each search phase, the agent places its carried tile either directly at p or at one of the two preceding nodes of the simple prefix. After placing the tile, the agent enters the fetch phase.

The position of the next tile is determined as follows. If *skipSearch* = true, then the agent places the tile at p and resets *skipSearch* to false. The agent also places the tile at p if the search region $R(p, p + R)$ contains no node that is marked or equal to s . Otherwise, the agent has found a marked node or the starting node inside the search region. This means that $R(p, p + R)$ contains a node of the simple prefix that was visited before p in the boundary traversal. Equivalently, the agent has detected a cycle in $H|_{\mathcal{E}}$ on which it can safely place a tile at a marked node.

It remains to specify where the agent places the tile in this case. The main concern is that placing a tile at a marked node cuts a cycle of $H|_{\mathcal{E}}$. If this placement marks a new node on the wrong side of this cut, then this newly marked node may later be mistaken for evidence of another cycle. To avoid this, the algorithm may place the tile one step further back along the simple prefix.

Let the agent's current position on the boundary path be $p = \sigma_i$. Then σ_{i-1} and σ_{i-2} are the two preceding nodes in the simple prefix; see Figure 5.4. If placing a tile at σ_{i-1} does not mark any node, then the agent places the tile at σ_{i-1} . Otherwise, it

places the tile at σ_{i-2} . In the latter case, the flag *skipSearch* is set to true whenever placing a tile at σ_{i-2} marks a node and σ_{i-3} is neither marked nor equal to s . The flag then ensures that the agent places a tile directly at σ_{i-3} in the next search phase. After this additional placement, the subsequent search phase again starts immediately after the last node of the simple prefix that is marked or equal to s . We prove this property as part of the analysis.

Fetch phase. In the fetch phase, the agent follows the right-hand rule until it reaches s . It recognizes s using the adjacent depot and the direction stored during the initialization. The agent then moves to the depot node p^0 , picks up a new tile, and returns to s . If s has no empty neighbors, then every node except s already contains a tile. In this case, the agent places its final tile at s and terminates. Otherwise, the agent follows the left-hand rule along the simple prefix until it reaches the next start position for the search phase. More precisely, it continues while its current position is marked or equal to s , or while the segment of its current position contains a marked successor. It then enters the next search phase.

Pseudocode. The corresponding pseudocode is given in Algorithm 4. Lines 1–5 implement the initialization. Afterwards, the agent repeatedly executes the main loop in lines 6–22. Within each iteration, lines 7–15 correspond to the search phase, and lines 16–22 correspond to the fetch phase.

In the pseudocode, p denotes the current position of the agent, and M denotes the set of marked nodes. The symbols L and R denote the next directions according to the left-hand rule and the right-hand rule, respectively. As in the previous chapter, each movement command updates the agent’s position p immediately. Thus, the two tests in lines 13 and 14 are evaluated at the node reached by the preceding movement command. In these lines, *p marks a node* is used in the sense of Definition 5.2.

The condition involving successors in the fetch phase requires the agent to inspect the current segment, since a node may have several successors. Similarly, the tests involving M , whether placing a tile would mark a node, and the exploration of the search region $R(p, p + R)$ are local tests. Since the degree of H is at most six and the search region has constant radius, these tests involve only constantly many nodes and can be performed with constant memory in constant time.

5.3. Analysis

Throughout this section, we keep the structural assumptions on the surface graph from the problem definition. In particular, H is a triangulation of a closed three-dimensional surface, its edges have only constantly many possible orientations, and the neighborhood of each node is a chordless simple cycle.

Coatable configurations. We analyze *COATSURFACE* for a slightly more general class of initial configurations than in the problem definition. Let $C^0 = (\mathcal{T}^0, p^0)$ be an initial configuration with $p^0 \in \mathcal{T}^0$, where the depot is located at p^0 and contains at least $|V_H| - |\mathcal{T}^0|$ tiles. In the problem definition, p^0 is the only initially tiled node.

Algorithm 4: *COATSURFACE*

```

1 pick up tile
2 move to an arbitrary node  $s \in N(p^0)$ 
3 store the direction of  $p^0$  w.r.t.  $s$ 
4  $a(s) \leftarrow p^0$ ;  $skipSearch \leftarrow \text{false}$ 
5 move  $L$ 
6 while true do
7   if  $skipSearch$  then
8     | place tile;  $skipSearch \leftarrow \text{false}$ 
9   else if  $R(p, p+R) \cap (M \cup \{s\}) = \emptyset$  then
10    | place tile
11   else
12     | move  $R$ 
13     | if  $p$  marks a node then move  $R$ 
14     | if  $p$  marks a node and  $p+R \notin M \cup \{s\}$  then  $skipSearch \leftarrow \text{true}$ 
15     | place tile
16   while  $p \neq s$  do
17     | move  $R$ 
18   move to  $p^0$ ; pick up tile; move to  $s$ 
19   if  $N_{\mathcal{E}}(s) = \emptyset$  then
20     | place tile; return
21   while  $p \in M \cup \{s\}$  or  $S(p)$  contains a marked successor of  $p$  do
22     | move  $L$ 

```

Here, we allow $\mathcal{T}^0$ to contain additional initially tiled nodes. We also relax the degree assumption for these nodes: initially empty nodes must still have degree at most six, whereas initially tiled nodes may have any constant degree. This more general setting will be useful when applying the algorithm to the FCC lattice graph.

Definition 5.5. An initial configuration $C^0 = (\mathcal{T}^0, p^0)$ is coatable if

- (a) every initially empty node has degree at most six,
- (b) the depot has an empty neighbor, that is, $N_{\mathcal{E}}(p^0) \neq \emptyset$,
- (c) the set of marked nodes is empty, that is, $M^0 = \emptyset$, and
- (d) the subgraph $H|_{\mathcal{E}^0}$ induced by the initially empty nodes is connected.

The initial configurations from the problem definition are special cases of coatable configurations. Indeed, in the problem definition every node of H has degree at most six, so every initially empty node has degree at most six. Moreover, the only initially tiled node is p^0 , so the depot-neighbor condition is immediate. Every empty node has at most one tiled neighbor and is therefore unmarked initially. Finally, since the neighborhood of each node forms a simple cycle, H has no cut node. Thus, $H|_{\mathcal{E}^0} = H|_{V_H \setminus \{p^0\}}$ is connected.

Proof structure. Unless stated otherwise, all notions defined from the set of tiled nodes are understood with respect to the current step t of the execution, that is, with respect to $\mathcal{T}^t$. For a set of nodes $W \subseteq V_H$, let $\|W\|$ denote the number of connected components in $H|_W$.

We aim to maintain the following five properties as invariants.

Definition 5.6. P1: *Marked nodes occur only on the simple prefix, that is, $M \subseteq \tau$.*

P2: *For every marked node, the tail of its segment is marked or equal to s , that is, $t(v) \in M \cup \{s\}$ for every $v \in M$.*

P3: *The subpath of τ from s to the last marked node on τ induces no cycle in H . That is, for any $i < j \leq k$ with $\tau_k \in M$, if $\tau_j \in N(\tau_i)$, then $j = i + 1$.*

P4: *The subgraph induced by the empty neighbors of any marked node contains exactly two connected components, that is, $\|N_{\mathcal{E}}(v)\| = 2$ for every $v \in M$.*

P5: *There is no marked node, or τ contains an unmarked node different from s whose segment has no successor. That is, $M = \emptyset$, or there is an i such that $\tau_i \notin M \cup \{s\}$ and $h(\tau_i) = \tau_i$.*

Observe that all five properties hold initially, since $M^0 = \emptyset$. The proof proceeds in three steps. First, we show that P5 guarantees progress after the agent picks up a tile at the depot: the fetch phase reaches a node at which the next search phase can start, and the agent then either returns to the depot or terminates. Second, we show that $H|_{\mathcal{E}}$ remains connected as long as P1–P4 are maintained. Third, we prove that the invariants are maintained throughout the execution of *COATSURFACE*, except for a temporary violation explicitly handled by the flag *skipSearch*.

Together, these steps imply that the agent repeatedly returns to the depot after placing a tile, while preserving reachability of all remaining empty nodes. Since each iteration places one tile and V_H is finite, the algorithm eventually reaches a step in which s has no empty neighbor. At that point, connectivity implies that no other empty node remains, and after placing the final tile at s , all nodes contain a tile.

Progress between depot visits. We first show that P5 is sufficient for progress between two visits to the depot. Whenever the agent picks up a tile at the depot, this tile is placed and the agent either returns to the depot or terminates.

Lemma 5.7. *Suppose that P5 holds in a step t in which the agent picks up a tile at p^0 . Then there is a step $t^+ > t$ in which the agent either enters p^0 again or terminates. Moreover, between steps t and t^+ , the agent places this tile.*

Proof. After picking up a tile at the depot, the agent returns to s . If $N_{\mathcal{E}}(s) = \emptyset$, then it places this tile at s and terminates. Otherwise, the agent follows the left-hand rule during the fetch phase. It continues moving as long as its current position is s , is marked, or its segment contains a marked successor.

By P5, either $M = \emptyset$, or there exists a node τ_i on τ that is neither marked nor equal to s and satisfies $h(\tau_i) = \tau_i$. In the first case, the agent leaves the fetch phase immediately after leaving s . In the second case, the node τ_i has no marked successor

in its segment, so the agent leaves the fetch phase at or before reaching τ_i . In both cases, the agent leaves the fetch phase after traversing a prefix of τ .

Let P be the prefix of τ traversed by the agent from s until it leaves the fetch phase, and let p be the last node of P . In the subsequent search phase, the agent places its tile either at p or at one of the two nodes immediately preceding p on P . Hence, after the tile placement, the path from s to the agent's current position is still a prefix of P and therefore simple. Since the agent obtained this path by following the left-hand rule, switching to the right-hand rule traverses it in reverse. Therefore, the agent reaches s and then moves to p^0 .

Thus, there is a step $t^+ > t$ such that the tile picked up in step t is placed before step t^+ , and in step t^+ the agent either enters p^0 again or terminates. $\square$

In the following, let t be a step in which the agent picks up a tile at p^0 . Let t^+ and t^- denote the next and previous step, respectively, in which the agent picks up a tile at p^0 or terminates, if such a step exists. Between two consecutive tile placements, all notions defined from the set of tiled nodes remain unchanged. Thus, when analyzing the effect of a tile placement, it is enough to refer to the steps t , t^+ , and t^- rather than to the exact step in which the tile is placed.

Remark 5.8. *To keep notation light, we may write a superscript $+$ or $-$ instead of t^+ or t^- . For example, $\mathcal{E}^+$ denotes $\mathcal{E}^{t^+}$, the set of empty nodes in step t^+ .*

Connectivity of empty nodes. We next analyze how tile placements affect the connectivity of $H|_{\mathcal{E}}$. The following auxiliary lemma identifies the point at which the search phase starts; we use it both here and later when proving that the invariants are maintained.

Lemma 5.9. *Suppose that P2 holds in step t and that, between steps t and t^+ , the agent leaves the fetch phase at some node τ_i . Then τ_{i-1} is the last node v on τ with $v \in M \cup \{s\}$.*

Proof. In the fetch phase, the agent starts at $s = \tau_0$ and follows the left-hand rule as long as its current position is marked, is equal to s , or its segment contains a marked successor. Thus, if it leaves the fetch phase at τ_i , then $i > 0$. Moreover, τ_{i-1} is marked or equal to s , and $S(\tau_i)$ contains no marked successor of τ_i . In particular, its head $h(\tau_i)$ is not marked.

Since τ is simple, it contains s only as its first node. Assume for contradiction that there is a marked node after τ_{i-1} . Let τ_k be the first such node. Since $S(\tau_i)$ contains no marked successor of τ_i , we have $\tau_k \notin S(\tau_i)$.

We trace the sequence of tails backwards along τ , starting from τ_k and ending at s . Let this sequence be $(v_0, \dots, v_m)$, where $v_0 = \tau_k$, $v_m = s$, and $v_j = t(v_{j-1})$ for every $1 \leq j \leq m$. By P2, this sequence is well-defined and, inductively, every v_j with $0 < j < m$ is marked.

The agent's anchor changes only *after* moving from a node w to a node w' , where w has no further successor in its current segment, i.e., $h(w) = w$. Consequently, w is the tail of w' , so $w = t(w')$. Hence, for every node u on τ , we have

$$h(t(u)) = t(u).$$

Since τ_i lies between s and τ_k on τ , the previous observation implies that $h(\tau_i)$ appears as a node in this sequence of tails. That is, there is an index j with $0 < j < m$ such that $v_j = h(\tau_i)$.

As argued above, $h(\tau_i)$ is not marked, which contradicts that every v_j with $0 < j < m$ is marked. Therefore, no marked node occurs after τ_{i-1} on τ . Since $\tau_{i-1} \in M \cup \{s\}$, it is the last node on τ with this property. $\square$

Using this auxiliary lemma, we now show that the first four invariants are sufficient to preserve connectivity of $H|_{\mathcal{E}}$ between steps t and t^+ .

Lemma 5.10. *Suppose that $H|_{\mathcal{E}}$ is connected and that P1–P4 hold in step t . Then $H|_{\mathcal{E}^+}$ is connected.*

Proof. There are three cases in which the agent places a tile. First, it may place the final tile at s in the fetch phase. This happens only if $N_{\mathcal{E}}(s) = \emptyset$, and therefore cannot disconnect $H|_{\mathcal{E}}$. Second, the agent may place a tile at an unmarked node during the search phase. By Definition 5.2, this also cannot disconnect $H|_{\mathcal{E}}$. It remains to consider the third case, in which the agent places a tile at a marked node during the search phase.

By Lemma 5.9, the agent enters the search phase at a node τ_i such that τ_{i-1} is the last node $w \in \tau$ with $w \in M \cup \{s\}$. The agent places a tile at a marked node only if the search region contains s or a marked node, that is,

$$R(\tau_i, \tau_{i-1}) \cap (M \cup \{s\}) \neq \emptyset.$$

In this case, it places the tile at one of the two nodes τ_{i-1} and τ_{i-2} . By P1 and Lemma 5.9, every node in $M \cup \{s\}$ occurs on τ at or before τ_{i-1} . Thus, $R(\tau_i, \tau_{i-1})$ contains a node of τ that was already visited during the preceding fetch phase.

Let τ_j be the last node of τ with $j < i - 1$ that is contained in $R(\tau_i, \tau_{i-1})$. Let P be a shortest path from τ_i to τ_j in $H|_{R(\tau_i, \tau_{i-1})}$. Then

$$C = P \circ (\tau_{j+1}, \tau_{j+2}, \dots, \tau_{i-1})$$

is a simple cycle in H , where $\circ$ denotes path concatenation.

It remains to show that placing a tile at τ_{i-1} or τ_{i-2} does not disconnect $H|_{\mathcal{E}}$. Since τ_{i-2} , τ_{i-1} , and τ_i are consecutive nodes of the boundary path, τ_i is the first empty node in clockwise order around $N(\tau_{i-1})$ starting at the anchor $a(\tau_{i-1})$, and τ_{i-2} is the first empty node in counterclockwise order. Since $a(\tau_{i-1})$ is tiled and τ_{i-1} is marked, these two nodes lie in different connected components of $H|_{N_{\mathcal{E}}(\tau_{i-1})}$. Analogously, if τ_{i-2} is marked, then τ_{i-3} and τ_{i-1} lie in different connected components of $H|_{N_{\mathcal{E}}(\tau_{i-2})}$.

In either case, the cycle C contains a node from each of the two connected components identified above. By P4, every empty neighbor of the node at which the tile is placed belongs to one of these two components. Hence, after placing the tile, all empty neighbors of this node lie in the same connected component of $H|_{\mathcal{E}^+}$. Thus, placing the tile does not increase the number of connected components. Since $H|_{\mathcal{E}}$ is connected, it follows that $H|_{\mathcal{E}^+}$ is connected. $\square$

Maintenance of the invariants. To show that P1–P5 are maintained, we first prove two auxiliary statements.

Lemma 5.11. *If v and w have a common tiled neighbor, then placing a tile at v cannot increase $\|N_{\mathcal{E}}(w)\|$, the number of connected components in $H|_{N_{\mathcal{E}}(w)}$.*

Proof. The claim is trivial if $v \notin N(w)$. Thus, assume that $v \in N(w)$. Since H is a triangulation, the edge $\{v, w\}$ is contained in precisely two triangular faces. Let u_1 and u_2 be the third nodes of these faces. Since $N(v)$ and $N(w)$ are chordless simple cycles, these are the only nodes adjacent to both v and w . Hence, $N(v) \cap N(w) = \{u_1, u_2\}$, and u_1, v , and u_2 are consecutive nodes on $N(w)$.

By assumption, v and w have a common tiled neighbor. Thus, at least one of u_1 and u_2 is tiled. Hence, $\|N_{\mathcal{T}}(w) \cup \{v\}\| \leq \|N_{\mathcal{T}}(w)\|$. By $\mathcal{E} = V_H \setminus \mathcal{T}$, the sets $N_{\mathcal{E}}(w)$ and $N_{\mathcal{T}}(w)$ are complementary in $N(w)$. Since $N(w)$ is a chordless simple cycle, they induce the same number of connected components. Therefore,

$$\|N_{\mathcal{E}}(w) \setminus \{v\}\| = \|N_{\mathcal{T}}(w) \cup \{v\}\| \leq \|N_{\mathcal{T}}(w)\| = \|N_{\mathcal{E}}(w)\|.$$

Thus, placing a tile at v cannot increase $\|N_{\mathcal{E}}(w)\|$. $\square$

We can now characterize the local structure around any marked node v for which placing a tile at v marks another node. Since v is marked, $H|_{N_{\mathcal{T}}(v)}$ contains at least two connected components, and, by the complement argument above, so does $H|_{N_{\mathcal{E}}(v)}$. If every connected component of $H|_{N_{\mathcal{E}}(v)}$ had size at most two, then every node in $N_{\mathcal{E}}(v)$ would share a common tiled neighbor with v . By Lemma 5.11, placing a tile at v could then not mark any node. Hence, one connected component of $H|_{N_{\mathcal{E}}(v)}$ has size at least three. By the degree bound, v has at most six neighbors. Thus, counting yields the following corollary.

Corollary 5.12. *Let $v \in M$. If placing a tile at v marks a node, then $H|_{N_{\mathcal{T}}(v)}$ contains two connected components of size one, and $H|_{N_{\mathcal{E}}(v)}$ contains two connected components of sizes one and three, respectively.*

It remains to show that P1–P5 are maintained from a depot visit in step t to the following depot visit in step t^+ . The only special case occurs when the flag *skipSearch* is set; in this case, the invariants are restored at the next depot visit after step t^+ , which we denote by t^{++} .

Lemma 5.13. *Suppose that P1–P5 hold in step t , and suppose that the agent does not terminate in step t^+ . Then either P1–P5 hold in step t^+ , or the flag *skipSearch* is true in step t^+ and P1–P5 hold in step t^{++} .*

Proof. We prove the lemma by distinguishing the type of node at which the tile is placed between steps t and t^+ . Since the agent does not terminate in step t^+ , the tile is placed during the search phase. We distinguish whether this node is unmarked, marked but does not mark another node, or marked and marks another node. In the last case, we further distinguish whether the preceding node on τ is unmarked or belongs to $M \cup \{s\}$. For clarity, we treat the cases in separate claims. Throughout all claims, let τ_i be the node at which the agent enters the search phase between steps t and t^+ . By Lemma 5.9, τ_{i-1} is the last node on τ that is contained in $M \cup \{s\}$.

Claim 1. *If P1–P5 hold in step t , and the agent places a tile at an unmarked node with $\text{skipSearch} = \text{false}$, then P1–P5 hold in step t^+ .*

Proof. We first show that P4 holds in step t^+ . Since the tile is placed at an unmarked node, it is placed at τ_i .

Since $\text{skipSearch} = \text{false}$ by assumption, the agent has searched $R(\tau_i, \tau_{i-1})$ and found neither s nor a marked node. Placing a tile at τ_i can only mark nodes in $N_{\mathcal{E}}(\tau_i)$, and the only possible node $w \in N_{\mathcal{E}}(\tau_i)$ that is already marked, that is, $\|N_{\mathcal{E}}(w)\| > 1$, is τ_{i-1} . Since τ_i and τ_{i-1} have a common tiled neighbor, placing a tile at τ_i cannot increase $\|N_{\mathcal{E}}(\tau_{i-1})\|$ by Lemma 5.11. Thus, P4 holds in step t^+ .

To show the remaining properties, we distinguish two cases based on the number of empty neighbors of τ_i .

Case 1: $|N_{\mathcal{E}}(\tau_i)| = 1$. By symmetry, τ_i is contained in a connected component of $H|_{N_{\mathcal{E}}(\tau_{i-1})}$ of size one. By P4, placing a tile at τ_i unmarks τ_{i-1} if τ_{i-1} is marked, and it cannot mark any other node because τ_i has no other empty neighbor. Thus, P1–P4 hold trivially in step t^+ . The node τ_{i-1} either has no successor in its segment $S(\tau_{i-1})$, or τ_i is such a successor. In both cases, after placing a tile at τ_i , it has no successor. Hence, $h^+(\tau_{i-1}) = \tau_{i-1}$, and P5 holds.

Case 2: $|N_{\mathcal{E}}(\tau_i)| > 1$. Consider a node $\tau_j \in \tau$ that is at distance at most three from τ_i in $H|_{\mathcal{E} \setminus \{\tau_{i-1}\}}$. If such a node exists, then by Definition 5.4, the search region $R(\tau_i, \tau_{i-1})$ contains its tail $t(\tau_j)$. Since $R(\tau_i, \tau_{i-1})$ contains neither s nor a marked node, P2 implies that $j > i - 1$.

Therefore, the subpath of τ from s to τ_{i-1} does not change from step t to step t^+ . Moreover, since the agent places a tile at τ_i , every node in $N_{\mathcal{E}}(\tau_i)$ is contained in τ^+ . Within $H|_{\mathcal{E} \setminus \{\tau_{i-1}\}}$, the distance between τ_i and any neighbor of a node in $N_{\mathcal{E}}(\tau_i)$ is at most two. Hence, these neighbors are among the nodes that are at distance at most three, and P3 holds in step t^+ . Placing a tile at τ_i can only mark nodes in $N_{\mathcal{E}}(\tau_i)$, so P1 holds in step t^+ as well.

It remains to show P2 and P5. Placing a tile at τ_i cannot unmark τ_{i-1} , that is, $\tau_{i-1} \in M^+ \cup \{s\}$. Otherwise, τ_{i-1} would be contained in a connected component of $H|_{N_{\mathcal{E}}(\tau_i)}$ of size one, which, since we assume $|N_{\mathcal{E}}(\tau_i)| > 1$, contradicts that τ_i is unmarked. Moreover, placing a tile at τ_i cannot mark τ_{i+1} by Lemma 5.11; hence, $\tau_{i+1} \notin M^+ \cup \{s\}$.

The node τ_{i-1} is the first node and τ_{i+1} is the last node in the consecutive sequence of empty nodes in the clockwise order of $N(\tau_i)$. Thus, for every $v \in N_{\mathcal{E}}(\tau_i)$ with $v \neq \tau_{i-1}$, we have

$$t^+(v) = \tau_{i-1} \in M^+ \cup \{s\} \quad \text{and} \quad h^+(v) = \tau_{i+1} \notin M^+ \cup \{s\}.$$

Therefore, P2 and P5 hold in step t^+ . $\triangleleft$

Claim 2. *If P1–P5 hold in step t , and the agent places a tile at a marked node that does not mark any node, then P1–P5 hold in step t^+ .*

Proof. We first show that P1–P4 hold in step t^+ . There are two cases.

Case 1: *The tile is placed at τ_{i-1} .* Then placing the tile trivially unmarks τ_{i-1} , since it is no longer empty in step t^+ .

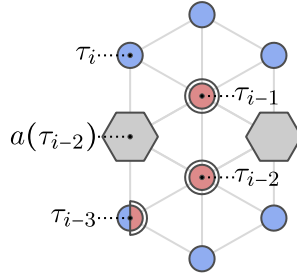


Figure 5.5: Common local structure used in the proofs of Claims 3 and 4. The figure shows the neighborhoods of τ_{i-1} and τ_{i-2} when the agent places a tile at the marked node τ_{i-2} and τ_{i-2} marks another node. Node τ_{i-3} may be marked or unmarked.

Case 2: The tile is placed at τ_{i-2} . Then the algorithm moved one additional step by the right-hand rule, so τ_{i-1} marks a node. By Corollary 5.12, $H|_{N_{\mathcal{E}}(\tau_{i-1})}$ contains two connected components of sizes one and three. Assume for contradiction that τ_{i-2} lies in the component of size three. Then τ_i lies in the component of size one. By symmetry, $H|_{N_{\mathcal{E}}(\tau_i)}$ has a connected component of size one as well. Thus, the empty neighbors of τ_i either induce at least two connected components, so τ_i is marked, or τ_i has only one empty neighbor. The former contradicts Lemma 5.9, and the latter contradicts that the tile is placed at τ_{i-2} instead of τ_i . Hence, τ_{i-2} lies in the component of size one. Therefore, placing a tile at τ_{i-2} unmarks both τ_{i-1} and τ_{i-2} .

In both cases, the subpath of τ^+ from s to the last marked node in τ^+ is a prefix of the subpath of τ from s to the last marked node in τ . Moreover, all previously marked nodes outside this prefix are unmarked in step t^+ , and placing the tile does not mark any node. Hence, P1–P4 hold in step t^+ .

It remains to show P5. For this, we treat the two cases above uniformly. Let τ_k be the node at which the tile is placed.

If the connected component of $H|_{N_{\mathcal{E}}(\tau_k)}$ that contains τ_{k-1} has size one, then placing a tile at τ_k unmarks τ_{k-1} by P4. Thus, $h^+(\tau_{k-1}) = \tau_{k-1}$, and P5 holds.

Otherwise, this component has size at least two. Let $\tau_j \neq \tau_{k-1}$ be a node in this component. By P3, we have $j > k$. Hence, $\tau_j \notin M$ by P1. Since, by assumption, placing a tile at τ_k does not mark any node, it follows that $\tau_j \notin M^+$. Since $h^+(\tau_k^+)$ lies in this component, in particular $h^+(\tau_k^+) \notin M^+$. Hence, P5 holds in step t^+ . $\triangleleft$

It remains to consider the case in which the tile is placed at a marked node that also marks another node. Since the search phase is entered at τ_i , such a tile is placed at τ_{i-2} . Thus, both τ_{i-1} and τ_{i-2} are marked nodes that mark another node.

By Corollary 5.12, both $H|_{N_{\mathcal{E}}(\tau_{i-1})}$ and $H|_{N_{\mathcal{E}}(\tau_{i-2})}$ contain two connected components of sizes one and three. The node τ_i must lie in the component of $H|_{N_{\mathcal{E}}(\tau_{i-1})}$ of size three. Otherwise, either τ_i is marked, which contradicts that the agent enters the search phase at τ_i , or τ_{i-1} is the only empty neighbor of τ_i , which contradicts that the tile is not placed at τ_i . Thus, τ_{i-2} lies in the component of $H|_{N_{\mathcal{E}}(\tau_{i-1})}$ of size one. By symmetry, τ_{i-1} lies in a component of $H|_{N_{\mathcal{E}}(\tau_{i-2})}$ of size one. In particular, τ_i , τ_{i-1} , τ_{i-2} , and τ_{i-3} are all contained in the same segment $S(\tau_{i-2})$. Hence, up to rotation, the neighborhoods of τ_{i-1} and τ_{i-2} are exactly as in Figure 5.5.

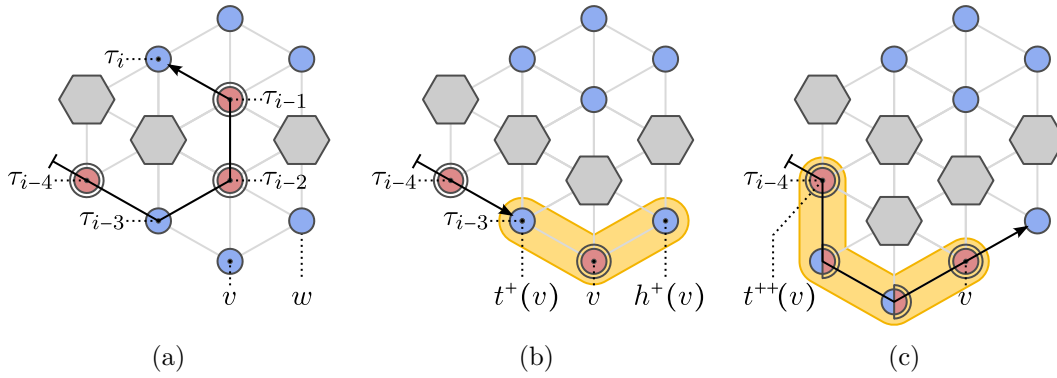


Figure 5.6: (a) Local configuration in the proof of Claim 3 in step t . (b) After step t^+ , the agent next enters the search phase at τ_{i-3} with `skipSearch` = true and places its tile there. (c) The resulting configuration in step t^{++} .

Claim 3. *If τ_{i-3} is unmarked, then P1–P5 hold in step t^{++} .*

Proof. It remains to determine the neighborhood of $a(\tau_{i-2})$. For ease of reference, see Figure 5.6. The assumption that $\tau_{i-3} \notin M \cup \{s\}$ implies that τ_{i-2} has another predecessor besides τ_{i-3} , as otherwise $\tau_{i-3} = t(\tau_{i-2}) \notin M \cup \{s\}$, contradicting P2. Thus, the segment $S(\tau_{i-2})$ contains the node τ_{i-4} as well.

Next, we show that τ_{i-1} and τ_{i-2} were marked by $a(\tau_{i-2})$ in some step $t' < t$. Initially, $M^0 = \emptyset$, so the nodes τ_{i-1} and τ_{i-2} were marked by one of the two common tiled neighbors of τ_{i-1} and τ_{i-2} . Let u be the node that marked both τ_{i-1} and τ_{i-2} in step t' . By Definition 5.5, it holds that $|B(u)| \leq 6$. By the neighborhood structure deduced above for τ_{i-1} and τ_{i-2} , $H|_{N_{\mathcal{E}}(u)}$ contains a connected component of size four. Hence, $H|_{N_{\mathcal{T}}(u)}$ is connected and contains at most two nodes. Since the agent never picks up a tile outside of the depot, this implies that u was not marked in step t' . Node u marks both τ_{i-1} and τ_{i-2} , and τ_{i-2} is visited before τ_{i-1} in step t . Hence, $a(\tau_{i-2}) = u$ and thus $|N(a(\tau_{i-2}))| \leq 6$. Therefore, $S(\tau_{i-2})$ contains five nodes σ_j with $j \in \{i-4, \dots, i\}$, and $a(\tau_{i-2})$ has one tiled neighbor. Up to rotation, this is precisely the local configuration depicted in Figure 5.6.

Second, we consider the situation after the tile is placed at τ_{i-2} between steps t and t^+ , and show that P2 holds in step t^{++} . Let v be the node marked by τ_{i-2} , and let w be the other node in the connected component of $H|_{N_{\mathcal{E}}(\tau_{i-2})}$ distinct from τ_{i-1} , τ_{i-3} , and v . As shown in Figure 5.6a, placing a tile at τ_{i-2} unmarks τ_{i-1} and marks v . It cannot mark w or τ_{i-3} , since both have a common tiled neighbor with τ_{i-2} . Hence, in step t^+ , the local configuration is exactly as depicted in Figure 5.6b. Since $t^+(v) = \tau_{i-3}$ and $\tau_{i-3} \notin M^+$, P2 is violated. The agent therefore enters the search phase with `skipSearch` set to true at τ_{i-3} between steps t^+ and t^{++} without visiting v . In this case, it places a tile at τ_{i-3} without searching for marked nodes in $R(\tau_{i-3}, \tau_{i-4})$. Afterwards, $t^{++}(v) = \tau_{i-4}$, so P2 holds again in step t^{++} . For completeness, Figure 5.6c shows the local configuration with respect to step t^{++} .

Third, we show that P1 and P3 to P5 are maintained. Recall that we already concluded that τ_{i-1} and τ_{i-2} were marked by $a(\tau_{i-2})$ in some prior step $t' < t$. Note that exactly two nodes in $N(\tau_{i-3})$ contain tiles in step t^+ ; see Figure 5.6b. Since

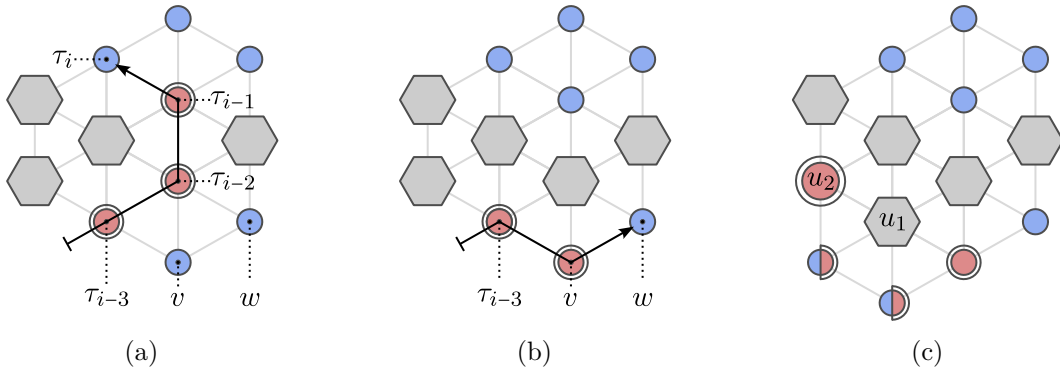


Figure 5.7: (a) Local configuration in the proof of Claim 4 in step t . The node τ_{i-3} is depicted as marked, but may instead be s . (b) The resulting configuration in step t^+ after placing a tile at τ_{i-2} . (c) Restatement of Figure 5.6c with new labels.

$N(a(\tau_{i-2}))$ contains only one tiled node in step t , by contraposition it follows that in step t' the tile at $a(\tau_{i-2})$ was placed with $skipSearch = false$. No node from $N_{\mathcal{E}}(a(\tau_{i-2}))$ was ever tiled before step t , and $a(\tau_{i-2})$ cannot mark τ_i in step t' as they have a common tiled neighbor. It follows that no tile was placed between steps t' and t , that is, $t' = t^-$. This implies that $R^-(a(\tau_{i-2}), \tau_{i-4})$ contains neither s nor any node that is marked in step t^- . The only nodes that become marked between steps t^- and t^{++} and remain marked in step t^{++} are contained in $N_{\mathcal{E}}(\tau_{i-3})$. Hence, P1, P3, and P4 hold in step t^{++} analogously to Claim 1, and P5 holds since $w \notin M^{++}$ and $h^{++}(w) = w$. $\triangleleft$

Claim 4. *If $\tau_{i-3} \in M \cup \{s\}$, then P1–P5 hold in step t^+ .*

Proof. The neighborhoods of τ_{i-1} and τ_{i-2} are exactly as in Figure 5.5. It remains to determine the neighborhood of $a(\tau_{i-2})$. If $a(\tau_{i-2})$ has only one tiled neighbor, then the proof reduces to the proof of Claim 3, except that $skipSearch$ is not set to true and P2 already holds in step t^+ . Hence, it remains to consider the case where $a(\tau_{i-2})$ has two tiled neighbors, which, up to rotation, is exactly the configuration depicted in Figure 5.7a.

Assume that $\tau_{i-3} = s$. By Lemma 5.9, τ_i is the first node $v \in \tau$ with $v \notin M \cup \{s\}$. Hence, there does not exist a marked node besides τ_{i-1} and τ_{i-2} . Let v be the node marked by τ_{i-2} , and let w be the node in $N_{\mathcal{E}}(\tau_{i-2})$ that is distinct from τ_{i-1} , τ_{i-3} , and v . As shown in Figure 5.7a, placing a tile at τ_{i-2} unmarks τ_{i-1} and marks v . By Lemma 5.11, it cannot mark w . Then, in step t^+ , the simple prefix is given by $\tau^+ = (s, v, w, \dots)$ with $M^+ = \{v\}$ and $h^+(w) = w \notin M^+$; see Figure 5.7b. Thus, P1–P5 hold in step t^+ .

Assume that $\tau_{i-3} \neq s$. Then, by the assumption of this claim, $\tau_{i-3} \in M$. Note that $H|_{N_{\mathcal{E}}(\tau_{i-3})}$ and $H|_{N_{\mathcal{T}}(\tau_{i-3})}$ both contain a connected component of size at least two. Since τ_{i-3} is empty, it holds that $|N(\tau_{i-3})| \leq 6$, and since τ_{i-3} is marked, $H|_{N_{\mathcal{E}}(\tau_{i-3})}$ and $H|_{N_{\mathcal{T}}(\tau_{i-3})}$ must each contain another component of size one. In particular, $H|_{N_{\mathcal{T}}(\tau_{i-3})}$ contains no connected component of size larger than two.

In the proof of Claim 3, we showed that, up to rotation, there is only one local configuration in which a tile is placed with $skipSearch = \text{true}$. For ease of reference, the configuration after the tile is placed is depicted in Figure 5.7c, with new labels on the nodes. Let u_1 be the node at which a tile is placed with $skipSearch = \text{true}$ in that configuration, and let u_2 be the node from which the agent moves to u_1 before placing the tile. As shown in Figure 5.7c, $H|_{N_{\mathcal{T}}(u_2)}$ contains a connected component of size at least three. By contraposition, $a(\tau_{i-2})$ must have been placed with $skipSearch = \text{false}$, since we showed that $H|_{N_{\mathcal{T}}(\tau_{i-3})}$ cannot contain a connected component of size larger than two. This case again reduces to the proof of Claim 3, except that $skipSearch$ is not set to true and P2 already holds in step t^+ . $\triangleleft$

The claims cover all non-terminating placements: the tile is placed either at an unmarked node, at a marked node that does not mark another node, or at a marked node that marks another node. In the last case, the preceding node on τ either belongs to $M \cup \{s\}$ or it does not. Thus, the case distinction is complete. $\square$

We can now prove correctness. All five invariants hold in the initial configuration. By Lemma 5.13, they are restored after each iteration of the main loop, possibly after one additional placement when $skipSearch$ is set. During this additional placement, the agent places a tile at a node that is neither marked nor equal to s . Thus, this placement cannot disconnect $H|_{\mathcal{E}}$. Together with Lemma 5.10, it follows that $H|_{\mathcal{E}}$ remains connected until termination.

By Lemma 5.7, whenever the agent picks up a tile at the depot, it either places that tile and returns to the depot, or places its final tile and terminates. Since one tile is placed in each such iteration and V_H is finite, the algorithm eventually terminates in some step T . The algorithm terminates only when s has no empty neighbor. Since $H|_{\mathcal{E}^{T-1}}$ is connected, no other empty node can remain. Hence, after the agent places its final tile at s , we have $\mathcal{T}^T = V_H$. This yields the main theorem.

Theorem 5.14. *For every coatable initial configuration $C^0 = (\mathcal{T}^0, p^0)$ on a surface graph H satisfying the assumptions from the problem definition, a finite automaton agent executing COATSURFACE solves the coating problem.*

5.3.1. Runtime

We now analyze the runtime of COATSURFACE and show that it is worst-case optimal. Since the agent senses only its current node p and the nodes in $N(p)$, exploring the search region $R(p, q)$ may require additional steps. By definition, $R(p, q)$ consists only of segments containing nodes reachable from p within distance at most three in $H|_{\mathcal{E} \setminus \{q\}}$. Hence, each node in $R(p, q)$ has distance at most five from p in H . Since the edges of H have constantly many possible orientations, every node has constant degree. Thus, $R(p, q)$ has constant size and can be explored in $\mathcal{O}(1)$ steps.

Each execution of the fetch phase takes $\mathcal{O}(|\tau|)$ steps, as the agent traverses a subpath of τ twice. Since τ is simple, we have $|\tau| = \mathcal{O}(n)$, where $n = |V_H|$. The agent places at most n tiles until $V_H = \mathcal{T}$. Hence, it performs at most n executions of the search phase and the fetch phase, for a total of $\mathcal{O}(n^2)$ steps.

For the lower bound, consider an agent with unlimited memory and global vision. Such an agent may reach any node via a shortest path instead of moving along the

boundary of the tiled nodes. Nevertheless, every tile is initially located at the depot and must be transported to its final position.

Consider a surface graph H that is the triangulation of an object resembling a straight line. Let w be a node at maximum distance from p^0 , so that $d_H(p^0, w) = \Theta(n)$, and let P be a shortest path from p^0 to w . In any final configuration, every node on P contains a tile. Initially, all these tiles are located at p^0 . Thus, the total distance over which tiles must be transported is at least the minimum cost of moving one tile from p^0 to each node of P , that is, the earth mover's distance between these two tile distributions. This cost is

$$\sum_{u \in P} d_H(p^0, u) = \sum_{i=1}^{\Theta(n)} i = \Theta(n^2).$$

Since each movement step can move at most one carried tile by one edge, any agent requires $\Omega(n^2)$ steps in the worst case. Thus, *COATSURFACE* is worst-case optimal.

5.4. Coating in the FCC Lattice Graph

We now apply *COATSURFACE* to the FCC lattice graph $G = (V, E)$. The object whose surface is to be coated is modeled by a connected set of nodes $\theta \subseteq V$. Nodes in θ are considered initially tiled and impassable by the agent.

We assume that holes in θ are sufficiently large. Formally, for any two nodes $v, w \in \theta$ with $d_v, w \leq 2$, we require $d_\theta(v, w) \leq 2$. Thus, any two object nodes that are close in the FCC lattice are also connected within θ by a short path. This excludes thin gaps in the object that would create local surface features too small for the following construction.

The depot and the initial position of the agent are located at a boundary node of the object, that is, $p^0 \in B(\theta)$. We define the *coating layer* L as the node set of the connected component of $G|_{B(\theta)}$ that contains p^0 . This excludes boundary nodes that are separated from the depot by the object, such as nodes on the inner surface of a hollow sphere. The goal is to place one tile at each node in L .

From the coating layer to a surface graph. The induced subgraph $G|_L$ is not necessarily a triangulation of a closed three-dimensional surface. Moreover, even if it is, nodes may have degree up to eight. We therefore first define an auxiliary graph on L from which we later obtain a surface graph satisfying our assumptions.

Let $G_L = (L, E_L)$ be the subgraph of $G|_L$ that contains exactly those edges $\{v, w\}$ for which v and w have adjacent neighbors in the object. Formally,

$$E_L = \{\{v, w\} \in E \mid \text{there are } u_v \in N_\theta(v) \text{ and } u_w \in N_\theta(w) \text{ with } d_\theta(u_v, u_w) \leq 1\}.$$

The difference between $G|_L$ and G_L is illustrated in Figure 5.8. By construction, an agent moving along G_L remains locally connected to the object. We may therefore view G_L as embedded on the surface of the three-dimensional object. This embedding contains triangular and square faces.

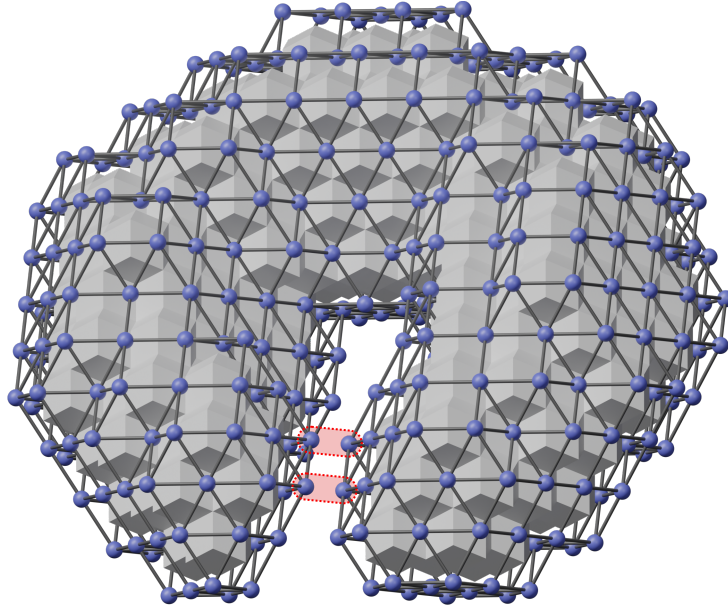


Figure 5.8: Example of the auxiliary graph G_L with its triangular and square faces. In contrast to $G|_L$, G_L contains no edge between either pair of red-circled nodes.

Up to rotation, square faces occur in exactly the following three orientations:

- (1) $v, v + NE, v + NE + USE, v + USE,$
- (2) $v, v + NW, v + NW + UNE, v + UNE,$
- (3) $v, v + N, v + N + UW, v + UW.$

Apart from rotation, Figure 5.9 depicts all possible local arrangements of faces in G_L around a common node. If two square faces share an edge, then they have the same orientation. We obtain H from G_L by adding one diagonal edge to each square face, choosing the diagonal orientation consistently for all square faces of the same type (1)–(3). Every edge of H is either an edge of G or it connects two nodes at distance two in G . Thus, each step in H can be simulated by at most two steps in G , and the additional memory needed to remember an intermediate node is constant.

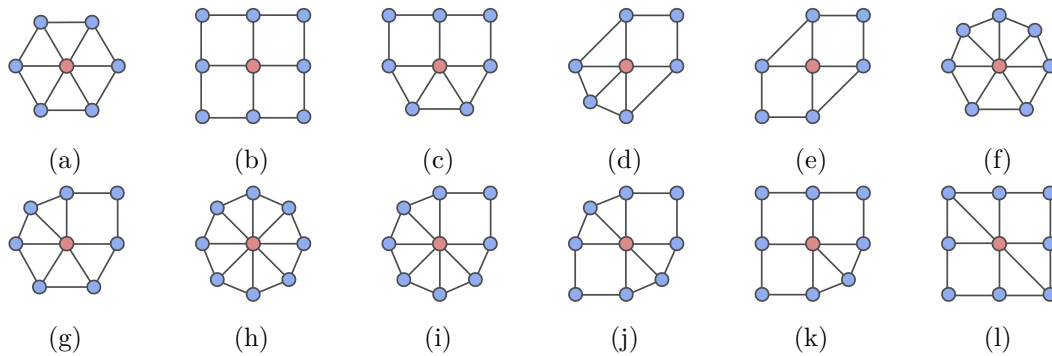


Figure 5.9: All possible arrangements of faces around a node of G_L , up to rotation.

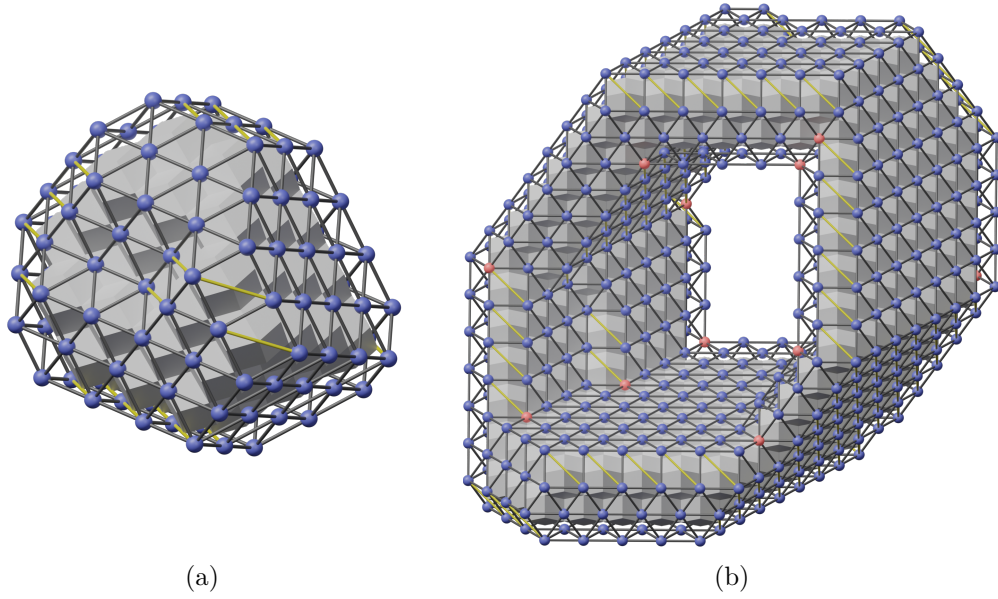


Figure 5.10: Examples of objects and their surface graphs for the two object classes. (a) The surface graph H of a simple object. (b) The surface graph H of a locally simple object. Edges of G_L are shown in black, while diagonal edges that occur only in H are shown in yellow. High-degree nodes are highlighted in red.

Classification of objects. We distinguish two classes of objects; see Figure 5.10. For simple objects, *COATSURFACE* can be applied directly. For the more general class of locally simple objects, we slightly adapt the algorithm.

A node of H is *high-degree* if its degree exceeds six. A *high-degree component* is a connected component of the subgraph of H induced by the high-degree nodes.

Definition 5.15. An object θ is *simple* if every node of the corresponding surface graph H has degree at most six. It is *locally simple* if every high-degree component has size at most two, and each high-degree component has distance at least three from every other high-degree component and from the depot p^0 .

For every object in either class, the neighborhood of each node in H forms a chordless simple cycle. This can be checked from Figure 5.9, which lists all possible neighborhoods up to rotation. If θ is simple, then every node of H has degree at most six by definition. Thus, the initial configuration on H is coatable according to Definition 5.5, and *COATSURFACE* can be applied directly. In terms of Figure 5.9, simple objects are precisely those for which only the local arrangements (a)–(e) occur.

Coating locally simple objects. The local arrangements in Figure 5.9 show that every node of H has degree at most eight, and hence constant degree. For locally simple objects, high-degree nodes occur only as isolated nodes or adjacent pairs, and distinct high-degree components are separated by distance at least three.

If all high-degree nodes were tiled initially, then the initial configuration would be coatable. Indeed, by the separation condition, the neighborhood of each empty node

contains nodes from at most one high-degree component. If an empty node is adjacent to two high-degree nodes, then these two nodes belong to the same component and are adjacent in its neighborhood. Thus, the tiled neighbors of every initially empty node induce a connected subgraph, and no node is marked initially.

We therefore adapt *COATSURFACE* as follows. Whenever the agent would place a tile at some node during the original algorithm, it first searches for an empty high-degree node within distance at most three in H . If it finds such a node v , it places the tile at v instead. By the separation condition, if v is the first node of its high-degree component to receive a tile, then there is no tiled node within distance at most three of v . Thus, placing a tile at v cannot mark any node and cannot disconnect $H|_{\mathcal{E}}$.

It remains to consider the case in which the high-degree component of v consists of two nodes. Let w be the other node of this component. If w receives a tile in a later step, then v is the only tiled node within distance at most three of w . Since H is a triangulation, the edge $\{v, w\}$ is contained in two triangular faces. Hence, there are exactly two empty nodes adjacent to both v and w , and v and w are connected in the neighborhood of each such node. Therefore, placing a tile at w cannot mark any node. Moreover, it cannot disconnect $H|_{\mathcal{E}}$, since $B(\{v, w\})$ contains only empty nodes in that step.

Each tile placed at a high-degree node is disconnected from all other tiles within H , except possibly from the other node of the same high-degree component. Thus, the simple prefix τ remains unchanged. Consequently, in the next execution of the search phase, the agent again attempts to place a tile at the same node as before.

The additional search is performed only when the agent places a tile. Since this happens at most n times and the search is restricted to a constant-size neighborhood, the asymptotic runtime remains $\mathcal{O}(n^2)$. This yields the following theorem.

Theorem 5.16. *In the FCC lattice graph, the extended COATSURFACE described above solves the coating problem for every surface graph H that corresponds to a simple or locally simple object in $\mathcal{O}(n^2)$ steps.*

5.4.1. Emulating Coatable Surface Graphs

We finally consider arbitrary triangulations that do not necessarily satisfy the structural assumptions used in the analysis above. In particular, the neighborhood of a node need not be a chordless simple cycle, and nodes may have degree up to an arbitrary constant Δ .

If we strengthen the model slightly, we can still solve the coating problem on such surface graphs by emulating a surface graph that satisfies our assumptions. For this, we assume that the depot provides a constant number of distinguishable tile types from which the agent can choose. For simplicity, we model this as if the agent chooses the tile type when placing a tile. In an implementation, the agent may have to return to the depot, pick up a tile of the desired type, and then return to the placement node. Since each fetch phase already traverses the current simple prefix a constant number of times, these additional traversals increase the runtime only by a constant factor.

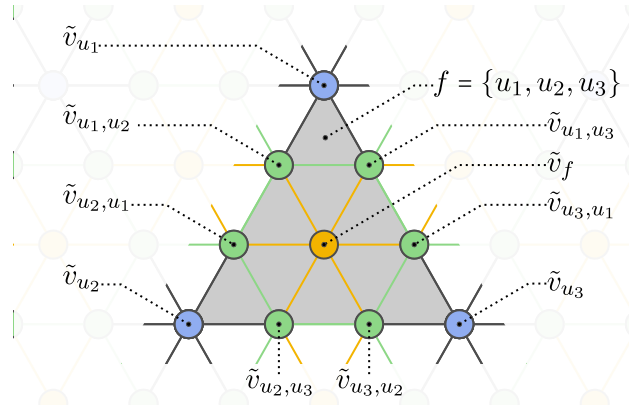


Figure 5.11: A triangular face f of H and the virtual nodes and edges in $\tilde{H}$ that correspond to this face, its corner nodes, and the edges between them.

Consider an arbitrary triangulation $H = (V_H, E_H)$ of constant degree Δ and an initial configuration C^0 . We construct a virtual surface graph $\tilde{H} = (\tilde{V}_H, \tilde{E}_H)$ with a virtual initial configuration $\tilde{C}^0$ such that the neighborhood of each node in $\tilde{H}$ is a chordless simple cycle and $\tilde{C}^0$ is coatable. During the construction, we define a partial surjective function $\mathcal{R}: \tilde{V}_H \rightarrow V_H$ that maps virtual nodes to original nodes. We show that an agent operating on H with initial configuration C^0 and $2^{2\Delta}$ distinguishable tile types can emulate an agent executing *COATSURFACE* on $\tilde{H}$ with initial configuration $\tilde{C}^0$ such that, throughout the emulation, $\mathcal{R}(\tilde{p}) = p$.

Virtual graph construction. The virtual graph $\tilde{H}$ is obtained by subdividing each triangular face of H into nine triangular faces; see Figure 5.11. The node set $\tilde{V}_H$ contains a virtual node $\tilde{v}_u$ for each node $u \in V_H$, two virtual nodes $\tilde{v}_{u,w}$ and $\tilde{v}_{w,u}$ for each edge $\{u, w\} \in E_H$, and a virtual node $\tilde{v}_f$ for each triangular face f of H .

For each edge $\{u, w\} \in E_H$, the edge set $\tilde{E}_H$ contains the three virtual edges

$$\{\tilde{v}_u, \tilde{v}_{u,w}\}, \quad \{\tilde{v}_{u,w}, \tilde{v}_{w,u}\}, \quad \{\tilde{v}_{w,u}, \tilde{v}_w\}.$$

For each triangular face $f = \{u_1, u_2, u_3\}$ of H , and for every pair of distinct nodes $u_i, u_j \in f$, the edge set $\tilde{E}_H$ contains the edge

$$\{\tilde{v}_f, \tilde{v}_{u_i, u_j}\}.$$

Thus, $\tilde{v}_f$ is incident to the six virtual nodes $\tilde{v}_{u_i, u_j}$ with $u_i, u_j \in f$ and $u_i \neq u_j$.

Moreover, for every $u_i \in f$, let u_j and u_k be the two other nodes of f . Then $\tilde{E}_H$ contains the edge

$$\{\tilde{v}_{u_i, u_j}, \tilde{v}_{u_i, u_k}\}.$$

Thus, each face contributes three edges between virtual edge-nodes.

We now define the partial function $\mathcal{R}: \tilde{V}_H \rightarrow V_H$ that maps virtual nodes to original nodes. For every virtual node $\tilde{v}_{u,w}$ corresponding to an edge $\{u, w\} \in E_H$, define

$$\mathcal{R}(\tilde{v}_{u,w}) = u.$$

To define $\mathcal{R}$ on virtual face nodes $\tilde{v}_f$, we assign each such node to one of the three original nodes incident to its face. Fix an arbitrary order on the vectors $\vec{X}_1, \dots, \vec{X}_m$ that occur as edge directions in the embedding of H . Let π denote this order, that is, $\pi(\vec{X}_i) = i$. For a triangular face $f = \{u_1, u_2, u_3\}$ of H , consider the six directed edge vectors $\vec{u}_i - \vec{u}_j$ with $u_i, u_j \in f$ and $u_i \neq u_j$. Let $\vec{u}_i - \vec{u}_j$ be the first of these vectors according to π . We define

$$\mathcal{R}(\tilde{v}_f) = u_i.$$

Finally, we define the virtual initial configuration $\tilde{C}^0$ by placing a tile at every virtual node $\tilde{v}_u$ that corresponds to an original node $u \in V_H$, and by setting the depot location to $\tilde{v}_{p^0}$. That is,

$$\tilde{C}^0 = (\{\tilde{v}_u \mid u \in V_H\}, \tilde{v}_{p^0}).$$

Emulation and final result. We next verify that the virtual construction satisfies the assumptions of the analysis and can be emulated on the original graph.

Lemma 5.17. *The virtual initial configuration $\tilde{C}^0$ is coatable with respect to $\tilde{H}$.*

Proof. We first show that every initially empty virtual node has degree at most six. Since each face of H is triangular and two virtual nodes are added for each edge of H , we have $|N(\tilde{v}_f)| = 6$ in $\tilde{H}$ for every face f of H .

Consider a virtual node $\tilde{v}_{u,w}$ corresponding to an edge $\{u, w\}$ of H . Let $f_1 = \{u, w, w_1\}$ and $f_2 = \{u, w, w_2\}$ be the two faces incident to this edge. Then $\tilde{v}_{u,w}$ is adjacent to the two virtual face nodes $\tilde{v}_{f_1}$ and $\tilde{v}_{f_2}$, to the two virtual edge-nodes $\tilde{v}_{u,w_1}$ and $\tilde{v}_{u,w_2}$, and to $\tilde{v}_{w,u}$ and $\tilde{v}_u$. Hence, $|N(\tilde{v}_{u,w})| = 6$ in $\tilde{H}$. The remaining virtual nodes are precisely the nodes $\tilde{v}_u$ added for original nodes $u \in V_H$, and these are initially tiled. Thus, every initially empty node of $\tilde{H}$ has degree at most six.

By construction, the initially tiled virtual nodes are pairwise at distance at least three, that is, $d(\tilde{v}, \tilde{w}) \geq 3$ for any distinct $\tilde{v}, \tilde{w} \in \tilde{T}^0$. Since $\tilde{H}$ is connected, it follows that $H|_{\tilde{\mathcal{E}}_0}$ is connected. Moreover, every initially empty virtual node has at most one initially tiled neighbor, and hence no node is marked initially. The depot has an empty neighbor by construction. Therefore, all conditions of Definition 5.5 are satisfied. $\square$

Theorem 5.18. *Let H be a triangulation of a closed three-dimensional surface with constant degree Δ . A finite automaton agent operating on H with $2^{2\Delta}$ distinguishable tile types can solve the coating problem in $\mathcal{O}(\Delta^2 n^2)$ steps.*

Proof. We construct the virtual graph $\tilde{H}$ and the virtual initial configuration $\tilde{C}^0$ as described above. By Lemma 5.17, the configuration $\tilde{C}^0$ is coatable, so *COATSURFACE* can be executed on $\tilde{H}$. It remains to show how an agent on H emulates this virtual execution.

For each original node $u \in V_H$, the emulation stores which of the virtual nodes that are mapped to u by $\mathcal{R}$ contain a tile. This information is encoded by the distinguishable tile types. We first describe this encoding and then show how virtual moves and tile placements are emulated on H .

Let $\tilde{F} \subseteq \tilde{V}_H$ be the set of virtual nodes $\tilde{v}_f$ that correspond to faces f of H . Since $H|_{\tilde{V}_H \setminus \tilde{F}}$ is a subdivision of H , it can be embedded in the same three-dimensional surface as H using vectors that are collinear to vectors in the embedding of H . Thus, we can use the same fixed order π from the construction of $\tilde{H}$.

For each node $u \in V_H$, let

$$x(u) = (x_1, \dots, x_{2\Delta})$$

be a bit sequence that encodes which virtual nodes $\tilde{v} \in \tilde{V}_H$ with $\mathcal{R}(\tilde{v}) = u$ contain a tile, where 0 represents an empty virtual node and 1 represents a virtual node containing a tile. By construction of $\tilde{H}$, there are at most 2Δ virtual nodes $\tilde{v}$ with $\mathcal{R}(\tilde{v}) = u$, so 2Δ bits suffice. The order of the bits in $x(u)$ is uniquely determined by π : the first Δ bits encode virtual nodes corresponding to edges of H , and the remaining bits encode virtual nodes corresponding to faces of H . There is no bit for the virtual node $\tilde{v}_u$, since $\mathcal{R}$ is undefined on $\tilde{v}_u$. This node contains a tile initially and continues to contain a tile until termination.

Consider an agent on H that has access to $2^{2\Delta}$ distinguishable tile types. Each tile type represents one bit sequence of length 2Δ . The agent emulates an execution of *COATSURFACE* on $\tilde{H}$ with initial configuration $\tilde{C}^0$ as follows.

If the virtual agent moves from $\tilde{v}$ to $\tilde{w}$, then the emulating agent moves from $\mathcal{R}(\tilde{v})$ to $\mathcal{R}(\tilde{w})$, unless $\mathcal{R}(\tilde{v}) = \mathcal{R}(\tilde{w})$. If the virtual agent places a tile at $\tilde{v}$ and $\mathcal{R}(\tilde{v})$ does not contain a tile, then the emulating agent places a tile at $\mathcal{R}(\tilde{v})$ whose type represents the bit sequence in which only the bit for $\tilde{v}$ is set to 1. If $\mathcal{R}(\tilde{v})$ already contains a tile, then the emulating agent replaces this tile by a tile whose bit sequence additionally sets the bit for $\tilde{v}$ to 1. If the virtual agent picks up a tile at the depot and the emulating agent carries no tile, then the emulating agent also picks up a tile.

By Theorem 5.14 and Lemma 5.17, the virtual agent solves the coating problem on $\tilde{H}$. At termination of the virtual execution, every virtual node contains a tile. Since $\mathcal{R}$ is surjective, for every original node $u \in V_H$ there is a virtual node $\tilde{v}$ with $\mathcal{R}(\tilde{v}) = u$. The tile at $\tilde{v}$ is represented in the emulation by a tile at u whose type encodes that $\tilde{v}$ contains a tile. Thus, the emulation has a tile at every original node and solves the coating problem on H .

The virtual graph has size $|\tilde{V}_H| = \mathcal{O}(\Delta n)$. By the runtime analysis of *COATSURFACE*, the virtual execution therefore takes

$$\mathcal{O}(|\tilde{V}_H|^2) = \mathcal{O}(\Delta^2 n^2)$$

steps. Each virtual step is emulated with only constant overhead, so the emulation also takes $\mathcal{O}(\Delta^2 n^2)$ steps. $\square$

Since Δ is constant, the number $2^{2\Delta}$ of tile types is constant as well. Moreover, for constant Δ , the runtime bound $\mathcal{O}(\Delta^2 n^2)$ simplifies to $\mathcal{O}(n^2)$. Thus, a finite automaton agent with constantly many distinguishable tile types can solve the coating problem on every triangulation of a closed three-dimensional surface of constant degree in quadratic time.

5.5. Conclusion

In this chapter, we studied the problem of coating surfaces with tiles in the hybrid model. To separate the combinatorial aspects of surface traversal from the geometric details of the FCC lattice, we first formulated the problem on an abstract surface graph. For this setting, we presented a single-agent algorithm that solves the coating problem in $\mathcal{O}(n^2)$ steps for surface graphs of degree at most six. We also showed that this runtime is asymptotically optimal, even in comparison with agents that have global knowledge and unrestricted memory.

The main difficulty of the problem lies in exploring the surface while preserving access to the depot, the only source of additional tiles. Our algorithm resolves this difficulty by using the tile configuration itself to implicitly mark and maintain traversable structures on the surface. This enables a constant-memory agent to systematically explore the surface and place tiles until all nodes are coated.

We then returned to the FCC lattice graph and studied how the abstract surface-graph formulation arises from the actual coating layer around a three-dimensional object. We first derived a suitable surface graph from that layer and showed that, for simple and locally simple objects, *COATSURFACE* can be applied directly or with only a small modification. For more general settings, we constructed suitable virtual surface graphs on which the algorithm can be emulated. This emulation runs in $\mathcal{O}(\Delta^2 n^2)$ steps and requires $2^{2\Delta}$ distinguishable tile types. Since Δ is bounded by a constant in the FCC lattice graph, this again yields a quadratic-time solution.

Several directions for future work remain. A first question is whether locally simple objects form the largest class on which our approach can be applied directly. Our current analysis applies to locally simple objects by ensuring that the modified execution is equivalent to one in which all high-degree components are tiled initially and no node is marked at the start. This is guaranteed when every high-degree component has size at most two and distinct high-degree components are sufficiently far apart. Already three tiled nodes in a single high-degree component may mark another node, so extending the analysis beyond locally simple objects seems to require a more refined treatment of such components. A second direction concerns the strengthened model with distinguishable tile types. Although $2^{2\Delta}$ is still a constant, it is quite large, and a more direct focus on this model may lead to substantially improved emulations or entirely different coating algorithms.

Part II.

Modular Self-Reconfigurable Robots

The Sliding Square Model

This chapter introduces the sliding square model used throughout the second part of this thesis. Modular self-reconfigurable robots are motivated by settings in which robotic structures must adapt their shape to changing environments or tasks. The sliding square model, originating in the work of Fitch et al. [FBR03], is a classical abstraction of such systems and represents them as connected configurations of identical square modules on a two-dimensional grid.

The model reflects central physical restrictions of modular self-reconfigurable robotic systems: modules can only move by using other modules as support, and movements are restricted by local space constraints. In the sliding square model, these restrictions are captured by two types of movement. A module can either slide along an adjacent module or perform a convex transition, where it moves around the corner of a neighboring module; see Figures 6.1b and 6.1c. In both cases, the positions intersected by the movement trajectory must be empty. Additionally, the configuration must remain globally connected throughout the execution, so that the robotic structure does not break apart as modules move. Together, these restrictions make the model well suited for studying algorithmic questions in settings where many simple modules collectively transform one shape into another.

Most previous work on the sliding square model studies reconfiguration planning in a centralized setting, where a global controller computes a schedule of moves. In this setting, canonical shapes provide a common strategy for universal reconfiguration. To reconfigure from a source shape to a target shape, it suffices to transform both shapes into the same canonical shape. The schedule from the source is then executed forward, and the schedule from the target is reversed. Canonical shapes are typically chosen to be compact and hole-free, since they provide simple targets and avoid the geometric complications associated with holes.

In this thesis, we study the sliding square model from a distributed perspective and focus on transforming arbitrary connected configurations into a rhombus. The rhombus is compact and hole-free and therefore serves as a natural canonical shape for reconfiguration. Each module operates autonomously, has only constant memory, communicates only with neighboring modules, and observes only local information. Under these restrictions, even forming simple shapes becomes challenging, since modules must preserve connectivity without global knowledge of the configuration.

We begin by formally defining configurations and the admissible moves of the sliding square model. We then introduce the distributed assumptions used in this thesis, including the communication model, local memory, and symmetry-breaking assumptions. Next, we define the reconfiguration problem studied in the next chapter. Finally, we review related work on modular self-reconfigurable robots and, in particular, on reconfiguration in the sliding square model.

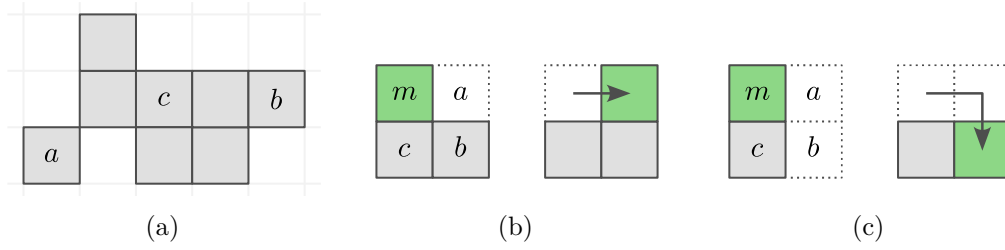


Figure 6.1: Grid terminology and move-set. (a) A subset of grid cells. The cell c has side-adjacent cells to the east, south, and west, and corner-adjacent cells to the southeast and northwest. The cells a and b are not adjacent to c . Since a shares no side with any other cell in the set, the set is not side-connected. (b) A module m slides into cell a . (c) A module m performs a convex transition into cell b .

6.1. Model Definition

In the sliding square model, all positions and movements are defined with respect to the two-dimensional square grid. We therefore first introduce the grid terminology used throughout this part of the thesis.

Grid Terminology. We consider the infinite square grid with set of cells C ; see Figure 6.1a for an illustration of the following notions. Two distinct cells $c_1, c_2 \in C$ are *side-adjacent* if they share an edge and *corner-adjacent* if they share a corner but no edge. They are *adjacent* if they are side-adjacent or corner-adjacent.

The *neighborhood* $N(c)$ of a cell c contains all cells adjacent to c . We describe the relative positions of adjacent cells using the compass directions north, east, south, and west for side-adjacent cells, and northeast, southeast, southwest, and northwest for corner-adjacent cells.

A *path* of length k from a cell c_0 to a cell c_k is a tuple $(c_0, c_1, \dots, c_k)$ such that c_i is side-adjacent to c_{i+1} for all $0 \leq i < k$. A *corner-path* is defined analogously, except that consecutive cells are only required to be adjacent. A set of cells is *side-connected* if, for any two cells in the set, there exists a path from one cell to the other that is fully contained in the set. It is *corner-connected* if the same condition holds for corner-paths. A *side-connected component* is a maximal side-connected set of cells, and a *corner-connected component* is a maximal corner-connected set of cells.

For example, in Figure 6.1a, there is a corner-path from a to b but no path, and the set contains two side-connected components and one corner-connected component.

Remark 6.1. As in the first part of this thesis, we slightly abuse notation and treat paths as sets when referring to their contents. For example, we write $c \in P$ if the cell c appears in the path P .

The Sliding Square Model. We consider a finite set of n identical square modules. Each module occupies exactly one cell of the grid, and each cell contains at most one module. A cell is *occupied* if it contains a module and *empty* otherwise. We denote by $M \subseteq C$ the set of all occupied cells.

Remark 6.2. *For simplicity, whenever there is no risk of ambiguity, we refer interchangeably to a module and the cell it occupies. In particular, we use M both for the set of occupied cells and for the corresponding set of modules.*

The modules have the computational capabilities of deterministic finite automata, share a common compass, and are activated according to a fully synchronous schedule. Time is divided into discrete rounds, and in each round every module simultaneously executes one *Look-Compute-Move* cycle.

In the look phase, a module observes its neighborhood. For every cell in this neighborhood, it determines whether the cell is empty or occupied and, if it is occupied, observes the state of the module in that cell.

In the compute phase, a module chooses a state transition based only on its current state and the information gathered during the look phase.

Finally, in the move phase, a module performs the action associated with the chosen state transition. This action is either to stay in place, to perform a *slide*, or to perform a *convex transition*; see Figures 6.1b and 6.1c. A slide moves a module to a side-adjacent cell while using neighboring modules as support, whereas a convex transition moves a module around the corner of a side-adjacent supporting module. Formally, let $m \in M$ be a module and consider a 2×2 block of cells containing m . Let (m, a, b, c) denote these four cells in cyclic order, either clockwise or counterclockwise. If a is empty and both b and c contain modules, then m may slide into cell a . If both a and b are empty and c contains a module, then m may perform a convex transition into cell b .

Conflicts can arise when several modules attempt to move along intersecting trajectories or into the same cell. We assume that such conflicts are resolved arbitrarily by the scheduler: one of the conflicting movements is executed, while the remaining modules stay in place.

Formally, each module is represented by a deterministic finite automaton with transition function

$$Q \times (Q \cup \{\perp\})^8 \rightarrow Q \times A.$$

Here, Q is the finite set of module states. We use N, E, S, W, NE, SE, SW , and NW for the corresponding compass directions, and define

$$A = \{\text{stay}\} \cup \{\text{slide}_d \mid d \in \{N, E, S, W\}\} \cup \{\text{convex}_d \mid d \in \{NE, SE, SW, NW\}\}$$

as the set of possible actions. The second component of the input describes the contents of the eight neighboring cells: for each neighboring cell, the input contains either the state of the module in that cell or the symbol $\perp$, representing an empty cell. The output consists of the next state of the module and one of the actions in A .

A *configuration* is a tuple (M, s, l) , where $M \subseteq C$ denotes both the set of occupied cells and the corresponding set of modules, $s: M \rightarrow Q$ assigns a state to each module, and $l \in M$ is the cell initially occupied by the leader module. A configuration is *side-connected* if its set of occupied cells is side-connected.

Although modules are finite automata, we describe algorithms textually and at a higher level of abstraction. In particular, we allow each module to store a constant number of variables, each with a constant-size domain. This is equivalent to a finite-state description and only serves to make the algorithms easier to present.

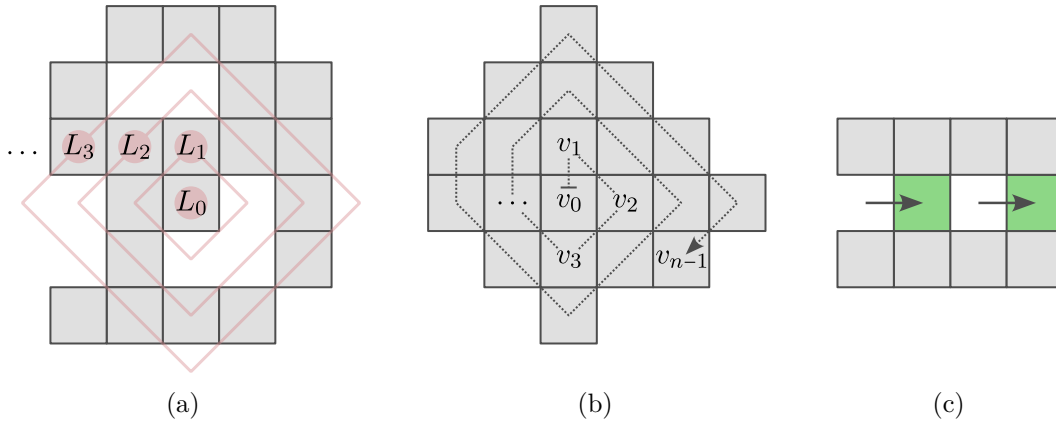


Figure 6.2: Target shape used by the reconfiguration algorithm. (a) Layers around the center cell, with the label of each layer placed at its connector cell. (b) A rhombus of size $n = 20$, with every cell occupied. (c) A violation of the connectivity constraint: the two indicated moves are valid individually, but executing them simultaneously disconnects the set of non-moving modules.

We also describe communication in terms of constant-size messages between neighboring modules. This does not give the modules any additional power. A module that sends a message to a neighbor encodes the message, together with the direction of the intended recipient, in its own state. In the next round, the recipient observes this state during its look phase and thereby receives the message. Since messages have constant size and directions are taken from a constant-size set, this communication mechanism can be encoded in the finite state space.

6.2. Problem Statement

We denote by $d(c_1, c_2)$ the Manhattan distance between two cells $c_1, c_2 \in C$. Let $v \in C$ be an arbitrary cell. For $i \geq 0$, the i -th layer around v is the set

$$L_i(v) = \{c \in C \mid d(v, c) = i\};$$

see Figure 6.2a. For each layer, we define a distinguished *connector* cell. The connector of $L_0(v)$ is v . For $i > 0$, the connector of $L_i(v)$ is the cell north of the westernmost cell in $L_{i-1}(v)$. A *rhombus* of size n with center v is a tuple

$$\mathcal{R} = (v_0, v_1, \dots, v_{n-1})$$

with $v_0 = v$. It is obtained by appending the layers around v in increasing order of their index, starting with $L_0(v)$. Within each layer, cells are ordered clockwise starting from the corresponding connector. Thus, the rhombus consists of consecutive layers around its center, possibly with a partially filled outermost layer; see Figure 6.2b.

We assume that the initial configuration (M^0, s^0, l) is side-connected and contains a unique leader module at cell $l \in M^0$. All other modules start in the same state, that is, $s^0(m) = s^0(m')$ for all $m, m' \in M^0 \setminus \{l\}$. The leader state is distinct from this common initial state, that is, $s^0(l) \neq s^0(m)$ for all $m \in M^0 \setminus \{l\}$.

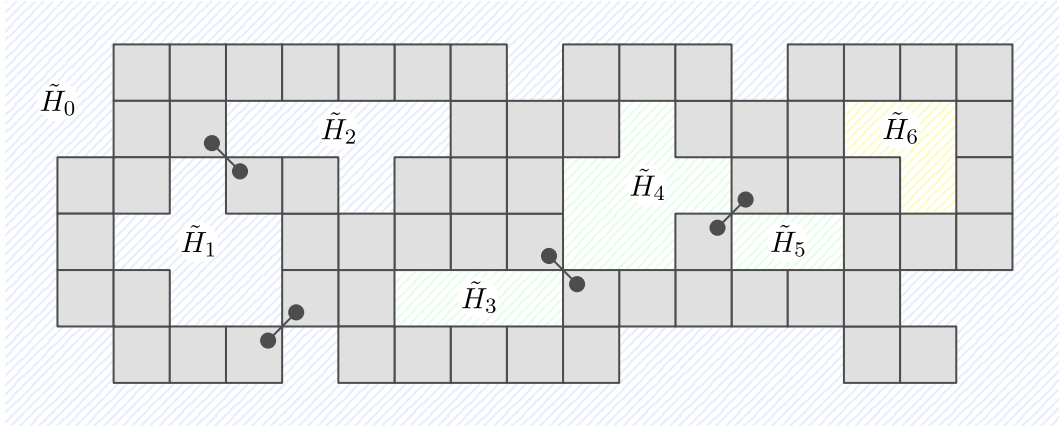


Figure 6.3: An example configuration that contains three holes, seven pseudo-holes, and four critical pairs. The pseudo-hole $\tilde{H}_0$ is infinite; together with $\tilde{H}_1$ and $\tilde{H}_2$, it forms the infinite hole on the exterior. The pseudo-holes $\tilde{H}_3$, $\tilde{H}_4$, and $\tilde{H}_5$ form another hole, while $\tilde{H}_6$ is both a hole and a pseudo-hole.

Executing a distributed algorithm yields a sequence of configurations

$$(M^0, s^0, l), (M^1, s^1, l), \dots, (M^T, s^T, l),$$

where each consecutive pair corresponds to one synchronous round. We require that, in every round, the subset of modules that do not move remains side-connected. This condition is also called the *backbone property* and prevents the modular robot from breaking apart during the execution; see Figure 6.2c.

The goal is to reach a configuration (M^T, s^T, l) in which all modules are terminated and occupy exactly the cells of a rhombus of size n with center l .

6.2.1. Additional Terminology

A *pseudo-hole* is a maximal side-connected set of empty cells, and a *hole* is a maximal corner-connected set of empty cells; see Figure 6.3. Intuitively, these are empty regions fully enclosed by modules, where holes allow corner contacts between modules. Since the set of modules is finite, there is exactly one infinite exterior hole and one infinite exterior pseudo-hole. All other holes and pseudo-holes are finite.

Every hole can be partitioned into one or more pseudo-holes. Two distinct pseudo-holes $\tilde{H}$ and $\tilde{H}'$ are *adjacent* if there exist adjacent cells $e \in \tilde{H}$ and $e' \in \tilde{H}'$. In this case, the common neighborhood $N(e) \cap N(e')$ contains exactly two occupied cells c and c' . We call (c, c') the *critical pair* of $\tilde{H}$ and $\tilde{H}'$. Moreover, we call e the *bridge* from $\tilde{H}$ to $\tilde{H}'$ and e' the *bridge* from $\tilde{H}'$ to $\tilde{H}$.

For any two pseudo-holes $\tilde{H}$ and $\tilde{H}'$, there is at most one critical pair and hence at most one bridge in each direction. Indeed, if there were two distinct bridges from $\tilde{H}$ to $\tilde{H}'$, then $\tilde{H} \cup \tilde{H}'$ would contain a corner-path cycle that fully encloses one of the cells of a critical pair. This would contradict side-connectivity of the configuration.

Finally, for a set of cells $V \subseteq C$, we define its *boundary* as

$$B(V) = \left(\bigcup_{v \in V} N(v) \right) \setminus V.$$

By definition, the boundary of a hole contains only occupied cells, whereas the boundary of a pseudo-hole may also contain empty cells from adjacent pseudo-holes.

6.3. Related Work

Dumitrescu and Pach give the first universal reconfiguration algorithm for connected configurations of sliding squares [DP04]. Their algorithm uses a straight line as a canonical intermediate shape and requires $\mathcal{O}(n^2)$ moves, which is worst-case optimal for sequential reconfiguration. Moreno and Sacristán later adapt this approach to an in-place setting, where modules remain within the bounding boxes of the input and target configurations, again using $\Theta(n^2)$ sliding moves [MS20]. Akitaya et al. give an input-sensitive in-place algorithm using $\mathcal{O}(nP)$ moves, where P is the maximum perimeter of the bounding boxes of the input and target configurations [Aki+22]. They also show that minimizing the number of sliding moves is NP-hard.

In three dimensions, modules slide over faces or rotate around edges of neighboring cubes. Fitch et al. study heterogeneous reconfiguration, where modules have unique identifiers and the target configuration specifies which module occupies each target position [FBR03]. They give a centralized algorithm using $\mathcal{O}(n^2)$ moves. In contrast, most later work on sliding cubes focuses on identical modules. In this setting, Abel et al. give the first universal reconfiguration algorithm in three and higher dimensions [Abe+24]. Their algorithm uses $\mathcal{O}(n^2)$ moves, matching the worst-case lower bound, and they also provide an in-place variant. Kostitsyna et al. study in-place compaction of sliding cubes, where the goal is to transform a configuration into a compact canonical shape [Kos+24]. They give an asymptotically optimal algorithm whose number of moves is proportional to the sum of coordinates over all modules. More recently, Akitaya et al. combine universal reconfiguration with input-sensitive bounds for sliding cubes, obtaining algorithms whose number of moves depends on the coordinate sums of both the input and target configurations [AKS25].

Distributed and Parallel Reconfiguration in Sliding Models. All algorithms discussed so far are centralized and compute schedules in which modules move sequentially. Distributed reconfiguration in the sliding square model has also been studied, although under different model assumptions. Fitch et al. give a distributed variant for the heterogeneous reconfiguration problem mentioned above, with runtime $\mathcal{O}(n^3)$ [FBR03]. This variant assumes a leader module and requires each module to store the target configuration, using $\mathcal{O}(n)$ memory per module. Dumitrescu et al. give a distributed algorithm that uses global communication to reconfigure horizontally convex configurations into a line [DSY04]. Here, horizontally convex means that the intersection of the configuration with any horizontal line is connected or empty. In particular, this assumption forbids holes in the input configuration.

Hurtado et al. give a universal distributed algorithm for two-dimensional lattice-based modular robotic systems, including square-grid systems [Hur+15]. Their algorithm preserves connectivity and runs in $\mathcal{O}(n)$ synchronous rounds. However, it relies on substantially stronger assumptions than the model considered in the next chapter. They assume a common compass, one leader with $\mathcal{O}(n)$ memory, $\mathcal{O}(\log n)$ memory in all other modules, and sensing and communication within distance two. They also relax the move-set by allowing modules to step over one another during a convex transition, which corresponds to allowing the move in Figure 6.1c even if cell a is occupied. In particular, this allows modules to move freely through critical pairs, which is not possible in the standard move-set considered in the next chapter.

Parallel reconfiguration has mainly been studied in the centralized setting. Wolters proposes parallelization methods for sliding squares [Wol24]. These methods are evaluated experimentally, but do not come with full correctness proofs for universal reconfiguration. Akitaya et al. later give the first worst-case optimal parallel result for sliding squares, achieving $\mathcal{O}(P)$ rounds, where P is the bounding-box perimeter [Aki+25]. They also prove strong hardness results, including NP-completeness of deciding reconfiguration in a constant number of rounds in several settings. In three dimensions, Akitaya et al. prove that deciding whether the optimal number of rounds is one or two is NP-hard, and give an input-sensitive algorithm with worst-case optimal $\mathcal{O}(n)$ rounds [Aki+26].

Restricted and Extended Variants. One variant of the model allows only sliding moves and disallows convex transitions. In this setting, Kawano studies reconfiguration in three dimensions using $2 \times 2 \times 2$ meta-modules [Kaw15]. A meta-module is a small group of modules that is treated as a larger movable unit. This increases the mobility of the system, but requires the input configuration to be organized into such meta-modules. Kawano later develops tunneling-based reconfiguration algorithms for the same setting [Kaw20]. In these algorithms, modules move only through the cells that are occupied by the start or target configuration, which makes them suitable for applications with severe space constraints. This line of work culminates in a linear-time reconfiguration algorithm [Kaw23].

Meta-modules have also been used more broadly to transfer reconfiguration algorithms between different types of modular robots. Aloupis et al. give a linear reconfiguration algorithm for cube-style modular robots arranged in $2 \times 2 \times 2$ meta-modules, achieving linear parallel time and in-place reconfiguration [Alo+09]. More generally, Parada et al. propose compact three-dimensional meta-module designs for other types of modules and show that they can simulate the capabilities of expanding and contracting systems such as Crystalline and Telecube robots [PSS21].

Another restriction is to remove the sliding movement primitive entirely. In pivoting and rotating models, modules instead rotate around corners or edges of neighboring modules. Depending on the move-set, such rotations may require additional empty cells along the trajectory of the rotating module. This often makes reconfiguration harder. Sung et al. study pivoting cube modular robots and give sufficient conditions under which configurations can be reconfigured into a line [Sun+15]. Feshbach and Sung extend this line of work to three-dimensional configurations with non-convex holes and show how to reconfigure them in $\mathcal{O}(n^2)$ moves [FS21].

Akitaya et al. study pivoting in two dimensions and show that the complexity depends strongly on the underlying lattice and the allowed pivot moves [Aki+21b]. In particular, pivoting square modules lead to PSPACE-hard reconfiguration problems, whereas universal reconfiguration is possible for hexagonal modules when a stronger class of pivot moves is allowed. For three-dimensional rhombic dodecahedral modules, Akitaya and Stock prove PSPACE-hardness in a restricted pivoting model [AS23]. Auxiliary modules, that is, additional modules not belonging to the input or target configuration, can restore universality in otherwise restricted pivoting settings. Akitaya et al. prove that five such modules suffice for universal reconfiguration by pivot moves [Aki+21a]. Connor et al. show that a constant number suffices to achieve worst-case optimal $\mathcal{O}(n^2)$ reconfiguration in a related pivoting model [CMS24]. Michail et al. study a broader programmable-matter model in which modules may rotate, slide, or use both types of moves, and analyze how the available movement primitives affect distributed reconfiguration [MSS19].

Other Models. Beyond square and cube sliding models, many other modular robot systems have been studied. Crystalline robots and Telecube-style systems use modules that expand and contract to reconfigure [RV01; VYS02]. For such systems, distributed and parallel reconfiguration algorithms are known, including algorithms that preserve connectivity during parallel module movements [BR03; VYS02].

Other lattice geometries and module shapes have also been considered. Walter et al. study parallel reconfiguration of hexagonal metamorphic robots [WTA05]. Their approach first computes a path through the target configuration that modules can occupy and traverse in parallel, and then uses this path to fill the remaining target cells. Naz et al. consider cylindrical modules arranged in a two-dimensional hexagonal lattice and give a parallel, asynchronous, distributed reconfiguration algorithm [Naz+16]. Odem et al. study triangular modules occupying cells of the triangular lattice and use sampling-based planning to compute reconfiguration schedules [OHM23].

A related line of work studies the 3D Catom model, where spherical modules are arranged in the FCC lattice. Modules move by rotating on the surface of neighboring modules, using them as pivots; thus, motion is constrained by the local geometry and requires free space along the rotation path. Within this model, Thalamy et al. develop scaffold-based methods, where an internal scaffold is constructed first and then coated with an outer layer of modules to obtain the desired object shape [Tha+19; Tha+20; TPB20].

Finally, related questions are studied in coordinated motion planning for swarms of robots that move in parallel. Fekete et al. consider connected grid configurations of robots that move continuously and in parallel, prove hardness results, and give approximation algorithms with bounded stretch [Fek+21b]. Although these robots are not modular squares or cubes, this work shares the goal of coordinating parallel motion under connectivity constraints.

Distributed Rhombus Formation

This chapter studies the problem of transforming an arbitrary side-connected configuration of square modules into a rhombus. The rhombus is compact and hole-free and therefore serves as a natural canonical shape for reconfiguration. We consider this problem in a distributed setting: modules have constant memory, communicate only with neighboring modules, and observe only local information. In addition, the initial configuration contains a unique leader module, which is necessary to break initial symmetries. We also assume that all modules share a common compass. This assumption is only needed to address neighboring modules consistently when messages are encoded in module states. Otherwise, the algorithm relies only on common chirality.

We present a distributed rhombus formation algorithm. The algorithm uses the original move-set of the sliding square model, consisting of slides and convex transitions, and preserves connectivity throughout the execution. This contrasts with previous distributed approaches, which rely on relaxed movement rules. Without these relaxations, a central difficulty arises: in some configurations, no module has a connectivity-preserving path to its position in the rhombus. The algorithm resolves this difficulty by moving modules that cannot enter the rhombus to suitable intermediate positions instead, thereby creating configurations in which rhombus cells can eventually be filled. The challenge is to coordinate this process without global knowledge of the configuration.

The main algorithm is sequential in nature: although the system is distributed, at most one module moves at a time. It transforms any side-connected configuration into a rhombus in $\mathcal{O}(n^2)$ rounds, which is worst-case optimal for algorithms that construct the target shape sequentially. To improve this runtime, we then introduce two parallel variants. Both maintain an auxiliary spanning-tree structure that allows modules to identify certain connectivity-preserving moves using only local information. Leaf modules of this tree move in parallel whenever possible; in rounds without parallel progress, the sequential part of the algorithm still advances the construction. The first variant permits a leaf module to move in parallel only if the move improves its depth in the spanning tree, without explicitly computing or storing this depth. The second variant permits any locally detected connectivity-preserving move of a leaf module. We prove correctness of the parallelization framework and evaluate the runtime of both variants through simulations.

Chapter Outline. We begin in Section 7.1 with the distributed rhombus formation algorithm with sequential movement. Its correctness and runtime are analyzed in Section 7.2. In Section 7.3, we extend the sequential algorithm with a framework for parallel movement. The two resulting parallel variants and their simulation-based

runtime evaluation are presented in Section 7.4. Finally, Section 7.5 summarizes the results and discusses directions for future work.

7.1. A Distributed Rhombus Formation Algorithm

We now describe the distributed rhombus formation algorithm. Figure 7.1 shows an example execution. The algorithm organizes modules into four phases: *HEAD*, *TAIL*, *ACTIVE*, and *PASSIVE*. At a high level, the *HEAD* module identifies the next cell to be filled in the rhombus, and the *ACTIVE* module moves to this cell. Modules in phase *TAIL* have already permanently joined the rhombus, while *PASSIVE* modules have not yet joined. Initially, the leader module is the *HEAD*, while all other modules are *PASSIVE*. In every round, there is exactly one *HEAD* module and at most one *ACTIVE* module. Every module that is not *ACTIVE* remains stationary and participates only in communication.

For the definitions and invariants used in the algorithm, we treat some moving modules as if their cells were empty. In round t , let M^t denote the set of all modules and let $A^t \subseteq M^t$ denote the set of modules whose cells are considered empty. In the sequential algorithm, A^t contains the *ACTIVE* module if such a module exists and is empty otherwise. In the parallel variants introduced later, A^t may also contain additional modules that are allowed to move in round t .

Remark 7.1. *Unless stated otherwise, all geometric notions, such as holes and boundaries, are defined with respect to the reduced set $M^t \setminus A^t$.*

Request Forwarding. The *HEAD* first identifies its immediate successor in the rhombus order. We refer to this cell as the *target cell*. The *HEAD* can determine the target cell from the local arrangement of neighboring *TAIL* modules, since *TAIL* modules always occupy a prefix of the rhombus order. If the target cell is already occupied, then the module in this cell becomes the new *HEAD*, while the previous *HEAD* becomes *TAIL*. If the target cell is empty, the *HEAD* initiates a request message to find a *PASSIVE* module that can become *ACTIVE* and move toward the target cell.

Messages are exchanged only between side-adjacent modules. A request message travels counterclockwise along the boundary of the hole that contains the target cell with respect to $M^t \setminus A^t$. We refer to this hole as the *target hole* and to its boundary as the *target boundary*. Whenever a module receives a request, it knows the neighbor from which the request was received. It can therefore determine the next forwarding direction locally by considering the counterclockwise order of its neighboring cells, starting from the sending module. The first side-adjacent module encountered in this order is the next module on the target boundary in counterclockwise direction. If a request fully traverses the target boundary without encountering a *PASSIVE* module, the *HEAD* terminates. All other modules terminate in subsequent rounds once they observe a terminated neighbor.

When a request reaches a *PASSIVE* module m , the module locally checks whether it can be selected to move toward the target cell. A module m is *removable from the target boundary* in round t if m is side-adjacent to the target hole and $M^t \setminus \{m\}$ is side-connected. For brevity, we call such a module *removable*. This global notion is

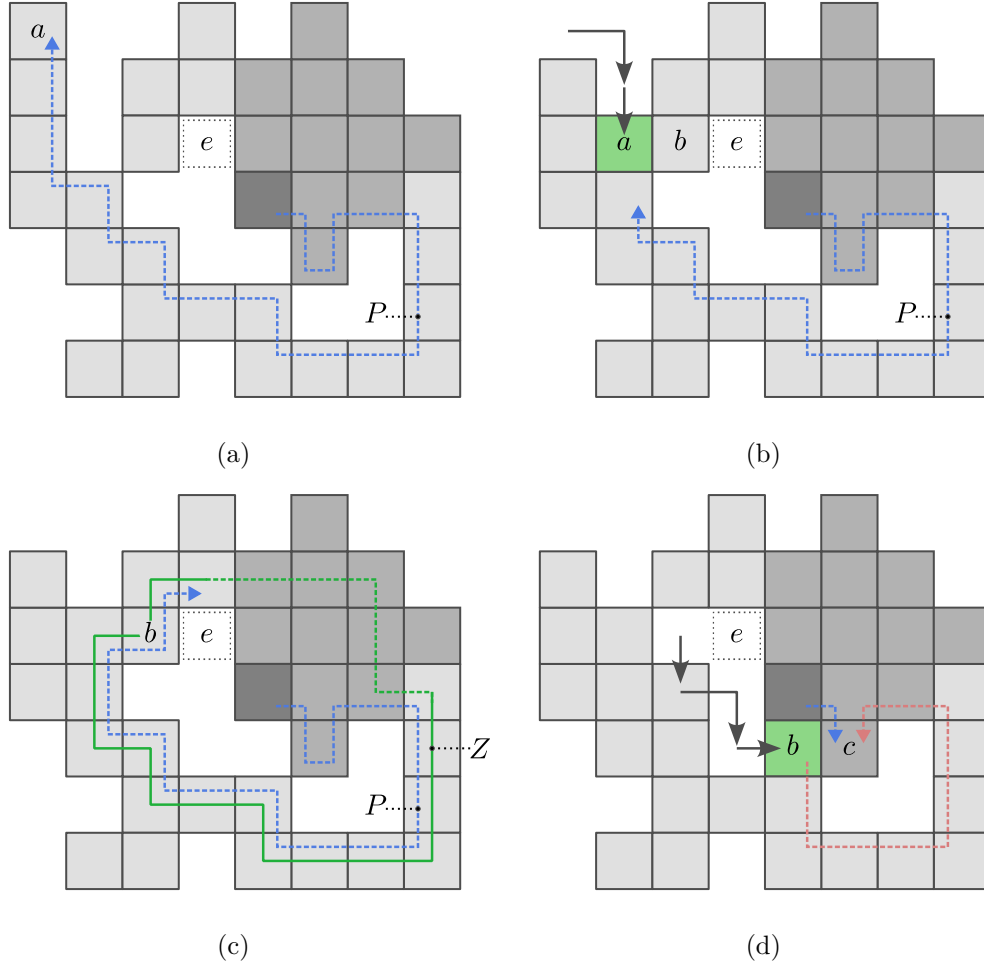


Figure 7.1: Example execution of the rhombus formation algorithm; *PASSIVE* modules are light gray, the *HEAD* is dark gray, and all other modules are *TAIL* modules. (a) The *HEAD* determines the next target cell e and sends a request that eventually reaches a simply removable module a . (b) Module a becomes *ACTIVE* and moves along the request path P in reverse until it is blocked by the critical-pair module b . Since b does not lie on P , module a becomes *PASSIVE* and forwards the request to b . (c) The next forwarding direction contains a *TAIL* module. An activation message travels backward along P , and b is the first bridged removable module to become *ACTIVE*. (d) Module b is blocked by the critical-pair module c , whose predecessor on P equals its successor. It sends a cleanup message backward along P . Once this message reaches c , module b can continue to move.

used only in the analysis; the algorithm instead relies on local sufficient conditions for selecting removable modules. The first such condition is simple removability.

Definition 7.2. *A module m is simply removable if*

- (a) *m is side-adjacent to the target hole, and*
- (b) *for every pair m_1, m_2 of modules side-adjacent to m , there exists a path from m_1 to m_2 within $N(m) \cap M^t$.*

If m is simply removable, it becomes *ACTIVE*. Otherwise, it forwards the request counterclockwise along the target boundary. Since simple removability is only a sufficient condition for removability from the target boundary, the request may reach no simply removable *PASSIVE* module, even though it reaches a removable one. To handle this situation, modules store the requests they receive.

Each stored request records the direction from which it was received and, once the request is forwarded, the direction in which it was sent. For a fixed target cell, a module can receive at most four requests, one from each side-adjacent neighbor. Moreover, before a new request is initiated, all information from the previous request is erased. Hence, constant memory suffices to store all requests relevant to the current target cell.

Definition 7.3. *In round t , a request path $P = (m_1, m_2, \dots, m_k)$ is a maximal directed path along the target boundary with respect to $M^t \setminus A^t$ such that, for each $1 < i \leq k$, module m_i stores a request from m_{i-1} or receives one at the start of round t . Moreover, each directed edge (m_{i-1}, m_i) occurs at most once in P .*

Definition 7.4. *Let $P = (m_1, m_2, \dots, m_k)$ be a request path. A passive segment of P is a maximal subpath*

$$S = (m_i, m_{i+1}, \dots, m_j)$$

*with $1 < i \leq j \leq k$ such that every module in S is *PASSIVE*.*

A request path may visit the same module more than once, but each directed cycle occurs at most once. In the analysis, we show that the algorithm maintains a stronger invariant: the request path contains at most one passive segment, this segment is simple, and every *PASSIVE* module storing a request belongs to it. Consequently, whenever a *PASSIVE* module receives a request, it is the last module of both the request path and the passive segment in that round.

Since a module observes the states of its neighbors, it can determine which neighboring modules belong to the request path. This information allows modules to identify certain local structures that imply removability even when no simply removable module is found. We use the following criterion.

Definition 7.5. *Let $Z = (m_0, m_1, \dots, m_{k-1})$ be a simple cycle of modules. A module m_i with $0 \leq i < k$ is bridged removable if*

- (a) *m_i is side-adjacent to the target hole, and*
- (b) *for every module m side-adjacent to m_i , there exists a path from m to m_{i-1} or a path from m to m_{i+1} within $N(m_i) \cap M^t$, where indices are taken modulo k .*

7.1.1. Activating Bridged Removable Modules

We now describe how the algorithm proceeds when the request cannot find a simply removable module. This procedure is initiated when the last module m_k on the request path P is *PASSIVE* and the module in the next forwarding direction is either already on P or not *PASSIVE*. In both cases, the algorithm identifies a simple cycle of modules and searches it in reverse request order for a bridged removable module.

Case 1: The request cycles back to itself. Suppose first that the module in the next forwarding direction from m_k is a *PASSIVE* module $m_l \in P$, where $l < k$. That is, the request would be forwarded to a module that already belongs to the request path. Since the passive segment S is unique and simple, the subpath

$$(m_l, m_{l+1}, \dots, m_k)$$

forms a simple cycle. The algorithm then traverses the request path in reverse order, starting at m_k . For indices

$$i = k, k-1, \dots, l,$$

the current module m_i checks whether it is bridged removable with respect to this cycle. If m_i is bridged removable, then it becomes *ACTIVE* and the procedure stops. Otherwise, it deletes the request received from m_{i-1} and sends an activation message to its predecessor m_{i-1} , which continues the procedure. We show in the analysis that this process always finds a bridged removable module before the activation message reaches m_l .

Case 2: The request reaches a non-PASSIVE module. Now suppose that the module in the next forwarding direction from m_k is not *PASSIVE*, that is, it is a *HEAD* or *TAIL* module. Since m_k is *PASSIVE*, the passive segment $S = (m_i, \dots, m_k)$ is non-empty. By maximality of S , the predecessor m_{i-1} of this segment is also non-*PASSIVE*. Let P' be a shortest path from the module in the next forwarding direction to m_{i-1} within the set of non-*PASSIVE* modules. Since shortest paths are simple and S contains only *PASSIVE* modules, the paths S and P' are disjoint. Their concatenation therefore forms a simple cycle.

The algorithm now proceeds as in the first case. Starting at m_k , the modules of S are checked in reverse order for bridged removability, and each module that is not bridged removable deletes its stored request. The first bridged removable module becomes *ACTIVE* and stops the procedure. As shown in the analysis, this process finds such a module before the activation message reaches m_{i-1} whenever the target hole has size at least two. If the target hole consists of a single cell, the algorithm deviates slightly. For clarity, we postpone this special case until after the analysis of larger target holes in Section 7.2.

Eventually, a *PASSIVE* module becomes *ACTIVE* and deletes its stored request. The *ACTIVE* module then moves through the target hole by slides and convex transitions, following the request path P in reverse direction. During this movement, the *ACTIVE* module instructs adjacent modules to delete their stored requests, so that it remains side-adjacent to the current last module of P .

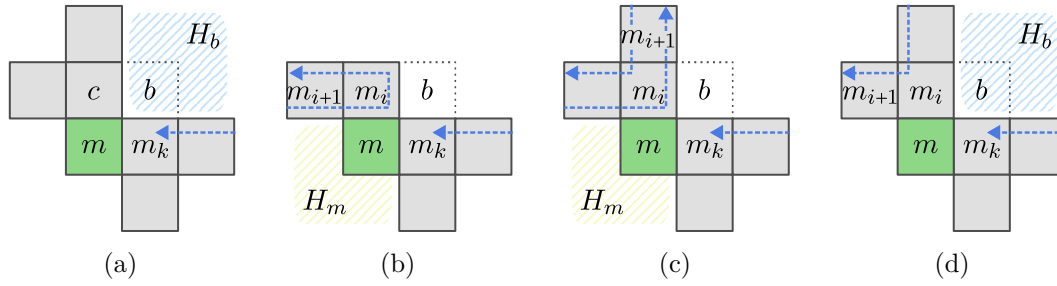


Figure 7.2: Cases 1 and 2 for determining whether the target cell lies in H_m or H_b when the *ACTIVE* module m is blocked by a critical pair (m_k, c) . The arrows indicate the request path P ; the labeled hole contains the target cell.

7.1.2. Moving Past Critical Pairs

It may happen that the movement of the *ACTIVE* module is blocked by a critical pair (m_k, c) , where m_k is the last module on the request path P . At this point, the current cell m of the *ACTIVE* module is one of the bridge cells of this critical pair. Let $b \neq m$ be the other bridge cell. If a module occupied b , then the current cell m would belong to a hole, which we denote by H_m . Similarly, if a module occupied m , then the bridge cell b would belong to a hole, which we denote by H_b ; see Figure 7.2. Since any two adjacent pseudo-holes are separated by a unique critical pair, the holes H_m and H_b are disjoint. Hence, exactly one of them contains the target cell.

To proceed, the *ACTIVE* module determines whether H_m or H_b contains the target cell. The *ACTIVE* module can determine whether c occurs on the request path from the state of c . In that case, it can also read the relative positions of the predecessors and successors of c on the request path from this state. This allows the following local case distinction, whose correctness is shown in Lemma 7.8.

Case 1: $c \notin P$. Then the target cell lies in H_b ; see Figure 7.2a.

Case 2: c occurs on the request path after m_1 . Consider all occurrences of c on the request path after m_1 , that is, all indices $i > 1$ with $m_i = c$.

Case 2a: There exists an index $i > 1$ with $m_i = c$ such that $m_{i+1} = m_{i-1}$ or $m_{i+1} \notin N(m)$. Then the target cell lies in H_m ; see Figures 7.2b and 7.2c.

Case 2b: For every index $i > 1$ with $m_i = c$, we have $m_{i+1} \in N(m)$ and $m_{i+1} \neq m_{i-1}$. Then the target cell lies in H_b ; see Figure 7.2d.

Case 3: c occurs only as m_1 . Then $c = m_1$ is the module that initiated the request and stores the position of the target cell. The *ACTIVE* module can read this position from the state of m_1 . Assume without loss of generality that m_1 lies north of m . If the target cell lies south, southwest, or west of m_1 , or if it lies northwest of m_1 and the cell west of m_1 is empty, then the target cell lies in H_m ; see Figures 7.3a and 7.3b. Otherwise, it lies in H_b ; see Figures 7.3c and 7.3d.

The *ACTIVE* module now proceeds according to which of the two holes contains the target cell. If H_m contains the target cell, then m sends a cleanup message to m_k . Each module receiving this message forwards it to its predecessor on the request path and then deletes its last stored request. Eventually, the message reaches c , where it is dropped. Afterwards, m resumes its movement along the boundary of H_m .

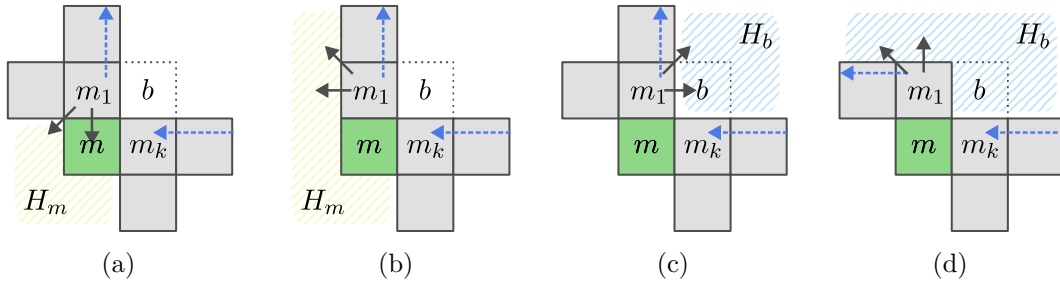


Figure 7.3: Case 3. The gray arrows indicate the direction of the target cell relative to m_1 . The case where the target cell lies northwest of m_1 and both the cells north and west of m_1 are occupied is omitted and analyzed in Lemma 7.8.

If H_b contains the target cell, then m becomes *PASSIVE* and proceeds as if it had received a request from m_k . That is, m stores a request from m_k and forwards this request to c in the next round. When m becomes *PASSIVE*, its cell is no longer considered empty. Thus, all pseudo-holes contained in H_m are pruned from the target hole. This reduction of the target hole is essential for showing that all requests terminate in linear time.

7.2. Analysis

We prove that the distributed rhombus formation algorithm transforms any initially side-connected configuration (M^0, s^0, l) with a unique leader in cell $l \in M^0$ into a rhombus of size n with center l . After $\mathcal{O}(n^2)$ rounds, each cell of this rhombus contains a terminated module. Throughout the execution, the algorithm preserves side-connectivity of the nonmoving modules.

Proof Structure. The analysis is organized around the progress of a single request for the current target cell. We first establish structural invariants of the request path, which ensure that every stored request belongs to this path and that all *PASSIVE* modules involved in the request form a single simple passive segment. We then show that, whenever the request cannot activate a simply removable module, the algorithm can find a bridged removable module on this segment, except possibly when the target hole consists of a single cell.

Using these properties, we prove that each request either fills the target cell or reduces the target hole to size one within $\mathcal{O}(n)$ rounds. In the remaining case, we use a secondary request to create a bridged removable module on the boundary of the single-cell target hole, which then moves into the target cell. Finally, since each filled target cell permanently extends the prefix of the rhombus occupied by *TAIL* modules, the algorithm terminates after $\mathcal{O}(n^2)$ rounds.

Structural Invariants. We begin with the invariant that relates the current *HEAD* position to the part of the rhombus that has already been completed. In particular, the *TAIL* modules always occupy a prefix of the rhombus.

Lemma 7.6. *Let $\mathcal{R} = (v_0, \dots, v_{n-1})$ be a rhombus with center l . If the *HEAD* module occupies cell v_i for some $0 \leq i < n$, then the cells occupied by *TAIL* modules are exactly the cells of the prefix $(v_0, \dots, v_{i-1})$ of $\mathcal{R}$.*

Proof. Initially, the leader is the *HEAD* module and occupies v_0 , while there are no *TAIL* modules. Hence, the claim holds for the empty prefix.

Assume that the claim holds at the start of an arbitrary round and that the *HEAD* module occupies v_i . If $i < n - 1$, then the next target cell is v_{i+1} . The *HEAD* module does not move, and *TAIL* modules neither move nor change phase. Thus, the claim can only change when the target cell becomes occupied and the *HEAD* role is passed on. Then the module in v_{i+1} becomes the new *HEAD*, while the previous *HEAD* becomes *TAIL*. Consequently, the cells occupied by *TAIL* modules become exactly the cells of the prefix $(v_0, \dots, v_i)$. The claim follows by induction over the rounds. $\square$

The following lemma was proven by Dumitrescu and Pach [DP04]. For ease of reference, we restate it using our terminology.

Lemma 7.7. *Let $\tilde{H}$ be a pseudo-hole, and let $c_1, c_2 \in \tilde{H}$ be two distinct cells that are side-adjacent to $B(\tilde{H})$. A module placed at cell c_1 can move to cell c_2 while remaining side-adjacent to cells in $B(\tilde{H})$ throughout its movement.*

We next prove the correctness of the local case distinction used when an *ACTIVE* module is blocked by a critical pair.

Lemma 7.8. *Let an *ACTIVE* module in cell m be blocked by a critical pair (m_k, c) with bridge cells m and b , where m_k is the last module on the request path. Let H_m be the hole that would contain m if b were occupied, and let H_b be the hole that would contain b if m were occupied. Then the local case distinction from Section 7.1.2 correctly determines whether the target cell lies in H_m or in H_b .*

Proof. Let $P = (m_1, \dots, m_k)$ be the request path. We define a sequence of empty cells adjacent to P . For each index i with $1 \leq i \leq k$, we consider the counterclockwise order of the cells in $N(m_i)$ between the predecessor and successor of m_i on the request path, and append all empty cells encountered in this order to the sequence. For m_1 , the order starts at the target cell and ends at m_2 . For m_k , the order starts at m_{k-1} and ends at the first side-adjacent module encountered after m_{k-1} . Let $S = (v_1, \dots, v_r)$ be the resulting sequence.

We say that a cell v is *recorded* by a module m_i if v is appended to S while considering the cells in $N(m_i)$. Moreover, we say that v is recorded *before* a cell w by m_i if both cells are recorded by m_i and v occurs before w in the corresponding counterclockwise order of cells in $N(m_i)$. The notion of being recorded *after* another cell is defined analogously.

The sequence S may contain repeated cells, but consecutive cells in S are adjacent, and every cell in S is adjacent to some module of the request path. Moreover, both bridge cells m and b occur in S because they are recorded by m_k . Let i_m and i_b be the first indices such that $v_{i_m} = m$ and $v_{i_b} = b$, respectively.

If $i_m < i_b$, then the prefix $(v_1, \dots, v_{i_m})$ does not contain b and is a corner-path from the target cell to m . Hence, the target cell lies in H_m . Analogously, if $i_b < i_m$,

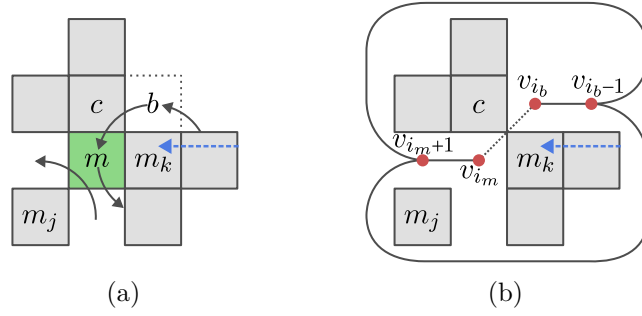


Figure 7.4: Contradiction argument for Case 1. (a) Under the assumption $i_m < i_b$, the module m is recorded before b by some module m_j with $j < k$. (b) This implies one of the two depicted closed curves through the common corner of m and b . In either case, the curve separates two occupied cells, contradicting side-connectivity.

then the prefix $(v_1, \dots, v_{i_b})$ does not contain m and is a corner-path from the target cell to b . Hence, the target cell lies in H_b .

We now consider the cases of the local case distinction from Section 7.1.2.

Case 1: $c \notin P$. Assume for contradiction that $i_m < i_b$.

Since m is recorded after b by m_k , the first occurrence of m in S cannot be recorded by m_k . It also cannot be recorded by c , since $c \notin P$. Hence, m must be recorded by some module m_j with $j < k$; see Figure 7.4a.

By construction of S , the subsequence $(v_{i_m}, \dots, v_{i_b})$ contains a corner-path from m to b that contains only empty cells. Since neither m_k nor c records b after m , the path does not contain an edge between m and b . Hence, there exists a closed simple curve contained in the union of the empty cells and passing through the common corner of m and b ; see Figure 7.4b.

The region enclosed by this curve separates the two modules of the critical pair. It contains either c but not m_k , or m_k but not c . In both cases, the curve separates two occupied cells, which contradicts side-connectivity of the configuration. Therefore, $i_b < i_m$, and the target cell lies in H_b .

Case 2: c occurs on the request path after m_1 . That is, there exists an index $i > 1$ with $m_i = c$. We consider all such occurrences of c .

Case 2a: There exists an index $i > 1$ with $m_i = c$ such that $m_{i+1} = m_{i-1}$ or $m_{i+1} \notin N(m)$. For this occurrence, the counterclockwise order of the cells in $N(m_i)$ implies that m_i records m . Moreover, whenever an occurrence of c records m , it also records b after m .

Assume for contradiction that $i_b < i_m$. Since m_i records b after m , the first occurrence of b in S cannot be recorded by m_i . Moreover, it cannot be recorded by m_k , since $k > i$. Hence, b must be recorded by some module m_j with $j < i$; see Figure 7.5a.

By construction of S , the subsequence $(v_{i_b}, \dots, v_{i_m})$ contains a corner-path from b to m that contains only empty cells. This path does not contain an edge between b and m : m_i records m before b , and the occurrence of m at position i_m precedes any occurrence of m recorded by m_k . Hence, as in Case 1, there exists a closed simple curve contained in the union of the empty cells and passing through the common

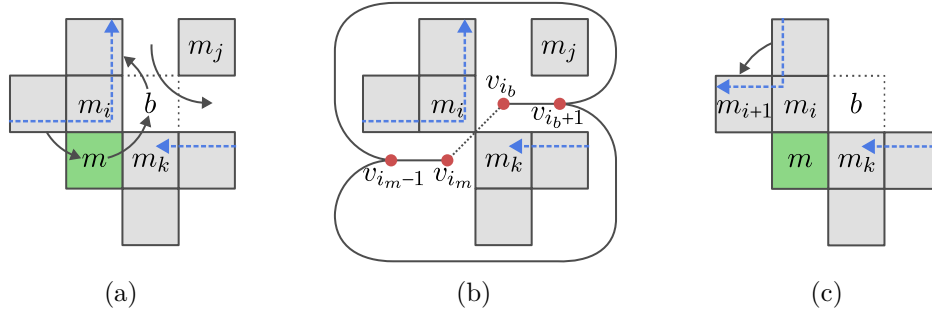


Figure 7.5: Contradiction argument for Case 2. (a) In Case 2a, if $i_b < i_m$, then some module m_j with $j < i$ records b before m . (b) This implies one of the two depicted closed curves through the common corner of m and b . In either case, the curve separates two occupied cells, contradicting side-connectivity. (c) In Case 2b, the module m_i records neither m nor b , so the proof is analogous to Case 1.

corner of m and b ; see Figure 7.5b. This again contradicts side-connectivity of the configuration. Therefore, $i_m < i_b$, and the target cell lies in H_m .

Case 2b: For every index $i > 1$ with $m_i = c$, we have $m_{i+1} \in N(m)$ and $m_{i+1} \neq m_{i-1}$. For each such occurrence, the bridge cells b and m occur in the counterclockwise order of $N(m_i)$ between m_{i+1} and m_{i-1} . Thus, no occurrence of c records either b or m ; see Figure 7.5c.

The remaining argument is analogous to Case 1. If $i_m < i_b$, then the first occurrence of m is recorded before the request reaches m_k and by no occurrence of c . Together with the subsequence $(v_{i_m}, \dots, v_{i_b})$, this yields the same separating curve as in Case 1, contradicting side-connectivity. Therefore, $i_b < i_m$, and the target cell lies in H_b .

Case 3: c occurs only as m_1 . Assume without loss of generality that c lies north of m ; the other cases are symmetric up to rotation. Since $c = m_1$, the module c initiated the request and stores the position of the target cell.

It is immediate from the local geometry that the target cell lies in H_m if it lies south, southwest, or west of m_1 , or if it lies northwest of m_1 and the cell west of m_1 is empty; see Figure 7.3. Similarly, the target cell lies in H_b if it lies north, northeast, or east of m_1 , or if it lies northwest of m_1 and the cell north of m_1 is empty.

It remains to consider the case where the target cell lies northwest of m_1 and both the north and west neighbors of m_1 are occupied; see Figure 7.6. Let H denote the hole containing the target cell, that is, the cell northwest of m_1 . Since m_1 occurs only once on P and the target cell lies northwest of m_1 , the module m_2 must lie west of m_1 . In particular, m_2 is side-adjacent to both H and H_m .

Assume for contradiction that $H = H_m$. Consider a simple directed curve that connects the midpoint of the northern side of m_2 to the midpoint of its southern side and lies entirely within H . Connecting its endpoints by a straight line segment inside the cell of m_2 yields a simple directed loop. Depending on the local configuration, this loop is either counterclockwise or clockwise, corresponding to the curves $\mathcal{C}_1$ and $\mathcal{C}_2$ in Figure 7.6. If the loop is counterclockwise, it encloses all successors of m_2 but not m_k . If it is clockwise, it encloses m_k but no successor of m_2 . Both cases contradict that m_k is a successor of m_2 . Hence, $H \neq H_m$.

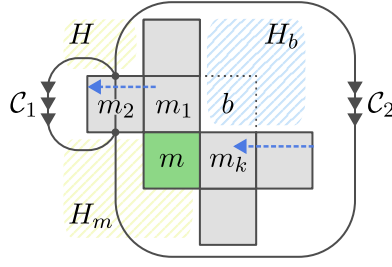


Figure 7.6: Remaining configuration for Case 3. The target cell lies northwest of m_1 in the hole H . Under the assumption $H = H_m$, the curves C_1 and C_2 show the two possible orientations of the loop used in the contradiction argument.

It remains to show that $H = H_b$. If m_k had no side-adjacent module other than m_{k-1} , then m_k would be simply removable and would have become *ACTIVE* before the request reached the current *ACTIVE* module. Hence, m_k has another side-adjacent module. Under our orientation, this means that the cell southeast of m is occupied. Moreover, the cell northwest of m is occupied by m_2 . Thus, there is no critical pair with m as a bridge cell besides (m_1, m_k) . Consequently, placing a module at m splits the target hole into exactly two holes. Since $H \neq H_m$, it follows that $H = H_b$.

Thus, in every case, the local case distinction identifies whether the target cell lies in H_m or in H_b . $\square$

We now state the main properties of the request path. These properties are used throughout the remaining analysis, and we show below that they are maintained as invariants throughout the execution. We refer to them as P1–P3.

Definition 7.9. Let t be an arbitrary round, and let $P = (m_1, m_2, \dots, m_k)$ be the request path in round t , defined with respect to the reduced set of modules $M^t \setminus A^t$. We consider the following properties:

- P1: For every module m that stores a request from some module m' , the directed edge (m', m) is contained in P .
- P2: The request path P contains at most one passive segment $S = (m_i, \dots, m_j)$. If such a segment exists, then $j = k$, that is, S ends at the last module of P .
- P3: The passive segment S is simple, that is, it contains each module at most once.

Lemma 7.10. The properties P1–P3 hold in every round during the execution of the rhombus formation algorithm.

Proof. We prove the claim by induction over the rounds of the algorithm.

Initially, no request has been initiated and no module stores a request. Hence, P1–P3 hold trivially. When a module m_1 initiates a request, the request path is $P = (m_1)$. Since no module has stored a request yet, P1 holds. Moreover, since *PASSIVE* modules do not initiate requests, P contains no passive segment, and P2 and P3 hold as well.

Now consider an arbitrary round in which P is non-empty, and assume that P1–P3 hold at the beginning of the round. We distinguish the possible changes to the

request path. A request may be forwarded, a *PASSIVE* module may become *ACTIVE*, adjacent modules may be instructed by the *ACTIVE* module to delete their stored requests, or the *ACTIVE* module may be blocked by a critical pair and therefore either become *PASSIVE* or send a cleanup message.

Case 1: A request is forwarded. Suppose the last module m_k of the request path forwards a request to some module m_{k+1} . If m_k is *PASSIVE*, then the request is forwarded only if m_{k+1} is *PASSIVE* and does not already belong to P . Thus, both P and the passive segment S are extended by a module that did not appear in P or S before. Hence, S remains simple, and P1–P3 are maintained.

If a non-*PASSIVE* module forwards a request, then S is empty by P2. After forwarding the request, S either remains empty or consists only of m_{k+1} . Hence, P1–P3 are maintained.

Case 2: A PASSIVE module becomes ACTIVE. Suppose a *PASSIVE* module m becomes *ACTIVE* after receiving either a request or an activation message. If m receives a request, then it is the last module of P by P2. If m receives an activation message, then this message was sent by its successor on the request path in the previous round. Only *PASSIVE* modules send activation messages, and after sending such a message they delete their last stored request. Since S is simple by P3, this successor has deleted its only request and no longer belongs to the request path. Thus, in both cases, m is the last module of the current request path.

When m becomes *ACTIVE*, it deletes its stored request. Hence, the request path in the next round is a prefix of the current request path. Therefore, P1–P3 are maintained.

Case 3: The ACTIVE module m moves without being blocked by a critical pair. Since m belongs to A^t , its cell is considered empty in the definition of the request path. Thus, its movement alone has no effect on P .

As argued above, m starts its movement side-adjacent to the last module of P . It then follows the request path in reverse direction. By Lemma 7.7, it can do so as long as the next cell it intends to move to lies in the same pseudo-hole as m . During this movement, m instructs adjacent modules to delete their last stored requests. Each such deletion makes the request path in the next round a prefix of the current request path, so P1–P3 are maintained. If the next cell does not lie in the same pseudo-hole as m , we consider m blocked; this case is handled next.

Case 4: The ACTIVE module m is blocked by a critical pair. It remains to consider the case where m is blocked by a critical pair and therefore either becomes *PASSIVE* or sends a cleanup message. Let (m_k, c) , b , H_m , and H_b be defined as in Lemma 7.8. By Lemma 7.8, m correctly determines whether the target cell lies in H_m or in H_b .

If H_m contains the target cell, then m sends a cleanup message to m_k . This message deletes the suffix $(m_i, \dots, m_k)$ of the request path, where $m_i = c$. Hence, the resulting request path is a prefix of the request path before sending the cleanup message, and P1–P3 are maintained.

If H_b contains the target cell, then m becomes *PASSIVE*. All stored requests lie on the boundary of H_b , so P1 holds. The module then stores a request from m_k and becomes the next module m_{k+1} on the request path in the following round, which maintains P2. Since m deleted all previously stored requests when it became *ACTIVE*,

this newly stored request is its only request. Thus, S remains simple, and P3 holds.

Finally, once an *ACTIVE* module reaches the target cell and no cleanup message is in transit, both the request path and the passive segment are empty. Hence, P1–P3 hold trivially, as in the initial case. This completes the induction. $\square$

Existence of Bridged Removable Modules. We call a request path *well-formed* if it satisfies P1–P3. Assuming a well-formed request path, we first show that a bridged removable module exists whenever the passive segment cycles back to itself.

Lemma 7.11. *Let $P = (m_1, m_2, \dots, m_k)$ be a well-formed request path with non-empty passive segment $S = (m_i, \dots, m_k)$. Suppose the next counterclockwise module along the target boundary after m_k is a module m_j with $i \leq j \leq k - 2$. Then at least one of the modules $m_j, m_{j+1}, \dots, m_k$ is bridged removable.*

Proof. Since $j \leq k - 2$, there is at least one module between m_j and m_k along the passive segment S . Otherwise, m_j would be the only side-adjacent neighbor of m_k , because it would be both the predecessor of m_k on P and the next counterclockwise module along the target boundary. In that case, m_k would be simply removable. Since P is well-formed, the passive segment S is simple by P3. Hence,

$$Z := (m_j, m_{j+1}, \dots, m_k)$$

is a simple cycle of modules.

If Z contains a critical pair (c_1, c_2) , then both c_1 and c_2 are bridged removable, and we are done. Hence, assume that Z contains no critical pair. Since Z is a subpath of the request path, it lies on the target boundary. Remove all modules side-adjacent to m_j except for the two modules of Z , namely m_{j+1} and m_k . The remaining modules decompose into at least two maximal side-connected sets. Let M_Z be the maximal side-connected set that contains Z ; see Figures 7.7a and 7.7b.

Let O be the set of points common to the cells occupied by modules in M_Z and the cells not occupied by modules in M_Z . Since Z is a simple cycle on the boundary of the target hole, no module of $M_Z \setminus Z$ contributes to O . Moreover, since Z contains no critical pair, O is a simple polygon whose vertices are corners of cells occupied by modules of Z . We distinguish whether this polygon encloses the target hole.

Case 1: O does not enclose the target hole. This case is illustrated in Figure 7.7a. Since we operate on the square grid, every exterior angle of O is $\pm \frac{\pi}{2}$. The sum of exterior angles of a simple polygon is 2π . Thus, O has at least four vertices with exterior angle $\frac{\pi}{2}$. Each such vertex corresponds to a module of Z whose cell shares two consecutive sides with cells not in Z . At most one of these cells can be occupied by m_j . Therefore, at least three modules of Z are bridged removable.

Case 2: O encloses the target hole. This case is illustrated in Figure 7.7b. The target cell is always adjacent to a non-*PASSIVE* module. Since Z consists only of *PASSIVE* modules, O must enclose at least one cell in addition to the target cell. Hence, the width or height of O is at least two cells. It follows that O either has two vertices with exterior angle $-\frac{\pi}{2}$ or contains a side of length at least two.

A vertex with exterior angle $-\frac{\pi}{2}$ corresponds to a module of Z whose cell shares two consecutive sides with cells not in Z , now with Z lying on the outside of the

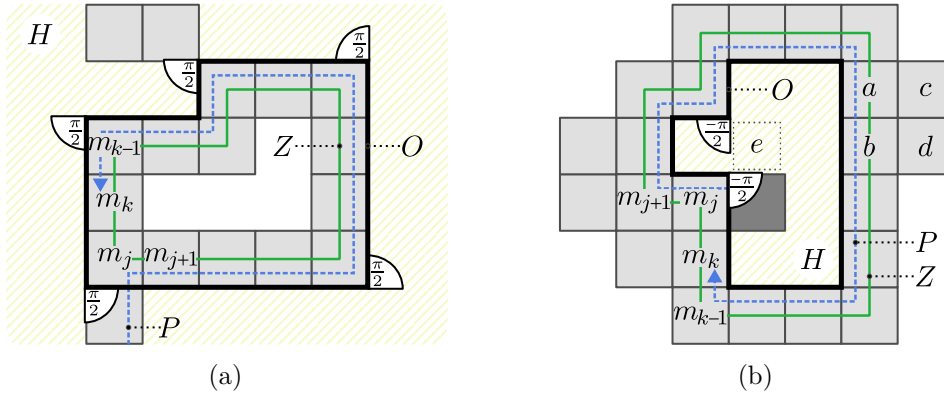


Figure 7.7: The two cases used in the proof of Lemma 7.11. The polygon O does not enclose the target hole H in (a), but encloses it in (b). The exterior angles relevant to the arguments in the proof are highlighted.

polygon. At most one such cell can be occupied by m_j . Thus, if O has two such vertices, at least one corresponding module of Z is bridged removable.

It remains to consider the case where O contains a side of length at least two. Then there are consecutive modules $a, b \in Z$ whose cells are both side-adjacent to the target hole. Let c and d be the cells side-adjacent to a and b , respectively, that are neither in the target hole nor occupied by modules of Z ; see Figure 7.7b. If c is empty, then a is bridged removable. If d is empty, then b is bridged removable. If neither c nor d is empty, then both a and b are bridged removable.

In all cases, Z contains a bridged removable module. $\square$

The previous lemma relies on the fact that the simple cycle consists entirely of *PASSIVE* modules. This is no longer true when the search for a bridged removable module is triggered because the next forwarding direction points to a non-*PASSIVE* module. To analyze this case, we use the following structural property.

A set of cells $V \subseteq C$ is *weakly convex* if, for any two cells $c_1, c_2 \in V$, there exists a shortest path from c_1 to c_2 that is fully contained in V . The following lemma shows that every rhombus is weakly convex, even after adding the neighborhood of its last cell. The weak convexity of the rhombus is used to show that if the target hole has size at least two, then a bridged removable module must exist on the passive segment. The extension to the neighborhood of the last cell is used for the remaining case where the target hole has size one, which is handled separately in Section 7.2.1.

Lemma 7.12. *For any $n \geq 0$, the set of cells $\{v_0, \dots, v_n\} \cup N(v_n)$ is weakly convex, where $v_0, \dots, v_n$ are the cells of a rhombus of size $n + 1$ with center v_0 .*

Proof. For a rhombus of size n , let $\mathcal{R}_n = \{v_0, \dots, v_{n-1}\}$ denote its set of cells, and let $R = \mathcal{R}_{n+1} \cup N(v_n)$. We prove weak convexity by constructing monotone paths. A path in the square grid is *monotone* if it does not contain edges in opposite directions. For example, a northwest-monotone path contains only north and west steps. Every monotone path in the square grid is a shortest path.

Weak convexity of $\mathcal{R}_n$. We first show by induction on n that $\mathcal{R}_n$ is weakly convex. For $n = 0$, the set $\mathcal{R}_n$ is empty, so the claim is trivial. Let $n > 0$, assume that $\mathcal{R}_{n-1}$

is weakly convex, and let $c_1, c_2 \in \mathcal{R}_n$. We show that there exists a shortest path from c_1 to c_2 that is fully contained in $\mathcal{R}_n$.

If $c_1, c_2 \in \mathcal{R}_{n-1}$, then the claim follows directly from the induction hypothesis. If $c_1 = c_2 = v_{n-1}$, the claim is trivial. Hence, it remains to consider the case where exactly one of c_1 and c_2 equals v_{n-1} . Without loss of generality, assume that $c_1 = v_{n-1}$ and $c_2 \in \mathcal{R}_{n-1}$.

First consider the case where c_1 is extremal within its layer. Without loss of generality, assume that c_1 is northernmost. Then every cell of $\mathcal{R}_{n-1}$ lies in the southeast or southwest quadrant relative to c_1 . We take one step south from c_1 into $\mathcal{R}_{n-1}$ and then follow a southeast- or southwest-monotone path within $\mathcal{R}_{n-1}$ to c_2 . This yields a monotone path from c_1 to c_2 inside $\mathcal{R}_n$, and hence a shortest path.

Now consider the case where c_1 is not extremal within its layer. Without loss of generality, assume that c_1 lies in the northwest quadrant of $\mathcal{R}_n$. Then c_1 has neighbors in $\mathcal{R}_{n-1}$ to the east and south. Moreover, no cell of $\mathcal{R}_n$ lies in the northwest quadrant relative to c_1 . Hence, depending on the position of c_2 relative to c_1 , either the east neighbor or the south neighbor of c_1 lies in a monotone direction toward c_2 . We take such a first step into $\mathcal{R}_{n-1}$ and then follow a monotone path inside $\mathcal{R}_{n-1}$ to c_2 . This yields a monotone path from c_1 to c_2 inside $\mathcal{R}_n$, and hence a shortest path.

Hence, every rhombus of size n is weakly convex.

Weak convexity of R . We now show that R is weakly convex. Let $c_1, c_2 \in R$. We show that there exists a shortest path from c_1 to c_2 that is fully contained in R .

If both cells lie in $\{v_n\} \cup N(v_n)$, then they lie in a full 3×3 square of cells, which is weakly convex. If both cells lie in $\mathcal{R}_{n+1}$, then the claim follows from the weak convexity of $\mathcal{R}_{n+1}$ shown above. Thus, assume without loss of generality that $c_1 \in N(v_n) \setminus \mathcal{R}_{n+1}$ and $c_2 \in \mathcal{R}_{n+1}$.

First consider the case where v_n is extremal within its rhombus layer. Without loss of generality, assume that v_n is northernmost. Then every cell of $\mathcal{R}_{n+1}$ lies in the southeast or southwest quadrant relative to any cell in $N(v_n) \setminus \mathcal{R}_{n+1}$. Hence, depending on the position of c_2 relative to c_1 , we can enter $\mathcal{R}_{n+1}$ by following a monotone path moving south, southeast, or southwest. Once inside $\mathcal{R}_{n+1}$, we follow a path of the same monotonicity to c_2 , which exists since $\mathcal{R}_{n+1}$ is weakly convex. The resulting path is monotone, hence shortest, and lies entirely in R .

Now consider the case where v_n is not extremal within its rhombus layer. Without loss of generality, assume that v_n lies in the northwest quadrant of its layer. Then no cell of $\mathcal{R}_{n+1}$ lies in the northwest quadrant relative to c_1 . If c_2 lies in the northeast quadrant relative to c_1 , we move east from c_1 into $\mathcal{R}_{n+1}$ and then follow a northeast-monotone path to c_2 within $\mathcal{R}_{n+1}$. If c_2 lies in the southwest quadrant relative to c_1 , we move south from c_1 into $\mathcal{R}_{n+1}$ and then follow a southwest-monotone path to c_2 within $\mathcal{R}_{n+1}$. Finally, if c_2 lies in the southeast quadrant relative to c_1 , we move east as long as c_2 is not strictly south of the current cell or until we enter $\mathcal{R}_{n+1}$. We then move south as long as c_2 is not strictly east of the current cell or until we enter $\mathcal{R}_{n+1}$. Once inside $\mathcal{R}_{n+1}$, we follow a southeast-monotone path to c_2 within $\mathcal{R}_{n+1}$. In all cases, the constructed path is monotone, hence shortest, and lies entirely in R .

Therefore, $R = \mathcal{R}_{n+1} \cup N(v_n)$ is weakly convex. $\square$

Lemma 7.13. *Let $P = (m_1, \dots, m_k)$ be a well-formed request path with non-empty passive segment $S = (m_i, \dots, m_k)$. Suppose the next counterclockwise module along the target boundary after m_k is not *PASSIVE*, and suppose that m_k is not simply removable. If no module in S is bridged removable, then the target hole has size one.*

Proof. We assume throughout the proof that no module in S is bridged removable and show that this forces the target hole to have size one.

Let m_l be the current position of the *HEAD* module in the rhombus. Let $\mathcal{F}$ denote the set of all *HEAD* and *TAIL* modules. By Lemma 7.6, the set $\mathcal{F}$ consists exactly of the first $l + 1$ cells of the rhombus. In the size-one target-hole case considered later, $\mathcal{F}$ is extended by the target cell and its neighborhood, and these cells are not considered *PASSIVE*. In both cases, $\mathcal{F}$ is weakly convex by Lemma 7.12.

Let m' be the next counterclockwise module along the target boundary after m_k . By assumption, m' is not *PASSIVE*, so $m' \in \mathcal{F}$. Let P' be a shortest path from m' to m_{i-1} that is fully contained in $\mathcal{F}$. Such a path exists by weak convexity. Since S contains only *PASSIVE* modules and is simple by P3, the paths P' and S are disjoint. Thus, their concatenation $Z = S \circ P'$ forms a simple cycle.

Let D be the set of all modules in S that are side-adjacent to the target hole. The proof first establishes three properties of D : the set D is non-empty, its modules cannot be part of critical pairs, and no two consecutive modules of S belong to D . We then use these properties to show that each module in D induces a local C-shape around its adjacent empty cell; see Figure 7.8. Because modules in D are not part of critical pairs, combining two such C-shapes forces all neighbors around this empty cell to be occupied, and therefore the target hole has size one.

Claim 1. *The set D is non-empty.*

Proof. First, assume that S consists only of the module m_k . Since every request is initiated by a module in $\mathcal{F}$, the predecessor m_{k-1} of m_k on the request path belongs to $\mathcal{F}$. If $m' = m_{k-1}$, then m_k has only one side-adjacent neighbor, contradicting the assumption that m_k is not simply removable. Hence, assume that $m' \neq m_{k-1}$.

If m' and m_{k-1} are not adjacent, then every shortest path from m' to m_{k-1} passes through $m_k \notin \mathcal{F}$, contradicting weak convexity of $\mathcal{F}$. If m' and m_{k-1} are adjacent, then weak convexity gives a shortest path from m' to m_{k-1} through a module corner-adjacent to m_k . This again contradicts that m_k is not simply removable. Thus, S contains at least two modules.

Since S is a contiguous subpath of the target boundary, at least one module of S is side-adjacent to the target hole. Hence, D is non-empty. $\triangleleft$

Claim 2. *For every module $c \in D$, exactly three modules are side-adjacent to c . Moreover, no module $c \in D$ is part of a critical pair.*

Proof. Let $c \in D$. Then c is side-adjacent to an empty cell in the target hole and to two modules on the cycle Z . If the fourth side-adjacent cell of c were empty, then c would be bridged removable, contradicting our assumption. Hence, the fourth side-adjacent cell is occupied, and exactly three modules are side-adjacent to c .

A module in a critical pair has at most two side-adjacent modules. Therefore, c cannot be part of a critical pair. $\triangleleft$



Figure 7.8: C-shapes around a cell e of the target hole. (a) The C-shape induced by a module $m \in D$ and the notation used in the proof of Lemma 7.13. (b) The union of the C-shapes induced by m and m^* . The dot marks the only uncovered cell of $N(e)$; if it is empty, then there is a critical pair side-adjacent to the target hole.

Claim 3. *If $m_j, m_{j+1} \in S$ are consecutive and $m_j \in D$, then $m_{j+1} \notin D$.*

Proof. Assume for contradiction that $m_j \in D$ and $m_{j+1} \in D$. By Claim 2, exactly three modules are side-adjacent to each of them.

The modules side-adjacent to m_j are its predecessor on Z , the module m_{j+1} , and an additional module $a \in N(m_{j+1})$. Similarly, the modules side-adjacent to m_{j+1} are its successor on Z , the module m_j , and an additional module $b \in N(m_j)$.

Thus, the modules side-adjacent to m_j contain a path from a to m_{j+1} through b that lies entirely in $N(m_j)$. Hence, m_j is bridged removable, contradicting our assumption. Therefore, no two consecutive modules of S belong to D . $\triangleleft$

We now analyze the local structure around an arbitrary module $m \in D$, which exists by Claim 1. Let e be the empty cell in the target hole that is side-adjacent to m . This cell is unique by Claim 2. We show that the occupied cells in $N(m) \cap N(e)$ form a C-shape containing another module of D distinct from m .

Label the cells in $N(m) \cap N(e)$ as in Figure 7.8a. Let a and b be the successor and predecessor of m on the cycle Z , respectively. Let c and d be the cells such that (m, a, c, e) and (m, b, d, e) are 4-cycles.

Case 1: $a, b \notin \mathcal{F}$. Then both c and d must be occupied; otherwise $a \in D$ or $b \in D$, contradicting Claim 3. The unique shortest path from c to d passes through the empty cell e . If both c and d belonged to $\mathcal{F}$, this would contradict weak convexity of $\mathcal{F}$. Hence, at least one of c and d lies outside $\mathcal{F}$. Since both are side-adjacent to e , at least one of them belongs to D .

Case 2: *Exactly one of a and b belongs to $\mathcal{F}$.* Without loss of generality, assume that $a \notin \mathcal{F}$ and $b \in \mathcal{F}$. As before, the cell c must be occupied; otherwise $a \in D$, contradicting Claim 3. Any shortest path from c to b passes through m or e , both of which are outside $\mathcal{F}$. Thus, weak convexity of $\mathcal{F}$ implies that $c \notin \mathcal{F}$. Since c is side-adjacent to e , it follows that $c \in D$.

To complete the C-shape, it remains to show that d is occupied. Assume for contradiction that d is empty. Let f_1, f_2 and f_3 be the cells in $N(e)$ that are side-adjacent to c, e and d , respectively; see Figure 7.8a. Any shortest path from one of the cells f_1, f_2, f_3 to $b \in \mathcal{F}$ passes through c, e , or d , all of which lie outside $\mathcal{F}$. Hence, none of f_1, f_2, f_3 belongs to $\mathcal{F}$.

Moreover, none of these cells can be empty. If f_1 were empty, then $c \in D$ would be bridged removable. If f_2 were empty, then c and f_1 would be consecutive modules in D , contradicting Claim 3. If f_3 were empty, then $f_2 \in D$ would be bridged removable. Therefore, f_2 and f_3 are consecutive modules in D , again contradicting Claim 3. Thus, d is occupied.

Case 3: $a, b \in \mathcal{F}$. This case cannot occur. Indeed, every shortest path from a to b passes through $m \notin \mathcal{F}$, contradicting weak convexity of $\mathcal{F}$.

We have shown that, for every $m \in D$, the occupied cells in $N(m) \cap N(e)$ form a C-shape that contains another module $m^* \in D$ with $m^* \neq m$. Consider the union of the two C-shapes induced by m and m^* ; see Figure 7.8b. This union covers all cells in $N(e)$ except for one cell that is corner-adjacent to e .

We claim that this remaining cell is occupied. If it were empty, then the two modules side-adjacent to both this cell and e would form a critical pair. These two modules are side-adjacent to the target hole. They cannot both belong to $\mathcal{F}$, since every shortest path between them passes through empty cells, contradicting weak convexity of $\mathcal{F}$. Hence, at least one of them belongs to S and therefore to D . This contradicts Claim 2. Thus, the remaining cell is occupied.

Therefore, all cells adjacent to e are occupied. Since e was an arbitrary cell of the target hole side-adjacent to a module in D , it follows that the target hole contains only e . Thus, the target hole has size one. $\square$

Filling the Target Cell. The following lemma shows that each request either fills the target cell or reduces the situation to the special case of a size-one target hole within a linear number of rounds.

Lemma 7.14. *Consider a round in which a module initiates a request for some target cell, and assume that at least one PASSIVE module exists. Then, within $\mathcal{O}(n)$ rounds, either the target cell is filled or the target hole consists only of the target cell.*

Proof. Let e be the target cell and assume that the target hole has size larger than one, as otherwise there is nothing to show.

Consider the first round after the request for e is initiated in which the request is not forwarded to the next module along the target boundary. Let $P = (m_1, m_2, \dots, m_k)$ be the request path and let S be its passive segment in that round. Let $\mathcal{F}$ be defined as in the proof of Lemma 7.13.

By weak convexity of $\mathcal{F}$, no PASSIVE module is fully enclosed by modules of $\mathcal{F}$. Moreover, the configuration remains side-connected throughout the execution, since only simply removable and bridged removable modules become ACTIVE. Since the request starts at a module in $\mathcal{F}$ and follows the target boundary, it must eventually encounter a PASSIVE module. Thus, S is non-empty in this round. By Lemma 7.10, the passive segment ends at the last module of P . Hence, m_k is PASSIVE.

Let m' be the next counterclockwise module along the target boundary after m_k . In this round, the request is not forwarded by m_k . This can happen only if one of the following holds:

1. m_k is simply removable,

2. $m' \in S$, or
3. m' is not *PASSIVE*.

In the first case, m_k becomes *ACTIVE*. In the second case, Lemma 7.11 implies that a module of S becomes *ACTIVE*. In the third case, Lemma 7.13 implies, since the target hole has size larger than one, that some module of S is bridged removable and becomes *ACTIVE*. Thus, in all cases, some *PASSIVE* module m_i with $i \leq k$ becomes *ACTIVE*.

From this point on, the cell of m_i is considered empty for all geometric notions. Let $\tilde{H}_i$ be the pseudo-hole containing the cell of m_i , and let $\tilde{H}_0$ be the pseudo-hole containing the target cell e . Since both simple removability and bridged removability require side-adjacency to the target hole, the module m_i is side-adjacent to the target hole when it becomes *ACTIVE*.

Case 1: $\tilde{H}_i \neq \tilde{H}_0$. Consider a shortest corner-path from the cell of m_i to the target cell e that lies entirely in the target hole. Let b_1 be the last cell of this path contained in $\tilde{H}_i$, and let b_2 be the first cell on the path not contained in $\tilde{H}_i$. The cells b_1 and b_2 are adjacent and lie in different pseudo-holes. Hence, they are the two bridge cells between $\tilde{H}_i$ and the adjacent pseudo-hole containing b_2 . By the uniqueness of the critical pair separating two adjacent pseudo-holes, b_1 and b_2 are unique. Let (c_1, c_2) be the critical pair adjacent to these bridge cells. At least one of c_1 and c_2 lies on the request path. Let j be the maximum index such that $m_j \in \{c_1, c_2\}$.

Since the cell of m_i and the bridge cell b_1 lie in the same pseudo-hole, Lemma 7.7 allows the *ACTIVE* module to move from its current cell to b_1 while remaining side-adjacent to the boundary of $\tilde{H}_i$. During this movement, it follows the request path in reverse direction and instructs adjacent modules to delete their stored requests until it is side-adjacent to m_j . This takes $\mathcal{O}(k - j)$ rounds.

Let $\mathcal{S}$ denote the number of square sides shared by an empty cell of the target hole and an occupied cell on its boundary. If an occupied cell belongs to the boundaries of several holes, each such side is assigned to exactly one of them. When the *ACTIVE* module becomes *PASSIVE*, the target hole is split into two holes, one of which contains the cell previously occupied by m_i . Before m_i became *ACTIVE*, the modules $m_{j+1}, \dots, m_k$ lay on the boundary of this separated hole. Hence, within $\mathcal{O}(k - j)$ rounds, the value of $\mathcal{S}$ decreases by $\Omega(k - j)$.

Initially, $\mathcal{S} = \mathcal{O}(n)$. Therefore, after $\mathcal{O}(n)$ rounds spent in this case, either the target hole has size one or the *ACTIVE* module reaches a situation with $\tilde{H}_i = \tilde{H}_0$.

Case 2: $\tilde{H}_i = \tilde{H}_0$. In this case, the *ACTIVE* module and the target cell lie in the same pseudo-hole. By Lemma 7.7, the *ACTIVE* module can move to the target cell while remaining side-adjacent to the boundary of this pseudo-hole. The request path has length at most $\mathcal{O}(n)$, so activating m_i and moving it along the request path in reverse direction to the target cell takes $\mathcal{O}(n)$ rounds.

Thus, within $\mathcal{O}(n)$ rounds, either the target cell is filled or the target hole consists only of the target cell. $\square$

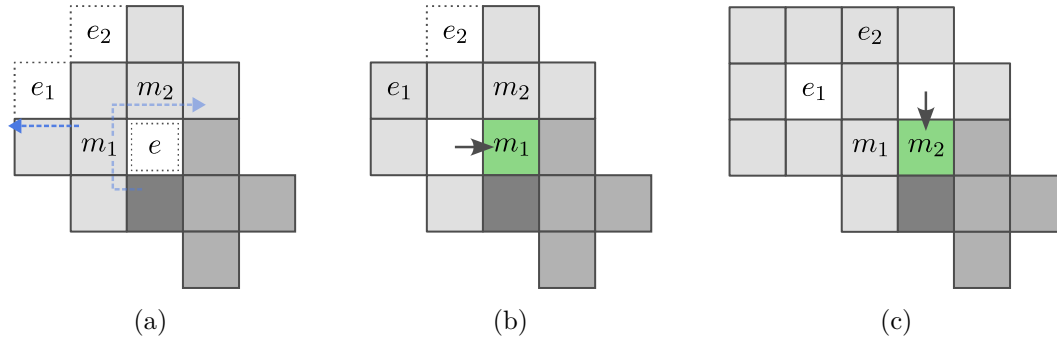


Figure 7.9: A size-one target hole without any bridged removable module in its boundary. Cells e_1 and e_2 are necessarily empty. (a) Module m_1 initiates a secondary request for cell e_1 . (b) If e_1 is filled, then m_1 becomes bridged removable and slides into e . (c) Otherwise, the hole containing e_1 is reduced to a single cell, which forces e_2 to be occupied; hence, m_2 becomes bridged removable and slides into e .

7.2.1. The Case of a Size-One Target Hole

It remains to handle the case where the target hole consists of a single cell e . By Lemma 7.13, this is the only case in which the search along the passive segment may fail to find a bridged removable module.

Let Z be the simple cycle formed by the eight modules on the target boundary, starting at the *HEAD* and following the request forwarding direction. Let m_1 and m_2 be the first two *PASSIVE* modules in Z that are side-adjacent to the target cell e . Since neither of them is bridged removable, there exist two empty cells e_1 and e_2 that are corner-adjacent to m_1 and m_2 , respectively, and lie in the same hole; see Figure 7.9.

After the activation message returns to the *HEAD* without finding a bridged removable module, module m_1 initiates a secondary request for cell e_1 . During this request, m_1 temporarily assumes the role of the *HEAD*, and all other modules of Z temporarily assume the role of *TAIL* modules.

The only proofs above that rely on the positions of *HEAD* and *TAIL* modules are those of Lemmas 7.13 and 7.14. In both proofs, we already allow the set $\mathcal{F}$ of cells considered to be occupied by non-*PASSIVE* modules to be extended by the target cell and its neighborhood. By Lemma 7.12, this extended set is weakly convex. Therefore, all preceding lemmas apply to the secondary request as well.

We can thus apply Lemma 7.14: within $\mathcal{O}(n)$ rounds, either the cell e_1 is filled or the hole containing e_1 is reduced to the single cell e_1 . In the first case, m_1 becomes bridged removable with respect to Z and can slide into e . In the second case, e_2 must be occupied, which implies that m_2 becomes bridged removable with respect to Z and can slide into e . In either case, some module moves into e . Afterwards, all modules of Z return to their original roles, and the algorithm proceeds as usual.

Corollary 7.15. *If the target hole consists of a single cell, then the target cell is filled within $\mathcal{O}(n)$ rounds.*

7.2.2. Correctness and Runtime

We now combine the previous lemmas to establish correctness and runtime of the sequential rhombus formation algorithm.

Theorem 7.16. *Let (M^0, s^0, l) be an arbitrary initially side-connected configuration of n modules with a unique leader in cell $l \in M^0$, and suppose that all other modules start in the same state. After $\mathcal{O}(n^2)$ rounds of the rhombus formation algorithm, each cell in the rhombus of size n with center l contains a terminated module, and side-connectivity of the nonmoving modules is preserved throughout the execution.*

Proof. Consider a round in which the *HEAD* initiates a request, and assume that at least one *PASSIVE* module exists. By Lemma 7.14, the request fills the target cell or reduces the target hole to a single cell within $\mathcal{O}(n)$ rounds. In the latter case, Corollary 7.15 implies that the target cell is filled within another $\mathcal{O}(n)$ rounds. Once the target cell is filled, the previous *HEAD* becomes a *TAIL* module and never assumes the *HEAD* role again, while the module occupying the target cell becomes the new *HEAD*. Hence, within $\mathcal{O}(n)$ rounds, the number of *PASSIVE* modules decreases by one. Since there are at most n *PASSIVE* modules, after $\mathcal{O}(n^2)$ rounds no *PASSIVE* module remains.

Now consider a round in which the *HEAD* initiates a request and no *PASSIVE* module remains. By Lemma 7.6, the cells occupied by *TAIL* modules form the prefix of the rhombus preceding the *HEAD*. Since no *PASSIVE* module remains, all cells of the rhombus of size n with center l are occupied. The request then traverses the entire target boundary in $\mathcal{O}(n)$ rounds, after which the *HEAD* terminates. Termination subsequently propagates through the rhombus within another $\mathcal{O}(n)$ rounds. Thus, after $\mathcal{O}(n^2)$ rounds in total, each cell of the rhombus of size n with center l contains a terminated module.

It remains to argue that side-connectivity of the nonmoving modules is preserved throughout the execution. A module becomes *ACTIVE* only if it is simply removable or bridged removable. In both cases, removing this module from the set of nonmoving modules preserves side-connectivity by definition. All other modules remain stationary during the movement of the *ACTIVE* module. Hence, side-connectivity of the nonmoving modules is preserved in every round. $\square$

7.3. Utilizing Parallelism

In the previous sections, we presented and analyzed a distributed rhombus formation algorithm that sequentially fills all cells of the rhombus in $\mathcal{O}(n^2)$ rounds. No sequential algorithm can be asymptotically faster in the worst case. To see this, consider a rhombus with $j + 1$ layers, where layer L_0 contains one cell and each layer L_i for $1 \leq i \leq j$ contains $4i$ cells. The total number of cells is therefore

$$n = 1 + \sum_{i=1}^j 4i = 2j^2 + 2j + 1.$$

Now place the modules in a straight line of length n starting at the center of the rhombus. Exactly $n - (j + 1) = \Theta(j^2)$ modules lie outside the rhombus, and the i -th

outer module must move distance $\Omega(i)$ to enter the rhombus. Hence, the number of rounds required by any sequential algorithm is at least

$$\sum_{i=1}^{\Theta(j^2)} i = \Omega(j^4) = \Omega(n^2).$$

To improve runtime, modules must therefore move in parallel. However, preserving connectivity during parallel movement is non-trivial. While simple removability can be detected locally, identifying bridged removable modules requires more extensive communication, and not all such modules can move simultaneously without disconnecting the configuration. Coordinating mutually exclusive movements would already require $\Omega(n^2)$ rounds under constant memory, so we pursue a different approach.

Spanning-tree strategy. We begin with a preprocessing phase in which a spanning tree $\mathcal{T}$ rooted at the leader is constructed. This can be done by a distributed breadth-first search in $\mathcal{O}(n)$ rounds using only constant memory per module. Afterwards, leaves of $\mathcal{T}$ move clockwise along hole boundaries whenever the move decreases their depth in $\mathcal{T}$. Intuitively, repeatedly decreasing the depth of leaves drives modules closer to the root of $\mathcal{T}$, and thus toward cells of smaller distance to the center of the rhombus. Depth values are never computed explicitly; each module stores only a parent pointer, and all movement decisions are made from its local neighborhood. As long as $\mathcal{T}$ remains a valid spanning tree, such movements cannot violate connectivity.

Nevertheless, locked configurations may still arise, that is, configurations in which this strategy makes no further progress. This can happen in two ways. First, no leaf is side-adjacent to a hole; this cannot be resolved locally, even with logarithmic memory to compute and store depth values; see Figure 7.10a. Second, a leaf cannot locally determine whether a move decreases its depth in $\mathcal{T}$; see Figure 7.10b.

To guarantee progress, we therefore run the sequential algorithm from the previous sections alongside the spanning-tree strategy. The additional information provided by $\mathcal{T}$ can also be used by the sequential algorithm: whenever a *PASSIVE* module checks whether it is simply removable, it also checks whether it is a leaf. In either case, removing the module preserves connectivity, and the module becomes *ACTIVE*. In particular, if a leaf lies on the boundary of the target hole, its clockwise movement along this boundary brings it closer to the request that is traversing the boundary counterclockwise.

Allowing leaves to move in parallel introduces interactions with the request and activation mechanism that do not occur in the purely sequential algorithm. Conversely, the sequential algorithm does not account for maintaining a spanning tree, which must now be preserved throughout the execution. To preserve the correctness guarantees established for the sequential algorithm in Theorem 7.16, we must address the following issues:

1. A request or activation message may activate a module that is not a leaf of $\mathcal{T}$. In this case, we locally redirect parent pointers so that $\mathcal{T}$ again becomes a valid spanning tree.

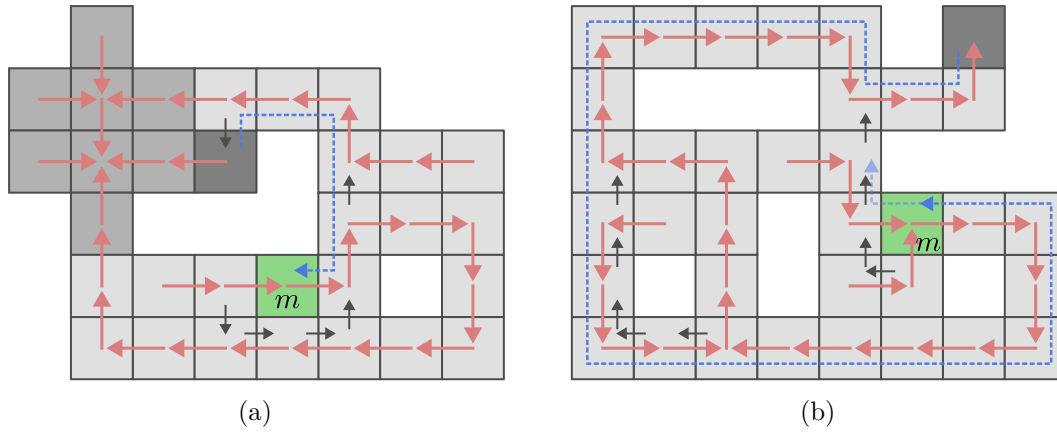


Figure 7.11: Local rerooting of tree edges during request forwarding, activation messages, and activation of a module. In (a), module m becomes *ACTIVE* after receiving a request. In (b), module m becomes *ACTIVE* after receiving an activation message sent by the module northwest of m . Solid red arrows indicate tree edges before rerooting, and small black arrows indicate the tree edges after rerooting.

We maintain the invariant that the subtree of $\mathcal{T}$ induced by all *HEAD* and *TAIL* modules is a shortest-path tree rooted at the center of the rhombus. To preserve this invariant, whenever the *HEAD* role is delegated to a new module, the new *HEAD* sets its parent pointer toward the center of the rhombus. This direction can be inferred from the relative position of the previous *HEAD* and the surrounding *TAIL* modules.

Rerooting children of simply removable modules. As a request is forwarded along the target boundary, every *PASSIVE* module receiving the request sets its parent pointer to the module from which it received the request. Let $\mathcal{T}'$ be the subgraph of $\mathcal{T}$ induced by the *HEAD* and *TAIL* modules and the modules on the request path. Restricted to the non-*PASSIVE* modules, $\mathcal{T}'$ is a shortest-path tree. Moreover, since the passive segment is simple by P3, the subgraph $\mathcal{T}'$ is itself a tree.

This structure makes it possible to identify ancestors of the last module on the request path locally, because they all lie on the request path. If a simply removable module m becomes *ACTIVE*, all modules side-adjacent to m lie in the same connected component of $N(m) \cap M$, where connectivity is with respect to side-adjacency. Thus, the children of m in $\mathcal{T}$ can be rerooted locally to one of its ancestors; see Figure 7.11a.

Rerooting children of bridged removable modules. For a bridged removable module m , the modules side-adjacent to m may lie in two distinct connected components of $N(m) \cap M$. One of these components contains the predecessor of m on the request path, which is the parent of m in $\mathcal{T}$ and therefore not a descendant of m . To reroot all children of m safely, we must also identify a non-descendant of m in the other component.

Let m_k be the last module on the request path, let m_{k+1} be the next module along the target boundary in counterclockwise order, and let m_{k-1} be the predecessor of m_k on the request path. Suppose that m_k would normally forward a request to m_{k+1} ,

but m_{k+1} is either a *HEAD* or *TAIL* module, or a *PASSIVE* module that already lies on the request path. In either case, m_{k+1} is not a descendant of m_k .

If m_k is not bridged removable, it deletes its stored request, sends an activation message to m_{k-1} , and sets its parent pointer to m_{k+1} . As with request forwarding, every module that receives an activation message sets its parent pointer to the module from which it received the message, unless it is bridged removable and becomes *ACTIVE*. Thus, when a bridged removable module eventually becomes *ACTIVE*, the modules in both connected components of its neighborhood contain locally identifiable non-descendants. All children of this module can therefore be rerooted to neighboring non-descendants; see Figure 7.11b.

7.3.2. Moving Through Leaves

In this subsection, we consider the cases in which the movement of an *ACTIVE* module is blocked by a leaf module that does not lie on the request path. In each case, the *ACTIVE* module makes progress by a local state swap: instead of moving into the blocked cell, it delegates the *ACTIVE* role to the blocking leaf.

First, consider a slide from cell m to cell a , where m contains the *ACTIVE* module; see Figure 7.12a. Let (m, a, b, c) form a clockwise 4-cycle in which c is the last and b is the second-last module on the request path P . If a is occupied by a leaf with $a \notin P$, then the module at m becomes *PASSIVE* and sets its parent pointer to c . The leaf at a becomes *ACTIVE* and sets its parent pointer to b , and c deletes its stored request from b .

Second, consider a convex transition from cell m to cell b , and let (m, a, b, c) be a clockwise 4-cycle in which c is the last module on the request path; see Figure 7.12b. If b is occupied by a leaf with $b \notin P$, we use the same strategy: the module at m becomes *PASSIVE*, the leaf at b becomes *ACTIVE*, and both set their parent pointers to c . If b is empty and a is occupied by a leaf with $a \notin P$, then the module at m becomes *PASSIVE*, while the leaf at a slides to b and becomes *ACTIVE*; see Figure 7.12c. Again, both modules set their parent pointers to c .

These cases cover all positions that must be empty for a slide or convex transition to be possible. In each case, progress is made by exchanging the *ACTIVE* role with a leaf and updating only parent pointers of leaves. Thus, the tree structure is preserved.

7.3.3. Ignoring Unattached Leaves

To preserve the request-path properties P1–P3, we extend the definition of the set A^t of cells treated as empty. In addition to the *ACTIVE* module, A^t contains every leaf module that does not lie on the request path and whose parent does not lie on the request path. Thus, such leaves are treated as empty cells in all geometric definitions, including holes, boundaries, and request paths.

This is justified for two reasons. First, by Section 7.3.2, the *ACTIVE* module can move through such leaves. Second, when a module m on the request path is removed, any leaf whose parent is not m still has a path to the root in $\mathcal{T}$. If this path does not contain m , then removing m does not affect the connectivity of that leaf to the root. If the path does contain m , then it also contains both a child of m and the parent of m , and the rerooting strategy from Section 7.3.1 preserves connectivity.

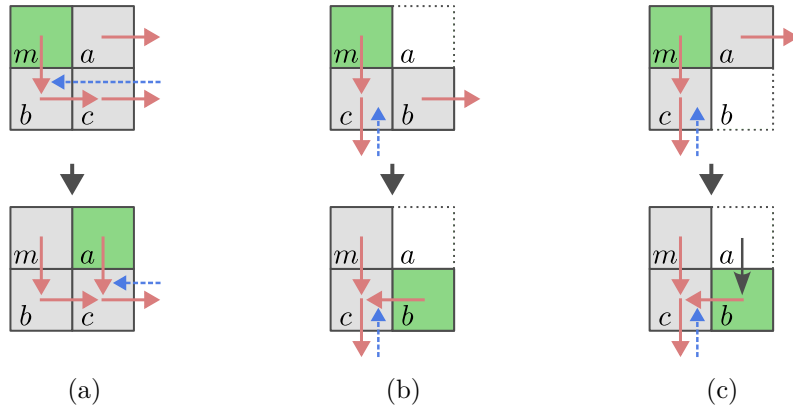


Figure 7.12: Local state swaps when an *ACTIVE* module at m is blocked by a leaf that does not lie on the request path P . (a) A slide into a is blocked. (b) A convex transition into b is blocked. (c) A convex transition is blocked by a leaf at a , while b is empty. The case where both a and b are occupied is analogous to (b).

The only remaining issue arises when a leaf sets its parent pointer to a module that has already forwarded a request. To prevent this, leaves may neither set their parent pointer to a module on the request path nor move while their current parent lies on the request path. However, a small edge case remains when a leaf sets its parent pointer to a module that receives a request in the same round.

Consider a counterclockwise 4-cycle $(m_k, m_{k-1}, m_{k-2}, m)$, where $m_k, m_{k-1}, m_{k-2} \in P$ are the last three modules on the request path and $m \notin P$ is a leaf of $\mathcal{T}$. The leaf m must have moved to its current position after m_{k-2} forwarded the request to m_{k-1} ; otherwise, the request would have been forwarded directly to m . For example, in Figure 7.13, the leaf m slides into its current position in the same round in which m_k receives a request. The case where m moves by a convex transition is analogous.

In this round, m_{k-1} may no longer lie on the target boundary. By the definition of the request path, the request path then ends at m_{k-2} , and P1 is temporarily violated. However, in the next round, m receives the request from m_k and becomes *ACTIVE*, since it is a leaf. The property is therefore immediately restored.

This edge case does not affect the analysis because the resulting configuration is the same as if m had first become *ACTIVE* and then moved into its current position. Moreover, if the leaf m does not form the above 4-cycle with three consecutive modules on the request path, then it receives the request in the same round regardless of whether it has just moved into its position or was already present when the request was initiated. Thus, all remaining cases behave as in the sequential algorithm.

7.3.4. Theoretical Guarantees

We now combine the arguments from the previous subsections and show that the guarantees of the sequential algorithm carry over to its parallel extension. The worst-case runtime bound remains $\mathcal{O}(n^2)$, but the experimental results in the next section show that the parallel extension performs significantly better on average.

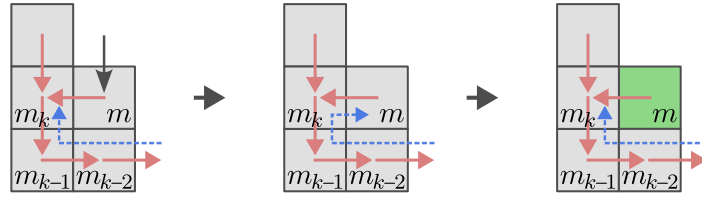


Figure 7.13: A leaf module m sliding adjacent to m_k in the same round in which m_k receives a request. In this situation, P1 may be violated temporarily. Activating m in the next round restores the property.

Theorem 7.17. *Under parallel execution, the rhombus formation algorithm satisfies the guarantees of Theorem 7.16: it reconfigures any initially side-connected configuration with a leader into a rhombus centered at the leader, preserves side-connectivity of the nonmoving modules in every round, and terminates within $\mathcal{O}(n^2)$ rounds.*

Proof. We argue that the lemmas used in the proof of Theorem 7.16 remain valid under parallel execution.

First, side-connectivity of the nonmoving modules is preserved, and $\mathcal{T}$ remains a valid spanning tree. By Section 7.3.1, whenever a non-leaf module becomes *ACTIVE*, its children are locally rerooted to neighboring non-descendants. Thus, after this module is removed from the set of nonmoving modules, the remaining nonmoving modules stay connected through $\mathcal{T}$. Moving a leaf preserves side-connectivity as well, because removing a leaf from a spanning tree cannot disconnect the remaining modules. Moreover, by Section 7.3.2, the local state swaps used when an *ACTIVE* module is blocked by a moving leaf update only parent pointers of leaves. Hence, they preserve the tree structure.

Second, by Section 7.3.3, every leaf that does not lie on the request path and whose parent does not lie on the request path is included in A^t and is therefore treated as empty in the geometric definitions. With this definition of A^t , the request-path properties from Lemma 7.10 remain valid. Leaves are not allowed to set their parent pointer to a module on the request path, and leaves whose parent already lies on the request path are not allowed to move. The only remaining edge case can temporarily violate P1. As shown in Section 7.3.3, this violation is resolved in the next round, after which the request path and the position of the *ACTIVE* module are the same as in the corresponding sequential execution.

The geometric arguments used to prove Lemmas 7.8, 7.11 and 7.13 depend only on these request-path properties and on the reduced set of modules $M^t \setminus A^t$. Hence, they continue to apply under parallel execution. Therefore, the progress lemma Lemma 7.14 also remains valid: within $\mathcal{O}(n)$ rounds, a request either fills its target cell or reduces the target hole to the size-one case. The size-one case is still resolved by Corollary 7.15: once the request path encloses a size-one target hole, the local configuration does not change, since modules on the request path are not allowed to move and no leaf may set its parent pointer to a module on the request path.

It follows that every target cell is eventually filled. By Lemma 7.6, the *TAIL* modules occupy a growing prefix of the rhombus, and once no *PASSIVE* module remains, termination proceeds as in the sequential algorithm. Thus, the algorithm

forms a rhombus centered at the leader and all modules eventually terminate.

For the runtime, parallel leaf movements never delay a request: they either occur independently of the request mechanism or are handled by swapping the state with the *ACTIVE* module. Hence, each request still completes within $\mathcal{O}(n)$ rounds. Since at most n target cells are filled, the total runtime remains bounded by $\mathcal{O}(n^2)$ rounds. $\square$

7.4. Parallel Variants and Experimental Evaluation

To further improve performance, we developed two variants, V1 and V2, that increase the degree of parallelism. Variant V1 allows leaf modules to perform state swaps similar to those of the *ACTIVE* module whenever their movement trajectories intersect. This enables progress in configurations such as the one in Figure 7.10b without waiting for requests. Variant V1 already yields a significant speedup over the purely sequential algorithm in most cases. Importantly, correctness does not depend on the specific movement choices of the leaves, as long as leaves are not allowed to set their parent pointer to a module on the request path. However, since leaf modules are restricted to moves that decrease their depth in $\mathcal{T}$, certain configurations still become locked. In such cases, the runtime is asymptotically no better than that of the sequential algorithm.

Variant V2 addresses this limitation by relaxing the depth-decrease requirement. Leaves move as in V1, with one addition: a leaf performs an available move even if it cannot locally verify that the move decreases its depth in $\mathcal{T}$. Such moves may temporarily increase the depth of a leaf and delay its individual progress, but they can also turn its parent into a leaf, thereby unlocking further parallel movement. To prevent unproductive cycles in which leaves repeatedly step over each other, we do not allow leaves to set their parent pointer to another *PASSIVE* leaf. In our experiments, V2 consistently outperforms V1 and appears to achieve linear runtime on average.

Simulation setup and evaluation. We evaluated the performance of the sequential algorithm and both parallel variants, V1 and V2, on structured and random configurations using a custom-built simulator. For each configuration type and size, we ran multiple independent simulations.

The structured input configurations are *lines*, where modules are arranged in a straight horizontal segment; *spirals*, generated by placing modules in a square spiral pattern; and *chains*, constructed as cascaded sequences of critical pairs. In all structured configurations, the initial leader is placed explicitly at a degree-one module. The top row in Figure 7.14 shows the number of rounds required to reconfigure a line and a chain into a rhombus, for configurations of size 10 to 120 in increments of 10. Spiral configurations are excluded from the plots because they follow the same trend as line configurations. We also include the Earth Mover’s Distance (EMD), which measures the minimum total movement required to transform the initial configuration into the target rhombus, as a lower bound for any sequential solution. The number of modules n is included as a lower bound for V1 and V2.

As expected, the runtime of the sequential algorithm grows quadratically on both line and chain configurations, which are worst-case inputs in the sequential setting. In

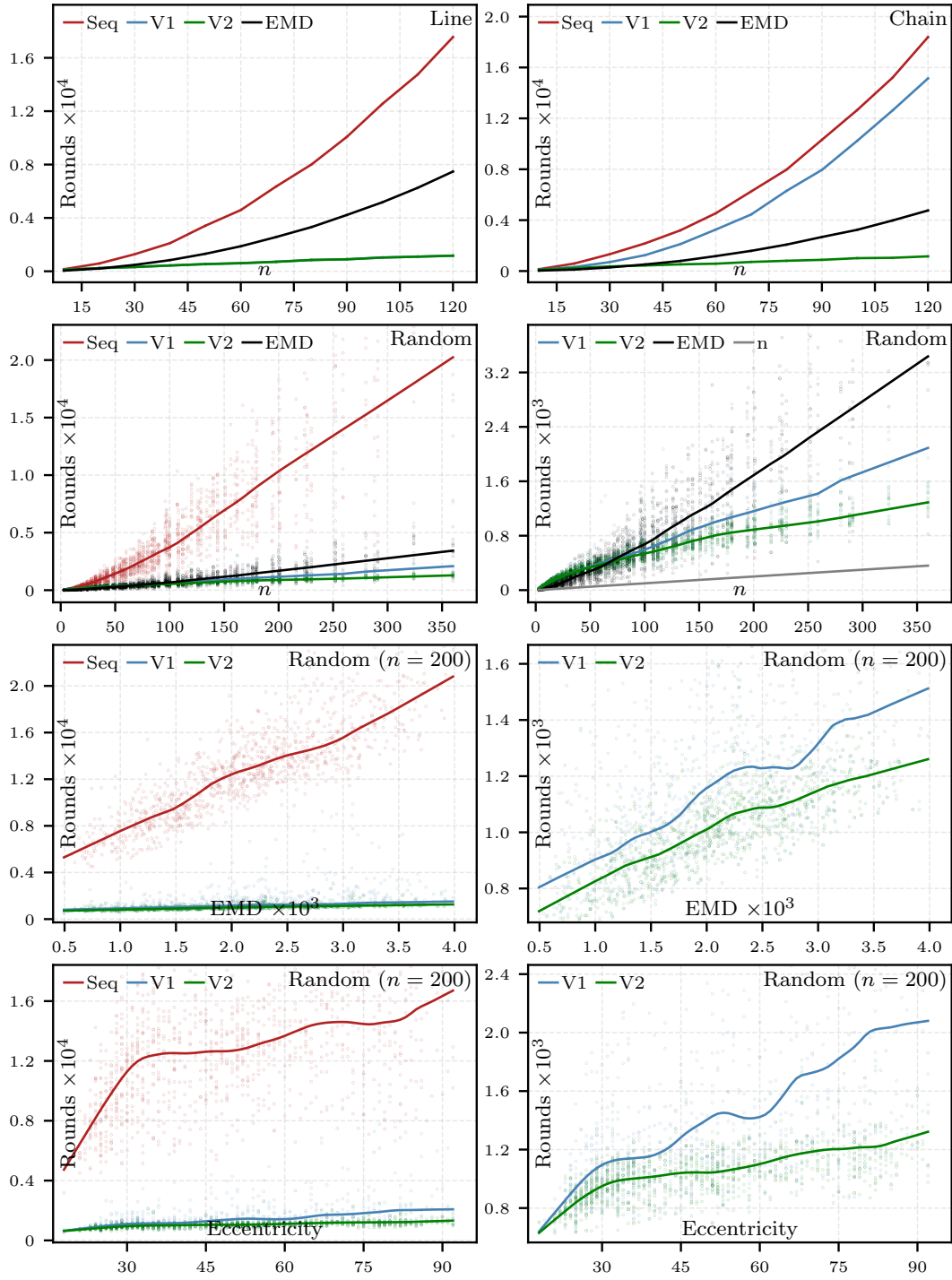


Figure 7.14: Runtime comparison between the sequential rhombus formation algorithm (Seq) and the two parallel variants V1 and V2. The top row shows line and chain configurations; the second row shows random percentage-based configurations. The third and fourth rows compare runtime against Earth Mover's Distance (EMD) and leader eccentricity for random configurations of fixed size $n = 200$. In the second, third, and fourth rows, the right plot shows the same data with the sequential runtime omitted for better resolution. Scatter points correspond to individual runs, while curves are obtained by locally weighted regression.

contrast, the runtimes of V1 and V2 coincide on line and spiral configurations, where leaves are not blocked during reconfiguration. For chain configurations, however, V1 yields only a constant-factor improvement over the sequential algorithm, while V2 appears to achieve linear runtime.

We used two different types of random input configurations:

1. *Percentage-based configurations.* Starting from a filled rectangle, modules are removed uniformly at random while preserving connectivity, until a specified percentage remains.
2. *Growth-based configurations using Perlin noise.* These configurations are grown from a single seed module. In each step, all empty cells side-adjacent to existing modules are assigned a weight based on a fixed two-dimensional Perlin noise map. Among the 10 cells with the highest weights, one is selected uniformly at random and added to the configuration. This method produces irregular, organic shapes with many holes.

For each rectangle of dimensions (x, y) , with x ranging from 6 to 20 in steps of 2 and y ranging from x to 20 in steps of 2, we generated 10 configurations with 10%, 30%, 50%, 70%, and 90% of the original modules retained. This resulted in 1800 simulations in total. The second row in Figure 7.14 shows the results of these runs.

Both parallel variants significantly improve runtime over the purely sequential algorithm and even outperform the lower bound given by the Earth Mover's Distance. This is expected, since the parallel variants allow multiple modules to move simultaneously, whereas the EMD measures total movement. Moreover, V2 consistently outperforms V1 across all simulations. Thus, the speedup obtained from increased parallelism in V2 outweighs the benefit of enforcing shorter local paths in V1.

Influence of configuration parameters. To analyze which configuration parameters influence runtime, we generated over 1000 additional random configurations, both percentage-based and growth-based, each with 200 modules. For each configuration, we computed several metrics: diameter, number of holes and pseudo-holes, leader eccentricity, closeness, EMD toward the target rhombus, and perimeter. Here, leader eccentricity is the largest distance from the leader to any other module, closeness is the mean eccentricity over all modules, and perimeter is the number of edges between empty and occupied cells.

The plots indicate that only leader eccentricity, closeness, and EMD correlate noticeably with runtime. Since eccentricity and closeness show nearly identical trends and the remaining parameters appear uncorrelated, we include only the plots for eccentricity and EMD, shown in the third and fourth rows of Figure 7.14, respectively.

Runtime increases roughly linearly with EMD. This is expected because larger differences between the initial and target configurations require more total module movement. In contrast, runtime grows sublinearly with leader eccentricity, approximately following a square-root curve. This is likely because distant modules rarely move directly to the target cell. Instead, they often fill critical pairs along the way, enabling closer modules to move first. As a result, the rhombus grows outward and reduces the distance that distant modules must traverse in later rounds.

In the EMD plot, V2 shows a smooth, steady linear trend with few outliers, while V1 exhibits higher variance. This suggests that locked leaves, such as those constructed manually in earlier examples, occur frequently even in random configurations, causing delays under V1 but not under V2.

7.5. Conclusion

In this chapter, we presented a distributed reconfiguration algorithm for the sliding square model that transforms any side-connected initial configuration into a rhombus centered at a given leader. The algorithm operates under strict local communication and constant-memory constraints while preserving side-connectivity throughout the execution. Its sequential version runs in $\mathcal{O}(n^2)$ rounds, which is asymptotically optimal for sequential solutions in the worst case.

Although we focused on rhombus formation, the approach is not specific to the rhombus shape. The algorithm generalizes to any target shape whose cells admit an order that can be followed using only constant memory, provided that every prefix of this order together with the neighborhood of its last cell is weakly convex. Examples include lines and filled rectangles. Thus, the request-based approach may serve as a more general tool for connectivity-preserving reconfiguration toward canonical shapes in the sliding square model.

To improve practical performance, we introduced two parallel variants, V1 and V2. Both variants combine the sequential request mechanism with parallel movement of leaves in a spanning tree that is maintained throughout the execution. Our simulations show that both variants substantially outperform the sequential algorithm and even beat the Earth Mover's Distance lower bound for sequential runtimes. Variant V2 achieves the larger speedup. In contrast to V1, its runtime curve is smooth and has low variance, suggesting that it handles configurations that tend to become locked under V1. The experiments further indicate that runtime scales roughly linearly with EMD and sublinearly, approximately following a square-root curve, with leader eccentricity and closeness.

Several directions remain open. A natural next step is to extend the approach to target shapes that are not weakly convex during construction. In this setting, the boundary of the target hole may contain no module whose movement makes progress toward the target shape. Thus, progress may require temporarily relocating modules that have already reached their intended positions.

Another direction is to study reconfiguration from the rhombus into arbitrary target configurations. This would require at least one module with enough memory to store a description of the target shape. Such a module could compute a schedule that transforms the target configuration into the rhombus, reverse this schedule, and then instruct the other modules, using only small messages, how to move in order to construct the target configuration. In this way, the rhombus could serve as a canonical intermediate shape for more general reconfiguration tasks.

Finally, it would be interesting to develop lower bounds for distributed reconfiguration in the sliding square model, especially bounds that distinguish sequential algorithms from algorithms with controlled parallel movement.

Bibliography

- [Abd+20] 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 (Media Exposition)”. In: *36th International Symposium on Computational Geometry (SoCG 2020)* 164 (2020). DOI: 10.4230/LIPIcs.SoCG.2020.73.
- [Abe+24] Z. Abel, S. D. Akitaya Hugo A. and Kominers, M. Korman, and F. Stock. “A Universal In-Place Reconfiguration Algorithm for Sliding Cube-Shaped Robots in a Quadratic Number of Moves”. In: *40th International Symposium on Computational Geometry (SoCG 2024)*. Ed. by W. Mulzer and J. M. Phillips. Vol. 293. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 1:1–1:14. DOI: 10.4230/LIPIcs.SoCG.2024.1.
- [Aki+21a] H. A. Akitaya, E. M. Arkin, M. Damian, E. D. Demaine, V. Dujmović, R. Flatland, M. Korman, B. Palop, I. Parada, A. v. Renssen, and V. Sacristán. “Universal Reconfiguration of Facet-Connected Modular Robots by Pivots: The $O(1)$ Musketeers”. In: *Algorithmica* 83.5 (May 2021), pp. 1316–1351. DOI: 10.1007/s00453-020-00784-6.
- [Aki+21b] H. A. Akitaya, E. D. Demaine, A. Gonczi, D. H. Hendrickson, A. Hesterberg, M. Korman, O. Korten, J. Lynch, I. Parada, and V. Sacristán. “Characterizing Universal Reconfigurability of Modular Pivoting Robots”. In: *37th International Symposium on Computational Geometry (SoCG 2021)*. Ed. by K. Buchin and É. Colin de Verdière. Vol. 189. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 10:1–10:20. DOI: 10.4230/LIPIcs.SoCG.2021.10.
- [Aki+22] H. A. Akitaya, E. D. Demaine, M. Korman, I. Kostitsyna, I. Parada, W. Sonke, B. Speckmann, R. Uehara, and J. Wulms. “Compacting Squares: Input-Sensitive In-Place Reconfiguration of Sliding Squares”. In: *18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*. Ed. by A. Czumaj and Q. Xin. Vol. 227. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 4:1–4:19. DOI: 10.4230/LIPIcs.SWAT.2022.4.
- [Aki+25] H. A. Akitaya, S. P. Fekete, P. Kramer, S. Molaei, C. Rieck, F. Stock, and T. Wallner. “Sliding Squares in Parallel”. In: *33rd Annual European Symposium on Algorithms (ESA 2025)*. Ed. by A. Benoit, H. Kaplan,

- S. Wild, and G. Herman. Vol. 351. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 28:1–28:17. DOI: 10.4230/LIPIcs.ESA.2025.28.
- [Aki+26] H. A. Akitaya, J. Dorfer, P. Kramer, C. Rieck, G. Shahrouzi, and F. Stock. “Sliding Cubes in Parallel”. In: *arXiv preprint arXiv:2603.08537* (2026). DOI: 10.48550/arXiv.2603.08537.
- [AKS25] H. A. Akitaya, M. Korman, and F. Stock. “Input-Sensitive Reconfiguration of Sliding Cubes”. In: *Proc. of the 37th Canadian Conference on Computational Geometry (CCCG 2025)*. 2025, pp. 216–222.
- [Alo+09] G. Aloupis, S. Collette, M. Damian, E. D. Demaine, R. Flatland, S. Langerman, J. O’Rourke, S. Ramaswami, V. Sacristán, and S. Wuhler. “Linear reconfiguration of cube-style modular robots”. In: *Computational Geometry* 42.6 (2009), pp. 652–663. DOI: 10.1016/j.comgeo.2008.11.003.
- [Ang+06] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. “Computation in networks of passively mobile finite-state sensors”. In: *Distributed Computing* 18.4 (Mar. 2006), pp. 235–253. DOI: 10.1007/s00446-005-0138-3.
- [AS23] H. A. Akitaya and F. Stock. “Reconfiguration of 3D Pivoting Modular Robots”. In: (2023). DOI: 10.48550/arXiv.2304.09990.
- [Bec+25] A. T. Becker, S. P. Fekete, J. Friemel, R. Kosfeld, P. Kramer, H. Kube, C. Rieck, C. Scheffer, and A. Schmidt. “Efficient Reconfiguration of Tile Arrangements by a Single Active Robot”. In: *Proc. of the 37th Canadian Conference on Computational Geometry (CCCG 2025)*. 2025, pp. 322–330.
- [BK78] M. Blum and D. Kozen. “On the power of the compass (or, why mazes are easier to search than graphs)”. In: *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*. 1978, pp. 132–142. DOI: 10.1109/SFCS.1978.30.
- [BR03] Z. Butler and D. Rus. “Distributed planning and control for modular robots with unit-compressible modules”. In: *The International Journal of Robotics Research* 22.9 (2003), pp. 699–715. DOI: 10.1177/02783649030229002.
- [CDP21] J. Czyzowicz, D. Dereniowski, and A. Pelc. “Building a Nest by an Automaton”. In: *Algorithmica* 83.1 (Jan. 2021), pp. 144–176. DOI: 10.1007/s00453-020-00752-0.
- [CG20] A. R. Cheraghi and K. Graffi. “A Leader Based Coating Algorithm for Simple and Cave Shaped Objects with Robot Swarms”. In: *2020 5th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*. 2020, pp. 43–51. DOI: 10.1109/ACIRS49895.2020.9162610.

- [Che+14] 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: *DNA Computing and Molecular Programming*. Ed. by S. Murata and S. Kobayashi. Cham: Springer International Publishing, 2014, pp. 20–36. DOI: 10.1007/978-3-319-11295-4_2.
- [Che+23] H. Chen, C. Li, M. Mafarja, A. A. Heidari, Y. Chen, and Z. Cai. “Slime mould algorithm: a comprehensive review of recent variants and applications”. In: *International Journal of Systems Science* 54.1 (2023), pp. 204–235. DOI: 10.1080/00207721.2022.2153635.
- [Chi94] G. S. Chirikjian. “Kinematics of a metamorphic robotic system”. In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. 1994, 449–455 vol.1. DOI: 10.1109/ROBOT.1994.351256.
- [CMS24] M. Connor, O. Michail, and G. Skretas. “All for One and One for All: An O(1)-Musketeers Universal Transformation for Rotating Robots”. In: *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*. Ed. by A. Casteigts and F. Kuhn. Vol. 292. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 9:1–9:20. DOI: 10.4230/LIPIcs.SAND.2024.9.
- [CWG20] A. R. Cheraghi, G. Wunderlich, and K. Graffi. “General Coating of Arbitrary Objects Using Robot Swarms”. In: *2020 5th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*. 2020, pp. 59–67. DOI: 10.1109/ACIRS49895.2020.9162617.
- [DAn+20] G. D’Angelo, M. D’Emidio, S. Das, A. Navarra, and G. Prencipe. “Asynchronous Silent Programmable Matter Achieves Leader Election and Compaction”. In: *IEEE Access* 8 (Nov. 2020), pp. 207619–207634. DOI: 10.1109/ACCESS.2020.3038174.
- [Day+17] 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 - 13th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2017, Revised Selected Papers*. Ed. by A. Fernández Anta, T. Jurdzinski, M. A. Mosteiro, and Y. Zhang. Springer International Publishing, 2017, pp. 127–140. DOI: 10.1007/978-3-319-72751-6_10.
- [Day+18] J. J. Daymude, Z. Derakhshandeh, R. Gmyr, A. Porter, A. W. Richa, C. Scheideler, and T. Strothmann. “On the runtime of universal coating for programmable matter”. In: *Natural Computing* 17.1 (Mar. 2018), pp. 81–96. DOI: 10.1007/s11047-017-9658-6.
- [Day+20] J. J. Daymude, R. Gmyr, K. Hinnenthal, I. Kostitsyna, C. Scheideler, and A. W. Richa. “Convex Hull Formation for Programmable Matter”. In: *Proceedings of the 21st International Conference on Distributed Computing and Networking*. ICDCN ’20. Kolkata, India: Association for Computing Machinery, 2020. DOI: 10.1145/3369740.3372916.

- [Der+14] Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. “Amoebot - a new model for programmable matter”. In: *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '14. Prague, Czech Republic: Association for Computing Machinery, 2014, pp. 220–222. DOI: 10.1145/2612669.2612712.
- [Der+15a] 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*. NANOCOM' 15. Boston, MA, USA: Association for Computing Machinery, 2015. DOI: 10.1145/2800795.2800829.
- [Der+15b] Z. Derakhshandeh, R. Gmyr, T. Strothmann, R. Bazzi, A. Richa, and C. Scheideler. “Brief Announcement: On the Feasibility of Leader Election and Shape Formation with Self-Organizing Programmable Matter”. In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*. PODC '15. Donostia-San Sebastián, Spain: Association for Computing Machinery, 2015, pp. 67–69. DOI: 10.1145/2767386.2767451.
- [Der+16] 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*. SPAA '16. Pacific Grove, California, USA: Association for Computing Machinery, 2016, pp. 289–299. DOI: 10.1145/2935764.2935784.
- [Der+17] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. “Universal coating for programmable matter”. In: *Theoretical Computer Science* 671 (2017), pp. 56–68. DOI: 10.1016/j.tcs.2016.02.039.
- [DiL+18] G. A. Di Luna, P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta. “Line Recovery by Programmable Particles”. In: *Proceedings of the 19th International Conference on Distributed Computing and Networking*. ICDCN '18. Varanasi, India: Association for Computing Machinery, 2018. DOI: 10.1145/3154273.3154309.
- [DiL+20a] G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. “Shape formation by programmable particles”. In: *Distrib. Comput.* 33.1 (Feb. 2020), pp. 69–101. DOI: 10.1007/s00446-019-00350-6.
- [DiL+20b] G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. “Mobile RAM and Shape Formation by Programmable Particles”. In: *Euro-Par 2020: Parallel Processing*. Ed. by M. Malawski and K. Rządca. Cham: Springer International Publishing, 2020, pp. 343–358. DOI: 10.1007/978-3-030-57675-2_22.

- [DP04] A. Dumitrescu and J. Pach. “Pushing squares around”. In: *Proceedings of the Twentieth Annual Symposium on Computational Geometry*. SCG ’04. Brooklyn, New York, USA: Association for Computing Machinery, 2004, pp. 116–123. DOI: 10.1145/997817.997838.
- [DSY04] A. Dumitrescu, I. Suzuki, and M. Yamashita. “Motion planning for metamorphic systems: feasibility, decidability, and distributed reconfiguration”. In: *IEEE Transactions on Robotics and Automation* 20.3 (2004), pp. 409–418. DOI: 10.1109/TRA.2004.824936.
- [FBR03] R. Fitch, Z. Butler, and D. Rus. “Reconfiguration planning for heterogeneous self-reconfiguring robots”. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 3. 2003, 2460–2467 vol.3. DOI: 10.1109/IROS.2003.1249239.
- [Fek+21a] 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: *Algorithmica* 83.1 (Jan. 2021), pp. 387–412. DOI: 10.1007/s00453-020-00761-z.
- [Fek+21b] S. P. Fekete, P. Keldenich, R. Kosfeld, C. Rieck, and C. Scheffer. “Connected Coordinated Motion Planning with Bounded Stretch”. In: *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*. Ed. by H.-K. Ahn and K. Sadakane. Vol. 212. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 9:1–9:16. DOI: 10.4230/LIPIcs.ISAAC.2021.9.
- [Fek+22] S. P. Fekete, E. Niehs, C. Scheffer, and A. Schmidt. “Connected Reconfiguration of Lattice-Based Cellular Structures by Finite-Memory Robots”. In: *Algorithmica* 84.10 (Oct. 2022), pp. 2954–2986. DOI: 10.1007/s00453-022-00995-z.
- [Fel+22] M. Feldmann, A. Padalkin, C. Scheideler, and S. Dolev. “Coordinating Amoebots via Reconfigurable Circuits”. In: *Journal of Computational Biology* 29.4 (2022), pp. 317–343. DOI: 10.1089/cmb.2021.0363.
- [FLS26] J. Friemel, D. Liedtke, and C. Scheffer. “Tile Reconfiguration by a Finite Automaton”. In: *WALCOM: Algorithms and Computation*. Ed. by E. Di Giacomo and D. Mondal. Singapore: Springer Nature Singapore, 2026, pp. 512–526. DOI: 10.1007/978-981-95-7127-7_34.
- [For+23] O. Formoso, G. Trinh, D. Catanoso, I.-W. Park, C. Gregg, and K. Cheung. “MMIC-I: A Robotic Platform for Assembly Integration and Internal Locomotion through Mechanical Meta-Material Structures”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 7303–7309. DOI: 10.1109/ICRA48891.2023.10161263.
- [Fra+05] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. “Graph exploration by a finite automaton”. In: *Theoretical Computer Science* 345.2 (2005). Mathematical Foundations of Computer Science 2004, pp. 331–344. DOI: 10.1016/j.tcs.2005.07.014.

- [FS21] D. Feshbach and C. Sung. “Reconfiguring Non-Convex Holes in Pivoting Modular Cube Robots”. In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 6701–6708. DOI: 10.1109/LRA.2021.3095030.
- [Gar+22] J. Garcia, M. Yannuzzi, P. Kramer, C. Rieck, and A. T. Becker. “Connected Reconfiguration of Polyominoes Amid Obstacles using RRT”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 6554–6560. DOI: 10.1109/IROS47612.2022.9981184.
- [Gar+24] J. Garcia, M. Yannuzzi, P. Kramer, C. Rieck, S. P. Fekete, and A. T. Becker. “Reconfiguration of a 2D Structure Using Spatio-Temporal Planning and Load Transferring”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 8735–8741. DOI: 10.1109/ICRA57147.2024.10611057.
- [Gar+25] J. Garcia, J. Friemel, R. Kosfeld, M. Yannuzzi, P. Kramer, C. Rieck, C. Scheffer, A. Schmidt, H. Kube, D. Biediger, S. P. Fekete, and A. T. Becker. “Moving Matter: Using a Single, Simple Robot to Reconfigure a Connected Set of Building Blocks”. In: *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*. 2025, pp. 2260–2266. DOI: 10.1109/CASE58245.2025.11163739.
- [Gmy+18] R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, and C. Scheideler. “Shape Recognition by a Finite Automaton Robot”. In: *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Ed. by I. Potapov, P. Spirakis, and J. Worrell. Vol. 117. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018, 52:1–52:15. DOI: 10.4230/LIPIcs.MFCS.2018.52.
- [Gmy+20] R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler, and T. Strothmann. “Forming tile shapes with simple robots”. In: *Natural Computing* 19.2 (June 2020), pp. 375–390. DOI: 10.1007/s11047-019-09774-2.
- [Gre+24] C. Gregg, D. Catanoso, O. Formoso, I. Kostitsyna, M. Ochalek, T. Olatunde, I. Park, F. Sebastianelli, E. Taylor, G. Trinh, and K. Cheung. “Ultralight, strong, and self-reprogrammable mechanical metamaterials”. In: *Science robotics* 9 (Jan. 2024), eadi2746. DOI: 10.1126/scirobotics.adi2746.
- [HL19] A. Heuer-Jungemann and T. Liedl. “From DNA Tiles to Functional DNA Materials”. In: *Trends in Chemistry* 1.9 (Dec. 2019), pp. 799–814. DOI: 10.1016/j.trechm.2019.07.006.
- [HLS24] K. Hinnenthal, D. Liedtke, and C. Scheideler. “Efficient Shape Formation by 3D Hybrid Programmable Matter: An Algorithm for Low Diameter Intermediate Structures”. In: *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*. Ed. by A. Casteigts and F. Kuhn. Vol. 292. Leibniz International Proceedings in Informatics (LIPIcs). 2024, 15:1–15:20. DOI: 10.4230/LIPIcs.SAND.2024.15.

- [HLS25] K. Hinnenthal, D. Liedtke, and C. Scheideler. “Efficient shape formation by 3D hybrid programmable matter: An algorithm for low diameter intermediate structures”. In: *Theoretical Computer Science* 1057 (2025), p. 115552. DOI: 10.1016/j.tcs.2025.115552.
- [Hof81] F. Hoffmann. “One pebble does not suffice to search plane labyrinths”. In: *Fundamentals of Computation Theory*. Ed. by F. Gécseg. FCT ’81. Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, pp. 433–444. DOI: 10.1007/3-540-10854-8_47.
- [HRS20] K. Hinnenthal, D. Rudolph, and C. Scheideler. “Shape Formation in a Three-dimensional Model for Hybrid Programmable Matter”. In: *Proc. of the 36th European Workshop on Computational Geometry (EuroCG 2020)*. 2020.
- [Hur+15] F. Hurtado, E. Molina, S. Ramaswami, and V. Sacristán. “Distributed reconfiguration of 2D lattice-based modular robotic systems”. In: *Auton. Robots* 38.4 (Apr. 2015), pp. 383–413. DOI: 10.1007/s10514-015-9421-8.
- [JC17a] B. Jenett and K. Cheung. “BILL-E: Robotic Platform for Locomotion and Manipulation of Lightweight Space Structures”. In: Jan. 2017. DOI: 10.2514/6.2017-1876.
- [JC17b] B. Jenett and D. Cellucci. “A mobile robot for locomotion through a 3D periodic lattice environment”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 5474–5479. DOI: 10.1109/ICRA.2017.7989644.
- [Jen+19] B. Jenett, A. Abdel-Rahman, K. Cheung, and N. Gershenfeld. “Material-Robot System for Assembly of Discrete Cellular Structures”. In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 4019–4026. DOI: 10.1109/LRA.2019.2930486.
- [Kaw15] H. Kawano. “Complete reconfiguration algorithm for sliding cube-shaped modular robots with only sliding motion primitive”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 3276–3283. DOI: 10.1109/IROS.2015.7353832.
- [Kaw20] H. Kawano. “Distributed tunneling reconfiguration of cubic modular robots without meta-module’s disassembling in severe space requirement”. In: *Robotics and Autonomous Systems* 124 (2020), p. 103369. DOI: 10.1016/j.robot.2019.103369.
- [Kaw23] H. Kawano. “Linear-Time Reconfiguration of Sliding-Only Cubic Modular Robots Under Severe Space Requirements”. In: *IEEE Robotics and Automation Letters* 8.12 (2023), pp. 8176–8183. DOI: 10.1109/LRA.2023.3315208.
- [KLS24] I. Kostitsyna, D. Liedtke, and C. Scheideler. “Universal Coating by 3D Hybrid Programmable Matter”. In: *Structural Information and Communication Complexity*. Ed. by Y. Emek. Cham: Springer Nature Switzerland, 2024, pp. 384–401. DOI: 10.1007/978-3-031-60603-8_21.

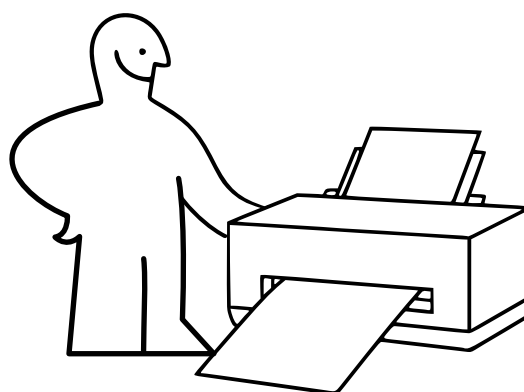
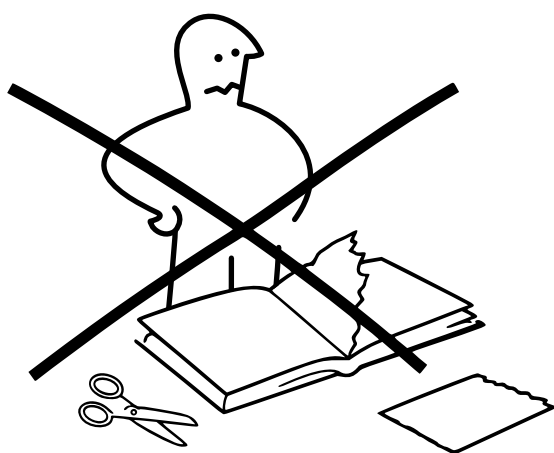
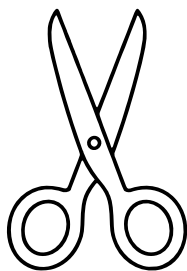
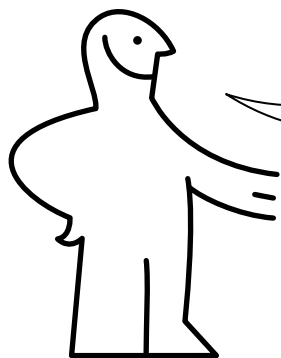
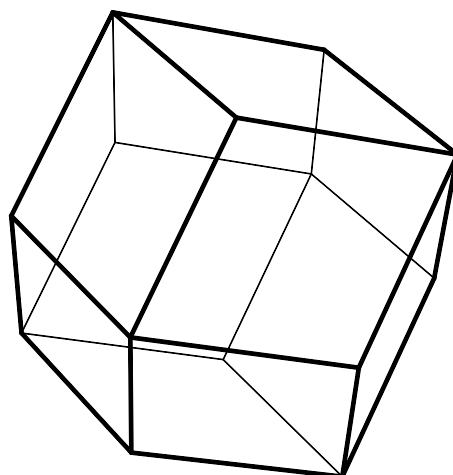
- [KLS26a] I. Kostitsyna, D. Liedtke, and C. Scheideler. “Distributed Rhombus Formation of Sliding Squares”. In: *Theoretical Computer Science* (2026). submitted (special issue).
- [KLS26b] I. Kostitsyna, D. Liedtke, and C. Scheideler. “Invited Paper: Distributed Rhombus Formation of Sliding Squares”. In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by S. Bonomi, P. S. Mandal, P. Robinson, G. Sharma, and S. Tixeuil. Cham: Springer Nature Switzerland, 2026, pp. 325–342. DOI: 10.1007/978-3-032-11127-2_26.
- [Kos+24] I. Kostitsyna, T. Ophelders, I. Parada, T. Peters, W. Sonke, and B. Speckmann. “Optimal In-Place Compaction of Sliding Cubes”. In: *19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024)*. Ed. by H. L. Bodlaender. Vol. 294. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 31:1–31:14. DOI: 10.4230/LIPIcs.SWAT.2024.31.
- [KPM21] K. Kant, D. Pattanayak, and P. S. Mandal. “Fort Formation by an Automaton”. In: *2021 International Conference on COMMunication Systems & NETworkS (COMSNETS)*. 2021, pp. 540–547. DOI: 10.1109/COMSNETS51098.2021.9352839.
- [KPS22] I. Kostitsyna, T. Peters, and B. Speckmann. “Brief Announcement: An Effective Geometric Communication Structure for Programmable Matter”. In: *36th International Symposium on Distributed Computing (DISC 2022)*. Ed. by C. Scheideler. Vol. 246. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 47:1–47:3. DOI: 10.4230/LIPIcs.DISC.2022.47.
- [KPS23] I. Kostitsyna, T. Peters, and B. Speckmann. “Fast Reconfiguration for Programmable Matter”. In: *37th International Symposium on Distributed Computing (DISC 2023)*. Ed. by R. Oshman. Vol. 281. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 27:1–27:21. DOI: 10.4230/LIPIcs.DISC.2023.27.
- [KSW22] I. Kostitsyna, C. Scheideler, and D. Warner. “Fault-Tolerant Shape Formation in the Amoebot Model”. In: *28th International Conference on DNA Computing and Molecular Programming (DNA 28)*. Ed. by T. E. Ouldridge and S. F. J. Wickham. Vol. 238. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 9:1–9:22. DOI: 10.4230/LIPIcs.DNA.28.9.
- [MS20] J. Moreno and V. Sacristán. “Reconfiguring sliding squares in-place by flooding”. In: *Proc. 36th European Workshop on Computational Geometry (EuroCG)*. 2020, p. 32.

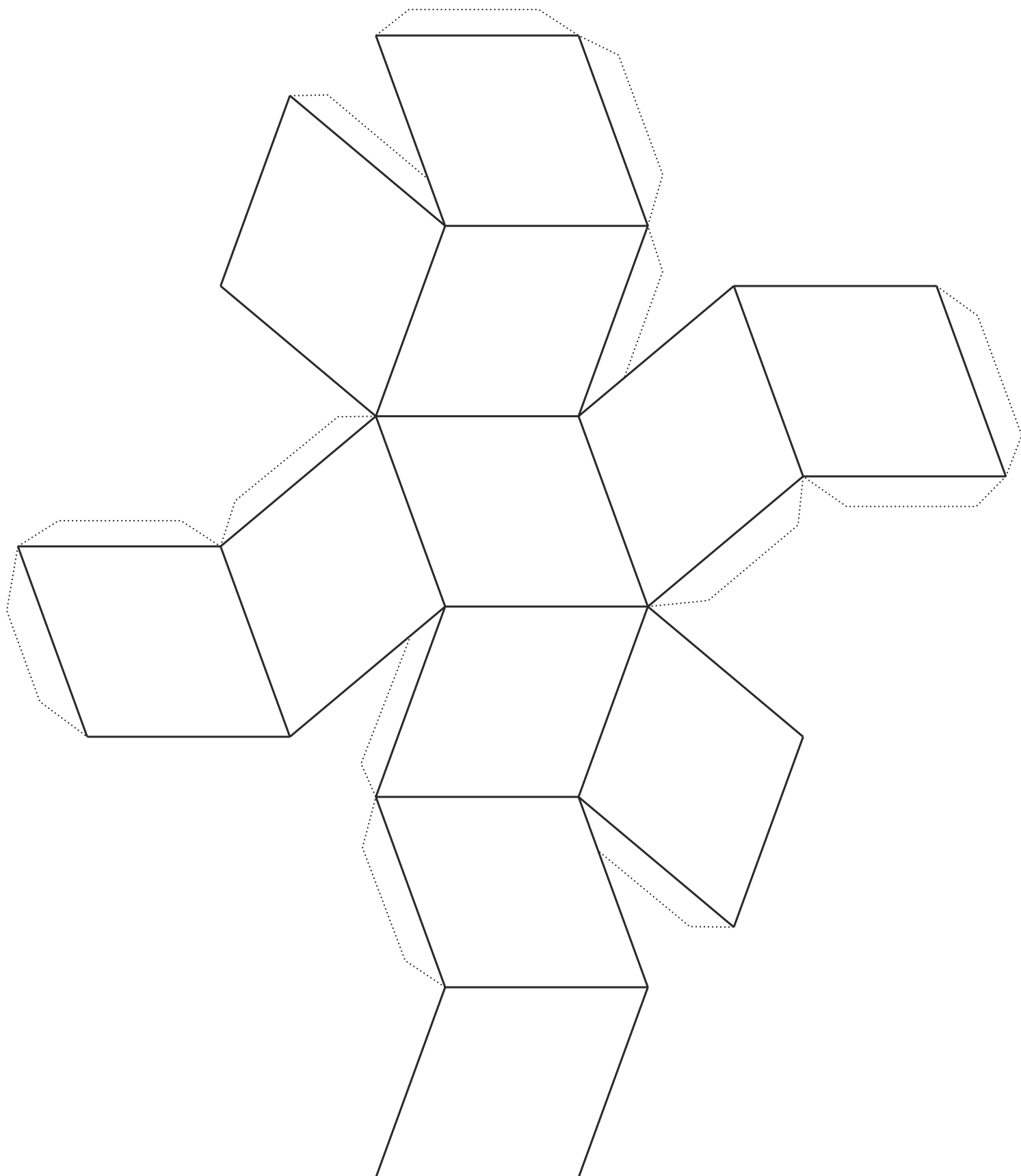
- [MSS19] O. Michail, G. Skretas, and P. G. Spirakis. “On the transformation capability of feasible mechanisms for programmable matter”. In: *Journal of Computer and System Sciences* 102 (2019), pp. 18–39. DOI: 10.1016/j.jcss.2018.12.001.
- [Naz+16] A. Naz, B. Piranda, J. Bourgeois, and S. C. Goldstein. “A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots”. In: *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*. 2016, pp. 254–263. DOI: 10.1109/NCA.2016.7778628.
- [NFS23] N. Nokhanji, P. Flocchini, and N. Santoro. “Dynamic Line Maintenance by Hybrid Programmable Matter”. In: *International Journal of Networking and Computing* 13.1 (2023), pp. 18–47. DOI: 10.15803/ijnc.13.1_18.
- [Nie+20] 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: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 8252–8259. DOI: 10.1109/ICRA40945.2020.9196700.
- [NP24] A. Navarra and F. Piselli. “Silent Programmable Matter: Coating”. In: *27th International Conference on Principles of Distributed Systems (OPODIS 2023)*. Ed. by A. Bessani, X. Défago, J. Nakamura, K. Wada, and Y. Yamauchi. Vol. 286. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 25:1–25:17. DOI: 10.4230/LIPIcs.OPODIS.2023.25.
- [NP25] A. Navarra and F. Piselli. “Coating in SILBOT with One Axis Agreement”. In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by T. Masuzawa, Y. Katayama, H. Kakugawa, J. Nakamura, and Y. Kim. Cham: Springer Nature Switzerland, 2025, pp. 177–192. DOI: 10.1007/978-3-031-74498-3_13.
- [NPP25] A. Navarra, F. Piselli, and G. Prencipe. “Line formation and scattering in silent programmable matter”. In: *Journal of Parallel and Distributed Computing* 204 (2025), p. 105129. DOI: 10.1016/j.jpdc.2025.105129.
- [NS20] N. Nokhanji and N. Santoro. “Line Reconfiguration by Programmable Particles Maintaining Connectivity”. In: *Theory and Practice of Natural Computing*. Ed. by C. Martín-Vide, M. A. Vega-Rodríguez, and M.-S. Yang. Cham: Springer International Publishing, 2020, pp. 157–169. DOI: 10.1007/978-3-030-63000-3_13.
- [OHM23] S. Odem, S. Hacoheh, and O. Medina. “An RRT That Uses MSR-Equivalence for Solving the Self-Reconfiguration Task in Lattice Modular Robots”. In: *IEEE Robotics and Automation Letters* 8.5 (2023), pp. 2922–2929. DOI: 10.1109/LRA.2023.3261705.

- [Par+23] I.-W. Park, D. Catanoso, O. Formoso, C. Gregg, M. Ochalek, T. Olatunde, F. Sebastianelli, P. Spino, E. Taylor, G. Trinh, and K. Cheung. “SOLL-E: A Module Transport and Placement Robot for Autonomous Assembly of Discrete Lattice Structures”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2023, pp. 10736–10741. DOI: 10.1109/IROS55552.2023.10341479.
- [Pat14] M. J. Patitz. “An introduction to tile-based self-assembly and a survey of recent results”. In: *Natural Computing* 13.2 (June 2014), pp. 195–224. DOI: 10.1007/s11047-013-9379-4.
- [PKS25] A. Padalkin, M. Kumar, and C. Scheideler. “Reconfiguration and locomotion with joint movements in the amoebot model”. In: *Autonomous Robots* 49.3 (Aug. 2025), p. 22. DOI: 10.1007/s10514-025-10204-9.
- [PSS21] I. Parada, V. Sacristán, and R. I. Silveira. “A new meta-module design for efficient reconfiguration of modular robots”. In: *Autonomous Robots* 45.4 (May 2021), pp. 457–472. DOI: 10.1007/s10514-021-09977-6.
- [RV01] D. Rus and M. Vona. “Crystalline Robots: Self-Reconfiguration with Compressible Unit Modules”. In: *Autonomous Robots* 10.1 (Jan. 2001), pp. 107–124. DOI: 10.1023/A:1026504804984.
- [Sun+15] C. Sung, J. Bern, J. Romanishin, and D. Rus. “Reconfiguration planning for pivoting cube modular robots”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 1933–1940. DOI: 10.1109/ICRA.2015.7139451.
- [Tha+19] P. Thalamy, B. Piranda, F. Lassabe, and J. Bourgeois. “Scaffold-Based Asynchronous Distributed Self-Reconfiguration By Continuous Module Flow”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Macau, China: IEEE Press, 2019, pp. 4840–4846. DOI: 10.1109/IROS40897.2019.8967775.
- [Tha+20] P. Thalamy, B. Piranda, F. Lassabe, and J. Bourgeois. “Deterministic scaffold assembly by self-reconfiguring micro-robotic swarms”. In: *Swarm and Evolutionary Computation* 58 (2020), p. 100722. DOI: 10.1016/j.swevo.2020.100722.
- [TPB20] P. Thalamy, B. Piranda, and J. Bourgeois. “3D Coating Self-Assembly for Modular Robotic Scaffolds”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 11688–11695. DOI: 10.1109/IROS45743.2020.9341324.
- [Tra20] W. K. R. Traversat. “Universal Coating by Programmable Matter in 3D”. Master’s thesis. Eindhoven University of Technology, 2020.
- [VYS02] S. Vassilvitskii, M. Yim, and J. Suh. “A complete, local and parallel reconfiguration algorithm for cube style modular robots”. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. Vol. 1. 2002, 117–122 vol.1. DOI: 10.1109/ROBOT.2002.1013348.

- [Wol24] M. Wolters. “Parallel Algorithms for Sliding Squares”. Master’s Thesis. Utrecht University, 2024.
- [Woo+13] 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: *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*. ITCS ’13. Berkeley, California, USA: Association for Computing Machinery, 2013, pp. 353–354. DOI: 10.1145/2422436.2422476.
- [WPN14] J. Werfel, K. Petersen, and R. Nagpal. “Designing Collective Behavior in a Termite-Inspired Robot Construction Team”. In: *Science* 343.6172 (2014), pp. 754–758. DOI: 10.1126/science.1245842.
- [WTA05] J. Walter, E. Tsai, and N. Amato. “Algorithms for fast concurrent re-configuration of hexagonal metamorphic robots”. In: *IEEE Transactions on Robotics* 21.4 (2005), pp. 621–631. DOI: 10.1109/TR0.2004.842325.

RHÖMBI





B