5-2020

Algorithmic Assembly of Nanoscale Structures

Austin Luchsinger
The University of Texas Rio Grande Valley

ALGORITHMIC ASSEMBLY OF NANOSCALE STRUCTURES

A Thesis

by

AUSTIN LUCHSINGER

Submitted to the Graduate College of
The University of Texas Rio Grande Valley
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2020

Major Subject: Computer Science

ALGORITHMIC ASSEMBLY OF NANOSCALE STRUCTURES

A Thesis
by
AUSTIN LUCHSINGER

COMMITTEE MEMBERS

Dr. Robert Schweller
Chair of Committee

Dr. Bin Fu
Committee Member

Dr. Tim Wylie
Committee Member

Dr. Jingru Zhang
Committee Member

May 2020

ABSTRACT

Luchsinger, Austin, Algorithmic Assembly of Nanoscale Structures. Master of Science (MS), May, 2020, 129 pp., 9 tables, 55 figures, 27 references.

The development of nanotechnology has become one of the most significant endeavors of our time. A natural objective of this field is discovering how to engineer nanoscale structures. Limitations of current top-down techniques inspire investigation into bottom-up approaches to reach this objective. A fundamental precondition for a bottom-up approach is the ability to control the behavior of nanoscale particles. Many abstract representations have been developed to model systems of particles and to research methods for controlling their behavior. This thesis develops theories on two such approaches for building complex structures: the self-assembly of simple particles, and the use of simple robot swarms. The concepts for these two approaches are straightforward. Self-assembly is the process by which simple particles, following the rules of some behavior-governing system, naturally coalesce into a more complex form. The other method of bottom-up assembly involves controlling nanoscale particles through explicit directions and assembling them into a desired form. Regarding the self-assembly of nanoscale structures, we present two construction methods in a variant of a popular theoretical model known as the 2-Handed Tile Self-Assembly Model. The first technique achieves shape construction at only a constant scale factor, while the second result uses only a constant number of unique particle types. Regarding the use of robot swarms for construction, we first develop a novel technique for reconfiguring a swarm of globally-controlled robots into a desired shape even when the robots can only move maximally in a commanded direction. We then expand on this work by formally defining an entire hierarchy of shapes which can be built in this manner and we provide a technique for doing so.

ACKNOWLEDGMENTS

I am forever indebted to Robert Schweller and Tim Wylie for the mentorship, guidance, and support they have given me. I thank Robbie for constantly pushing me, and teaching me to think like an academic. I thank Tim for introducing me to the world of academic research, and always being there to offer sound advice in times of doubt. Together, the both of them have given me not only the skillset, but also the opportunities to succeed. I have found this combination to be an essential part of good leadership, and theirs has opened doors for me that would have likely remained shut otherwise.

I thank Bin Fu for serving my defense committee and for his fascinating Algorithms and Theory of Computation courses. I also thank Jingru Zhang for serving my defense committee and for her valuable Computational Geometry course. I wish to express my gratitude to Cameron Chalk, Eric Martinez, and all of the members of the Algorithmic Self-Assembly Research Group—past and present—for creating a friendly, inviting, and creative workspace that seeded some of the most stimulating conversatons I have been a part of.

I thank my mom and dad for always believing in me and for laying the foundation for my success. I would not be the person I am today were it not for the love and guidance I received from them and the rest of my family. I thank my wife's family for the support and love they have given me since the moment we met. My mother-in-law Adela has been especially caring, and the completion of this thesis is largely due to her support. I am not articulate enough to capture the full acknowledgment my wife Samantha deserves, so I will have to show my thanks by returning the love, encouragement, and guidance she offers me every day.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

Personal voice vs. the use of "we": The reader should be aware that several sections of this document use "we" (except when the author's personal voice is expected) to emphasize that most of the work herein was done in collaboration with others.

1.1 Engineering Nanoscale Structures Through Self-Assembling Particles

Chapters II and III provide results in the area of algorithmic self-assembly. In theoretical self-assembly models, systems of particles naturally come together by following the rules of a given model. A primary area of study within the field of self-assembly is how to design systems which efficiently self-assemble into general shapes. The efficiency of a tile self-assembly system can be measured by various metrics, the most natural of which is the number of distinct types of system monomers. The earliest model in which this was studied– the *abstract tile assembly model* (aTAM) –is defined by a collection of *tile types* which may attach to a seed assembly via *glues* on their edges, and a *temperature* parameter which dictates what glue strengths are sufficient for attachment. The work of Soloveichik and Winfree [26] showed that any shape, if scaled up sufficiently, may be self-assembled within the aTAM using an optimal number of tile types – based on the *Kolmogorov* or *descriptional* complexity of shape. This seminal result presented a concrete connection between the descriptional complexity of a shape and the efficiency of self-assembling the shape, and represents an elegant example of the potential connections between algorithmic processes and the self-assembly of matter. The only drawback with this result is the extremely large scale factor required by construction: the scale factor to build a shape S is at least linear in $|S|$, and is typically far greater in their construction.

This motivated several efforts to reduce the scale factor by taking advantage of more powerful self-assembly models. A result by Demaine, Patitz, Schweller, and Summers [15] was able to improve upon the scale factor (while maintaining optimal tile types) by considering general shape assembly within the *staged RNAse* self-assembly model which allowed tiles to be separated into different bins and mixed over distinct stages. Furthermore, an option to "delete" tiles of a certain type is allowed in their model. This, however, still didn't achieve the best possible scale factor of $O(1)$. This wouldn't be seen until the work of Schiefer and Winfree [23], where they introduced *chemical reaction network tile assembly model* (CRN-TAM) in which chemical reaction networks and abstract tile assembly systems combine and interact by allowing CRN species to activate and deactivate tiles, while tile attachments may introduce CRN species. Although the result provides a great scale factor, the CRN-TAM constitutes a substantial jump in model complexity and power. Several results followed ([14], [18], [27]) which were all able to achieve a $O(1)$ scale factor using the powerful *staged* approach introduced by [13] and used in [15]. While each of these approaches offers important algorithmic insights, the system complexity of these systems (which includes the staging algorithms) greatly exceeds the descriptional complexity of the goal shape in a typical case.

The Contributions of This Work. In Chapter II we present the design for a self-assembly system which can self-assemble an arbitrary shape at only a constant-scale factor while still using an optimal number of tile types. For this result we use a variation of the *the two handed tile assembly model* (2HAM), which allows for negative or repulsive forces (also seen in [16, 19, 20, 21, 25]). Studies of these repulsive forces are motivated by experimental implementation of self-assembly systems which exhibit this behavior [22]. At a high level, our construction takes as input the description of a shape, and uses a Turing machine construction from [25] to extract directional instructions to build a path through a scaled version of the shape. The self-assembly process then causes bits of information to "walk" along the Turing machine tape and deposit new "pixels" at the end of the path being constructed. The shape is filled in once the path is completed, thus resulting in the construction of a constant scaled shape.

In Chapter III we show how we can transfer the complexity of a tile system from its tile types

to its temperature. We give one tile set with a constant number of distinct tile types which satisfies the following: given any finite connected shape $S \subset \mathbb{Z}^2$, there exists an assignment of strengths between glues (a glue function) and a temperature τ such that the system uniquely assembles S. The system encodes the shape in its temperature parameter τ and its glue function. Then, by utilizing the inclusion of one negative-strength glue type, the system assembles a width-τ assembly. This width-τ assembly is utilized as a seed for a tile set designed by [26] which "runs" the program encoded by the seed to assemble the shape. This work is the first to show that any shape can be built with a constant number of distinct tile types (where the glues are a function of τ) at any scale without a staged model[1], i.e., it is the first to achieve this in a fully "hands-off" model which requires no experimenter intervention during the assembly process.

1.2 Engineering Nanoscale Structures Through Manually-Controlled Particles

Chapters IV and V use controllable particles (i.e. robots) towards the construction of nanoscale shapes. When considering the use of robots at the nanoscale, power and bandwidth limitations often make individual robot control infeasible. Thus, abstract models of motion planning started considering global signals that control all robots uniformly. Perhaps the simplest (first proposed in [9]) consists of movable particles (an abstract representation of robots) that exist on a 2D grid environment with "open" and "blocked" spaces. These particles are controlled by global signals which uniformly move all particles in a particular direction when given a movement command (unless movement is prevented by a blocked space). In [9], the movement commands cause particles to move one unit distance in the given direction. Motivated by further limitations, another version of motion planning with global signals was considered in [5]. The authors analyze the complexity of steering particles through an environment when movement commands require them to move maximally in a direction. This spurred further investigation into computation and complexity of relocating particles [7] (more recently [2, 3, 4, 6]).

After gaining a better understanding of the power of this model, efforts were steered toward

[1] In the staged self-assembly model, the results of [12] give a construction which can effectively use $\mathcal{O}(1)$ tile types to assemble the shape by increasing the number of bins and stages used.

the problem of engineering particular environments to reconfigure particles into desired shapes [17]. While [9] discusses the reconfiguration of robots into particular forms, [17] has an emphasis on "building shapes" with these particles. Following, [8], formally analyze and improve on the results from [17] by creating fixed-shape "micro-factories" (or *shape builders*) which are capable of constructing a shape from a particular class of shapes (later named "drop-shapes") by attaching particles to each other using maximal-movement commands. In [24], they investigated the natural next-step to improve the efficiency with which engineered environments could build their particular shapes.

The Contributions of This Work. Chapter IV introduces the concept of *universal constructors* –environments where particle a configuration can transform into another from a given universe of configurations– and presents two universal constructors under maximal movements. The first is capable of building any shape (up to a given size), but allows for a relaxed notion of shape construction where "extra" particles are allowed to exist in the environment (*weak* construction). The second considers a version of shape construction in which all particles in the environment must be considered in the final configuration (*strong* construction), and provide a universal constructor which can strongly build any "drop shape" (up to a certain size). The second set of results for this work investigates the previously mentioned complexity of controlling these particle swarms. The *occupancy* problem asks if a given location on a board can be occupied by a polyomino. The *relocation* problem asks if a specific tile is able to be relocated to a given location. The *reconfiguration* problem asks whether a given board configuration may be reconfigured into a second given configuration. We show these problems to be PSPACE-complete, even when polyominoes are restricted to be 1×1 and a single 2×2 piece that never stick to each other.

Chapter V continues the work from [4] where we formalize the notion of hierarchical construction (first started in [24]) by presenting a hierarchy of shape classes. We also present a $O(n^4 \log n)$-time algorithm for determining if a given hole-free shape is in this hierarchy. To accompany this result, we present the design for a configuration that is *strongly universal* for any shape within the hierarchy. This result constitutes the most general set of buildable shapes under

maximal-movement in the literature, with previous work consisting of just the first level of our hierarchy. The second main result of this chapter is where we resolve an open problem from [5] by showing that the occupancy problem, contrary to expectation, is PSPACE-complete even when restricting all polyominoes to be 1×1 tiles.

1.3 Additional Research

In addition to the investigations presented in this thesis, I have also been involved in other lines of work such as generalized active self-assembly models [11], information-secure computation in self-assembly [10], relocating robots with limited directions [3], and game complexity [1].

BIBLIOGRAPHY

[1] C. L. A. L. E. M. F. M. A. R. ANGEL A. CANTU, ARTURO GONZALEZ AND T. WYLIE., *Tile pattern-building games on a grid are pspace-complete (short abstract).*, in The 21st Japan Conference on Discrete and Computational Geometry, Graphs, and Games, 2018.

[2] J. BALANZA-MARTINEZ, D. CABALLERO, A. CANTU, L. GARCIA, A. LUCHSINGER, R. REYES, R. SCHWELLER, AND T. WYLIE, *Full tilt: Universal constructors for general shapes with uniform external forces*, in 30^{th} Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'19, 2019.

[3] J. BALANZA-MARTINEZ, D. CABALLERO, A. A. CANTU, T. GOMEZ, A. LUCHSINGER, R. SCHWELLER, AND T. WYLIE, *Relocation with uniform external control in limited directions*, in The 22^{nd} Japan Conference on Discrete and Computational Geometry, Graphs, and Games, JCDCGGG'19, 2019, pp. 39–40.

[4] J. BALANZA-MARTINEZ, T. GOMEZ, D. CABALLERO, A. LUCHSINGER, A. A. CANTU, R. REYES, M. FLORES, R. T. SCHWELLER, AND T. WYLIE, *Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces*, in Proc. of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA'20, 2020, pp. 2625–2641.

[5] A. T. BECKER, E. D. DEMAINE, S. P. FEKETE, G. HABIBI, AND J. MCLURKIN, *Reconfiguring massive particle swarms with limited, global control*, in Algorithms for Sensor Systems, 2014, pp. 51–66.

[6] A. T. BECKER, E. D. DEMAINE, S. P. FEKETE, J. LONSFORD, AND R. MORRIS-WRIGHT, *Particle computation: complexity, algorithms, and logic*, Natural Computing, 18 (2019), pp. 6751–6756.

[7] A. T. BECKER, E. D. DEMAINE, S. P. FEKETE, AND J. MCLURKIN, *Particle computation: Designing worlds to control robot swarms with only global signals*, in 2014 IEEE Inter. Conf. on Robotics and Automation (ICRA), 2014, pp. 6751–6756.

[8] A. T. BECKER, S. P. FEKETE, P. KELDENICH, D. KRUPKE, C. RIECK, C. SCHEFFER, AND A. SCHMIDT, *Tilt assembly: Algorithms for micro-factories that build objects with uniform external forces*, Algorithmica, (2018).

[9] A. T. BECKER, G. HABIBI, J. WERFEL, M. RUBENSTEIN, AND J. MCLURKIN, *Massive uniform manipulation: Controlling large populations of simple robots with a common input signal*, in 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nov 2013, pp. 520–527.

[10] A. A. CANTU, A. LUCHSINGER, R. T. SCHWELLER, AND T. WYLIE, *Covert computation in self-assembled circuits*, in 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece, C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, eds., vol. 132 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 31:1–31:14.

[11] C. CHALK, A. LUCHSINGER, E. MARTINEZ, R. SCHWELLER, A. WINSLOW, AND T. WYLIE, *Freezing Simulates Non-freezing Tile Automata*, Springer International Publishing, Cham, 2018, pp. 155–172.

[12] C. CHALK, E. MARTINEZ, R. SCHWELLER, L. VEGA, A. WINSLOW, AND T. WYLIE, *Optimal staged self-assembly of general shapes*, Algorithmica, (2017).

[13] E. D. DEMAINE, M. L. DEMAINE, S. P. FEKETE, M. ISHAQUE, E. RAFALIN, R. T. SCHWELLER, AND D. L. SOUVAINE, *Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues*, Natural Computing, 7 (2008), pp. 347–370.

[14] E. D. DEMAINE, S. P. FEKETE, C. SCHEFFER, AND A. SCHMIDT, *New geometric algorithms for fully connected staged self-assembly*, in DNA Computing and Molecular Programming, vol. 9211 of Lecture Notes in Computer Science, 2015, pp. 104–116.

[15] E. D. DEMAINE, M. J. PATITZ, R. T. SCHWELLER, AND S. M. SUMMERS, *Self-assembly of arbitrary shapes using RNAse enzymes: Meeting the kolmogorov bound with small scale factor*, in STACS 2011: Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science, 2011.

[16] D. DOTY, L. KARI, AND B. MASSON, *Negative interactions in irreversible self-assembly*, Algorithmica, 66 (2013), pp. 153–172.

[17] S. MANZOOR, S. SHECKMAN, J. LONSFORD, H. KIM, M. J. KIM, AND A. T. BECKER, *Parallel self-assembly of polyominoes under uniform control inputs*, IEEE Robotics and Automation Letters, 2 (2017), pp. 2040–2047.

[18] J. MAŇUCH, L. STACHO, AND C. STOLL, *Step-wise tile assembly with a constant number of tile types*, Natural Computing, 11 (2012), pp. 535–550.

[19] M. J. PATITZ, T. A. ROGERS, R. T. SCHWELLER, S. M. SUMMERS, AND A. WINSLOW, *Resiliency to multiple nucleation in temperature-1 self-assembly*, in Proceedings of the 22nd International Conference on DNA Computing and Molecular Programming (DNA), vol. 9818 of LNCS, Springer, 2016, pp. 98–113.

[20] M. J. PATITZ, R. T. SCHWELLER, AND S. M. SUMMERS, *Exact shapes and Turing universality at temperature 1 with a single negative glue*, in DNA Computing and Molecular Programming, vol. 6937 of LNCS, 2011, pp. 175–189.

[21] J. H. REIF, S. SAHU, AND P. YIN, *Complexity of graph self-assembly in accretive systems and self-destructible systems*, Theoretical Comp. Sci., 412 (2011), pp. 1592–1605.

[22] P. W. K. ROTHEMUND, *Using lateral capillary forces to compute by self-assembly*, Proceedings of the National Academy of Sciences, 97 (2000), pp. 984–989.

[23] N. SCHIEFER AND E. WINFREE, *Universal Computation and Optimal Construction in the Chemical Reaction Network-Controlled Tile Assembly Model*, Springer International Publishing, Cham, 2015, pp. 34–54.

[24] A. SCHMIDT, S. MANZOOR, L. HUANG, A. T. BECKER, AND S. FEKETE, *Efficient parallel self-assembly under uniform control inputs*, IEEE Robotics and Automation Letters, (2018), pp. 1–1.

[25] R. SCHWELLER AND M. SHERMAN, *Fuel efficient computation in passive self-assembly*, in SODA 2013: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2013, pp. 1513–1525.

[26] D. SOLOVEICHIK AND E. WINFREE, *Complexity of self-assembled shapes*, SIAM Journal on Computing, 36 (2007), pp. 1544–1569.

[27] S. M. SUMMERS, *Reducing tile complexity for the self-assembly of scaled shapes through temperature programming*, Algorithmica, 63 (2012), pp. 117–136.

CHAPTER II

SELF-ASSEMBLY OF SHAPES AT CONSTANT SCALE USING REPULSIVE FORCES

Collaborators: Robert Schweller and Tim Wylie. **My contribution:** I created the conceptual steps for the construction process as well as the tileset details (including the inequality equations). The text was written by all. **This chapter was published as:** Austin Luchsinger, Robert Schweller, and Tim Wylie, "Self-Assembly of Shapes at Constant Scale using Repulsive Forces," Natural Computing, 18(1), 93-105, 2019.

2.1 Abstract

The algorithmic self-assembly of shapes has been considered in several models of self-assembly. For the problem of *shape construction*, we consider an extended version of the Two-Handed Tile Assembly Model (2HAM), which contains positive (attractive) and negative (repulsive) interactions. As a result, portions of an assembly can become unstable and detach. In this model, we utilize fuel-efficient computation to perform Turing machine simulations for the construction of the shape. In this paper, we show how an arbitrary shape can be constructed using an asymptotically optimal number of distinct tile types (based on the shape's Kolmogorov complexity). We achieve this at $O(1)$ scale factor in this straightforward model, whereas all previous results with sublinear scale factors utilize powerful self-assembly models containing features such as staging, tile deletion, chemical reaction networks, and tile activation/deactivation. Furthermore, the computation and construction in our result only creates constant-size garbage assemblies as a byproduct of assembling the shape.

2.2 Introduction

A fundamental question within the field of self-assembly, and perhaps the most fundamental, is how to efficiently self-assemble general shapes with the smallest possible set of system monomers. This question has been considered in multiple models of self-assembly. Soloveichek and Winfree [15] first showed that any shape S, if scaled up sufficiently, is self-assembled within the *abstract tile assembly model* (aTAM) using $O(\frac{K(S)}{\log K(S)})$ tile types, where $K(S)$ denotes the *Kolmogorov* or *descriptional* complexity of shape S with respect to some universal Turing machine, which matches the lower bound for this problem. This seminal result presented a concrete connection between the descriptional complexity of a shape and the efficiency of self-assembling the shape, and represents an elegant example of the potential connections between algorithmic processes and the self-assembly of matter. The only drawback with this result is the extremely large scale factor required by construction: the scale factor to build a shape S is at least linear in $|S|$, and is typically far greater in their construction. To lay claim as a true universal shape building scheme for potential experimental application, a much smaller scale factor is needed. Unfortunately, example shapes exist (long thin rectangles for example) which prove that the aTAM cannot build all shapes at $o(|S|)$ scale in the minimum possible $O(\frac{K(S)}{\log K(S)})$ tile complexity. This motivates the quest for small scale factors in more powerful self-assembly models.

The next result by Demaine, Patitz, Schweller, and Summers [5] considers general shape assembly within the *staged RNAse* self-assembly model. In this model, system tiles are separated into separate bins and mixed over distinct stages of the algorithm in a way that models realistic laboratory operations. In addition, each tile type in this model is of type DNA or RNA, and the staging permits the addition of an RNAse enzyme at any step in the staging, thereby dissolving all tiles of type RNA, leaving DNA tiles untouched. By adding the powerful operations of separate bins, sequential stages, and tile deletion, [5] achieves general shape construction within optimal $O(\frac{K(s)}{\log K(S)})$ tile complexity using only a constant number of bins and stages, and only a logarithmic scale factor. This leap in scale factor reduction constituted a great improvement, but required a very powerful model with both staging and tile dissolving. In addition, the holy grail of $O(1)$ scale factor

remained elusive.

The next entry into the quest for Kolmogorov optimal shape assembly at small scale comes from a recent work by Schiefer and Winfree [13]. Schiefer and Winfree introduce the *chemical reaction network tile assembly model* (CRN-TAM) in which chemical reaction networks and abstract tile assembly systems combine and interact by allowing CRN species to activate and deactivate tiles, while tile attachments may introduce CRN species. This powerful interaction allowed the construction of Kolmogorov optimal systems for the assembly of general shapes at $O(1)$ scale. Although the result provides a great scale factor, the CRN-TAM constitutes a substantial jump in model complexity and power.

In this paper we study the optimal shape building problem within one of the simplest, and most well studied models of self-assembly: *the two handed tile assembly model* (2HAM), where system monomers are 4-sided tiles with glue types on each edge. Assembly in the 2HAM proceeds whenever two previously assembled conglomerations of tiles, or assemblies, collide along matching glue types whose strength sums to some temperature threshold. Our only addition to the model is the allowance of negative strength (i.e., *repulsive*) glues, an admittedly powerful addition based on recent work [6, 8, 9, 10, 14], but an addition motivated by biology [12] that maintains the *passive* nature of the model as system monomers are static, state-less pieces that simply attract or repulse based solely on surface chemistry (Figure 2.1). While the negative glue 2HAM has been used for works such as fuel-efficient computation [14] and recently universal shape replication [1], it is also one of the simplest models where the general shape assembly problem has been considered. Our result is on par with the best possible result: we show that any connected shape S is self-assembled at $O(1)$-scale in the negative glue 2HAM within $O(\frac{K(S)}{\log K(S)})$ tile types, which is met by a matching lower bound.

Our Approach. We achieve our result by combining the *fuel efficient Turing machine* construction published in SODA 2013, [14], with a number of novel negative glue based gadgets. At a high level, the fuel efficient Turing machine system extracts a description of a path that walks the pixels of the constant-scaled shape from a compressed initial binary string. From there, the

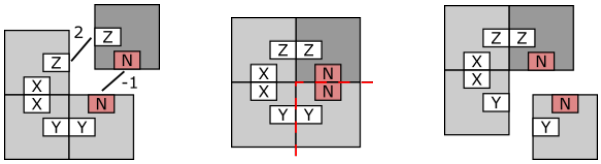
steps of the path are translated into *walker* gadgets which conceptually walk along the surface of the growing path and eventually deposit an additional pixel in the specified direction, with the aid of *path extension* gadgets. When all pixels have been placed, the path through the shape is filled, resulting in a scaled version of the original shape.

Additional Related Work. Additional work has considered assembly of $O(1)$-scaled shapes by breaking the assembly process up into a number of distinct stages. In particular, [3] introduce the *staged self-assembly* model in which intermediate tile assemblies grow in separate bins and are mixed and split over a sequence of distinct stages. This approach is applied to achieve $O(1)$-scaled shapes with $O(1)$ tiles types, but a large number of bins and stages which encode the target shape. In [4] this approach is pushed further to achieve tradeoffs in terms of bin complexity and stage complexity, while maintaining construction of a final assembly with no unbonded edges. In [7] similar constant-scale results are obtained in the *step-wise self-assembly* model in which tile sets are added in sequence to a growing seed assembly. Finally, in [16] $O(1)$-scaled shapes are assembled with $O(1)$ tile types by simply adjusting the temperature of a given system over multiple assembly stages. While each of above *staged* approaches offers important algorithmic insights, the large number of stages required by each makes the approaches infeasible for large shapes. Furthermore, the system complexity of these systems (which includes the staging algorithms) greatly exceeds the descriptional complexity of the goal shape in a typical case.

Paper layout. Our construction consists of a number of detailed gadgets for specific tasks. Presentation is thus organized incrementally to walk through a version of each gadget (with symmetry there may be multiple). Section 2.3 gives the preliminary definitions and background. In Section 2.4 we provide a high-level overview of the entire process as a guide for the rest of the paper. We then show the details of our construction with the gadgets and methods we use for constructing a straight line of the path (Section 2.5) and turning corners in the path (Section 2.6). Additional gadget variations are detailed in Section 2.7. Section 2.8 provides the analysis of our construction, with the lower bound on tile complexity for shape assembly presented in Section 2.9, and details for pushing our construction to achieve a matching upper bound in Section 2.10. Then we conclude in

Section 2.11.

2.3 Definitions and Model



(a) (b) (c)

Figure 2.1: This figure introduces notation for our constructions, as well as a simple example of negative glues. On each tile, the glue label is presented. Red (shaded) labels represent negative glues, and the relevant glue strengths for the tiles can be found in the captions. For caption brevity, for a glue type X we denote $str(X)$ simply as X (e.g., $X+Y = str(X)+str(Y)$). In this temperature $\tau = 1$ example, $X = 2$, $Y = 1$, $Z = 2$ and $N = -1$. (a) The three tile assembly on the left attaches with the single tile with strength $Z+N = 2-1 = \tau$ resulting in the 2×2 assembly shown in (b). However, this 2×2 assembly is unstable along the cut shown by the dotted line, since $Y+N = 1-1 < \tau$. Then the assembly is breakable into the assemblies shown in (c).

In this section we first define the two-handed tile self-assembly model with both negative and positive strength glue types. We also formulate the problem of designing a tile assembly system that constructs a constant-scaled shape given the optimal description of that shape.

Tiles and Assemblies. A tile is an axis-aligned unit square centered at a point in $\mathbb{Z}^2$, where each edge is labeled by a *glue* selected from a glue set Π. A *strength function* $\text{str} : \Pi \to \mathbb{N}$ denotes the *strength* of each glue. Two tiles equal up to translation have the same *type*. A *positioned shape* is any subset of $\mathbb{Z}^2$. A *positioned assembly* is a set of tiles at unique coordinates in $\mathbb{Z}^2$, and the positioned shape of a positioned assembly A is the set of those coordinates.

For a given positioned assembly Υ, define the *bond graph* G_Υ to be the weighted grid graph in which each element of Υ is a vertex and the weight of an edge between tiles is the strength of the matching coincident glues or 0.[1] A positioned assembly C is said to be τ-*stable* for positive integer τ provided the bond graph G_C has min-cut at least τ, and C is said to be *connected* if every pair of vertices in G_C has a connecting path using only positive strength edges.

For a positioned assembly A and integer vector $\vec{v} = (v_1, v_2)$, let $A_{\vec{v}}$ denote the positioned

[1]Note that only matching glues of the same type contribute a non-zero weight, whereas non-equal glues always contribute zero weight to the bond graph. Relaxing this restriction has been considered as well [2].

assembly obtained by translating each tile in A by vector $\vec{v}$. An *assembly* is a translation-free version of a positioned assembly, formally defined to be a set of all translations $A_{\vec{v}}$ of a positioned assembly A. An assembly is τ-stable if and only if its positioned elements are τ-stable. An assembly is *connected* if its positioned elements are connected. A *shape* is the set of all integer translations for some subset of $\mathbb{Z}^2$, and the shape of an assembly A is defined to be the set of the positioned shapes of all positioned assemblies in A. The *size* of either an assembly or shape X, denoted as $|X|$, refers to the number of elements of any positioned element of X.

Breakable Assemblies. An assembly is τ-*breakable* if it can be cut into two pieces along a cut whose strength sums to less than τ. Formally, an assembly C is *breakable* into assemblies A and B if A and B are connected, and the bond graph $G_{C'}$ for some assembly $C' \in C$ has a cut (A', B') for $A' \in A$ and $B' \in B$ of strength less than τ. We call A and B *pieces* of the breakable assembly C.

Combinable Assemblies. Two assemblies are τ-*combinable* provided they may attach along a border whose strength sums to at least τ. Formally, two assemblies A and B are τ-*combinable* into an assembly C provided $G_{C'}$ for any $C' \in C$ has a cut (A', B') of strength at least τ for some $A' \in A$ and $B' \in B$. We call C a *combination* of A and B.

Note that A and B may be combinable into an assembly that is not stable (and thus breakable). This is a key property that is leveraged throughout our constructions. See Figure 2.1 for an example. For a system $\Gamma = (T, \tau)$, we say $A \rightarrow_1^\Gamma B$ for assemblies A and B if either A is τ-breakable into pieces that include B, or A is τ-combinable with some producible assembly to yield B, or if $A = B$. Intuitively this means that A may grow into assembly B through one or fewer combination or break reactions. We define the relation $\rightarrow^\Gamma$ to be the transitive closure of $\rightarrow_1^\Gamma$, ie., $A \rightarrow^\Gamma B$ means that A may grow into B through a sequence of combination or break reactions.

Producibility and Unique Assembly. A *two-handed tile assembly system (2HAM system)* is an ordered pair (T, τ) where T is a set of single tile assemblies, called the *tile set*, and $\tau \in \mathbb{N}$ is the *temperature*. Assembly proceeds by repeated combination of assembly pairs, or breakage of unstable assemblies, to form new assemblies starting from the initial tile set. The *producible assemblies* are those constructed in this way. Formally:

Definition 1 (2HAM Producibility)**.** *For a given 2HAM system $\Gamma = (T, \tau)$, the set of* producible assemblies *of Γ, denoted $\mathtt{PROD}_\Gamma$, is defined recursively:*

- *(Base) $T \subseteq \mathtt{PROD}_\Gamma$*
- *(Combinations) For any $A, B \in \mathtt{PROD}_\Gamma$ such that A and B are τ-combinable into C, then $C \in \mathtt{PROD}_\Gamma$.*
- *(Breaks) For any assembly $C \in \mathtt{PROD}_\Gamma$ that is τ-breakable into A and B, then $A, B \in \mathtt{PROD}_\Gamma$.*

Definition 2 (Terminal Assemblies)**.** *A* terminal *assembly of a 2HAM system is a producible assembly that cannot break and cannot combine with any other producible assembly. Formally, an assembly $A \in \mathtt{PROD}_\Gamma$ of a 2HAM system $\Gamma = (T, \tau)$ is* terminal *provided A is τ-stable (will not break) and not τ-combinable with any producible assembly of Γ (will not combine).*

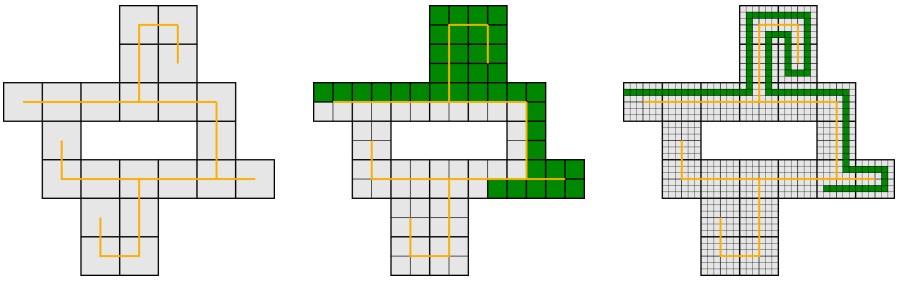
Definition 3 (Unique Assembly - with bounded garbage)**.** *A 2HAM system $\Gamma = (T, \tau)$* uniquely *produces an assembly A with garbage bound $c \in \{0\} \bigcup \mathbb{Z}^+$ provided that A is terminal, and for all $B \in \mathtt{PROD}_\Gamma$ such that $|B| > c$, $B \rightarrow^\Gamma A$.*

Definition 4 (Unique Shape Assembly - bounded garbage)**.** *A 2HAM system $\Gamma = (T, \tau)$* uniquely assembles *a finite shape S with garbage bound $c \in \{0\} \bigcup \mathbb{Z}^+$ if for every $A \in \mathtt{PROD}_\Gamma$ such that $|A| > c$, there exists a terminal $A' \in \mathtt{PROD}_\Gamma$ of shape S such that $A \rightarrow^\Gamma A'$.*

Definition 5 (Kolmogorov Complexity)**.** *The* Kolmogorov complexity *(or* descriptional complexity*) of a shape S with respect to some fixed universal Turing machine U is the smallest bit string such that U outputs a list of exactly the positions in some translation of shape S when provided the bit string as input. We denote this value as $K(S)$.*

2.4 Concept/Construction Overview

This section presents a high-level overview of the shape construction process. First, we will present the conceptual overview, which explains the fundamental ideas behind our shape self-assembly process. Then, we will show a high-level look at how our construction implements this process.

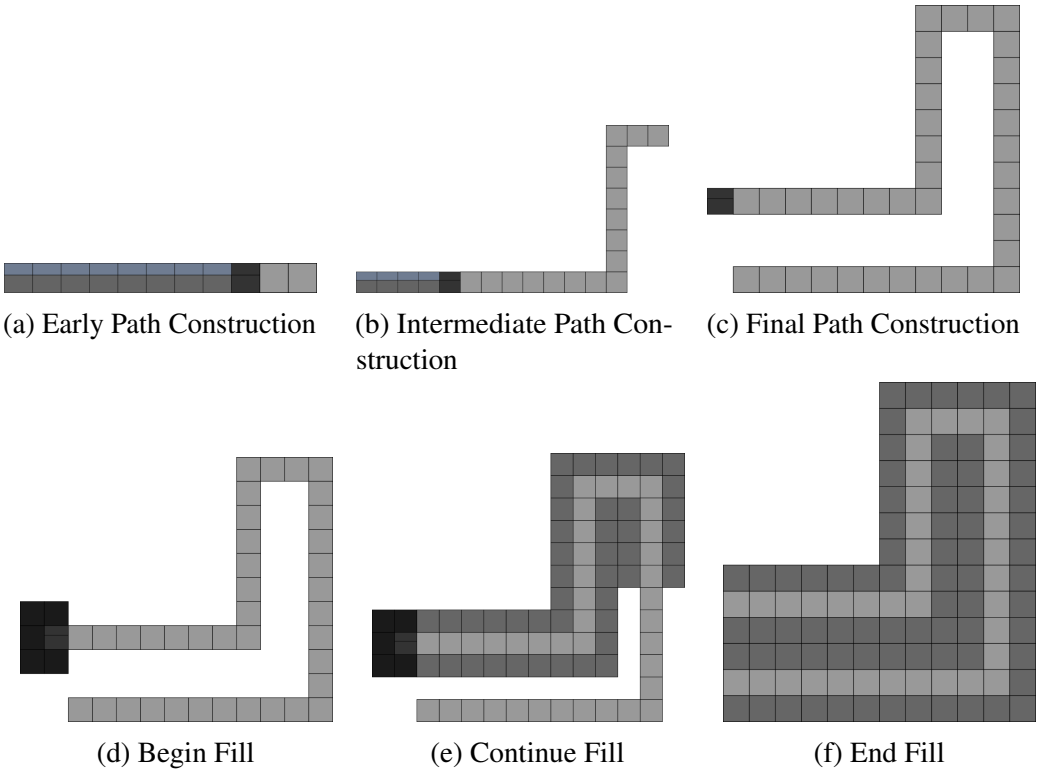


(a) Non-scaled shape X with spanning tree.

(b) Shape X at scale 2 with spanning tree.

(c) Shape X at scale 6 with spanning tree.

Figure 2.2: The Turing machine calculates a spanning tree of the tiles in the shape (a), scales the shape in order to allow a path around the spanning tree (b), and further scales the shape for the gadgets (c).



(a) Early Path Construction

(b) Intermediate Path Construction

(c) Final Path Construction

(d) Begin Fill

(e) Continue Fill

(f) End Fill

Figure 2.3: (a)-(c) The process is repeated until all information has been read/removed from the tape. (d)-(f) The final step is *Path Filling* the shape.

2.4.1 Conceptual Overview

Starting with the Kolmogorov-optimal description of a shape (as a base b string, $b > 2$), we simulate a Turing machine which converts any base b string into its equivalent base 2 representation (Sec. 2.10) We then simulate another Turing machine that takes the binary description of a shape, finds a spanning tree for that shape, and outputs a path around that spanning tree as a set of

instructions (forward, left, right) starting from a beginning node on the perimeter.

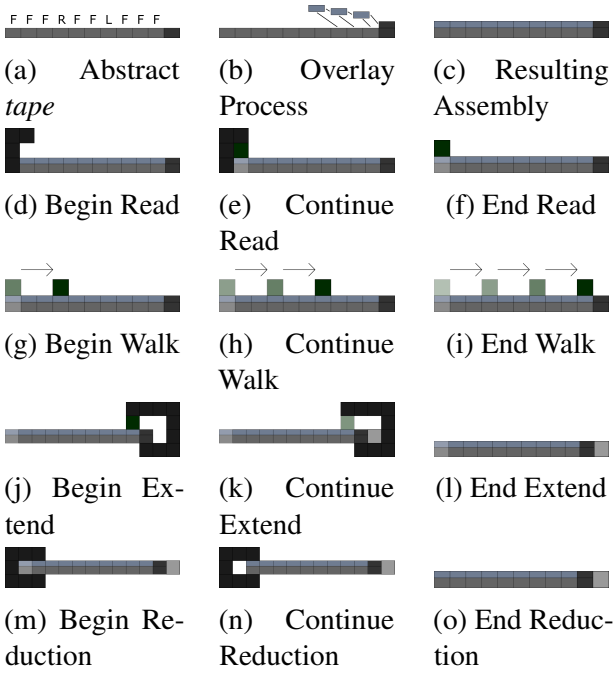
A simple depth-first search will find the spanning tree for any shape. Scaling the shape to scale 2 creates a perimeter *path* that outlines the spanning tree, and assembles the shape. Scaling again, this time by a multiple of 3, now allows space for the perimeter path with an equal-sized space buffer on both sides (Fig. 2.2). This buffer is required as it allows sufficient space for our construction gadgets to "walk" along the perimeter path being built.

Process Overview:

1. Given the Kolmogorov-optimal description of a shape, run a base conversion Turing machine to get its binary equivalent.
2. Given that binary string, run another Turing machine that outputs the description of a path around the shape's spanning tree as a set of instructions (forward, left, right).
3. Given those instructions, build the path. Our construction begins with a *tape* containing this *path* description for a scale 24 shape.

2.4.2 Construction Overview

The construction overview begins at step 3 of the conceptual overview, using the output from step 2. Throughout this paper, we will be referring to this output as the *tape*, meaning the fuel-efficient Turing machine tape with *path*-building instructions encoded on it. This *tape* is detailed in Section 2.5.



(a) Abstract *tape* (b) Overlay Process (c) Resulting Assembly

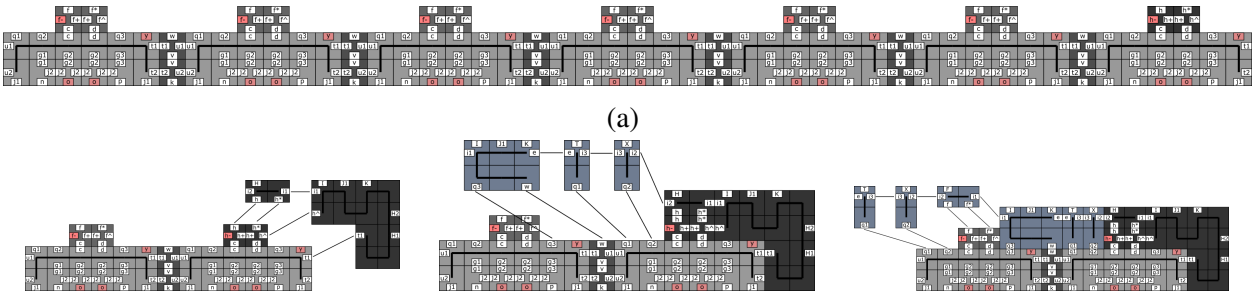
(d) Begin Read (e) Continue Read (f) End Read

(g) Begin Walk (h) Continue Walk (i) End Walk

(j) Begin Extend (k) Continue Extend (l) End Extend

(m) Begin Reduction (n) Continue Reduction (o) End Reduction

Figure 2.4: (a)-(c) The *overlay* process covers the tape while making the data readable on top. (d)-(f) *Reading* the leftmost piece of data and creating an information block (depicted in green). (g)-(i) *Information Walking* on the path to the end where the information is used. (j)-(l) When the information block reaches the end of the path, the block triggers a *Path Extension*. (m)-(o) Once the information has been read, *Tape Reduction* removes that piece of the tape.



(a)

(b) (c) (d)

Figure 2.5: (a) A completed *tape* consisting of all *forward* instructions. (b) Overlay Initiator gadget attaching to tape. (c-d) Overlay fillers begin covering all *tape* sections from right to left.

Construction Steps Overview:

1. **Overlay.** The overlay process is the first step in shape construction. Figure 2.4a-c shows an abstraction of how the output from step 2 in the concept overview gets covered during the overlay process. The overlay initiator gadget can only attach to a completed tape. This begins a series of cooperative attachments that will cover the tape. Each bit of information on

the tape is covered by its corresponding overlay piece, and thus is readable on the top of the overlay. The overlay process is finished once the entire tape is covered.

2. **Reading.** After the overlay process is complete, information can be extracted from the tape through the read process (Figs. 2.4d-f). Information can only be extracted from the covered leftmost section of the tape if it has not already been read. When a tape section is read, information is extracted from the tape and a corresponding information block is created.

3. **Information Walking.** Once the information block is created, it begins walking until it reaches the end of the tape/path (Figs. 2.4g-i). Walking gadgets allow the information to travel down the entire path.

4. **Path Extension.** When an information block cannot travel any further, the path is extended (Figs. 2.4j-l). The path can be extended forward, left, or right. The direction of the path extension is dependent on which information block is at the end of the path. After the path is extended, the information block is removed from the path.

5. **Tape Reduction.** Once information is extracted from the tape and sent down the path, one tape section is removed (Figs. 2.4m-o). Only tape sections that have been read are removed, which then allows the next section to be read. This process continues until every section of the tape is read/removed.

6. **Repeat.** Repeat the tape read, information walk, path extend, and tape reduction processes until all path instructions have been read (Figs. 2.3a-c).

7. **Path Filling.** The final tape section that gets read begins the shape fill process (Figs. 2.3d-f). In this process, the path is padded with tiles which fill it in and results in the final shape.

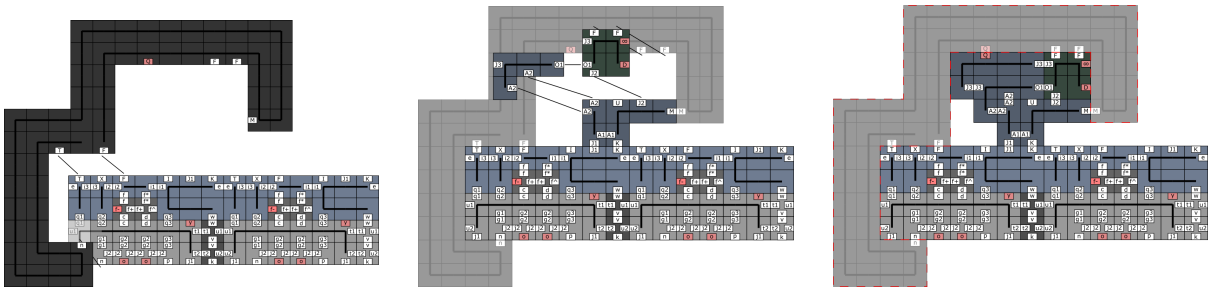
2.5 Construction Details

In this section, we detail the steps of our construction process.This is the process by which information is read from the *tape* and portions of the *path* are assembled.

We will also cover the gadgets required for each step, and review the tape construction from the fuel-efficient Turing machine used in [14]. This construction uses pre-constructed assemblies called gadgets. These gadgets are designed to work in a temperature $\tau = 10$ system. In our figures, a perpendicular black line through the middle of the edge of two adjacent tiles indicates a unique $2\tau = 20$ strength bond[2]. Each gadget provides a different function to the shape creation process.

Turing Machine Tape. A detailed look at a fuel-efficient Turing machine *tape* is seen in Figure 2.5a. Notice each tape section has a pair of tiles on top of it where the information is stored. Each pair of dark grey tiles on top of the tape sections represents a piece of information describing the *path*.

The Overlay Initiator Gadget attaches to the end of the completed *tape*, and begins the overlay process (Fig. 2.5b-d). Each bit of information on the tape is covered by a corresponding overlay section, allowing the information to be read on top of the overlay. This process continues, section by section, until the entire tape is covered. Once finished, the overlay layer will act as an interface, allowing the gadgets to use the information on the *tape*.



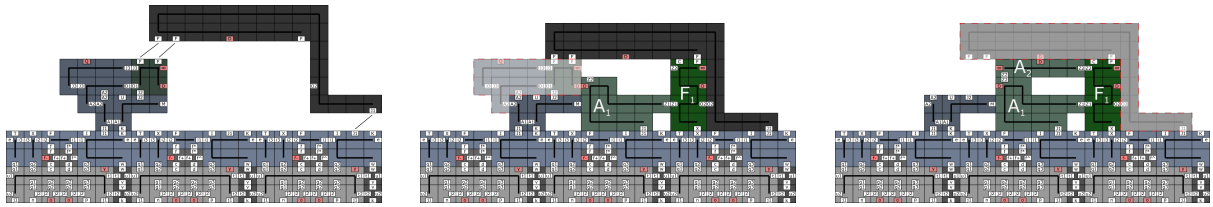
(a) (b) (c)

Figure 2.6: (a) The Read Gadget attaches ($n+T+F = 2+7+1 \geq \tau$). (b) The information block attaches ($F+F+J2 = 1+1+8 \geq \tau$), along with the read-helpers.(c) After all read helpers have attached, the read gadget becomes unstable and detaches ($F+F+M+n+T+F+Q = 1+1+1+2+7+1-7 \leq \tau$).

Read. The read gadget is required for "reading" the Turing machine *tape*. Essentially, this gadget extracts the information that is relayed from the *tape* through the overlay blocks. The read process (Fig. 2.6a-c) can only begin if the leftmost tape section has not previously been read. Once attached,

[2]The strongest detaching force used in our construction is a τ strength detachment, and since the internal bonds of our gadgets are meant to withstand even the strongest repulsive force, it follows that those bonds must be of strength at least 2τ.

the gadget allows the attachment of an information block (corresponding to the information being read) that will be used to carry the build instructions through the rest of our construction. Once the information block is present, the remaining read-helpers can attach. The final helper destabilizes the read gadget, allowing it to fall off and expose the newly attached information block. The read gadget was designed to produce this information block, alter the *tape* section that is being read (making it unreadable), and then detach from the assembly. This design ensures that each *tape* section is only read once, and allows us to transfer the instructions to other locations in our construction via the walking gadgets. A particular cut in the read process is of note, as it may result in an infinite attachment/detachment loop (between Fig. 2.6b-c). This is merely a peculiarity which has no impact on our construction.

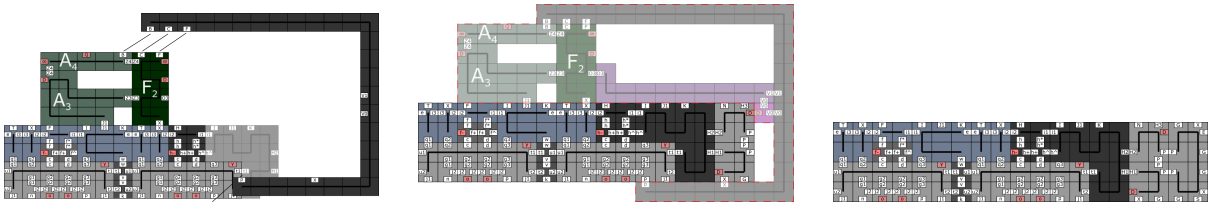


(a) (b) (c)

Figure 2.7: (a) A Walking Gadget attaches to the overlay and the information block ($F+F+J1 = 1+1+8 \geq \tau$). (b) The negative interaction between the D glues destabilizes the old information block, along with the two walking-helpers ($J2+A2+A2+F+F+D = 8+2+2+1+1-7 \leq \tau$). (c) Once the second walking-helper is attached, the walking gadget becomes unstable ($F+O2+J1+D = 1+7+8-7 \leq \tau$).

Information Walking. The walking gadgets begin the information walking process (Fig. 2.7), which allows instructions to travel throughout our construction. After a tape section has been read and an information block has been placed, a walking gadget can attach. Once attached, the walking gadget allows a new information block (of the same type) to attach, while also detaching the the previous information block. Notice that this detachment will always be $O(1)$ size. After the previous information is removed, the walking gadget detaches as well, allowing the new info block to interact with other gadgets. Thus, the same information has traveled from the *tape*, through the overlay, and is now traveling along the *tape*. This process is repeated until the information has traveled to the end of the *path*, at which point it is used to construct the next *path* portion. This method is desirable

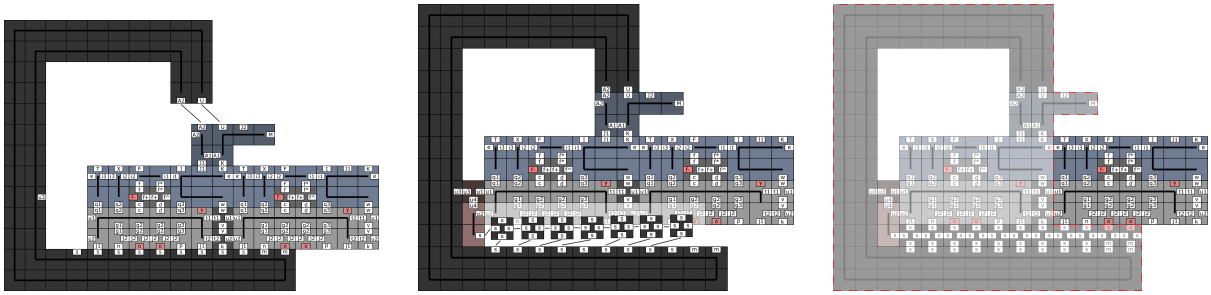
because it does not allow duplicate readable instructions to be attached to the path at any time.



(a) (b) (c)

Figure 2.8: (a) The forward-extension gadget attaches to the information block and Turing tape ($B+C+F+p=3+4+1+2\geq\tau$). (b) The second extension-helper comes with the negative D glue that causes targeted destabilization ($X+p+J1+X+D=2+2+8+2-7\leq\tau$). The extension gadget and its helpers, along with the information block and its helpers are no longer stable along their tape-overlay edges. (c) The final result is a one path-pixel extension of the path.

Path Extension. After the information block has reached the end of the *path*, a path extension gadget can attach to the assembly. Once attached, the gadget allows the *path* extension process (Fig. 2.8) to begin, which extends the *path* in a given direction (forward, left, or right) based on the instruction carried by the information block. The extension gadget "reads" the information block, and then extends the path in the given direction. Afterwards, the extension helpers destabilize the information block and extension gadget, causing a $O(1)$ sized detachment. We designed the extension gadget to essentially replace an instruction block with a corresponding *path* portion. This design allows us to attach a $O(1)$ sized *path* portion for each instruction read from the *tape*.

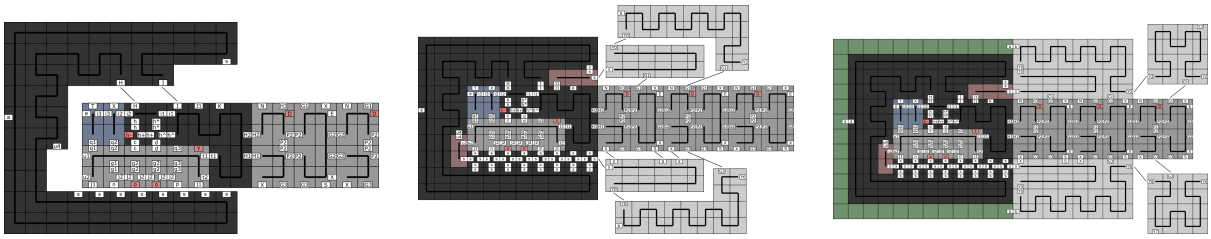


(a) (b) (c)

Figure 2.9: (a) The tape reduction gadget attaches to the read-helpers ($A2+U=2+8\geq\tau$). (b) Filler tiles attach ($s+s=8+8\geq\tau$), and create a strong bond to the tape reduction gadget. (c) The two negative o glues cause a strong targeted destabilization of the previously read tape section ($e+u1+u2+o+o=3+8+8-5-5\leq\tau$).

Tape Reduction. After a tape section has been read, we no longer need it. Instead of continuing

to grow the assembly, we can remove $O(1)$ size portions of the *tape* as it is being read. This is where the tape reduction gadget initiates the tape reduction process (Fig. 2.9).The attachments left behind by the read/walk processes allow the tape reduction gadget to attach to a tape section that has already been read. The gadget then removes itself, along with the previously read tape section, exposing the next section of the tape for reading. This technique is desirable because it allows us to break apart the *tape* into $O(1)$ sized pieces as we use it. As the *tape* is reduced, the *path* continues to grow until there are no more *tape* sections to be read.



(a) (b) (c)

Figure 2.10: (a) The fill initiator gadget attaches to the tape ($H+I=8+5\geq\tau$). (b) The first topside fillers attach ($s+G1=8+8\geq\tau, Y0+G1=9+8\geq\tau$), as well as the underside fillers ($X+S+X=2+5+2\geq\tau, Y0+G3=9+8\geq\tau$) (c) The topside and underside line filler blocks continue to attach ($Y2+G1=9+8\geq\tau, Y6+G3=9+8\geq\tau$).

Filling. After the entire *path* has been built, all previous tape sections will have been read/removed, save for one. The fill initiator gadget then attaches to the final tape section (Fig. 2.10), and begins the fill process. A series of cooperative attachments flood the sides (above and below) the path we've constructed. The initiator gadget, as well as the final tape section, remain to become the first pixel of the shape.

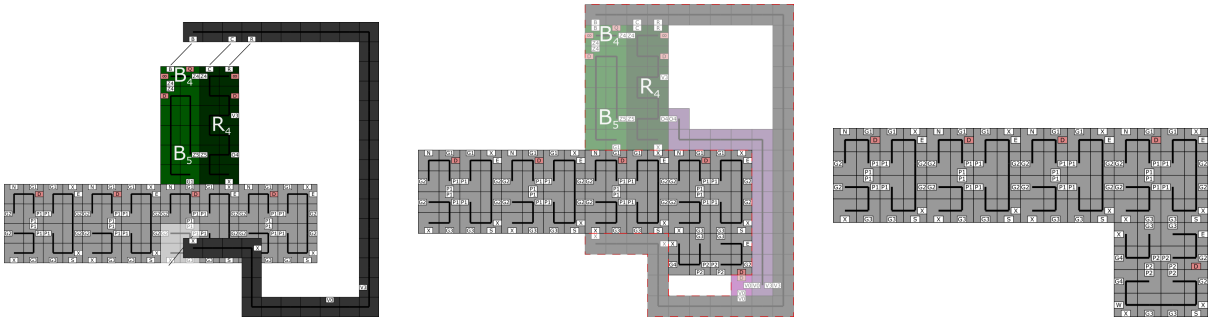
2.6 Turning Corners

The previous section detailed all of the tools needed for our construction to build straight lines. This section shows the mechanisms required for the *path* to turn left or right during its construction. The process by which the information is extracted and moves along the *path* is identical to that of Section 2.5. The key difference is how the information is used once it gets to the end of the *path*. Previously, forward extension and walking gadgets were used to extend the *path*.

Here, specific gadgets are used in order to turn the path left and right.

(a) (b) (c)

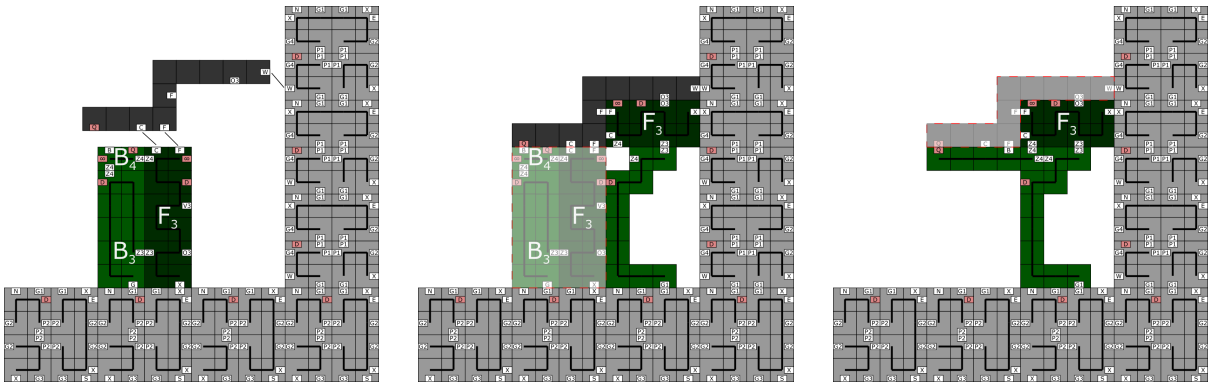
Figure 2.11: (a) The left-extension gadget attaches to the information block and shape path ($B+C+L+X=3+4+1+2\geq\tau$). (b) The negative D glue on the second extension-helper causes targeted destabilization. The extension gadget and its helpers, along with the information block and its helpers are no longer bound to the path with sufficient strength. ($X+X+G1+X+D=2+2+8+2-7\leq\tau$) (c) The final result is a one path-pixel extension of the path to the left of the direction the info block was walking.



(a) (b) (c)

Figure 2.12: (a) The right-extension gadget attaches to the information block ($B+C+R+X=3+4+1+2\geq\tau$). (d) The second extension-helper comes with a negative D glue that causes targeted destabilization. The extension gadget and its helpers, along with the information block and its helpers are no longer bound to the path with sufficient strength($X+X+G1+X+D=2+2+8+2-7\leq\tau$). (f) The final result is a one path-pixel extension of the path to the right of the direction the info block was walking.

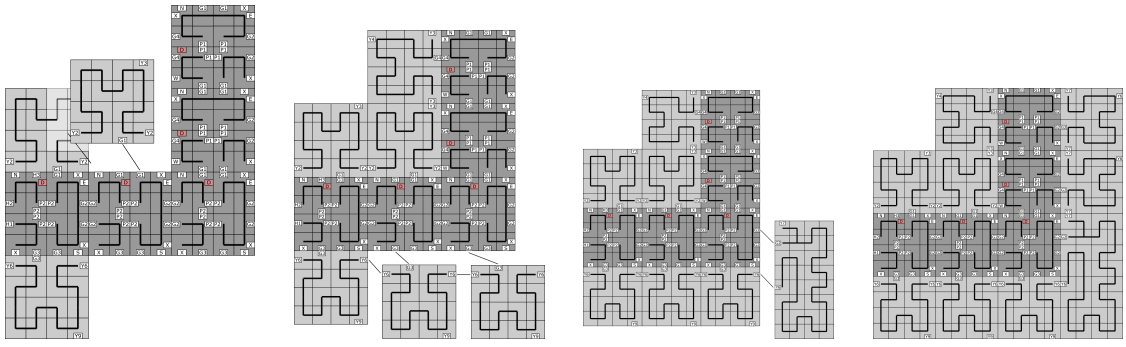
Extend Left and Right. The process for extending left and right is very similar to extending forward. Just as before, the information block must walk to the end of the path before it can be used for extension. The difference now, is the direction of the extension. The *left/right* information block allows the left/right-extension gadget to attach and extend the path using the same mechanics as forward-extension. (Figs. 2.11 and 2.12). The extension gadget reads the information block, extends the path in the given direction, and then detaches all but the newly extended path.



(a) (b) (c)

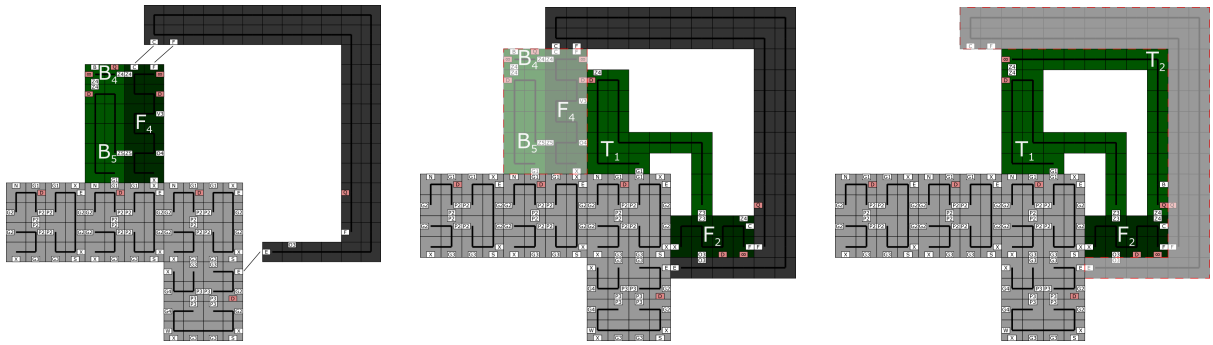
Figure 2.13: The left-walking gadget attaches to the path and the information block ($C+F+W = 4+1+5 \geq \tau$). (b) The negative interaction between the D glues destabilizes the old information block, along with the two walking-helpers($X+G1+C+F+D = 2+8+4+1-7 \leq \tau$). (c) Once the second walking-helper is attached, the walking gadget becomes unstable due to the negative Q glues ($W+O3+F+Q = 5+7+1-4 \leq \tau$).

Walk Left and Right. The walk-(left/right) procedures also utilize the same mechanics as walking forward, but with slightly different gadgets(Figs. 2.13 and 2.14). The information block only allows the correct walking gadget to attach and begin the walking process. Once attached, the walking gadget allows a new information block (of the same type) to attach, while also detaching the previous info block. After the previous information is removed, the walking gadget detaches as well, allowing the new info block to interact with other gadgets. Thus, the same information has traveled along a left or right path turn.



(a) (b) (c) (d)

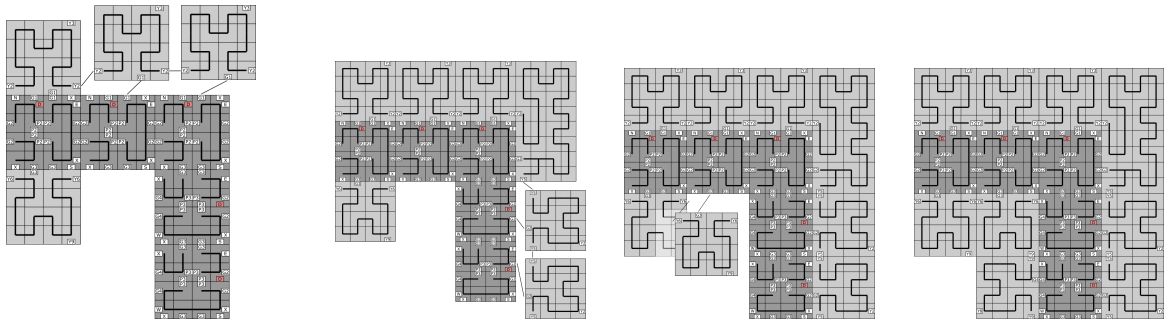
Figure 2.15: (a) The northern topside fillers attach until they encounter a left corner ($Y2+G1 = 9+8 \geq \tau$). (b) The southern underside fillers attach until they reach a corner as well ($Y6+G3 = 9+8 \geq \tau$).(c) The underside south-east filler attaches ($Y6+G2 = 9+8 \leq \tau$) (d) A completely filled left turn.



(a) (b) (c)

Figure 2.14: The right-walking gadget attaches to the path and the information block ($C+F+W = 4+1+5 \geq \tau$). (b) The negative interaction between the D glues destabilizes the old information block, along with the two walking-helpers ($X+G1+C+F+D = 2+8+4+1-7 \leq \tau$). (c) Once the second walking-helper is attached, the walking gadget becomes unstable due to the negative Q glues($F+O3+E+Q = 1+7+5-4 \leq \tau$).

Fill Left. The filler blocks continue attaching until they encounter a corner (Fig. 2.15). For left turns, the topside fillers encounter a concave corner, while the underside fillers encounter a convex corner. The design of the filler blocks allows them to simply transition from one block type to the next for concave corners. Convex corners, however, require a filler transition block to start filling in the new direction. Again, there are unique sets of filler blocks for filling along the topside and underside of the *path*.



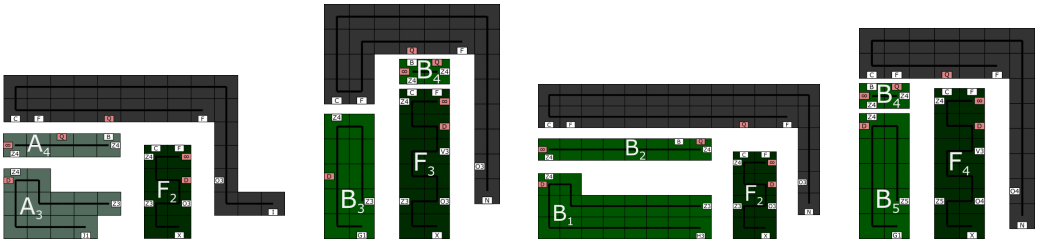
(a) (b) (c) (d)

Figure 2.16: (a) The northern topside fillers attach until they encounter a right corner ($Y2+G1 = 9+8 \geq \tau$). (c) The eastern topside fillers attach ($Y5+G2 = 9+8 \geq \tau$).(d) The south underside fillers attach until they reach a corner as well ($Y6+G2 = 9+8 \leq \tau$) (f) A completely filled right turn.

Fill Right. The right-fill process is a reflection of the left-turn process (Fig. 2.16). Here, the topside

fillers encounter the convex corner, and the underside fillers encounter the concave corner. Both filler types are designed to flood their respective sides of the *path*.

2.7 Gadget Variations

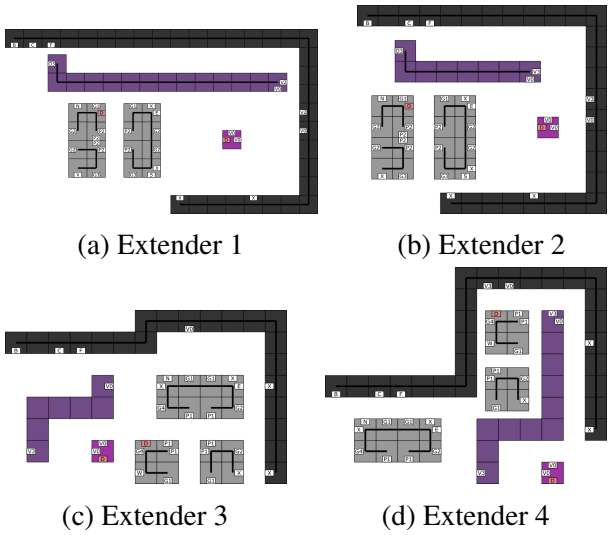


(a) Walker 1 (b) Walker 3 (c) Walker 2 (d) Walker 4

Figure 2.17: These are the walking gadget variations. Each variation also has specific versions for the different instructions that are carried by the information blocks. While slightly different, all walking gadgets utilize the same mechanics shown in Section 2.5. The changes here are due to *what* they are walking on, be it the *tape* or the *path*.

Walkers 1 & 2 The first walking gadget (Fig 2.17a) is used for all subsequent steps along the *tape*, after the standard walking gadget (Sec 2.5) has executed the information block's initial step. The second walking gadget (Fig 2.17c) allows the information block to transition from walking on the *tape*, to walking on the *path*. This gadget is required because single *tape* and *path* sections differ in size and glue types.

Walkers 3 & 4 Once an information block has transitioned from the *tape* to the *path*, the third walking gadget (Fig 2.17b) allows it to take an initial step on the *path*. The fourth walking gadget (Fig 2.17d) allows the information block to continue walking along the *path*. These gadgets are required because a single *path* section is shorter than a single *tape* section, which these gadgets account for.



(a) Extender 1 (b) Extender 2

(c) Extender 3 (d) Extender 4

Figure 2.18: As with the walking gadgets, all extension gadgets are mechanically the same as the gadget introduced in Section 2.5. While their function of these gadgets are the same (to extend the *path* by one section), the primary difference is in their geometry.

Extenders 1 & 2 Two gadgets (Figure 2.18a-b) are required to extend the path after the initial extension gadget (Sec 2.5) builds the first *path* portion. Extender 1 (Fig 2.18a) is designed to build the second *path* portion. Once the second *path* portion has been built, the extender 2 (Fig 2.18b) carries out all remaining forward extensions, with the exception of two special cases after a left turn.

Extenders 3 & 4 After a left extend (Sec 2.6, two more extender variations (Fig 2.18c-d) are required to extend the *path* to a sufficient length that allows the walking/extending gadgets to be used. The first two of these forward extensions require extenders 3 and 4. Extender 3 (Fig 2.18c) extends the *path* in the new direction after the turn. Extender 4 (Fig 2.18d) then builds an additional *path* portion in the direction of the turn.

2.8 Constant Scaled Shapes

In this section, we formally state the results based on our construction.

Theorem 1. *For any finite connected shape S, there exists a 2HAM system* $\Gamma = (T_S, 10)$ *that uniquely produces S (with a* $O(1)$ *garbage bound) at a* $O(1)$ *scale factor, and* $|T_S| = O(\frac{K(S)}{\log K(S)})$.

Proof. We show this by constructing a 2HAM system $\Gamma = (T_S, 10)$. One portion of T_S consists of the tile types which assemble a higher base Kolmogorov-optimal description of S (Section 2.10).

This portion of T_S consists of $O(\frac{K(S)}{\log K(S)})$ tile types, as analyzed in Section 2.10. Another portion of T_S consists of the tile types needed to assemble a fuel-efficient Turing machine, as described by [14], that performs a simple base conversion to binary using $O(\frac{K(S)}{\log K(S)})$ tile types, as analyzed in Section 2.10. The next portion of T_S consists of the tile types required to assemble another fuel-efficient Turing machine that finds and outputs the description of a path around the spanning tree of S. This Turing machine is of $O(1)$ size, and thus adds $O(1)$ tile types using the method from [14]. The final portion of T_S consists of the tile types that construct the gadgets and assemblies shown in Section 2.5. With the number of tile types used for computing the *path* description and for our construction process being $O(1)$, our final tile complexity is $O(\frac{K(S)}{\log K(S)})$.

Now, consider assembly A to be the fully constructed *tape* assembly (Section 2.5) encoded with *path*-building instructions specific to S. Also, suppose assembly B is some *terminal* assembly that has shape S at a constant scale factor. To assemble A, we had to start with an assembly which represents a higher base description of S. By using the fuel-efficient Turing machines of [14], we can transition this higher-base assembly into A. During this process the only assembly larger than 56 tiles is the Turing machine tape on which the computation is being performed, which inevitably becomes A.

Note that the remaining assembly process of Γ follows the details of Section 2.5. This process was designed so that two assemblies are *combinable* only if at least one of those assemblies is at most a constant size (70 tiles), and every *breakable* assembly can only break into two subassemblies if one of those assemblies is at most another constant size (118 tiles). In our construction, the only assemblies which are not bounded by this 118 tile constant are A, B, and any intermediate assembly that consists of some portion of the *tape*, and some partially assembled section of the final shape. Of these, B is the only terminal assembly.

While A and the intermediate assemblies continue engaging in a series of attachments and detachments, the *tape* continues to get smaller and the *path* continues to grow. The attachment and detachment of $O(1)$ size pieces with these assemblies will continue until we reach the terminal assembly B, at which time A will have been disassembled into chunks of constant-sized garbage.

Therefore, we see that $A \rightarrow^{\Gamma} B$. □

2.9 Lower Bound

Here we present a brief argument for the lower bound of $\Omega(\frac{K(S)}{\log K(S)})$ on the tile types needed to assemble a scaling of a shape S, under the assumption of constant bounded temperature, constant bounded glue strengths, and a fairly reasonable assumption that the system to build S does not grow infinitely many distinct producible assemblies. This argument is essentially the same as what is presented in [2, 11, 15], and we refer the reader there for a more detailed explanation.

Theorem 2. *Consider a 2HAM system $\Gamma = (T, \tau)$ such that τ is bounded by some constant, and each glue used within T has an absolute strength value bounded by a constant, and the number of producible assembles of Γ is finite. If Γ uniquely produces a scale-c version of a shape S for some constant garbage bound, then $|T| = \Omega(\frac{K(S)}{\log K(S)})$.*

Proof. Note that a 2HAM system $\Gamma = (T, \tau = O(1))$ can be uniquely represented with a string of $O(|T| \log |T|)$ bits. In particular, each tile may be encoded as a list of its 4 glues, and each glue may be represented by a $O(\log |T|)$-bit string taken from an indexing of the maximum possible $4|T|$ distinct glue types of the system, along with a constant number of bits encoding the glue strength. The constant bounded temperature incurs an additional additive constant. Given this representation, consider a simulation program that inputs a negative glue 2HAM system, and outputs the positions of any uniquely produced scale-c shape (with up to $O(1)$ garbage), if one exists. Utilizing the assumption of a finite set of producible assemblies, this simulator could achieve this by simply generating all producible assemblies in a brute force manner and halting upon verifying that all such assemblies grow into a terminal scaling of shape S. This simulator, along with the $O(|T| \log |T|)$ bit encoding of a system Γ which assembles S at scale c, constitute a program which outputs the positions of S, and is thus lower bounded in bits by $K(S)$. Therefore $K(S) \leq d|T| \log |T|$ for some constant d, implying $|T| = \Omega(\frac{K(S)}{\log K(S)})$. □

2.10 Extension to $\frac{K(S)}{\log K(S)}$

The starting assembly for our shape construction algorithm is the *tape* assembly from [14] with a binary string as its value. For a binary string $A = a_0 \dots a_{k-1}$, such an assembly can be constructed in a straightforward manner using $O(k)$ tile types (simply place a distinct tile for each position in the assembly, for example). However, by using a base conversion trick, we can take advantage of the fact that each tile type is asymptotically capable of representing slightly more than 1 bit in order to build the string in $O(k/\log k)$ tile types. To achieve this, first we consider the base-b representation $B = b_0 \dots b_{d-1}$ of the string A for some higher base $b > 2$. Note that the number of digits of this string is $d \leq \lceil \frac{k}{\lfloor \log_2 b \rfloor} \rceil = O(\frac{k}{\log b})$. We are able to assemble this shorter string (by brute force with distinct tile types at each position) with only $O(d)$ tile types.

Next, we consider a Turing machine which converts any base b string into its equivalent base 2 representation. Such a Turing machine can be constructed using $O(b)$ transition rules. Therefore, we can apply the result of [14] to run this Turing machine on the initial tape assembly representing string B to obtain string A. The cost of this construction in total is $O(d)$ tiles to construct the initial tape assembly, plus $O(b)$ tiles to implement the rules of the conversion Turing machine[3], for a total of $O(d+b)$ tiles.

Finally, we select $b = \lceil \frac{k}{\log k} \rceil = O(\frac{k}{\log k})$, which yields $d = O(\frac{k}{\log k - \log\log k}) = O(\frac{k}{\log k})$, implying that the entire tile cost of setting up the initial tape assembly representing binary string B is $O(b+d) = O(\frac{k}{\log k})$ tile types. In our case $k = O(K(S))$ where $K(S)$ denotes the Kolmogorov complexity of shape S for some given universal Turing machine, and so we achieve our final tile complexity of $O(\frac{K(S)}{\log K(S)})$.

2.11 Conclusion

In this work, we considered the optimal shape building problem in the negative glue 2-handed assembly model, and provided a system that allows the self-assembly of general shapes at scale 24. Shape construction has been studied in more powerful self-assembly models such as the

[3]The formal theorem statement of [14] cites the product of the states and symbols of the Turing machine as the tile type cost. However, the actual cost is the number of transition rules, which is upper bounded by this product.

staged RNA assembly model and the chemical reaction network-controlled tile assembly model. However, our result constitutes the first example of optimal general shape construction at constant scale in a *passive* model of self-assembly where no outside experimenter intervention is required, and system monomers are state-less, static pieces which interact solely based on the attraction and repulsion of surface chemistry.

Our work opens up a number of directions for future work. We have not considered a runtime model for this construction, so analyzing and improving the *running time* for constant-scaled shape self-assembly in the 2-handed assembly is one open direction. Another is determining the lowest necessary temperature and glue strengths needed for $O(1)$ scale shape construction. We use temperature value 10 for the sake of clarity, and have not attempted to optimize this value.

BIBLIOGRAPHY

[1] C. CHALK, E. D. DEMIANE, M. L. DEMAINE, E. MARTINEZ, R. SCHWELLER, L. VEGA, AND T. WYLIE, *Universal shape replicators via self-assembly with attractive and repulsive forces*, in Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17), 2017.

[2] Q. CHENG, G. AGGARWAL, M. H. GOLDWASSER, M.-Y. KAO, R. T. SCHWELLER, AND P. M. DE ESPANÉS, *Complexities for generalized models of self-assembly*, SIAM Journal on Computing, 34 (2005), pp. 1493–1515.

[3] E. D. DEMAINE, M. L. DEMAINE, S. P. FEKETE, M. ISHAQUE, E. RAFALIN, R. T. SCHWELLER, AND D. L. SOUVAINE, *Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues*, Natural Computing, 7 (2008), pp. 347–370.

[4] E. D. DEMAINE, S. P. FEKETE, C. SCHEFFER, AND A. SCHMIDT, *New geometric algorithms for fully connected staged self-assembly*, in DNA Computing and Molecular Programming, vol. 9211 of Lecture Notes in Computer Science, 2015, pp. 104–116.

[5] E. D. DEMAINE, M. J. PATITZ, R. T. SCHWELLER, AND S. M. SUMMERS, *Self-assembly of arbitrary shapes using RNAse enzymes: Meeting the kolmogorov bound with small scale factor*, in STACS 2011: Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science, 2011.

[6] D. DOTY, L. KARI, AND B. MASSON, *Negative interactions in irreversible self-assembly*, Algorithmica, 66 (2013), pp. 153–172.

[7] J. MAŇUCH, L. STACHO, AND C. STOLL, *Step-wise tile assembly with a constant number of tile types*, Natural Computing, 11 (2012), pp. 535–550.

[8] M. J. PATITZ, T. A. ROGERS, R. T. SCHWELLER, S. M. SUMMERS, AND A. WINSLOW, *Resiliency to multiple nucleation in temperature-1 self-assembly*, in Proceedings of the 22nd International Conference on DNA Computing and Molecular Programming (DNA), vol. 9818 of LNCS, Springer, 2016, pp. 98–113.

[9] M. J. PATITZ, R. T. SCHWELLER, AND S. M. SUMMERS, *Exact shapes and Turing universality at temperature 1 with a single negative glue*, in DNA Computing and Molecular Programming, vol. 6937 of LNCS, 2011, pp. 175–189.

[10] J. H. REIF, S. SAHU, AND P. YIN, *Complexity of graph self-assembly in accretive systems and self-destructible systems*, Theoretical Comp. Sci., 412 (2011), pp. 1592–1605.

[11] P. ROTHEMUND AND E. WINFREE, *The program-size complexity of self-assembled squares*, 2000, pp. 459–468.

[12] P. W. K. ROTHEMUND, *Using lateral capillary forces to compute by self-assembly*, Proceedings of the National Academy of Sciences, 97 (2000), pp. 984–989.

[13] N. SCHIEFER AND E. WINFREE, *Universal Computation and Optimal Construction in the Chemical Reaction Network-Controlled Tile Assembly Model*, Springer International Publishing, Cham, 2015, pp. 34–54.

[14] R. SCHWELLER AND M. SHERMAN, *Fuel efficient computation in passive self-assembly*, in SODA 2013: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2013, pp. 1513–1525.

[15] D. SOLOVEICHIK AND E. WINFREE, *Complexity of self-assembled shapes*, SIAM Journal on Computing, 36 (2007), pp. 1544–1569.

[16] S. M. SUMMERS, *Reducing tile complexity for the self-assembly of scaled shapes through temperature programming*, Algorithmica, 63 (2012), pp. 117–136.

CHAPTER III

SELF-ASSEMBLY OF ANY SHAPE WITH CONSTANT TILE TYPES USING HIGH TEMPERATURE

Collaborators: Cameron Chalk, Robert Schweller, and Tim Wylie. **My contribution:** CC developed the initial concept for this construction, while I generated the tileset details. The text was written by all. **This chapter was published as:** Cameron Chalk, Austin Luchsinger, Robert Schweller, and Tim Wylie, "Self-Assembly of Any Shape with Constant Tile Types using High Temperature," In Proceedings of the 26th Annual European Symposium on Algorithms, 2018.

3.1 Abstract

Inspired by nature and motivated by a lack of top-down tools for precise nanoscale manufacture, self-assembly is a bottom-up process where simple, unorganized components autonomously combine to form larger more complex structures. Such systems hide rich algorithmic properties—notably, Turing universality—and a self-assembly system can be seen as both the object to be manufactured as well as the machine controlling the manufacturing process. Thus, a benchmark problem in self-assembly is the unique assembly of shapes: to design a set of simple agents which, based on aggregation rules and random movement, self-assemble into a particular shape and nothing else. We use a popular model of self-assembly, the 2-handed or hierarchical tile assembly model, and allow the existence of repulsive forces, which is a well-studied variant. The technique utilizes a finely-tuned temperature (the minimum required affinity required for aggregation of separate complexes).

We show that calibrating the temperature and the strength of the aggregation between the tiles, one can encode the shape to be assembled without increasing the number of distinct tile

types. Precisely, we show one tile set for which the following holds: for any finite connected shape S, there exists a setting of binding strengths between tiles and a temperature under which the system uniquely assembles S at some scale factor. Our tile system only uses one repulsive glue type and the system is growth-only (it produces no unstable assemblies). The best previous unique shape assembly results in tile assembly models use $\mathcal{O}(\frac{K(S)}{\log K(S)})$ distinct tile types, where $K(S)$ is the Kolmogorov (descriptional) complexity of the shape S.

3.2 Introduction

Due to the limited tool set for precise fabrication at the nanoscale, the bottom-up approach of self-assembly is an attractive area of research. Such bottom-up approaches, such as *DNA origami* [18], allow for the assembly of nanoscale materials with detailed, precisely designed shapes and patterns. Abstract self-assembly models are used to predict the behavior of systems wherein simple, separate entities form larger complex structures based on a simple rule set for movement and/or attachment using only local interactions and no global leader. Such systems include swarm robotics and molecular self-assembly, particularly self-assembling nucleic acid structures such as *DNA tiles* [9].

A common benchmark in such models, which aims at the manufacture of precise nanoscale structures, is the self-assembly of *shapes*. In our case, a shape is defined simply as a finite, connected subset of $\mathbb{Z}^2$. The model studied herein is the *two-handed tile assembly model* (also called the *hierarchical tile assembly model*) [1]. In this model, the separate entities are *tiles*, adorned with *glues*. The intuition is that tiles wander about randomly, and when tiles with matching glues meet, the tiles bind to form a larger assembly; further, such larger structures wander about and may bind to other larger assemblies or tiles.

The main measure of complexity is the number of unique types of tiles necessary and sufficient to uniquely assemble the shape, termed the *tile complexity*. The *temperature* of a system, denoted by τ, is the minimum required binding strength between two entities to enforce a stable attachment; the sum of the strengths between the shared glues of two assemblies must meet or exceed the temperature. Some studies of the model include *negative-strength glues* [2, 8, 12, 13, 14, 15, 20],

which are repulsive forces which act against a particular bond between two assemblies. Studies of these repulsive forces are motivated by experimental implementation of self-assembly systems which exhibit this behavior [17].

Our contributions. We give one tile set with a constant number of distinct tile types which satisfies the following: given any finite connected shape $S \subset \mathbb{Z}^2$, there exists an assignment of strengths between glues (a glue function) and a temperature τ such that the system uniquely assembles S. The system encodes the shape in its temperature parameter τ and its glue function. Then, by utilizing the inclusion of one negative-strength glue type, the system assembles a width-τ assembly. This width-τ assembly is utilized as a seed for a tile set designed by [23] which "runs" the program encoded by the seed to assemble the shape. This work is the first to show that any shape can be built with a constant number of distinct tile types (where the glues are a function of τ) at any scale without a staged model[1], i.e., it is the first to achieve this in a fully "hands-off" model which requires no experimenter intervention during the assembly process.

Previous results. For self-assembling a shape S, we list the previously known results, which do not use negative glues and use $\mathcal{O}(1)$ temperature unless otherwise specified. Let $K(S)$ be the Kolmogorov complexity[2] of S, and let $T(S)$ be the (smallest) runtime of a Kolmogorov-optimal program outputting S. A tile complexity of $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ is known, using a scale factor of $T(S)$ [23]. With negative-strength glues, a tile complexity of $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ is known, using a $\mathcal{O}(1)$ scale factor [12]. In a *staged* version of the model, where several self-assembly systems are run in parallel across a series of *bins* and then mixed together in *stages*, a tile complexity of $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ at scale factor $T(S)$ is known for $\mathcal{O}(1)$ bins and $\mathcal{O}(1)$ stages along with a method for (optimally) reducing the number of sufficient and necessary tile types by increasing the number of bins and stages [3].

In another staged version of the model, where tiles are partitioned into `DNA` and `RNA` types, and `RNA` types may be "washed away" at a given stage, a tile complexity of $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ at scale

[1] In the staged self-assembly model, the results of [3] give a construction which can effectively use $\mathcal{O}(1)$ tile types to assemble the shape by increasing the number of bins and stages used.

[2] The Kolmogorov complexity of S is the number of bits in the smallest program which outputs S w.r.t. a universal Turing machine. For more information on Kolmogorov complexity, see [11].

factor $\mathcal{O}(\log |S|)$ is known [7]. In a staged model where the self-assembly process is controlled by a chemical reaction network which activates and deactivates tiles' binding sites, a tile plus reaction network complexity of $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ at scale factor $\mathcal{O}(1)$ is known [19].

Related work in high-temperature[3] self-assembly. The first studies of utilizing temperature to encode information involved the "online", mid-assembly-process changing of temperatures [10, 24]. Our result utilizes a high temperature bonding threshold for self-assembly attachment, which we leverage to encode precise information for guiding the self-assembly process through precisely set glue strengths. A number of recent related works have also studied the effects of higher temperature self-assembly systems within various models. Within the aTAM, larger temperatures have been shown to affect the possible behavior of systems [4], and the tile complexity of self-assembled shapes [22]. Within the 2HAM, unique-assembly verification has been shown to be hard for high-temperature systems [21], while the dynamics of certain higher-temperature systems have been shown to be impossible to simulate at lower temperatures [6].

3.3 Definitions and Model

In this section we first define the two-handed tile self-assembly model with both negative and positive strength glue types. We also formulate the problem of designing a tile assembly system that constructs a constant-scaled shape given the optimal description of that shape.

Tiles and Assemblies. A tile is an axis-aligned unit square centered at a point in $\mathbb{Z}^2$, where each edge is labeled by a *glue* selected from a glue set Π. A *strength function* $\text{str} : \Pi \to \mathbb{Z}$ denotes the *strength* of each glue. Two tiles equal up to translation have the same *type*. A *positioned shape* is any subset of $\mathbb{Z}^2$. A *positioned assembly* is a set of tiles at unique coordinates in $\mathbb{Z}^2$, and the positioned shape of a positioned assembly A is the set of those coordinates. For a given positioned assembly Υ, define the *bond graph* G_Υ to be the weighted grid graph in which each element of Υ is a vertex and the weight of an edge between tiles is the strength of the matching coincident glues

[3] We say high-temperature self-assembly for consistency with previous literature. The term high temperature may be misleading; e.g., we are not attempting to model what happens in DNA-based self-assembly systems when the literal temperature of the system is raised to high values. Intuitively, higher temperature in this model implies more fine-grained glue strengths. Another natural way to think of high temperature is to fix the temperature to one, but allow rational glue strengths.

or 0.[4] A positioned assembly C is τ-*stable* for positive integer τ provided the bond graph G_C has min-cut at least τ.

For a positioned assembly A and integer vector $\vec{v} = (v_1, v_2)$, let $A_{\vec{v}}$ denote the positioned assembly obtained by translating each tile in A by vector $\vec{v}$. An *assembly* is a translation-free version of a positioned assembly, formally defined to be a set of all translations $A_{\vec{v}}$ of a positioned assembly A. An assembly is τ-stable if and only if its positioned elements are τ-stable. A *shape* is the set of all integer translations for some subset of $\mathbb{Z}^2$, and the shape of an assembly A is defined to be the set of the positioned shapes of all positioned assemblies in A. The *size* of either an assembly or shape X, denoted as $|X|$, refers to the number of elements of any positioned element of X.

Combinable Assemblies. Two assemblies are τ-*combinable* provided they may attach along a border whose strength sums to at least τ. Formally, two assemblies A and B are τ-*combinable* into an assembly C provided $G_{C'}$ for any $C' \in C$ has a cut (A', B') of strength at least τ for some $A' \in A$ and $B' \in B$. We call C a *combination* of A and B. Figure 3.1 gives examples of combinable and not combinable assemblies.

Figure 3.1: The black lines between two tiles indicate unique unimportant τ-strength bonds. If $\tau = 2$, $str(A) = 1$ and $str(B) = 1$, then the two assemblies in (a) are τ-combinable, since $str(A) + str(B) \geq \tau$ and the positioned assemblies may be translated such that the A and B glues are aligned— such a combination is termed *cooperative binding*, since neither the A nor B glue are alone sufficient to satisfy τ-combination. In (b), we consider two cases concerning the negative strength glue X. If $\tau = 2$, $str(C) = 2$, and $str(X) = -1$, then the assemblies in (b) are not τ-combinable since $str(C) + str(X) < \tau$. If $\tau = 1$, $str(C) = 2$, $str(D) = 2$, $str(E) = 1$ and $str(X) = -1$, then the assemblies are τ-combinable since $str(C) + str(X) \geq \tau$; however, the resultant assembly is unstable, since a cut along the X and E glue has strength $str(X) + str(E) < \tau$; this violates the valid growth-only system definition.

Two Handed Assembly Model: Growth-only Version A *two-handed tile assembly system (2HAM system)* is an ordered pair (T, τ) where T is a set of single tile assemblies, called the *tile set*, and $\tau \in \mathbb{N}$ is the *temperature*. In the *growth-only* model, assembly proceeds by repeated combination of assembly pairs to form new assemblies starting from the initial tile set. The *producible assemblies*

[4]Note that only matching glues of the same type contribute a non-zero weight, whereas non-equal glues always contribute zero weight to the bond graph. Relaxing this restriction has been considered [5].

are those constructed in this way.

Definition 6 (2HAM Producibility (growth-only))**.** *For a given 2HAM system* $\Gamma = (T, \tau)$*, the set of* producible assemblies *of* Γ*, denoted* $\mathtt{PROD}_\Gamma$*, is defined recursively:*

- *(Base)* $T \subseteq \mathtt{PROD}_\Gamma$
- *(Combinations) For any* $A, B \in \mathtt{PROD}_\Gamma$ *such that* A *and* B *are* τ*-combinable into* C*, then* $C \in \mathtt{PROD}_\Gamma$.

The inclusion of negative glues, in general, allows for unstable assemblies to be producible. In previous literature, such assemblies "detach", forming two new producible assemblies. We impose the following constraint on growth-only systems which disallows production of unstable assemblies which would fall apart. Satisfying the growth-only constraint argues that the system has simpler kinetics than a non-growth-only system since the system does not rely on detachment events.

For a system $\Gamma = (T, \tau)$, we say $A \to_1^\Gamma B$ for assemblies A and B if A is τ-combinable with some producible assembly to yield B, or if $A = B$. Intuitively this means that A may grow into assembly B through one or fewer combination. We define the relation $\to^\Gamma$ to be the transitive closure of $\to_1^\Gamma$, ie., $A \to^\Gamma B$ means that A may grow into B through a sequence of combinations.

Definition 7 (Valid Growth-Only System)**.** *A 2HAM system* $\Gamma = (T, \tau)$ *is a* valid growth-only *system if for all* $A \in \mathtt{PROD}_\Gamma$*,* A *is* τ*-stable.*

Definition 8 (Terminal Assemblies)**.** *A* terminal *assembly of a valid growth-only 2HAM system is a producible assembly that cannot combine with any other producible assembly. Formally, an assembly* $A \in \mathtt{PROD}_\Gamma$ *of a 2HAM system* $\Gamma = (T, \tau)$ *is* terminal *provided* A *is not* τ*-combinable with any producible assembly of* Γ.

We formalize what it means for a 2HAM system to uniquely build a given assembly or a given shape.

Definition 9 (Unique Assembly)**.** *A 2HAM system* uniquely *produces an assembly A if all producible assemblies have a growth path towards the terminal assembly A. Formally, a 2HAM system $\Gamma = (T, \tau)$* uniquely *produces an assembly A provided that A is terminal, and for all $B \in \mathtt{PROD}_\Gamma$, $B \to^\Gamma A$.*

Definition 10 (Unique Shape Assembly[5])**.** *A 2HAM system uniquely produces a shape S if all producible assemblies have a growth path to a terminal assembly of shape S. Formally, a 2HAM system $\Gamma = (T, \tau)$* uniquely assembles *a finite shape S if for every $A \in \mathtt{PROD}_\Gamma$, there exists a terminal $A' \in \mathtt{PROD}_\Gamma$ of shape S such that $A \to^\Gamma A'$.*

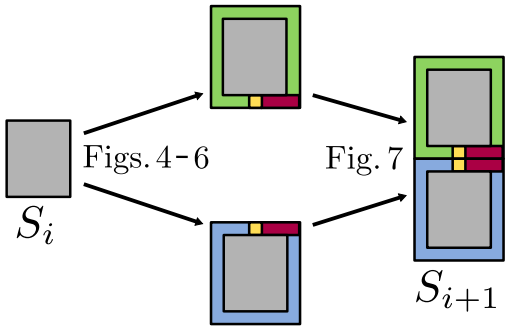


Figure 3.2: A simplified overview of the growing step. S_i is a width-$\Theta(i)$, height-$\Theta(2^i)$ assembly with particular exposed edge glues. S_i nondeterministically assembles one of two assemblies; a *top* and *bottom*. The top and bottom share one glue of strength $2\tau - 1$ shown in yellow, and i many -1 strength glues shown in red. Thus, the top and bottom bind with strength $2\tau - i - 1$, which is τ-stable only if $i < \tau$. The resultant assembly adds two width and doubles the height of S_i, so its dimensions are $\Theta(i+1) \times \Theta(2^{i+1})$. Further, its exposed glues allow the process to repeat.

3.4 Assembly of General Shapes with Constant Tiles

Here we give the main construction of the paper. First presented is our key contribution—assembly of a precise-width rectangle— detailed in Subsection 3.4.1, followed by its composition with established techniques from [23] for the main result in Subsection 3.4.2.

[5]Some previous literature calls this *strict self-assembly*, typically to contrast another definition, *weak self-assembly*; we choose the name unique shape assembly to contrast unique assembly.

3.4.1 Key idea: precise-width rectangle using $\mathscr{O}(1)$ tile types

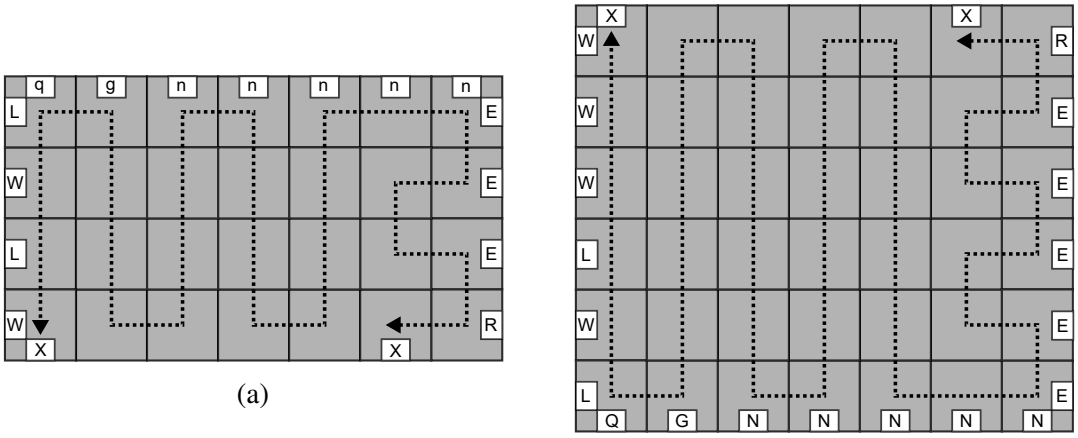


Figure 3.3: The base assembly, shown in two separate subassemblies; (a) shows the top subassembly, and (b) the bottom. The two subassemblies combine using cooperative binding at the strength-$\lceil \frac{\tau}{2} \rceil$ glues labeled X. The dotted line indicates distinct tile types which attach along the path with full τ strength glues. The snaking pattern ensures that each subassembly is complete before both X glues are available. Once these two subassemblies bind, the resultant assembly satisfies S_0.

Here, we present a construction for building a precise-width rectangle from a constant-bounded set of tile types. Note that the convention in this paper is width $\times$ height. Formally,

Lemma 1. *Given a temperature $\tau > 2$, there exists a negative glue, growth-only 2HAM tile system $\Gamma = \{T, \tau\}$ such that $|T| = \mathscr{O}(1)$ and Γ uniquely produces an assembly which is a $(18+4\tau) \times (2^{\tau+5}-6)$ rectangle.*

Proof. We give a proof by construction. Unless explicitly stated otherwise, all glues have strength $\lceil \frac{\tau}{2} \rceil$, so at least two matching glues are required for a τ-stable attachment. This is called *cooperative binding*. The construction is split into two steps: *growing* and *finishing*. Figure 3.2 shows a simplified overview of the growing step. The growing step of the construction concerns producing an assembly with width $7+2\tau$, through a process consisting of τ iterations of growth, each adding a constant-bounded width to the assembly. The $i+1^{\text{th}}$ iteration of growth is initiated by a combination of two assemblies with total binding strength $2\tau - i - 1$; thus, after the τ^{th} iteration of growth, the binding strength which would initiate the next iteration of growth has total binding strength $2\tau - \tau - 1 < \tau$, and growth halts.

The finishing step involves the system's "detecting" that the growth process has completed. This is achieved using the following technique: by adding a total strength of 1 shared between two assemblies at each iteration of growth, once the assemblies have completed τ repetitions of growth, they bind with strength τ. This step also gives the system its property of unique assembly of an assembly whose shape is a rectangle (and not just unique shape assembly). That is, there is exactly one terminal assembly of the system— as opposed to several terminal assemblies with the correct rectangular shape. Maintaining this property in this lemma is required to achieve the same property in Theorem 3.

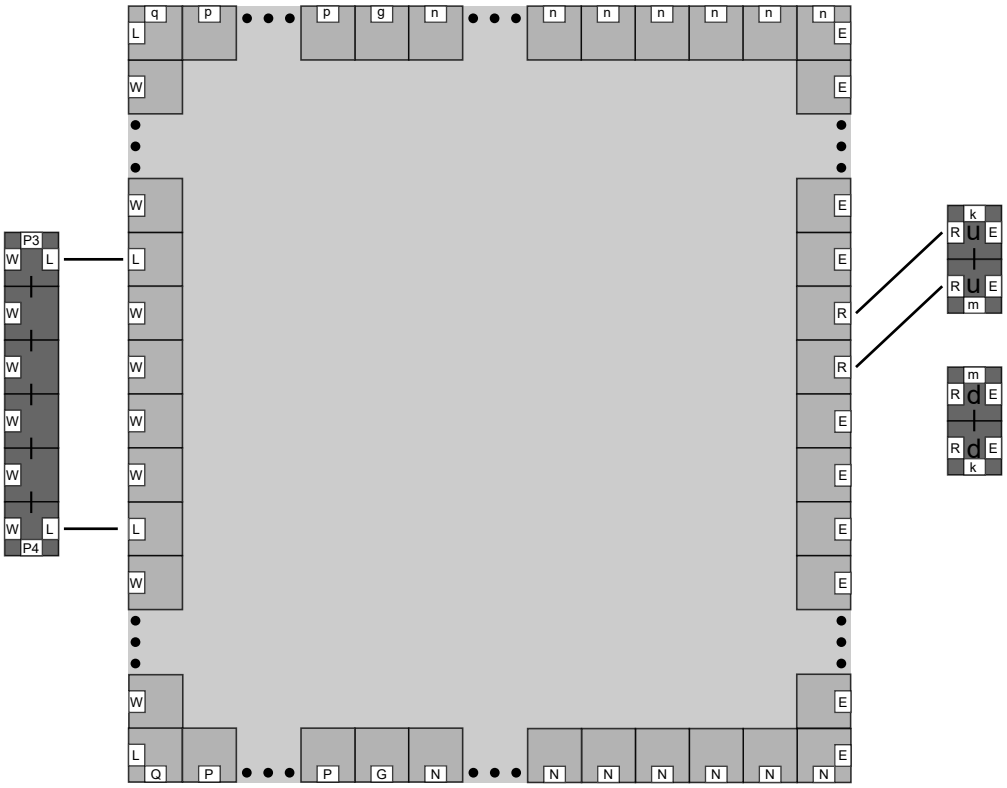


Figure 3.4: An assembly satisfying S_i. The dots indicate an omitted set of repeated tiles; e.g., the dots between the tiles exposing p indicate the omission of the i glues with label p. On the right are the keystones, which attach cooperatively using R glues. Only one keystone may attach, introducing nondeterminism; this is how the producibility of two assemblies, a top and bottom assembly, are implied by the production of one assembly satisfying S_i.

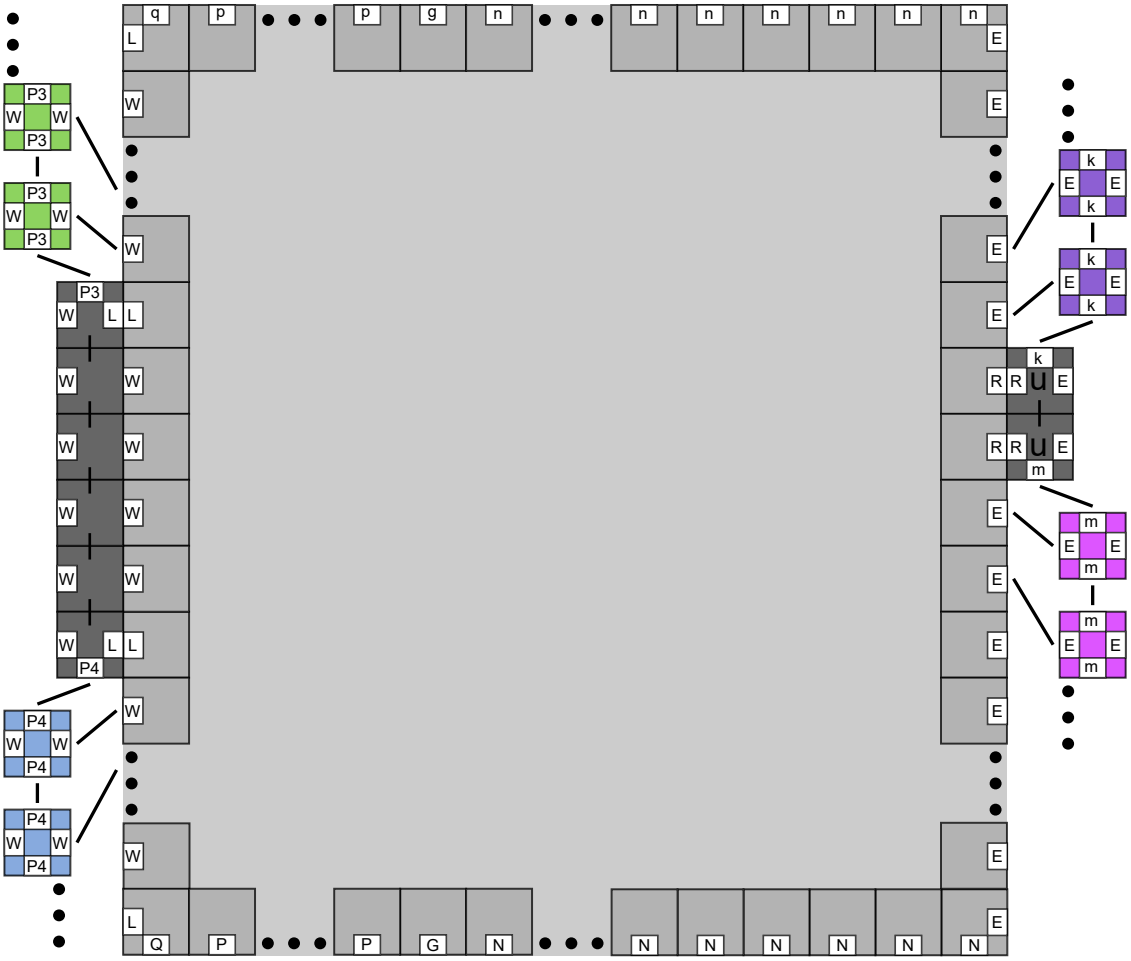


Figure 3.5: Once the keystone and supertile on the left have attached, a sequence of single tile attachments may occur using cooperative binding at newly available glues. The tiles are designed such that they may attach along arbitrarily long faces, as long as the appropriate glue is exposed (e.g., a W glue in the top-left tile's case).

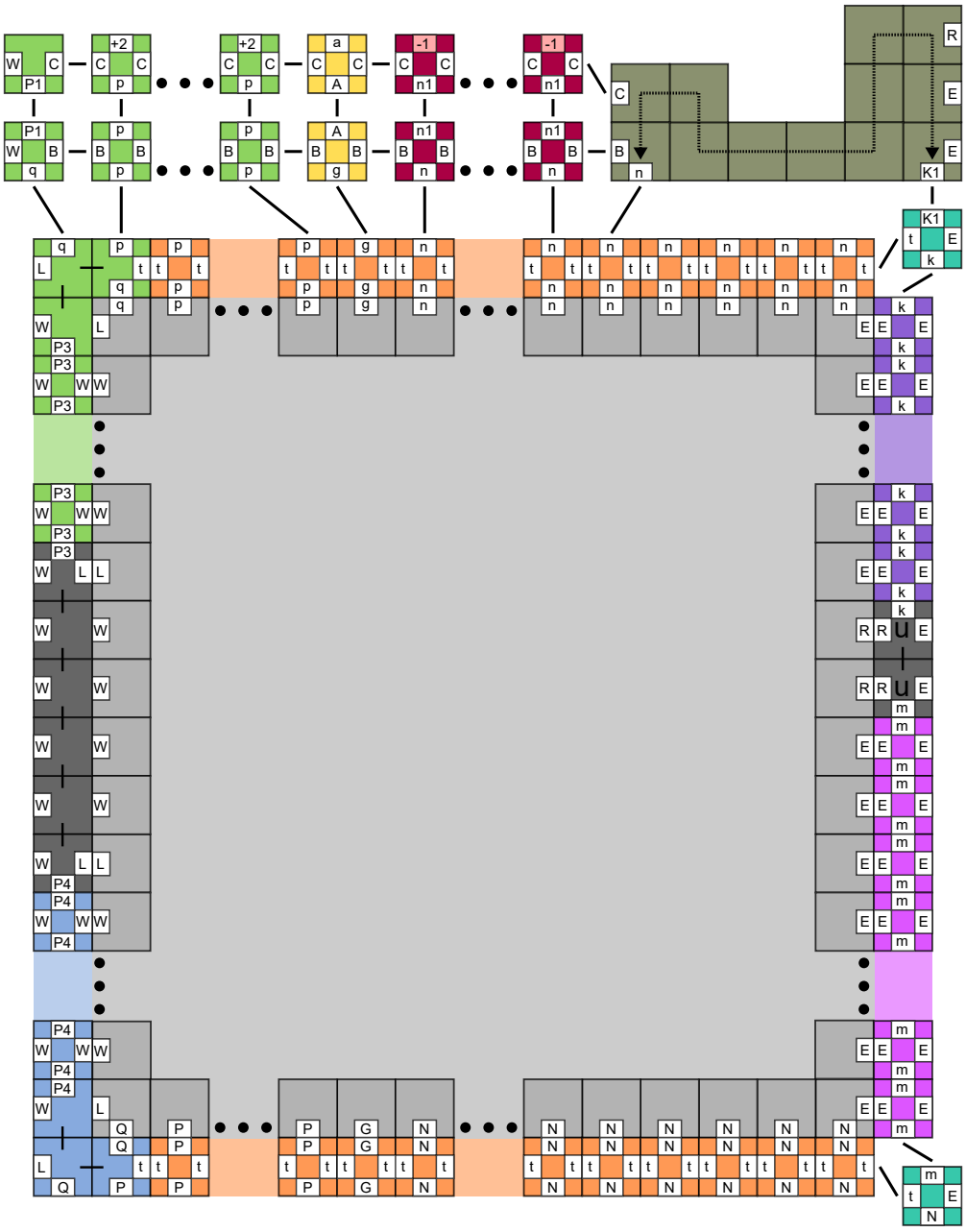


Figure 3.6: At the top-right and bottom-right corners, tiles attach which indicate that the left-hand side tile propagation has reached the right-hand side of the assembly. In the case of an assembly which has attached an up keystone, the tooth attaches on the top side of the assembly, and initiates a propagation of tiles along the top face. A tooth with complementary geometry will attach on the bottom side of the assembly if a down keystone attaches instead, as can be seen in Figure 3.7.

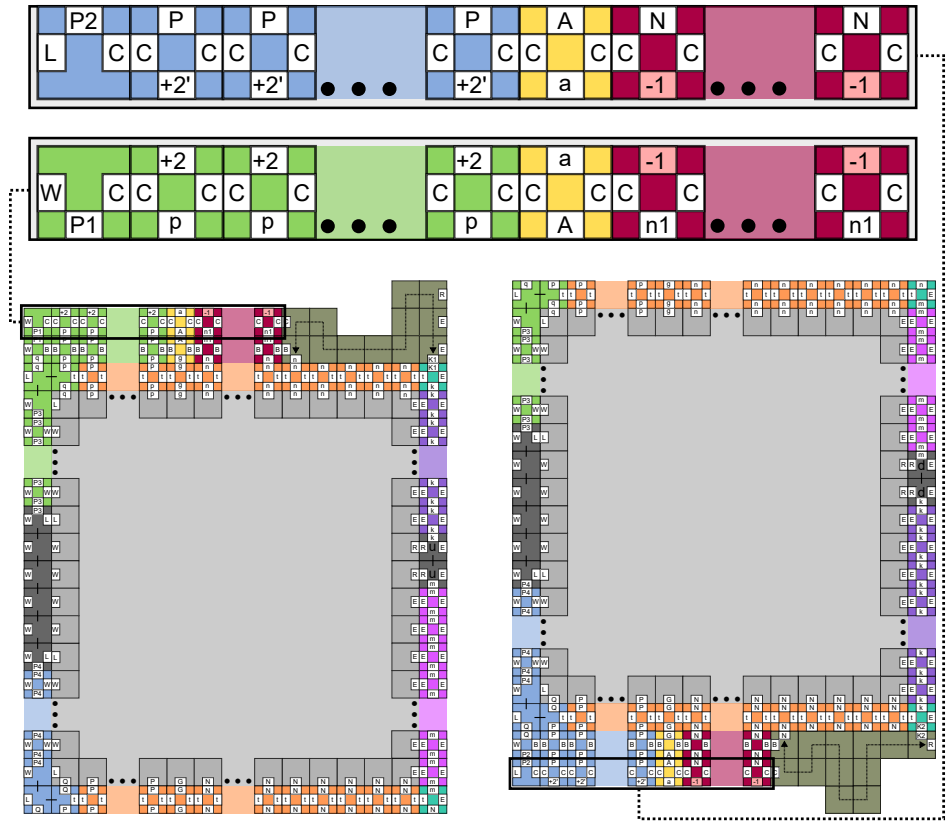


Figure 3.7: A bottom assembly (left) and top assembly (right). At the top of the figure are the tiles whose glues bond a bottom and top assembly; in particular, the a and -1 glues, with $str(a) = 2\tau - 1$ and $str(-1) = -1$. A top and bottom assembly grown from an assembly S_i each expose i glues with strength -1. Then the strength of the attachment between them is $2\tau - i - 1$, and is sufficient when $i < \tau$ but insufficient when $i = \tau$. Note that $+2$ and $+2'$ glues do not match; their purpose is described later in the finishing step.

Growing The construction is described and proven correct via induction. The induction is on iterations of rectangular assemblies with well-defined exposed glues, termed S_i. Formally, S_i refers to a $(7+2i) \times (2^{i+4} - 6)$ rectangular assembly with the following exposed glue labels, written as strings built by concatenating the glue labels in left-to-right/top-to-bottom order):

- North glues: $qp^i gn^{i+5}$
- East glues: $E^{2^{i+3}-5} R^2 E^{2^{i+3}-3}$
- South glues: $QP^i GN^{i+5}$
- West glues: $LW^{2^{i+3}-7} LW^4 LW^{2^{i+3}-7} L$

The goal is to show that if an assembly satisfying S_i is producible, then an assembly satisfying S_{i+1} is producible iff $i < \tau$. Further, only $\mathscr{O}(1)$ tile types are used in the inductive step. Then it suffices to show that S_0 is producible in $\mathscr{O}(1)$ tile types, implying S_τ is producible, which has width $\times$ height as in the lemma statement.

Base case. An assembly satisfying S_0 is shown to be trivially assembled by a set of $\mathscr{O}(1)$ tile types in Figure 3.3.

Inductive step. The next three paragraphs describe the inductive step. The goal is to show that if S_i is producible, then a top and bottom assembly are producible which can bind to produce S_{i+1} iff $i < \tau$. Consider an assembly with exposed glues satisfying S_i. The two R glues exposed allow attachment of a *keystone* assembly via cooperative binding as seen in Figure 3.4. There are two keystone types: *up* and *down*. Only one may attach. The L glues in the middle of the west-side exposed glues, spaced by four W glues, also allow the attachment of a supertile using cooperative binding. Once the keystone or west-side supertile has attached, a single tile type suffices to attach tiles along any long set of repeating glues on the assembly (e.g., the E glues on the east side), until the glue is no longer available, as seen in Figure 3.5. This type of single tile type attaching along arbitrary walls is termed *propagation* of a tile.

The tiles which propagate to the corners of the west side of the assembly allow the attachment of three tiles around the corner which allow propagation of tiles along the north and south faces of the assembly. Once these tiles propagate, depending on which keystone was attached to the assembly, the attachment of a *tooth* occurs on the corresponding face (e.g., on the north face if the up keystone was attached) as shown in Figure 3.6. The tooth is a supertile with a specific geometry which will be motivated later on. The tooth initiates a propagation of tiles along the corresponding face. The result is the production of two assemblies, one having attached the up keystone and attached all previously discussed propagating tiles and supertiles, and one having attached the down keystone and similar tiles. We refer to the former as a *top* assembly, and the latter as a *bottom* assembly.

When a top assembly and bottom assembly attach, the result is an assembly satisfying S_{i+1}.

A top assembly and bottom assembly are designed to attach iff $i < \tau$. When $i \geq \tau$, the binding strength between a top and bottom assembly is $\tau - 1$, and thus is insufficient. This design can be seen in Figure 3.7. Note that the -1 glues are propagated via the N-labeled glues from the base assembly. Since S_i has $i+5$ many N glues exposed, 5 of which are covered by the tooth, the bottom/top assembly which assembles from S_i exposes i many -1 glues. Then the bonding strength between top and bottom assemblies is $2\tau - 1 - i$, which is less than τ iff $i \geq \tau$. The complementary geometry of the teeth ensure that a top and bottom assembly which are assembled from S_i and S_j respectively, with $i \neq j$, cannot align their a glues and will not attach.

Dimensions of S_i. The base assembly satisfying S_0 is 7×10. When a top and bottom assembly attach, both have added 2 width in tiles; one tile width propagated on the west side, and one on the east. Then the width of S_i is $7 + 2i$. A top and bottom assembly which have combined add 6 height in tiles on top of doubling in height: 2 via tile propagation on the top and bottom, and 4 along where the top and bottom assemblies attach. Then the height of S_i, $h(i)$, is defined by the recurrence $h(i) = 2h(i-1) + 6$ with $h(0) = 10$. Solving the recurrence gives a height of $2^{i+4} - 6$. Then consider a combination of a top and bottom assembly which formed from some assembly satisfying S_{i-1}; the resultant dimension is $(7 + 2i) \times (2^{i+4} - 6)$.

Finishing When a top and bottom assembly combine to form an assembly satisfying S_τ, the growing step shows that the process will not continue to produce $S_{\tau+1}$. However, the attachment of a supertile on the west side, a keystone, and the resultant tile propagation still occurs. The teeth attach and so do the tiles which propagate resulting from the attachment of a tooth. Then an assembly satisfying S_τ implies the production of the corresponding top and bottom assemblies. These assemblies are not rectangular. This step involves detecting that the iterative process has reached τ repetitions, and the system should finish its rectangle.

Figure 3.8 gives an overview of the finishing step. The technique discussed in the growing phase is employed by two disjoint tile sets, one called *system 1* and the other *system 2*. The sets of glues on the tiles in the two systems are disjoint except for two glues: the -1 glue, and the $+2$ glue described but not used in the growing phase. In the growing phase, recall that on the north

face of a bottom assembly of system 1 which assembled from S_i, there are i many strength 2 glues labeled $+2$ exposed which are not used (recall Figure 3.7). These glues are designed to match with the corresponding glues in system 2. Then the strength of binding between these shared glues is $2i - i = i$. Thus, only when $i \geq \tau$ is this binding τ-stable. Similarly, system 1's top assembly attaches with the system 2's bottom assembly under the same constraint. These resultant assemblies expose cooperative binding locations which were not present before this attachment, allowing these two new assemblies to combine, and then fill into a rectangle using a $\mathscr{O}(1)$-sized tile set. Next, we give the dimensions of the completed rectangular assembly: system 1's top and system 2's bottom assembly, once attached, form a $(7+2(\tau+1)) = (9+2\tau) \times (2^{(\tau+1)+4} - 6) = (2^{\tau+5} - 6)$ assembly— this can be derived from the combination of two assemblies satisfying S_τ assembling into an assembly satisfying $S_{\tau+1}$ not in exposed glues, but in size. System 1's top and system 2's bottom are combined with system 1's bottom and system 2's top into one via a width-two column, resulting in a $2(9+2\tau)+2 = 18+4\tau \times 2^{\tau+5} - 6$ assembly. □ □

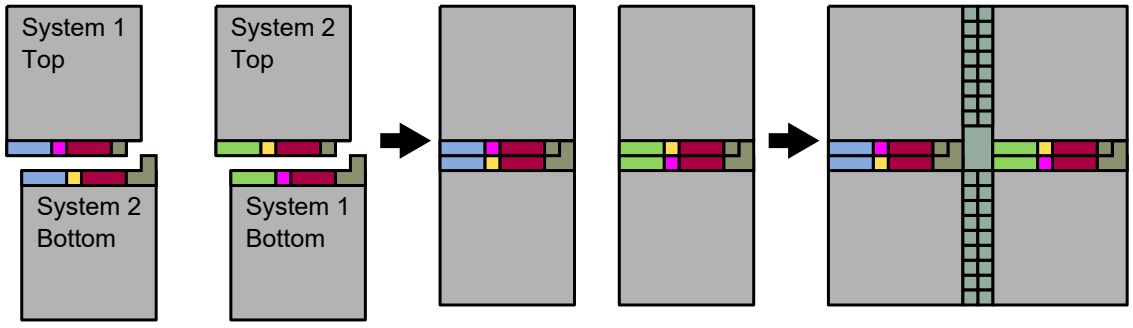


Figure 3.8: An overview of the finishing step. The system described in the growing step, denoted here as system 1, is repeated, denoted system 2, such that the only matching glues between the two systems are a strength-2 glue type and the one strength-(-1) glue type. Between a system 1 top/bottom and system 2 bottom/top assembled from S_i, the number of shared strength-2 glues and strength-(-1) glues is i, so the sum of strengths between shared glues is $2i - i = i$. This allows the top/bottom assembly of system 1 to make a τ-stable attachment to the bottom/top of system 2 only after each system assembles S_τ, thus detecting when the rectangle has $\Theta(\tau)$ width. Once these tops and bottoms attach, new cooperative binding locations initiate a constant-sized set of tiles to bind the two rectangles, simply to satisfy unique assembly of the rectangle.

3.4.2 From rectangle to shape

To assemble the target shape, a technique is combined with Lemma 1. The technique, shown by Soloveichik and Winfree [23], first assembles a *seed block* bearing a representation (a series of exposed glues) of a Kolmogorov-optimal program which computes the spanning tree of S. A $\mathscr{O}(1)$-sized tile set is used to "run" the program (via Turing machine (TM) simulation) from the seed block. The seed block assembles into a $c \times c$ assembly, logically representing one coordinate of S. Assembly proceeds from the seed block in a subset of the four cardinal directions depending on the spanning tree computed by the program. If the spanning tree has an edge in a direction to a coordinate adjacent to the seed block, a $c \times c$ *growth block* is assembled in that direction. Each time a growth block is assembled, the program is run again to determine which adjacent coordinates (w.r.t. the growth block) are connected by edges in the spanning tree; if so, a growth block is assembled in that direction. After assembling all growth blocks, the unique assembly is a $c \times c$ scaled version of S.

The seed block of [23] is assembled using $\mathscr{O}(\frac{K(S)}{\log K(S)})$ tile types. In our case, the seed block is assembled using the $\mathscr{O}(1)$-sized tile set of Lemma 1: we assemble a rectangle of width n where n is the length of a unary encoding of the Kolmogorov-optimal program. This seed is combined with the $\mathscr{O}(1)$-sized tile set which runs the program and assembles the shape. Figure 3.9 is a simplified overview of constructing a seed block compatible with a TM simulating tileset. The formal result is as follows:

Theorem 3. *Given a shape S, there exists a negative glue, growth-only* 2*HAM tile system* $\Gamma = \{T, \tau\}$ *with* $|T| = \mathscr{O}(1)$ *whose unique assembly has shape S at some scale factor.*

Proof. The system is a union of the Lemma 1 system and a subset of the TM simulating tile set of [23]. If the entire TM simulating tile set is added to the system, depending on the seed block, some tiles may never bind to the seed block, and thus do not grow into the target shape and violate unique assembly definition. We include the subset of the TM simulating tile set which will be used by the program encoded by the seed block. Observe that the unique assembly produced by the system of Lemma 1 is not a square, nor does it have any exposed glues designed to bind with the

TM simulation tile set. In order to assemble a square from the terminal assembly of Lemma 1, two such rectangular assemblies are assembled in parallel, one which is rotated 90 degrees— this "rotation" is w.r.t. the other rectangular assembly and the way the two assemblies will bind. These two assemblies combine and then "fill-in" to a square trivially using a constant-sized tile set— similar to propagation of tiles along an edge, cooperative binding can be used to add tiles between two assembled rectangles to assemble a square (technique first used in [16]). Once assembly of the square is complete, more tile propagation via another constant-sized tile set assembles a one-tile perimeter which exposes glues— described in the following paragraph— which allow the assembly to act as a seed block similar to the Soloveichik and Winfree construction [23].

Let P be the (binary) program used to assemble S via the construction of [23], R be some mapping from binary strings to unary strings, and R' be some mapping from unary strings $\{1\}^i$ with $i \in \mathbb{N}$ to unary strings $\{1\}^j$ with $j = 20 + 4m$ for $m \in \mathbb{N}$— the intuition for the mapping R' is to map arbitrary unary strings to numbers which are widths of assemblies assembled by Lemma 1.

Once the square seed block is assembled and a one-tile perimeter is attached, three glue types are exposed: 1, b, and λ. Across the length of filler tiles (those in the square which are not from the Lemma 1 construction), λ glues are placed; these symbols are ignored by the TM simulating tile set. The b glues are placed at the beginnings and ends of the edges of the square; these indicate where to start the TM simulation. Along the edges of the rectangles from the Lemma 1 construction, glues labeled 1 are placed; these 1 glues are the relevant glues logically. The TM simulation converts these to unary strings by R'^{-1}, and then to P by R^{-1}. Then the TM simulation tiles run the program P which assembles the shape. □ □

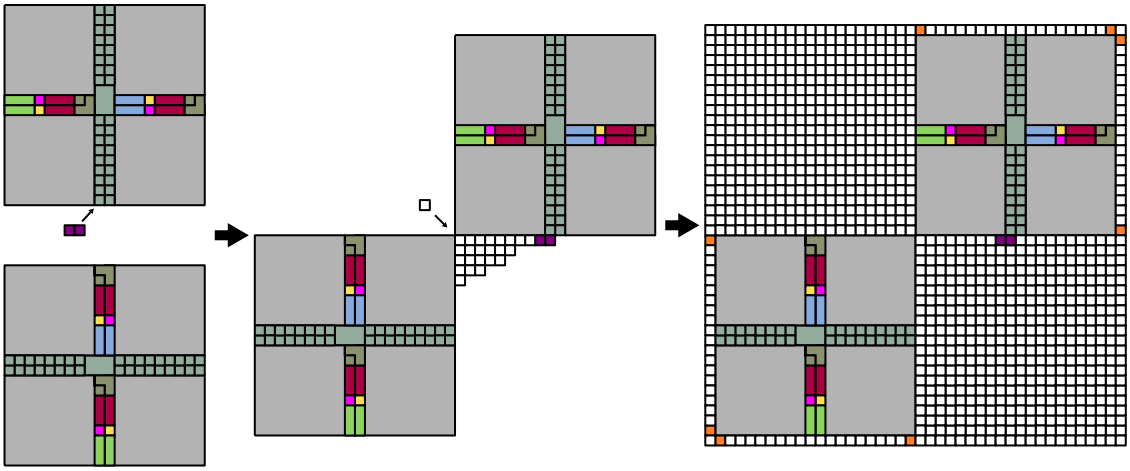


Figure 3.9: The combination of two Lemma 1 constructions into a seed block. The two-tile assembly in the first subfigure initializes the attachment of the set of white tiles, which indicate a constant-sized set of *filler* tiles which are used to fill in a full square. Once the square is filled in, new cooperative binding locations are exposed where the filler tiles meet the non-filler tiles. At this location, tiles begin to propagate, adding a one-tile perimeter to the assembly. The orange tiles on the outmost perimeter of the rightmost figure demarcate the beginning and ending of glues exposing the unary program which constructs the shape S via a TM simulation. The rest of the perimeter exposes glues which the TM simulation ignores.

3.5 Future Work

The most apparent direction for future work is to achieve the unique assembly of shapes at a $\mathcal{O}(1)$ scale factor with $\mathcal{O}(1)$ tile types. This result may be achieved through a combination of the techniques used in this work and in [12], which achieves $\mathcal{O}(1)$-scale factor with $\mathcal{O}(\frac{K(S)}{\log K(S)})$ tile types where $K(S)$ is the Kolmogorov complexity of the shape S. Their construction utilizes a dynamic behavior of negative glues not utilized in this work called "breaking": the combination of two assemblies may result in an unstable assembly, which then breaks into two assemblies—a formal model and some usages of breakage may be seen in [2, 8, 20]. Their usage of breaking involves performing a computation— via a self-assembly process which simulates a TM— which builds the shape pixel-by-pixel (using $\mathcal{O}(1)$-sized assemblies per pixel), and then breaks the TM simulating assembly into $\mathcal{O}(1)$-sized pieces, leaving the shape S at a $\mathcal{O}(1)$ scale factor (along with "small garbage" of $\mathcal{O}(1)$ size). That technique may be applicable to the construction given in this work in order to break the precise-width rectangles after they are used as input for the TM which outputs S.

Another direction might be to achieve the unique assembly of scaled shapes with $\mathscr{O}(1)$ tile types using only positive-strength glues. We have briefly discussed previous positive-strength results which use $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ tile types. Could this be lowered to $\mathscr{O}(1)$ tile types by calibrating the temperature and glue strengths, or is there some super-constant lower bound that cannot be breached?

BIBLIOGRAPHY

[1] S. CANNON, E. D. DEMAINE, M. L. DEMAINE, S. EISENSTAT, M. J. PATITZ, R. SCHWELLER, S. M. SUMMERS, AND A. WINSLOW, *Two hands are better than one (up to constant factors): Self-assembly in the 2HAM vs. aTAM.*, in STACS, N. Portier and T. Wilke, eds., vol. 20 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013, pp. 172–184.

[2] C. CHALK, E. D. DEMIANE, M. L. DEMAINE, E. MARTINEZ, R. SCHWELLER, L. VEGA, AND T. WYLIE, *Universal shape replicators via self-assembly with attractive and repulsive forces*, in Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17), 2017.

[3] C. CHALK, E. MARTINEZ, R. SCHWELLER, L. VEGA, A. WINSLOW, AND T. WYLIE, *Optimal staged self-assembly of general shapes*, Algorithmica, (2017).

[4] H.-L. CHEN, D. DOTY, AND S. SEKI, *Program size and temperature in self-assembly*, Algorithmica, 72 (2015), pp. 884–899.

[5] Q. CHENG, G. AGGARWAL, M. H. GOLDWASSER, M.-Y. KAO, R. T. SCHWELLER, AND P. M. DE ESPANÉS, *Complexities for generalized models of self-assembly*, SIAM Journal on Computing, 34 (2005), pp. 1493–1515.

[6] E. D. DEMAINE, M. J. PATITZ, T. A. ROGERS, R. T. SCHWELLER, S. M. SUMMERS, AND D. WOODS, *The two-handed tile assembly model is not intrinsically universal*, Algorithmica, 74 (2016), pp. 812–850.

[7] E. D. DEMAINE, M. J. PATITZ, R. T. SCHWELLER, AND S. M. SUMMERS, *Self-assembly of arbitrary shapes using RNAse enzymes: Meeting the kolmogorov bound with small scale factor*, in STACS 2011: Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science, 2011.

[8] D. DOTY, L. KARI, AND B. MASSON, *Negative interactions in irreversible self-assembly*, Algorithmica, 66 (2013), pp. 153–172.

[9] C. EVANS, *Crystals that Count! Physical Principles and Experimental Investigations of DNA Tile Self-Assembly*, PhD thesis, California Inst. of Tech., 2014.

[10] M.-Y. KAO AND R. T. SCHWELLER, *Reducing tile complexity for self-assembly through temperature programming*, in SODA 2006: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 571–580.

[11] M. LI AND P. VITANYI, *An Introduction to Komogorov Complexity and Its Applications (Second Edition)*, Springer Verlag, New York, 1997.

[12] A. LUCHSINGER, R. SCHWELLER, AND T. WYLIE, *Self-assembly of shapes at constant scale using repulsive forces*, in Unconventional Computation and Natural Computation, M. J. Patitz and M. Stannett, eds., Cham, 2017, Springer International Publishing, pp. 82–97.

[13] M. J. PATITZ, T. A. ROGERS, R. T. SCHWELLER, S. M. SUMMERS, AND A. WINSLOW, *Resiliency to multiple nucleation in temperature-1 self-assembly*, in Proceedings of the 22nd International Conference on DNA Computing and Molecular Programming (DNA), vol. 9818 of LNCS, Springer, 2016, pp. 98–113.

[14] M. J. PATITZ, R. T. SCHWELLER, AND S. M. SUMMERS, *Exact shapes and Turing universality at temperature 1 with a single negative glue*, in DNA Computing and Molecular Programming, vol. 6937 of LNCS, 2011, pp. 175–189.

[15] J. H. REIF, S. SAHU, AND P. YIN, *Complexity of graph self-assembly in accretive systems and self-destructible systems*, Theoretical Comp. Sci., 412 (2011), pp. 1592–1605.

[16] P. ROTHEMUND AND E. WINFREE, *The program-size complexity of self-assembled squares*, 2000, pp. 459–468.

[17] P. W. K. ROTHEMUND, *Using lateral capillary forces to compute by self-assembly*, Proceedings of the National Academy of Sciences, 97 (2000), pp. 984–989.

[18] ——, *Folding DNA to create nanoscale shapes and patterns*, Nature, 440 (2006), pp. 297–302.

[19] N. SCHIEFER AND E. WINFREE, *Universal Computation and Optimal Construction in the Chemical Reaction Network-Controlled Tile Assembly Model*, Springer International Publishing, Cham, 2015, pp. 34–54.

[20] R. SCHWELLER AND M. SHERMAN, *Fuel efficient computation in passive self-assembly*, in SODA 2013: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2013, pp. 1513–1525.

[21] R. SCHWELLER, A. WINSLOW, AND T. WYLIE, *Complexities for high-temperature two-handed tile self-assembly*, in DNA Computing and Molecular Programming, R. Brijder and L. Qian, eds., Cham, 2017, Springer International Publishing, pp. 98–109.

[22] S. SEKI AND Y. UKUNO, *On the behavior of tile assembly system at high temperatures*, Computability, 2 (2013), pp. 107–124.

[23] D. SOLOVEICHIK AND E. WINFREE, *Complexity of self-assembled shapes*, SIAM Journal on Computing, 36 (2007), pp. 1544–1569.

[24] S. M. SUMMERS, *Reducing tile complexity for the self-assembly of scaled shapes through temperature programming*, Algorithmica, 63 (2012), pp. 117–136.

CHAPTER IV

FULL TILT: UNIVERSAL CONSTRUCTORS FOR GENERAL SHAPES WITH UNIFORM EXTERNAL FORCES

Collaborators: Jose Balanza-Martinez, David Caballero, Angel A. Cantu, Luis Angel Garcia, Rene Reyes, Robert Schweller, and Tim Wylie **My contribution:** LAG presented the initial concept for the patterns and general shapes constructor. He and I then continued to work out the details for this result. I was also responsible for the initial creation of the relocation and reconfiguration gadgets used in the PSPACE-completeness proof. The text was written by all. **This chapter was published as:** Jose Balanza-Martinez, David Caballero, Angel A. Cantu, Luis Angel Garcia, Austin Luchsinger, Rene Reyes, Robert Schweller, and Tim Wylie, "Full Tilt: Universal Constructors for General Shapes with Uniform External Forces," In Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms, 2019.

4.1 Abstract

We investigate the problem of assembling general shapes and patterns in a model in which particles move based on uniform external forces until they encounter an obstacle. In this model, corresponding particles may bond when adjacent with one another. Succinctly, this model considers a 2D grid of "open" and "blocked" spaces, along with a set of slidable polyominoes placed at open locations on the board. The board may be tilted in any of the 4 cardinal directions, causing all slidable polyominoes to move maximally in the specified direction until blocked. By successively applying a sequence of such tilts, along with allowing different polyominoes to stick when adjacent, tilt sequences provide a method to reconfigure an initial board configuration so as to assemble a collection of previous separate polyominoes into a larger shape.

While previous work within this model of assembly has focused on designing a specific board configuration for the assembly of a specific given shape, we propose the problem of designing *universal configurations* that are capable of constructing a large class of shapes and patterns. For these constructions, we present the notions of *weak* and *strong* universality which indicate the presence of "excess" polyominoes after the shape is constructed. In particular, for given integers h, w, we show that there exists a strongly universal configuration with $\mathcal{O}(hw)$ 1×1 slidable particles that can be reconfigured to build any $h \times w$ patterned rectangle. We then expand this result to show that there exists a weakly universal configuration that can build any $h \times w$-bounded size connected shape. Following these results, which require an admittedly relaxed assembly definition, we go on to show the existence of a strongly universal configuration (no excess particles) which can assemble any shape within a previously studied "drop" class, while using quadratically less space than previous results.

Finally, we include a study of the complexity of motion planning in this model. We consider the problems of deciding if a board location can be occupied by any particle (occupancy problem), deciding if a specific particle may be relocated to another position (relocation problem), and deciding if a given configuration of particles may be transformed into a second given configuration (reconfiguration problem). We show all of these problems to be PSPACE-complete with the allowance of a single 2×2 polyomino in addition to 1×1 tiles. We further show that relocation and occupancy remain PSPACE-complete even when the board geometry is a simple rectangle if domino polyominos are included.

4.2 Introduction

The "tilt" model is an elegant and simple model of robotic motion planning and assembly proposed by Becker et. al. [7] with foundations in classical motion planning. The model consists of a 2D grid board with "open" and "blocked" spaces, as well as a set of slidable polyominoes placed at open locations on the board. At the micro or nano scale, individually instructing specific particles may be impossible. Thus, this model uses a global external force to give all movable particles the same instruction. This may be done through any external force such as a magnetic field or gravity.

In this work we assume gravity is the global force, and this is where the term *tilt* comes from. In the simplest form of the problem, the board may be tilted (as an external force) in any of the four cardinal directions, causing all slidable polyominoes to slide maximally in the respective direction until reaching an obstacle. These mechanics have proved to be interesting bases for puzzle games, as evidenced by the maximal movement of [1] and the global movement signals in [2]. By adding bonding glues on polyomino edges, as is done in self-assembly theory [13, 19, 22], the polyominoes may stick together after each tilt, enabling this model to be a framework for studying the assembly of general shapes.

In this work we propose a new type of problem: the design of board configurations that are *universal* for a class of general shapes or patterns. That is, we are interested in designing a single board configuration that is capable of being reconfigured to assemble any shape or pattern within a given set of shapes or patterns by applying the proper sequence of tilts. This problem is distinct from problems considered in prior work in which a given shape is assembled by encoding the particular shape into a specific board configuration (i.e. encoding the shape by way of placement of "blocked" locations and initial polyomino placement). As an analogy, prior work has focussed on building a specific purpose machine for building copies of a target shape via a simple repeating tilt sequence. Here, we propose to build something more akin to a computer printer in which any shape or pattern, provided sufficient fuel/ink/polyominoes, may be requested from the machinery, and the particular shape requested is encoded by a provided tilt sequence.

As our primary focus in this paper relates to the problem of reconfiguring board configurations, a natural computational question arises: what is the complexity of deciding if a given board configuration may be reconfigured into a second target board configuration? In [9], the authors show that minimizing the number of tilts to reconfigure between two configurations is PSPACE-complete. In this paper, we add to this growing understanding of reconfiguration complexity by showing that deciding if reconfiguration is possible is PSPACE-complete with 1×1 movable tiles and a single 2×2 movable polyomino. A related problem, which asks if a given location can be occupied by any polyomino, has also been considered. This problem was shown to be NP-hard [8] in the restricted

case of 1×1 polyominoes that never stick together (but is not known to be in NP). We also extend this line of work by showing this problem is PSPACE-Complete if polyominoes larger than than 1×1 tiles are allowed. We also show that this problem remains PSPACE-comeplete with different constraints such as limiting the number of larger polyominos and the board complexity.

4.2.1 Motivations

The beauty of the Tilt model is its simplicity combined with its depth. These features allow this model to be a framework for computation and assembly within a number of potential applications at various scales. A few examples at the macro, micro, and nano-scale are as follows.

Macro-scale The simplicity of the Tilt model allows for implementation with surprisingly simple components. For instance, Becker et al. have constructed a modular, reconfigurable board with both geometry and sliding components, which they have demonstrated with video walkthroughs [6]. Further, the components allow for attaching magnets to implement bonding between components. These bonded components can be viewed as polyominoes, which can even be sorted within a tilt system [15]. Even a basic set of Legos is capable of quickly implementing many of our proposed constructions. These systems can be represented realistically with games like Tilt by Thinkgames [3] and the marble based Labyrinth [11].

Micro-scale It has been shown how to control magneto-tactic bacterium via global signals to move bacteria around complex vascular networks via magnets [5], and how the same type of bacteria can be moved magnetically through a maze [16]. This has a plethora of medical uses, such as minimal invasive surgery, targeted drug payloads, subdermal micro-constructions, "targeted delivery of hemotherapeutic agents or therapeutic genes into malignant cells while sparing healthy cells" [21], and "medical intervention in targeted zones inaccessible to catheterization" [17].

Nano-scale Promising potential applications for Tilt Assembly systems may even be found in the emerging field of DNA nanotechnology. DNA walkers have been engineered to traverse programmed paths along substrates such as 2D sheets of DNA origami [14, 24]. Such walkers may be augmented with activation signals to drive the walker forward one step by way of a DNA

Result	Shape Class	Universal	Tilts	Size	Bonding Complexity	Geometry Complexity	Theorem
Fixed Shape	DROP	No	$O(hw)$[1]	$O(h^3w^3)$	2 labels	Connected	Thm. 6 in [10]
Universal Patterns	All	Strongly	$O(hwk)$	$O(hwk)$	k labels	Simple	Thm. 4
Universal Shapes	All	Weakly	$O(hw)$	$O(hw)$	1 label	Simple	Thm. 5
Universal Shapes	DROP	Strongly	$O(h^2w)$	$O(h^2w)$	2 labels	Connected	Thm. 6

Table 4.1: An overview of the shape construction results. The **Result** is the type of constructor achieved, and the **Shape Class** refers to the types of shapes that can be built. The **Tilts** are the number of board tilts, or external forces, required to build the shape. The **Size** refers to the size of the board necessary with respect to the size of the $h \times w$ bounding box of the shape being built. The **Bonding Complexity** is the number of distinct particle types that can attach to each other and are necessary to build the shape. The k labels needed for patterns is the number of desired types of particles in the pattern ($1 \leq k \leq |S|$). The **Geometry Complexity** refers to the complexity of the open and blocked pieces of the board based on the hierarchy provided in Section 4.3. The **Theorem** refers to where this information is from.

Problem	Shapes	Geometry	Complexity	Theorem
Relocation/Occupancy	1×1	Connected	NP-hard	1 in [8]
Reconfiguration Optimization	1×1	Connected	PSPACE-complete	10 in [8]
Relocation/Occupancy	1×1, 2×2	Connected	PSPACE-complete	8, 3
Reconfiguration	1×1, 2×2	Connected	PSPACE-complete	9

Table 4.2: An overview of the computational complexity results related to tilt assembly and our results. The **Problem** gives the computational question. The **Shapes** refer to the size of the polyominos that are allowed to move within the world. The **Geometry** column refers the complexity of the nonmovable tiles needed in the reduction. The **Complexity** refers to the computational complexity class that the problem was proven to be a member, and the **Theorem** is the reference to the result.

strand-displacement reaction [23]. By flooding the system with a given signal, the walker could be pushed to continuously walk forward until stopped by some form of blocked location. By further implementing four such signal reactions (one for each cardinal direction), and adding a specifically chosen signal type at each stage or step of the algorithm, a set of these DNA walkers become a nanoscale implementation of the Tilt Assembly model. If feasible, such an implementation implies our Tilt algorithms offer a novel technique for the construction of nanoscale shapes and patterns.

4.2.2 Our Contributions in Detail

We first show the existence of a universal configuration for building any $h \times w$ bounded shape or pattern. The result utilizes simple geometry (the set of open spaces on the board has

genus-0), only a single type of bonding particle, and is quadratically smaller in size than the corresponding non-universal construction from previous work [10]. Moreover, this is the first result in the literature that is capable of building any connected shape. However, in the case of general shape construction, we say this system is only *weakly* universal in that it allows for the inclusion of "helper" polyominoes that are not counted as part of the final shape as they do not stick to any other tile. Our next result is for *strong* universality in which only a single final polyomino of the desired shape is permitted. In this case we achieve a restricted class of shapes termed "Drop" shapes which are shapes buildable by dropping 1×1 polyominoes onto the outside of the shape from any of the four cardinal directions. Previous non-universal work has focused on both constructing and identifying members of the Drop shapes class [10]. A summary of universal shape construction results, along with closely related prior work, is provided in Table 4.1.

Our next set of results focuses on various decision problems within the tilt model. The *occupancy* problem asks if a given location on a board can be occupied by a polyomino. The *relocation* problem asks if a specific tile is able to be relocated to a given location. The *reconfiguration* problem asks whether a given board configuration may be reconfigured into a second given configuration. We show these problems to be PSPACE-complete, even when polyominoes are restricted to be 1×1 and a single 2×2 piece that never stick to each other. Our proofs rely on a reduction from a 2-tunnel gadget network game with different gadget types that was recently proven to be PSPACE-complete by Demaine, Grosof, Lynch, and Rudoy [12]. Previous work on occupancy has shown the problem to be NP-hard [8] even when restricted to 1×1 pieces that do not stick. A closely related problem of computing the minimum number of tilts needed to reconfigure between two board configurations has been shown to PSPACE-complete for 1×1 non-sticking pieces [8]. A summary of our complexity results, along with closely related prior work, is provided in Table 4.2.

[1]This technique permits "pipelined" construction for the creation of n copies of the target shape in amortized $O(1)$ number of tilts per copy.

4.3 Preliminaries

Board. A *board* (or *workspace*) is a rectangular region of the 2D square lattice in which specific locations are marked as *blocked*. Formally, an $m \times n$ board is a partition $B = (O, W)$ of $\{(x,y)|x \in \{1,2,\ldots,m\}, y \in \{1,2,\ldots,n\}\}$ where O denotes a set of *open* locations, and W denotes a set of *blocked* locations- referred to as "concrete" or "walls." We classify the different possible board geometries according to the following hierarchy:

- Connected: This hierarchy level was known as *complex*. We modify this class to allow for a proper hierarchy of shape classes. A board is said to have *connected* geometry if the set of open spaces O for a board is a connected shape.
- Simple: defines simple geometry based on the connectivity of W. We also modify the concept for hierarchical purposes. A connected board is said to be *simple* if O has genus-0.
- Monotone: A simple board is *monotone* if O is either horizontally monotone or vertically monotone.
- Convex: A monotone board is *convex* if O is both horizontally monotone and vertically monotone.
- Rectangular: A convex board is *rectangular* if O is a rectangle.

Our board definitions have changed since [4] in order to create the hierarchy shown above.

Tiles. A tile is a labeled unit square centered on a non-blocked point on a given board. Formally, a tile is an ordered pair (c,a) where c is a coordinate on the board, and a is an attachment label. Attachment labels specify which types of tiles will stick together when adjacent, and which have no affinity. For a given alphabet of labels Σ, and some *affinity* function $G : \Sigma \times \Sigma \to \{0,1\}$ which specifies which pairs of labels attract ($G(a,b) = 1$) and which do not ($G(a,b) = 0$), we say two adjacent tiles with labels a and b are *bonded* if $G(a,b) = G(b,a) = 1$.

Polyomino. A *polyomino* is a finite set of tiles $P = \{t_1, \ldots t_k\}$ that is 1) connected with respect to the coordinates of the tiles in the polyomino and 2) *bonded* in that the graph of tiles in P with edges connecting bonded tiles is connected. A polyomino that consists of a single tile is informally referred to as simply a “tile".

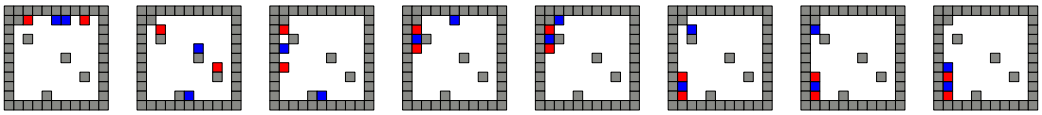
Configurations. A configuration is an arrangement of polyominoes on a board such that there are no overlaps among polyominoes, or with blocked board spaces. Formally, a configuration $C = (B, P = \{P_1 \ldots P_k\})$ consists of a board B, along with a set of non-overlapping polyominoes P that each do not overlap with the blocked locations of board B.

Step. A *step* is a way to turn one configuration into another by way of a global signal that moves all polyominoes in a configuration one unit in a direction $d \in \{N, E, S, W\}$ when possible without causing an overlap with a blocked location, or another polyomino. Formally, for a configuration $C = (B, P)$, consider the translation of all polyominoes in P by 1 unit in direction d. If no overlap with blocked board spaces occurs, then the new configuration is derived by first performing this translation, and then merging each pair of polyominoes that each contain one tile from a now (adjacently) bonded pair of tiles. If an overlap does occur, for each polyomino for which the translation causes an overlap with a blocked space, temporarily add these polyominoes to the set of blocked spaces and repeat. Once the translation induces no overlap with blocked spaces, execute the translation and merge polyominoes based on newly bonded tiles to arrive at the new configuration. If all polyominoes are eventually marked as blocked spaces, then the step transition does not change the initial configuration. If a configuration does not change under a step transition for direction d, we say the configuration is *d-terminal*. In the special case that a step causes a polyomino to “leave the board", we simply remove the polyomino from the configuration.

Tilt. A *tilt* in direction $d \in \{N, E, S, W\}$ for a configuration is executed by repeatedly applying a

step in direction $d \in \{N, E, S, W\}$ until a d-terminal configuration is reached. We say that a configuration C can be *reconfigured in one move* into configuration C' (denoted $C \to_1 C'$) if applying one tilt in some direction d to C results in C'. We define the relation $\to_*$ to be the transitive closure of $\to_1$. Therefore, $C \to_* C'$ means that C can be reconfigured into C' through a sequence of tilts.

Tilt Sequence. A *tilt sequence* is a series of tilts which can be inferred from a series of directions $D = \langle d_1, d_2, \ldots, d_k \rangle$; each $d_i \in D$ implies a tilt in that direction. For simplicity, when discussing a tilt sequence, we just refer to the series of directions from which that sequence was derived. Given a starting configuration, a tilt sequence corresponds to a sequence of configurations based on the tilt transformation. An example tilt sequence $\langle S, W, N, W, S, W, S \rangle$ and the corresponding sequence of configurations can be seen in Figure 4.1.



(a) Start (b) South (c) West (d) North (e) West (f) South (g) West (h) South

Figure 4.1: Tilt Example

Universal Configuration. A configuration C' is universal to a set of configurations $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ if and only if $C' \to_* C_i \; \forall \; C_i \in \mathscr{C}$.

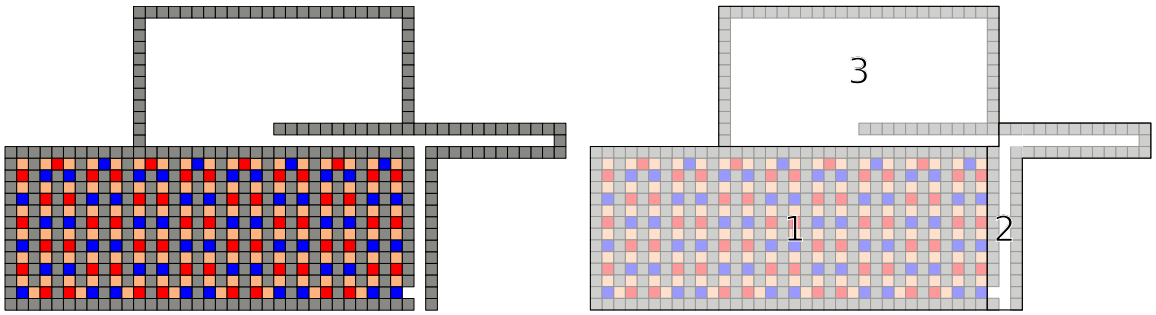
Configuration Representation. A configuration may be interpreted as having constructed a "shape" in a natural way. Define a shape to be a connected subset $S \subset Z^2$. A configuration *strongly* represents S if the configuration consists of a single polyomino whose tile coordinates are exactly the points of some translation of S. A weaker version (discussed in detail in section 4.4) allows for some "helper" polyominoes to exist in the configuration and not count towards the represented shape. In this case, we say a configuration *weakly* represents S. We extend this idea of shape representation to include patterns. We say a configuration represents a pattern if each attachment label used in

the representation corresponds to exactly one symbol of the pattern. For example, a configuration with attachment labels $\{a_1, a_2, \ldots, a_n\}$ represents a pattern with symbols $\{s_1, s_2, \ldots, s_n\}$ if the location of each tile $t_i \in C$ with attachment label a_i matches with the positions of symbol s_i in the pattern.

Universal Shape Builder. Given this representation, we say a configuration C' is *universal* for a set of shapes U if and only if there exists a set of configurations $\mathscr{C}$ such that 1) each $u \in U$ is represented by some $C \in \mathscr{C}$ and 2) C' is universal for $\mathscr{C}$. If each $u \in U$ is strongly represented by some $C \in \mathscr{C}$, we say C' is *strongly universal* for U. Alternately, if each $u \in U$ is weakly represented by some $C \in \mathscr{C}$, we say C' is *weakly universal* for U. In a similar way, a configuration can be universal for a set of patterns.

4.4 Patterns and General Shapes

In this section we present a shape builder which is universal for the set of all $h \times w$ binary-patterned rectangles. We then extend this approach to achieve a universal constructor for any connected shape, or even any patterned connected shape. We first cover a high-level overview of how the construction works and then formally state and prove the result.

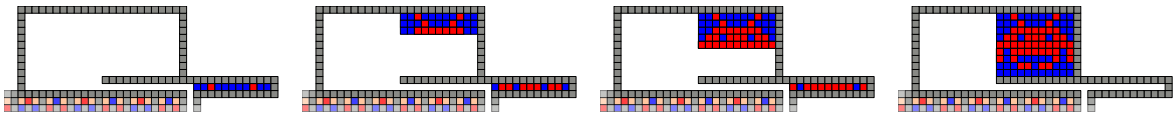


(a) Starting Configuration (b) Sections

Figure 4.2: An overview of the universal pattern and shape constructor. (a) The universal binary-patterned rectangle builder configuration in a starting configuration. (b) We refer to various sections of the configuration by different names. Section 1 is the *fuel chamber*, section 2 is the *loading chamber*, and section 3 is the *construction chamber*.

4.4.1 Binary-patterned Rectangles: Construction

The high-level idea behind this construction is simple: tiles are removed from the fuel chamber one at a time and are either used in "line assembly," or ejected from the system. Once a patterned line is complete, it is used for "rectangle assembly". Essentially, we are assembling a patterned rectangle pixle-by-pixel, row-by-row. For a given rectangle size ($h \times w$), we can construct a tilt assembly configuration (Figure 4.2) which can assemble any binary-patterned rectangle of that size.



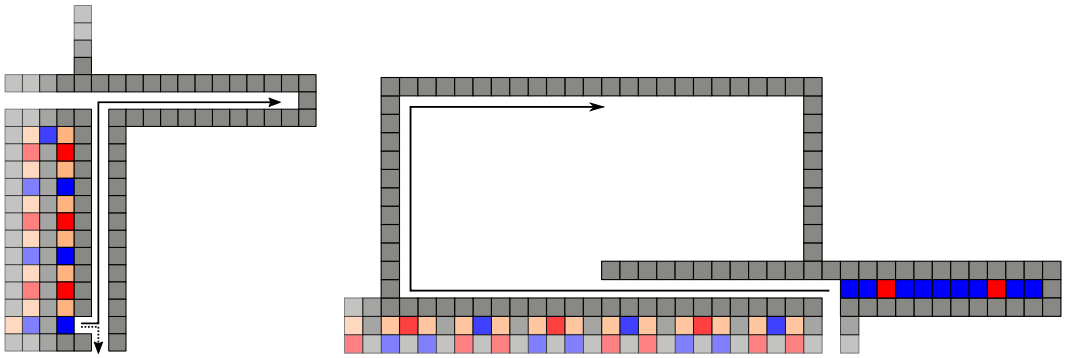
(a) (b) (c) (d)

Figure 4.3: The rectangle construction process at different points. Patterns are built row by row and then added to the final shape. (a) depicts an assembled row of the pattern. (b) and (c) show subsequent configurations after more rows have been added and (d) shows the final assembled pattern.

Fuel Chamber The fuel chamber is the portion of our system which contains the tiles that are used for construction. Our construction utilizes 2 types of tiles that attach to themselves and to each other, and 1 tile that does not attach to anything (we often refer to this tile as *sand*)[2]. We must be able to fill up the entire hw-sized porion of the construction chamber with either of the "sticky" tiles, so we need to allocate room for that many tiles in the fuel chamber. In the fuel chamber, each sticky tile must be separated by a sand tile, so we do not end up with "clumps" of fuel. Thus, we have $4hw$ tiles in our fuel chamber. To progress fuel through the fuel chamber, an input sequence of $\langle N, E, S \rangle$ is required. It is important to note that each of our commands in Table 4.3 ends with this sequence, thereby naturally advancing the fuel through the fuel chamber.

Loading Chamber The loading chamber is the next portion of the construction. It is here that tiles can either be removed from the configuration, or loaded into the upper part of the loading

[2]The term *sand* is inspired by the game Minecraft [18] in which blocks of type "sand" do not stick to adjacent blocks and will fall freely if nothing lies beneath them. Here, sand refers to 1×1 tiles that do not stick to any other tile and are not counted as part of the final assembly.



(a) Add/Remove Tile (b) Add Line

Figure 4.4: Basic sequences used in the constructor. The cyclic sequence to advance the fuel is $\langle S,E,N,E,S,E,\ldots\rangle$. (a) The sequence to add a tile to the line is $\langle E,N,E,S\rangle$. (b) The sequence to remove a tile from the system is $\langle E,S,N,E,S\rangle$.

chamber. As tiles are added to the loading chamber, one row (line) of the pattern is built. The vertical portion of this chamber is the same height as the fuel chamber, and the horizontal portion has a width of w. Once a line is complete, it is added to the construction chamber.

Construction Chamber The construction chamber is the portion of the tilt assembly system where our pattern will be constructed. Its dimensions are determined by the size of the rectangle to be constructed. The upper portion (with dimensions $h \times 2w$) is where the binary-patterned rectangle will be constructed. The lower portion (with dimensions $1 \times 2w$) is where the pre-assembled lines enter from the loading chamber. The construction chamber can be thought of as having two parts. The left portion is where lines are added to the already existing pattern, and the right portion is where the pattern is stored for other move sequences.

Line Construction The $1 \times w$ lines (rows of the rectangle) are constructed pixel-by-pixel (tile-by-tile) from right-to-left. For each pixel, the user decides if a blue or red tile should be placed and discards the other (as well as the separating sand). The sequences required for tile addition and removal are in Figure 4.4.

Rectangle Construction Once the line (row) is complete, the next step is to add it to the construction chamber. The line construction process is then repeated over and over again, with each new line being added to the construction chamber. Thus, the rectangle is built row-by-row from

top-to-bottom. The sequences for all moves are in Table 4.3.

Moves	Add tile	Remove tile	Add line
Tilts	$\langle E,N,E,S\rangle$	$\langle E,S,N,E,S\rangle$	$\langle W,N,E,S\rangle$

Table 4.3: Tilt sequences used for general shape and pattern construction. Note that a tile will not be removed during the Add Tile and Add Line commands (despite the presence of $\langle E,S\rangle$ in both) because of the preceding N command and the fact that the rightmost column of the fuel chamber will be missing at least 1 tile.

4.4.2 Patterned Rectangles and General Shapes: Formal Results

Theorem 4. *Given two positive integers $h,w \in \mathbb{Z}^+$, there exists a configuration C which is* strongly universal *for the set of patterns $U = \{u \mid u$ is an $h \times w$ rectangle, where each pixel has a label $x \in \{0,1\}\}$. This configuration has size $\mathcal{O}(hw)$ and uses $\mathcal{O}(hw)$ tilts to reconfigure into a configuration which strongly represents any pattern $u \in U$.*

Proof. We show this by constructing a configuration $C = (B,P)$, similar to that shown in Figure 4.2a, where B is the board and P is the set of 1×1 polyominoes in the fuel chamber in their starting configuration.

The three chambers of the board (fuel, loading, and construction) with scale based on h and w are:

- *Fuel chamber.* Let the fuel chamber of this configuration be of height $h+3$ and length $8w+2$. The fuel chamber is a rectangular area with staggered columns of blocked locations, so as to create a serpentine path of length $4hw$. Let this path be filled with tiles of alternating attachment labels a,b buffered with label ε, where $G(a,a) = 1, G(a,b) = 1, G(b,b) = 1$ and ε is the attachment label with no affinity. This allows enough room to have a single serpentine line of $4hw$ tiles that do not stick to each other.

- *Loading chamber.* The loading chamber must have a vertical hallway whose height is the same as the height of the fuel chamber ($h+2$). Let the loading chamber also contain a horizontal tunnel whose width is $w+2$.

- *Construction chamber.* The construction chamber can be described as a large set of open locations with a perimeter of blocked locations resembling the figures above. Let the construction chamber have height $h+2$ and width $2w+2$.

By observing the dimensions of the chambers, the height of the board is $2h+5$ and the width of the board is $9w+4$. So, the size of the board is $\mathcal{O}(hw)$.

Reconfiguration. Consider the set of configurations $\mathscr{C}'$, where each $c' \in \mathscr{C}'$ is similar to that shown in Figure 4.2a and *strongly* represents some $u \in U$ by a patterned-rectangle in the construction area. From configuration c, and the construction shown above, there exists a tilt sequence to construct any binary-colored line pixel-by-pixel. There also exists a tilt sequence to construct a rectangle with these binary-colored lines row-by-row. These sequences, along with that to discard any superfluous tiles, shows that there exists a complete sequence to build any rectangular binary pattern. Thus, $\forall\, c' \in \mathscr{C}', C \to_* c'$. □

General Patterns By increasing the size of the fuel chamber, we can easily generalize this result to k tile types (represented by labels or colors). Given k different labeled tiles for the pattern, the fuel chamber just needs to repeat the sequence of tiles in order, with sand in between each labeled tile, enough times to ensure the desired pattern can be built.

Corollary 1. *Given three positive integers $h,w,k \in \mathbb{Z}^+$, there exists a configuration C which is* strongly universal *for the set of patterns $U = \{u \mid u$ is an $h \times w$ rectangle, where each pixel has a label $x \in \{0,1,\ldots,k\}\}$. This configuration has size $\mathcal{O}(hwk)$ and uses $\mathcal{O}(hwk)$ tilts to reconfigure into a configuation which strongly represents any pattern $u \in U$.*

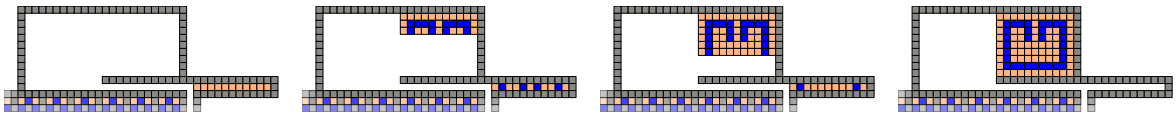
Proof. We use the construction from Theorem 4 and extend the fuel chamber to include k colors. The length of the serpentine fuel line would then be $\mathcal{O}(hwk)$ meaning the width of our board would now scale with k as well. Also, since each pixel requires the user to select a color and discard the others, the number of tilts would also scale with k. □

General Shapes With a simple extension of this result, we are able to achieve the construction of general shapes. By including sand in the building process and the finished pattern, we can

construct any connected shape. This is weakly built by our definition since there are non-attached tiles built along with it. Figure 4.5 shows this process.

Theorem 5. *Given two positive integers $h,w \in \mathbb{Z}^+$, there exists a configuration C which is* weakly universal *for the set of shapes $U = \{u | u \subseteq \{1,\ldots,h\} \times \{1,\ldots,w\}\}$. This configuration has size $\mathcal{O}(hw)$ and uses $\mathcal{O}(hw)$ tilts to reconfigure into a configuration which weakly represents any shape $u \in U$.*

Proof. Following Theorem 4, we know that we can construct any rectangular binary pattern. By removing one of the labeled ("sticky") tile types, we can create a binary pattern using only one labeled type and the ε (sand) tiles. Then the only connected portion of the shape is the parts with the one labeled tile type. Hence, we can build a connected shape surrounded by "sand." This process results in a shape builder that is universal for the set of polyomino shapes that fit within an $h \times w$ bounding box. □



(a) (b) (c) (d)

Figure 4.5: The shape construction process at different intervals. (a) The first row of the shape being built, however, the shape does not need this row and so only sand is added to the row. (b-c) Partially built shape with several disconnected sections of the shape being built simultaneously. (d) The finished shape with sand encasing the shape.

Patterned Shapes By keeping any number of labeled tiles, we can also build any patterned connected shape.

Corollary 2. *Given three positive integers $h,w,k \in \mathbb{Z}^+$, there exists a configuration C which is* weakly universal *for the set of shapes $U = \{u | u \subseteq \{1,\ldots,h\} \times \{1,\ldots,w\}$, and each pixel in u has a label $x \in \{0,1,\ldots,k\}\}$. This configuration has size $\mathcal{O}(hwk)$ and uses $\mathcal{O}(hwk)$ tilts to reconfigure into a configuration which weakly represents any shape $u \in U$.*

Proof. By the same argument as Theorem 1, we can add k colors to our fuel chamber, causing the board size and number of tilts to scale with k. □

4.4.3 Additional Notes

Keeping the Undesirable Tiles One aspect of the constructor that may be undesirable or infeasible for some pracitcal implemenations is the idea of removing pieces from the board. Although not shown explicitly, the constructor (and subsequently the tilt sequences) can easily be modified to handle the removed tiles in numerous alternate ways.

- Trash. The easiest solution is to have a chamber that the tiles are thrown down into such that they become trapped (this can be achieved with a large enough chamber with one opening at the center of the top wall). The chamber must be large enough to accommodate pieces arbitrarily sticking together, and still prevent tiles from returning to the constructor.

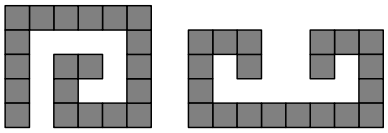
- Only Sand as Trash. Each labeled (colored) sticky tile can have its own fuel chamber still using sand to separate each piece. Each fuel chamber would be a $2n^2 \times 1$ vertical chamber that requires a unique tilt sequence to get a piece of fuel of that type out. The only trash is then sand, so the trash chamber would also only need to be $h \times w$ large.

- Reusing tiles. With the standard fuel tank, the unwanted tile could be routed down and around to the back of the fuel chamber to be reinserted. This recycling would only require an extra chamber of sand to put between labeled pieces that were recycled consecutively because the sand in between them was used in the shape.

Freeing the shape Another possible drawback of the constructor as presented is that the shape is trapped in the constructor itself. The constructor may be easily modified to allow for one shape to be released and another constructed. By enlarging the construction chamber to have a height of $2h$, and removing the top h tiles from the left wall, we can create an opening for the finished shape to leave the constructor. Thus, the *remove shape* tilt sequence $\langle N, W \rangle$ could be added, as it does not overlap with any of the pre-existing tilt sequences.

4.5 Drop Shapes

Next, we consider a class of shapes discussed in [10] which we refer to as *drop shapes*. A *drop shape* is any polyomino which is constructable by adding particles from any of the four cardinal directions *{N, E, S, W}* towards a fixed seed. For a formal definition of this class of shapes, see constructable polyominoes for the Tilt Assembly Problem (TAP) in [10]. Figure 4.6 shows an example of a valid and invalid drop shape. Here, we present a shape builder which is strongly universal for the set of drop shapes.



(a) Valid (b) Invalid

Figure 4.6: Drop Shapes examples. (a) This polyomino is buildable with the drop shape method, whereas (b) is a polyomino that is not a constructable drop shape. From a fixed single tile seed, it is not possible to build (b) by adding one tile at a time from a cardinal direction.

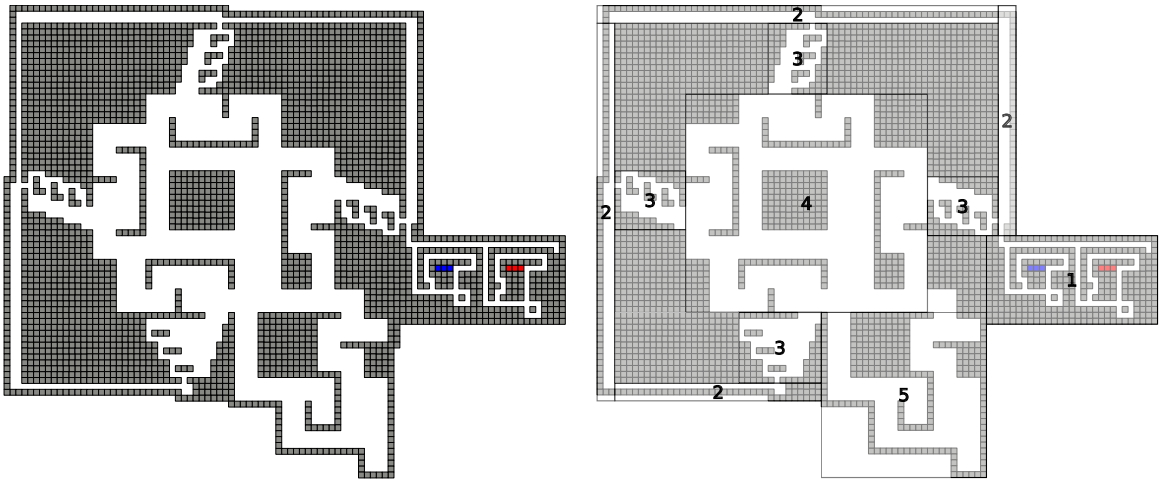
4.5.1 Universal Drop Shape Builder: Construction

Figure 4.7 is an example of our universal drop-shape builder for polyominoes fitting within a 4×4 bounding box. At a high-level, this construction works by following five phases, which are indicated by the labeled areas in Figure 4.7b.

1. Select a red or blue tile from the fuel chamber. Area 1 shows the two types of fuel that stick to each other. Here, we use a sequence to pick the one we want.

2. Choose which direction to add the tile from. We move the shape into the appropriate N, S, E, or W location and move the tile to the side we are adding from. As the area 2 labels show, we can move the new tile to the appropriate side of the shape.

3. Choose which column/row to add the tile on the shape. This example is for any 4×4 shape, so we choose columns (N/S) or row (E/W) to shoot the tile onto the shape where it will be added.



(a) Starting Configuration (b) Sections

Figure 4.7: (a) The universal drop-shape builder. (b) An overview of the parts of the drop shape builder. Area 1 is the *fuel chamber*, area 2 is the *selection chamber*, the area 3's are the *alignment chambers*, area 4 is the *construction chamber*, and area 5 is the *holding chamber*.

4. This area is where the shape is held when the new tile is added. There is a different holding area for each of the four directions.

5. In the last phase we move the shape to area 5 where it is held while we get the next tile to add. This holding chamber ensures the polyomino being built does not interfere with getting the next tile ready.

A full overview of the process with a flowchart and the necessary tilt sequences is shown in Figure 4.10a and Table 4.4, respectively.

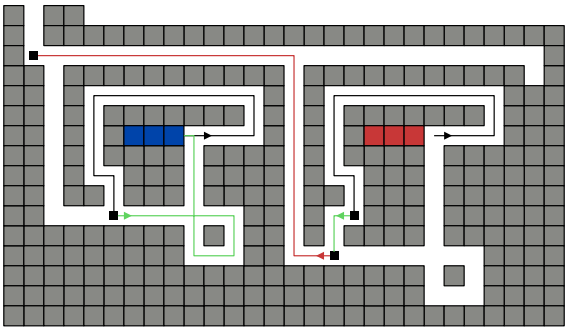


Figure 4.8: Tile selection gadget. Each tile is pulled out with the sequence $\langle E,N,W,S,E,S\rangle$, and stops at the first square. Then the left tile type (blue) is either pulled out of the gadget or put back in the storage area. This shows it being added back to the storage with $\langle E,S,W,N,W,S\rangle$. This sequence puts the next tile type (red) in a decision location. The red tile is selected with the sequence $\langle W,N,W,S,W,S\rangle$.

Fuel Chamber: Section 1 The fuel chamber consists of one or more tile reservoirs. Each reservoir contains a tile type with the same attachment label. The tilt sequence $\langle E,N,W,S,E,S\rangle$ extracts one fuel tile out of each reservoir. After extraction, we select which tile we want one at a time. For each extracted tile, we can either perform a tilt sequence to return the fuel tile to its reservoir, or to select it to be used in shape construction. Figure 4.8 shows an example of selecting the second type of fuel. Once a fuel tile has been selected with the sequence $\langle W,N,W,S,W,S\rangle$, the storing tilt sequence must be performed for the rest of the fuel tiles that were not selected. To store a fuel tile the tilt sequence is $\langle E,S,W,N,W,S\rangle$. Note that the position of a selected fuel piece remains the same after an application of the storage sequence.

On all iterations after the first, we must also consider the position of the polyomino/assembly which is being constructed. Before executing the tilt sequence to extract a tile, the polyomino is located in the northmost eastmost notch of the holding chamber. A quick observation will note that the tilt sequence required to traverse the holding chamber is identical to the tile extraction sequence. Also note that the position of the assembly after an extraction, like the selected fuel piece, is invariant after each application of the storage sequence. When the extracted fuel tile is to enter the selection chamber, the assembly will be ready to enter the construction chamber.

Selection Chamber: Section 2 After selecting the fuel tile and storing the rest of the extracted tiles, we now move the fuel tile into the selection chambers. This stage of the construction simply selects the direction of attachment. After tilting $\langle N \rangle$, the fuel enters the eastern selection chamber and the assembly enters the construction chamber. To select an attachment from the east, perform the tilt sequence $\langle E, S, W \rangle$ (notice that this will position the assembly to the west of the eastern alignment chamber), else tilt $\langle W \rangle$ to select the next direction. After this western tilt, to select an attachment from the north, perform the tilt sequence $\langle N, E, S \rangle$ (notice this also positions the assembly below the northern alignment chamber), else tilt $\langle S \rangle$ to select the next direction. Now, after this southern tilt, to attach from the west perform the tilt sequence $\langle W, N, E \rangle$ (this too positions the assembly to the east of the western alignment chamber), else to select the last direction tilt $\langle E \rangle$. At this point, the fuel must be attached from the south. The sequence $\langle S, W, N \rangle$ will prepare the fuel tile for an attachment from the south (while also positioning the assembly to the north of the southern alignment chamber).

Alignment Chambers: Sections 3 After an attachment direction has been chosen, the tile enters a gadget to select the row/column of the polyomino to target with the new tile. Figure 4.9 shows an example for a northern selection gadget for a 4×4 polyomino. The tilt sequence to align with a particular location varies with respect to the direction of attachment. If the rightmost pixel of a northern attachment is chosen, the tilts $\langle W, S \rangle$ would be performed. Each successive column is chosen by performing the sequence $\langle E, S, W, S \rangle$. This sequence is repeated, each time moving tile further down the gadget, until the desired location is reached, at which point the attachment sequence $\langle W, S \rangle$ is executed. The alignment sequence for each attachment chamber does not affect the position of the assembly (it will remain positioned in the attachment chamber). Note the southern alignment gadget is slightly different but serves the same purpose. This difference is so the alignment and extraction sequences do not overlap.

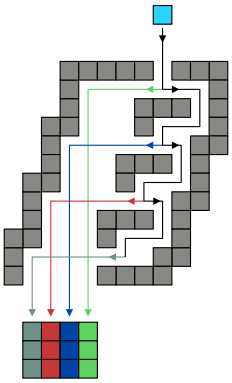


Figure 4.9: The column selection gadget for drop shapes. Assuming the shape to build is at a fixed location, this gadget allows any column to be selected to drop the new tile onto. The number of columns to drop from in this gadget determines the size of the shape we can build. Thus, this is for a drop shape within a 4×4 bounding box. This gadget is repeated on each of the four sides of the drop-shape constructor (with a slightly modified one on the south side).

Construction Chamber: Section 4 This central chamber is where the constructed assembly will be housed as it is being assembled. As we saw in the selection paragraph, the sequences required to position the assembly in front of a particular alignment chamber are the same as the sequences required to prepare the fuel piece to attach from that chamber. This, along with the alignment chamber process, allows us to pinpoint the attachment location of a fuel piece to the assembly.

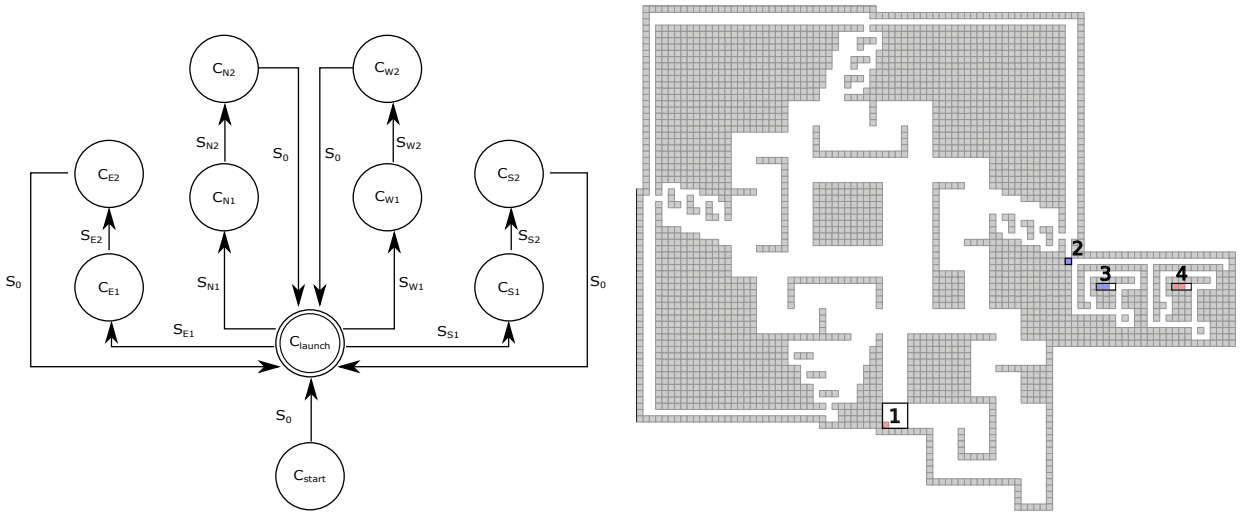
Holding Chamber: Section 5 Once the new tile has been added, we return the assembly to the holding chamber, which is where the assembly resides while the next fuel tile is being selected (Section 1). Once again, the target location in the holding chamber is the northmost westmost notch. As mentioned, the holding chamber's tilt sequences's are identical to that of the fuel chamber's. This ensures that we can select a fuel type without repositioning the assembly.

4.5.2 Universal Drop Shape Builder: Theorem and Proof

Theorem 6. *Given two positive integers $h, w \in \mathbb{Z}^+$, there exists a configuration C which is* strongly universal *for the set of drop shapes $U = \{u | u \subseteq \{1,\ldots,h\} \times \{1,\ldots,w\}\}$. WLOG, let $h \geq w$. This configuration has size $\mathcal{O}(h^2 w)$ and uses $\mathcal{O}(h^2 w)$ tilts to reconfigure into a configuration which strongly represents any shape $u \in U$.*

s_0	$\langle E,N,W,S,E,S\rangle + \langle E,S,W,N,W,S\rangle^i + \langle W,N,W,S,W,S\rangle + \langle E,S,W,N,W,S\rangle^j$
s_{E1}	$\langle N,E,S,W,S\rangle$
s_{N1}	$\langle N,W,N,E,S\rangle$
s_{W1}	$\langle N,W,S,W,N,E\rangle$
s_{S1}	$\langle N,W,S,E,S,W,N\rangle$
s_{E2}	$\langle S,W,N,W\rangle^j + \langle N,W,S,E,S,E,S\rangle$
s_{N2}	$\langle E,S,W,S\rangle^j + \langle W,S,E,N,E,S,E,S\rangle$
s_{W2}	$\langle N,E,S,E\rangle^j + \langle S,E,N,W,E,S,E,S,E,S\rangle$
s_{S2}	$\langle W,N\rangle^j + \langle E,N,W,S,W,N,E,S,E,S\rangle$

Table 4.4: The sequence of tilts used in the flowchart (Figure 4.10a). s_0 denotes the tile selection from $i+j+1$ different tile types and chambers. The sequence also places the current polyomino in the correct area for the next sequences (Figure 4.10b). For area 2 in Figure 4.7b, s_{E1}, s_{N1}, s_{W1}, s_{S1} represents choosing to attach the new tile from the east, north, west, or south side, respectively. Similarly, the alignment selection (area 3), from the east, north, west, or south side, is represented by s_{E2}, s_{N2}, s_{W2}, and s_{S2}, respectively. Note there may be up to $j-1$ columns.



(a) Flowchart

(b) Example C_{launch} Position

Figure 4.10: (a) A flowchart where each state represents a set of configurations and the symbols represent sequences that can move from one state to another. The sequences for each of the symbols is shown in Table 4.4. (b) An example configuration in the C_{launch} state. Configurations in C_{launch} always have the assembly located in box 1 at the rightmost bottom corner, the next fuel tile to be shot located in box 2, and all fuel pieces in the fuel chamber are in their proper reservoir pushed to the far left as depicted in box 3 and 4. A configuration *strongly* representing a drop shape $u \in U$ is in C_{launch} with no more tiles to launch and the fuel chamber empty.

Proof. This is a proof by construction. We begin with a configuration C where all chambers are empty except the fuel chambers. Following the process outlined in Figure 4.10a, we can extract the desired fuel piece, and the fuel piece can be moved from the alignment chamber to the shape via the construction chamber. We repeat this process to add single fuel pieces to the shape from any direction. After the fuel chambers are empty, we have generated a configuration C' from C by performing a series of tilts. Thus, $C \to_* C'$, and to show the transition from the starting configuration to the configuration that represents shape $u \in U$, we use the process shown in the flowchart of Figure 4.10a. □

4.6 PSPACE-complete Results

This section is a continuation of the work done by [7, 8, 9]. In their papers, they discuss the occupancy problem in a tilt-based system which asks whether a given location can be occupied by any tile on the board. We extend their line of investigation by introducing a set of new problems based on their work. We then go on to show PSPACE-completeness for the problems defined.

This section is divided into 4 main subsections. In Section 4.6.1, we present several natural decision problems for the tilt model. Section 4.6.2 is where we summarize the puzzle solvability problem from [12], which we use for our PSPACE-hardness reductions. In Section 4.6.3, we show that the relocation problem is PSPACE-complete, even when limiting all movable particles to 1×1's with a single 2×2. Finally, in Section 4.6.4 we show that the reconfiguration problem is PSPACE-complete, even with this same limitation on the size of the movable particles.

4.6.1 Problems in the Tilt Model

In this section we define the three tilt decision problems we consider.

Occupancy Problem The occupancy problem asks whether or not a given location can be occupied by any tile on the board. Formally, given a configuration $C = (B, P)$ and a coordinate $e \in B$, does there exist a tilt sequence such that $C \to_* C'$ where $C' = (B, P')$ and $\exists p \in P'$ that contains a tile with coordinate e?

Relocation The relocation problem asks whether a specified polyomino can be relocated to a particular position. That is, given a configuration, a polyomino within that configuration, and a translation of that polyomino, does there exist a sequence of tilts which moves the original polyomino to its translation?

Reconfiguration The reconfiguration problem asks whether a configuration can be reconfigured into another. Formally, given two configurations $C = (B, P)$ and $C' = (B, P')$, does there exist a tilt sequence such that $C \to_* C'$?

The occupancy problem was shown to be NP-Hard by [7, 8, 9] when only using 1×1 tiles. Their reduction also works to show NP-hardness for the relocation problem when only using 1×1 tiles. Additionally, they show that finding the *optimal* tilt sequence to reconfigure a board from one configuration to another is PSPACE-complete. In this section, we show that the relocation, reconfiguration, and occupancy problems are all PSPACE-complete. Our hardness proofs rely on a recent result in [12]. They indroduce the *puzzle solvability* problem, which asks whether or not a robot can traverse a system of specific types of gadgets, and show that it is PSPACE-Complete. We present a summary of this problem, and then show our reductions.

4.6.2 Puzzle Solvability

The model introduced in [12] is based on single-agent robot motion planning problems, where an agent navigates a given environment to a target destination. The authors introduce a general model of gadgets, of which the environment to be traversed may be comprised. We define several key components of these gadgets, as well as the specific types of gadgets themselves. We then state the puzzle solvability problem, which we later reduce from.

Gadget Components:

- Locations. A gadget consists of one or more *locations*, which are the points of entry and exit to the gadget.

- States. Each *state* s of the gadget defines a labeled directed graph on the locations, where a directed edge (a, b) with label s' means that the robot can enter the gadget at location a and

exit at location b, forcing the state to change to s'. The gadgets are defined by state spaces. A *state space* is a directed graph whose vertices are state and location pairs, where a directed edge from (s,a) to (s',b) means that the robot can traverse through the gadget from a to b if it is in state s, such that traversing will change the state of the gadget to s'. We use gadgets with at most two states.

- Toggle. A *toggle* is a tunnel that can only be traversed in a single direction as dictated by its state. Each toggle has 2 states dictating which direction a robot is allowed to traverse the tunnel. Tunnels are routes between two locations that the robot can traverse through. The state of the toggle gadget changes when the tunnel is traversed.

- Lock. A *lock* is a tunnel which has two states, locked and unlocked. The unlocked state allows bidirectional traversal. The locked state does not allow traversal whatsoever.

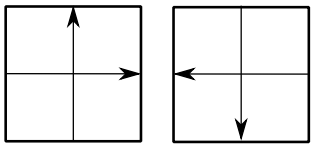
Gadgets and Puzzles:

- C2T. *Crossing 2-Toggle* is a gadget that has two toggle tunnels perpendicular to each other. Traversing either tunnel causes the gadget's state to change, which means the state (or direction) of both tunnels are changed (or reversed). Figure 4.11a shows a representation of both states of the C2T gadget.

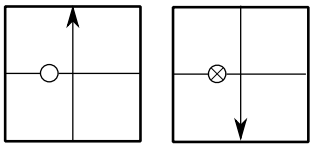
- CTL. The *Crossing Toggle-Lock* gadget also has two perpendicular tunnels. One tunnel is a toggle, and the other is a lock. Traversing the toggle switches the lock between its locked and unlocked state. Figure 4.11b shows the representation of both states of the CTL gadget.

- Puzzle. A *puzzle* is a problem posed as a system of interconnected gadgets, their initial states, the wires connecting them, and the robot's start and goal location. A puzzle is said to be solvable if there is a path from the start location to the goal location using only moves allowed by the wires and gadgets.

Puzzle Solvability Problem This problem simply asks if a given puzzle of gadgets is solvable. In other words, does there exists a sequence of moves that relocates the robot from its



(a) Crossing 2-Toggle (C2T)



(b) Crossing Toggle-Lock (CTL)

Figure 4.11: (a) The two states of the crossing 2-toggle gadget. Robot traversal through either tunnel toggles the gadget between these two states. (b) The two states of the crossing toggle-lock gadget. Robot traversal through the directed tunnel toggles the gadget between these two states (locked and unlocked).

start location to its goal location? If the puzzle consists of only C2T gadgets, we refer to this as the C2T puzzle solvability problem. The same goes for the CTL gadget.

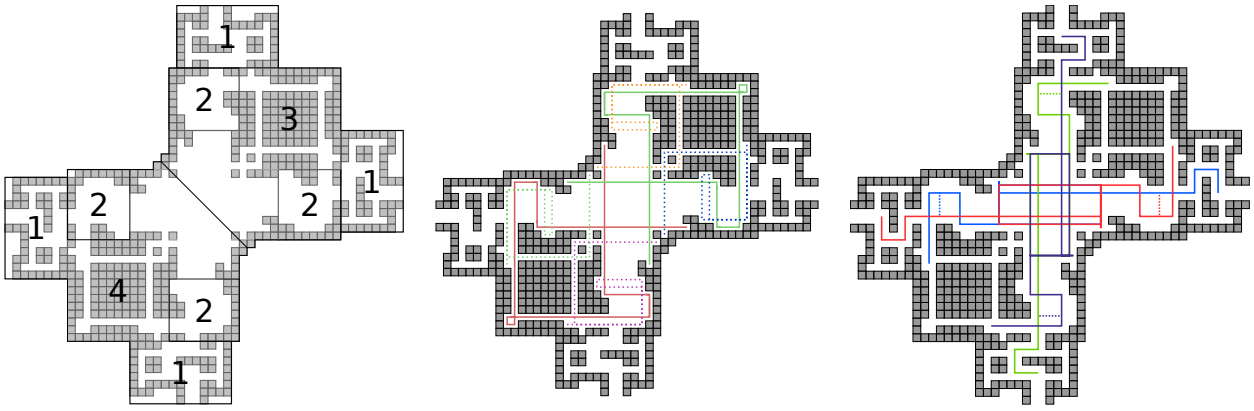
Theorem 7. *The C2T puzzle solvability problem and the CTL puzzle solvability problem are both PSPACE-complete.*

Proof. This was shown in Corollary 5.2 from [12]. □

4.6.3 Relocation

The work from [9] implies that the relocation problem is NP-hard when using only 1×1 movable particles. In this section, we show that the problem is PSPACE-complete when adding a single 2×2 polyomino.

Relocation Gadget Preliminaries Here we present a gadget in the tilt model that behaves like the C2T gadget described above. Figure 4.12 shows an overview of the gadget. Figure 4.12a shows the basic sections of the gadget which we describe below. The main idea is that a 1×1 tile (referred to as the *state tile*) is confined to one of two different paths within the gadget and a 2×2 polyomino (referred to as the *robot polyomino*) may only traverse the gadget in a specified direction if the state tile is in the correct corresponding path. To implement this dependency, we created locations within the gadget where the state tile and robot polyomino need to use each other's geometry to traverse the gadget. Figures 4.12b and 4.12c show the paths of the state tile and robot polyomino, respectively. We note some fundamental properties of the C2T gadget which we recreate in the full-tilt model. Figure 4.13 shows an example of a pathway not reachable by the



(a) Sections of Gadgets (b) Paths of the State Tile (c) Paths of the Robot Polyomino

Figure 4.12: (a) Relocation sections where 1) represents entrance/exits locations, 2) represents areas where the robot polyomino becomes stuck if unassisted by the state tile, and 3) and 4) represent the *NE* and *SE* state tile areas. State and robot paths are shown in (b) and (c), where the state tile is stuck on the solid lines and traverses through the dotted lines when changing states, and where the robot polyomino travels through one location to the next through the solid lines, unassisted by the state tile, or traverses the dotted lines if assisted by the state tile.

robot polyomino without the geometric assistance of a state tile. This is the same as the C2T gadget not allowing the robot's traversal through an exit location. In Figure 4.13 we depict an example of a state tile confined to a pathway that is moved into another with the assistance of the robot polyomino. This depicts one of two different *states* that the state tile can be in. This mimics the C2T's state changing function. By combining these elements we are able to create a complex gadget which is able to allow a robot to traverse through it only if the gadget is in the correct state.

Robot Polyomino The *robot polyomino* is a 2×2 polyomino that traverses through a puzzle. Figure 4.12c shows all paths of the robot in a gadget. The dotted lines show its path without the help of the state tile.

State Tile The relocation gadget has a *state tile*, which is a single tile trapped inside the gadget. The state tile can freely move in one of the solid lines shown in Figure 4.12b, where the dotted lines depict a pathway traversable by the state tiles only with the assistance of the traversing robot polyomino's geometry.

Relocation Gadget Our Relocation gadget consists of openings at its four cardinal directions which we will call N,E,S,W. The internal geometry is diagonally symmetrical and allows

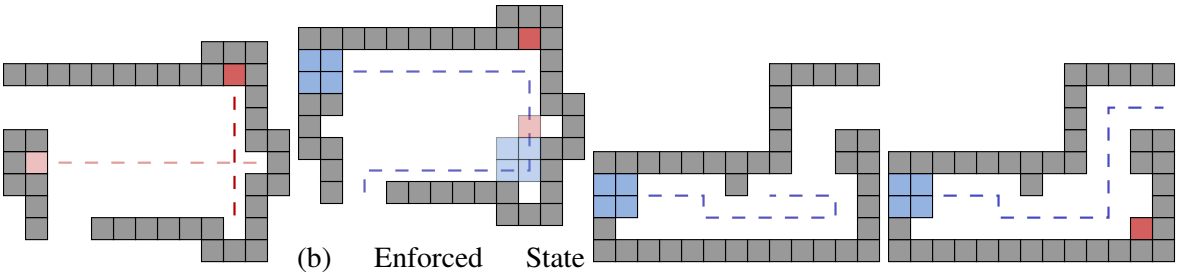


Figure 4.13: Relocation Properties. (a) A gadget with states 1 (dark) and 2 (light). (b) The robot-traversal "toggles" the gadget. (c) A gadget cannot be traversed by the robot alone. (d) The same gadget being robot-traversed with the help of the state tile.

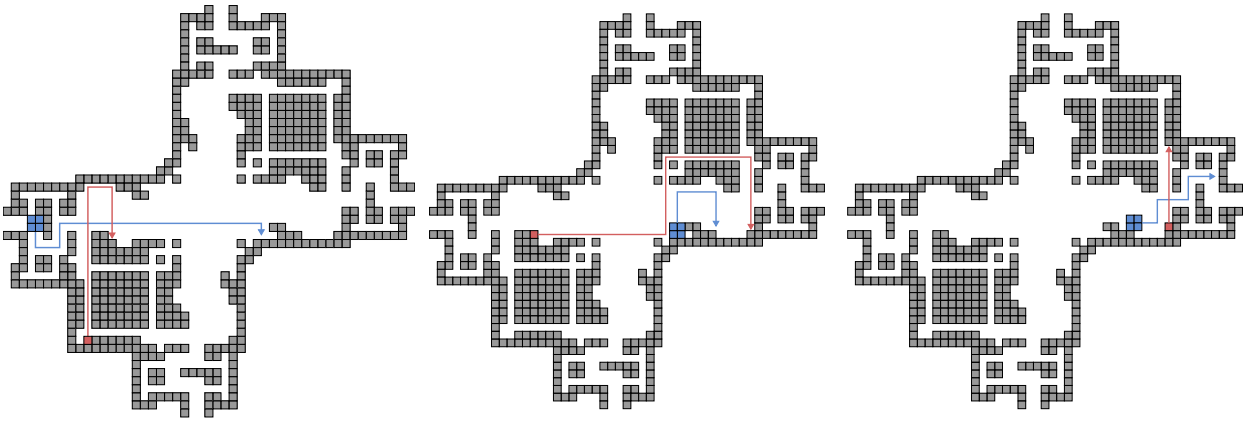
for traversal between its openings. Wires are made with 2-tile wide hallways attached at gadget openings. A high-level overview of the four relocation gadget sections is shown in Figure 4.13.

Entrance/Exit Chambers Marked as section 1 in 4.12a, the *entrance/exit chambers* represent *locations* in a C2T gadget. These chambers do not enforce any behavior or move sequence constraints on a robot polyomino, letting it move in or out of the chamber with no difficulty. However, these chambers have spaces on its sides designed to keep the state tile within the gadget. Once a state tile becomes stuck in the spaces, the gadget becomes inoperable. These chambers change from entrance to exit chambers depending on the state of the gadget, where if the state tile is in the *NE* side of the gadget, the *NE* chambers become entrance chambers and the *SW* side chambers become exit chambers and vice versa. These different states correspond to the depicted states of C2T gadgets in Figure 4.11.

Assistance Chamber Marked as section 2 in 4.12a, an *assistance chamber* is where the state tile meets the robot polyomino to assist it in its traversal through the gadget. The robot polyomino can reach the assistance chamber opposite of where it entered from. The only way a robot polyomino can move through an assistance chamber is if the state tile is in the correct path. See Figure 4.14 for an example travesal. If a robot polyomino enters a gadget whose state tile is in the opposite side, the robot polyomino becomes stuck inside the gadget. This forces the robot to enter through the correct entry points.

State Chambers Sections 3 and 4 in 4.12a are the *state chambers* of the relocation gadget. They store the state tiles and dictate the gadget's state. We refer to these states as the NE state/path and the SW state/path. See Figure 4.12b for the possible paths of the state tile in its two states. As shown in the figure, the $NE(SW)$ chamber allows a path for the state tile to move freely between its north (south) side and east (west) side.

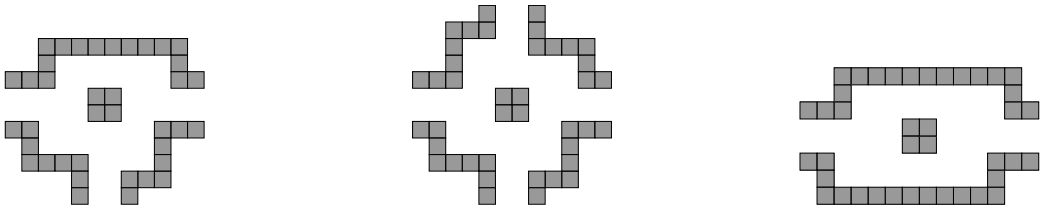
Directed Tunnels We say a *directed tunnel* (a,b) exists in our relocation gadget if a sequence of tilts exists such that we can relocate our robot polyomino from location a to location b. For example, the directed tunnel (W,E) exists if our robot polyomino can travel from the west entrance of our relocation gadget to the east entrance.



(a) $\langle S,E,N,E,S\rangle$ (b) $\langle E,N,E,S\rangle$ (c) $\langle E,N,E,N,E\rangle$

Figure 4.14: Example of a traversing sequence and state change when a gadget is in the state from Figure 4.11a. The robot polyomino enters the gadget in (a) and performs the sequence shown. The robot geometrically assists the state tile to traverse around in (b), and in (c) the state tile assists the robot polyomino via its geometry to exit through the opposite location. In the end the robot would have traversed the gadget, and the state tile would have gone from section 4's red path in Figure 4.12b to section 3's green path; Therefore, the gadget was toggled.

Intersections To allow robots to change directions in the paths between gadgets, we must have geometry to stop the robot and then choose which direction it will proceed. We place a 2×2 blocking piece of geometry in the middle of the wire and expand the surrounding area to allow the robot to make traversing decisions. Examples of 3-way and 4-way intersections are shown in Figures 4.15a and 4.15b, respectively.



(a) 3-way intersection (b) 4-way intersection (c) Intermediate Wire

Figure 4.15: Intersections of tunnels. The geometric block in the center stops the robot and allows it to choose any of the tunnels to move through. (a) Three tunnels intersecting. (b) A four-way intersection. (c) The intermediate wire used in a system of reconfiguration gadgets to "reset" the state tiles.

Relocation is PSPACE-complete Here, we show the hardness reduction for the relocation problem. The central idea of our reduction is that our relocation gadget has the same functionality as the C2T gadget. In order to prove this, we prove a series of lemmas (using a brute-force proof-by-computer) about how our gadget works. This equivalent functionality is then used towards the main theorem of this section. We begin with a simple recursive algorithm which, given an initial configuration, creates a tree of all reachable configurations.

Algorithm 1 Create Configuration Tree

```
function CREATETREE(startConfig)
    Create a new node for startConfig.
    Add startConfig to the list of visited configurations.
    for all d in {N,E,S,W} do
        Create new configuration by tiliting startConfig in direction d.
        if the new configuration is not in the list of visited configurations then
            Create a child node for direction d and call createTree on the new configuration.
```

Lemma 2. *Given relocation gadget configurations C and C', there exists a node for C' in the configuration tree created by* createTree*(C) if and only if $C \rightarrow_* C'$.*

Proof. Since algorithm 1 recursively calls itself until the base case where all four child configurations have already been visited, it is easy to see that all reachable configurations are generated by this algorithm. □

Forest Generation For the following lemmas, we use a brute-force proof-by-computer approach in which we generate a forest of trees (each created by calling algorithm 1) from each of

the configurations where the robot polyomino is located in any of the four entrance/exit chambers. In other words, a tree is generated for every combination of robot polyomino entrance/exit location and all possible state tile locations. We can easily modify algorithm 1 to flag nodes as north, east, south, or west nodes if their corresponding configuration has the robot polyomino located at the north, east, south, or west entrance/exit chambers. Sourcecode:

Lemma 3. *Directed tunnels* (N,S) *and* (E,W) *exist in a relocation gadget if and only if the gadget is in the NE state.*

Proof. To prove this, we show that the existance of directed tunnels (N,S) and (E,W) implies that the gadget is in the *NE* state, and that the gadget being in the *NE* state implies the existance of directed tunnels (N,S) and (E,W). By Lemma 2, we see that the (N,S) / (E,W) directed tunnels exist for any north (east) configuration which is the root of a tree that contains a south (west) node (i.e. there is a tilt sequence to move the robot polyomino from the north entrance to the south). For the first implication, we search the brute-force forest and verify that all trees which have a north (east) root and contain a south (west) node also begin with the state tile at some location in the *NE* path. For the second implication, we search the brute-force forest and verify that all trees which have a north (east) root and begin with the state tile at some location in the *NE* path contain a south (west) node. □

Lemma 4. *Directed tunnels* (S,N) *and* (W,E) *exist in a relocation gadget if and only if the gadget is in the SW state.*

Proof. We perform a similar forest-searching process as Lemma 3 and show a similar implication for directed tunnels (S,N) / (W,E) and gadget state *SW*. □

Lemma 5. *Directed tunnels* $(N,E),(N,W),(S,E),(S,W),(E,N),(E,S),(W,N),$ *and* (W,S) *do not ever exist in a relocation gadget (regardless of its state).*

Proof. For each directed tunnel $(a,b) \in \{(N,E),(N,W),(S,E),(S,W),(E,N),(E,S),(W,N),(W,S)\}$, we search the forest and verify that no tree exists which has root flagged for direction a and contains a node flagged for direction b. □

Lemma 6. *If a directed tunnel in a relocation gadget is traversed, the state of the gadget is toggled.*

Proof. From Lemma 5, we can deduce that the only directed tunnels which may exist in a relocation gadget are (N,S), (E,W), (S,N), and (W,E). Consider directed tunnel (N,S). We search the forest and find all trees which have a north root and contain a south node. By Lemma 3 we know that these trees have root configurations where the state tile is in the *NE* path. We verify that all south nodes in these trees represent configurations in which the state tile is in the *SW* path. We perform equivalent verifications for the remaining directed tunnels. □

Lemma 7. *The relocation gadget correctly implements the behavior of the C2T gadget.*

Proof. By Lemmas 3 and 4, we see that robot traversal through our relocation gadget is dictated by what state the gadget is in. Lemma 5 shows that this traversal must be in a straight line. Lemma 6 shows that robot traversal forces the gadget to change state. Thus, our relocation gadget has the same functionality as the C2T. □

Theorem 8. *The relocation problem is PSPACE-complete. Moreover, it remains PSPACE-complete when polyominoes are all* 1×1 *squares with a single* 2×2 *square.*

Proof. First, we observe that this problem is in PSPACE. To see this, consider a directed graph $G = (V,E)$ where each vertex is a configuration on the board B and each edge $e_{i,j} = (v_i, v_j)$ connects two vertices if there exists one tilt that reconfigures v_i to v_j. Clearly, a nondeterministic search of this graph will yield the answer in polynomial space, implying membership in NPSPACE. Since NPSPACE = PSPACE, we get membership in PSPACE.

We now show PSPACE-hardness by a reduction from the C2T puzzle solvability problem from [12]. Given an instance of a C2T puzzle, use the tools described above to create a tilt model configuration as follows: For each C2T gadget in the puzzle, create a relocation gadget such that the initial state of the relocation gadget matches the initial state of the C2T gadget. For each straight wire in the puzzle, create a 2-tile wide tunnel in the tilt configuration. For each 3/4-way wire intersection in the puzzle, create a 3/4-way tunnel intersection in the tilt configuration. For the goal location of the puzzle, create a 2×2 goal chamber in the the tilt configuration.

Lemmas 3 and 4 show that there exists a way for the robot polyomino to traverse our relocation gadget. This, along with the fact there clearly exists a move sequence to traverse each wire tunnel/intersection, shows that if the given C2T puzzle instance has a solution, then there exists a tilt sequence which relocates the robot polyomino to the goal chamber. Thus, a solution for the puzzle solvability problem implies a solution to the relocation problem.

Now, consider the case where the given C2T instance is not solvable. Since Lemma 7 shows that an individual relocation gadget has the same functionality as an individual C2T gadget, the only way that our relocation problem could be solvable is if the functionality of our relocation gadget changed when placed in a system of gadgets. The only way one relocation gadget could possibly influence another is if a state tile could leave one gadget and enter another. After generating the brute-force forest with a modified algorithm 1 (flagging any configurations in which the state tile exits the reconfiguration gadget) we search the forest and verify that no such configurations exist. Thus, no solution for the C2T puzzle solvability problem implies no solution for the relocation problem. By Theorem 7, we see that the relocation problem is PSPACE-complete.

□

Corollary 3. *The occupancy problem is PSPACE-complete. Moreover, it remains PSPACE-complete when polyominoes are all* 1×1 *squares with a single* 2×2 *square.*

Proof. This follows from the same construction that was used in Theorem 8. Since we know that no state tile can ever leave a relocation gadget, the only polyomino which could occupy the goal location is the robot polyomino. Thus, the same implications hold for the C2T puzzle solvability problem and the occupancy problem. □

4.6.4 Reconfiguration

In [9], it was shown that computing a shortest sequence of tilts required to transform one configuration into another is PSPACE-complete. In this section, we show that even the problem of determining if a reconfiguration sequence exists between two given configurations is PSPACE-complete.

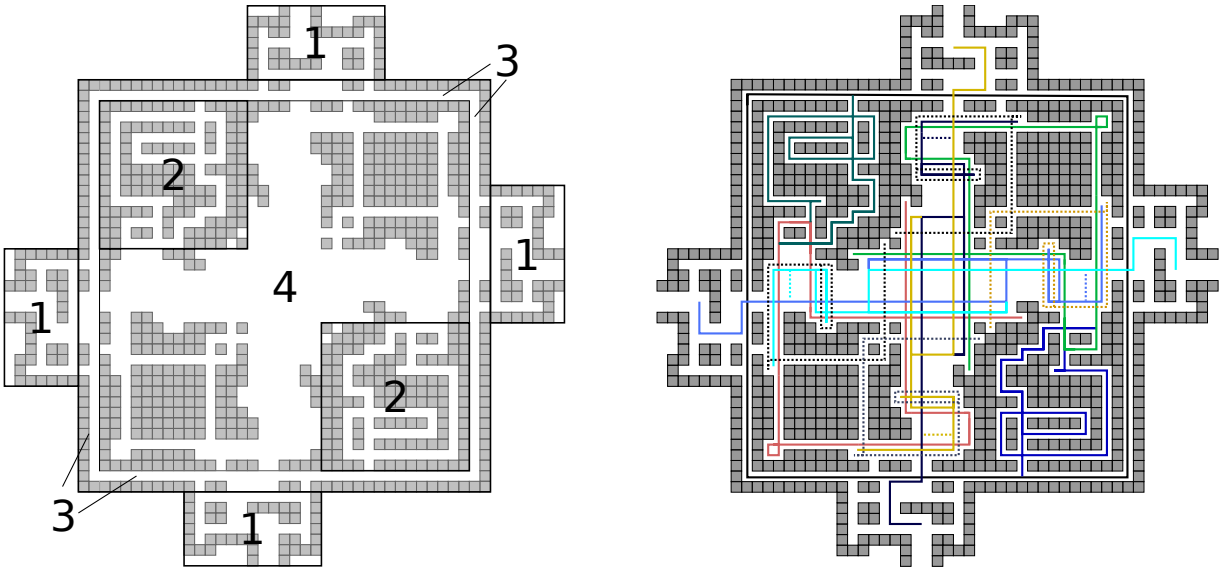
Reconfiguration Gadget Model Preliminaries For the reconfiguration problem we must provide a unique final configuration, as opposed to just a final location of a single polyomino. The previous gadget is insufficient for this problem as there is generally not a unique final configuration for all successful traversals, and it would not be polynomial time computable even in the cases that it were. The issue is that we would not know the state of each of the gadgets, and thus not know the locations of the state tiles throughout the system. We address this issue by extending the relocation gadget to allow for a final tilt sequence that moves all state tiles of all gadgets (regardless of the state) into one unique position in each gadget, thereby providing a polynomial time computable target configuration for the reduction.

Reconfiguration Gadget The reconfiguration gadget is a relocation gadget with additional geometry. Each state tile has additional pathways that lead to an inescapable chamber on the perimeter of the gadget. The robot polyomino otherwise uses the same paths as in the relocation gadget.

Reconfiguration Ring Each reconfiguration gadget has a *reconfiguration ring* located on its perimeter reachable by the state tiles only. These hallways help solve the reconfiguration problem as they convert all gadgets to one global unique configuration. To insert all state tiles to this ring, we can move the state tile through the new geometry (depicted in Figure 4.17a) into the reconfiguration ring, and render all gadgets inoperable.

Restarting Positions We define *restarting positions* (depicted by the solid state tiles in Figure 4.17a) as the locations where we initialize all state tiles of the same state. Applying a global motion signal will ensure that all state tiles (in the same state) will be in the same position at all times.

Intermediate Wire An *intermediate wire* (Figure 4.15c) is a small chamber gadget placed on wires between every pair of reconfiguration gadgets to ensure the state tiles will not enter the reconfiguration ring. The intermediate gadget gives the robot enough room to make tilts in all directions. After every reconfiguration gadget traversal, the robot can move freely in the intermediate



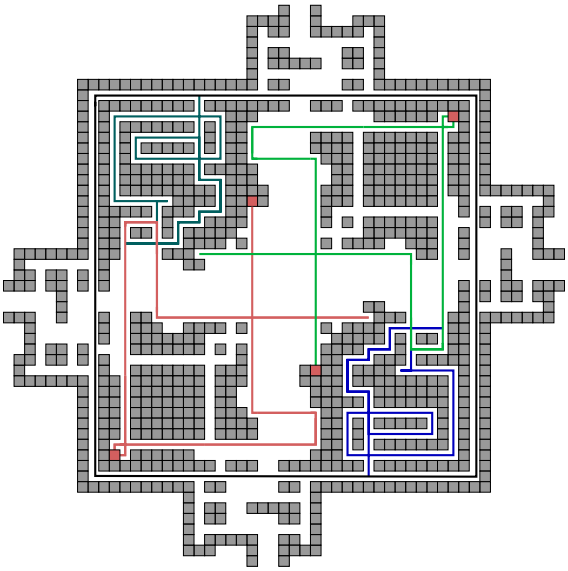
(a) Reconfiguration Gadget Sections (b) Reconfiguration Gadget Positions

Figure 4.16: (a) Sections 1 are the *entrance chambers*, sections 2 are the *pre-reconfiguration tunnels*, section 3 is the *reconfiguration tunnel*, section 4 is the *relocation gadget*. The solid lines depict the pathways the state tile and robot polyomino are free to move through, and the dotted lines depict the pathways which are only accessible through cooperation between the state tile and robot polyomino.

wire and reposition the state tiles to their restarting positions.

Reconfiguration is PSPACE-complete Using the same proof by computer described above, we show that in a system of reconfiguration gadgets the state tiles can be placed in their restarting positions by using the intermediate wire. We do this to avoid accidentally placing a state tile in the reconfiguration ring, caused by performing tilt sequences that move the robot around the system. We show that by placing intermediate wires before every reconfiguration gadget, we can traverse through one gadget to the next and place the state tiles back in their restarting positions. Therefore, everytime the robot enters a gadget the state tiles will be in one of the two restarting positions we depict in Figure 4.16b. When the robot has been relocated to its appropiate location, we can then proceed to move all state tiles to their reconfiguration ring. This converts all of the gadgets into one configuration.

Lemma 8. *For every system of gadgets there exists a sequence of tilts that allows the robot to traverse the gadget system without moving the state tile into the reconfiguration ring.*



(a) Reconfiguration state tile paths.

Figure 4.17: When the robot has reached its goal location, all the state tiles will be in one of the two state paths (red or light green). The user could then maneuver all state tiles through the paths (dark green and blue) and place them inside the reconfiguration ring. Doing so will convert all gadgets to a global configuration.

Proof. There exists a set of tilt sequences that allow the robot polyomino to traverse the gadget while preserving the state tile restarting positions defined above. Thus, depending on the direction traveled, every state tile will be in a specific location depending on the gadget's state as the polyomino is exiting the gadget, or trivially be one tilt away. This is true not only for the gadget being traversed, but also for every gadget in the system. This ensures that while traversing through the hallways no state tiles are forced into a reconfiguration ring. From the top-right and bottom-left state tile restarting positions, a total of 13 tilts are required to permanently trap the state tile in the reconfiguration ring, and tilts must be performed in a clockwise order to progress the tile through the reconfiguration hallway. The maximum number of hallways to connect two gadgets or a gadget to an intersection is five. Thus, at most five tilts are needed in a hallway traversal. This means no state tile can enter the reconfiguration ring while only traversing hallways. The intersections are designed to use counter-clockwise tilts in choosing a direction, and therefore does not allow progression through the chamber. Finally, since the entrance to every gadget is preceeded by an intermediate wire, we can perform tilts in these wires to reposition the state tiles to their optimal positions. In the

worst case scenario all state tiles are located in their gadget's reconfiguration hallway, however, we have shown that even from this scenario there is a tilt sequence that will reposition all state tiles into their optimal starting positions. □

Theorem 9. *The reconfiguration problem is PSPACE-complete. Moreover, it remains PSPACE-complete when polyominoes are all* 1×1 *squares with a single* 2×2 *square.*

Proof. Since the reconfiguration gadget shares the same structure and properties as the relocation gadget, it also behaves as a C2T gadget. We use an exhaustive computer simulation, as in the proof of the relocation gadget, to ensure the gadget works as described. Lemma 8 shows that solving the relocation problem is feasible using just reconfiguration gadgets. Therefore, after solving the relocation problem using reconfiguraton gadgets, we can move all state tiles into their reconfiguration ring. Doing so will move all state tiles into a specific location within each gadget that is specified as part of our final configuration D, which also has the required location of the robot. □

4.7 Future Work

There are a number of open questions and directions for future work that stem from our results. In the area of complexity, we have shown that relocation, occupancy, and reconfiguration problems are PSPACE-complete if both 1×1 tiles and a single 2×2 block are considered.The complexity of this problem is only known to be NP-hard if restricted to 1×1 movable blocks [8]. In [8] the authors showed that 1×1 blocks have substantial limitations such as the impossibility of implementing a certain type of fanout gate. This might be evidence that this variant is not PSPACE-complete. However, the existence of necessarily exponentially long tilt sequences for reconfiguration shown in [8] provides some evidence that the problem does not lie within the class NP. An alternate set of questions involve the trade off between of number of larger than 1×1 tiles and the board complexity. Does the problem remain PSPACE-complete when both are restricted, or does the problem become simpler, perhaps allowing for a polynomial time solution?

Another direction of future work is to explore strong universal configurations for classes

beyond the "drop" class. One plausible extension might entail building drop shapes as a subroutine and combining them together in a hierarchical "staged assembly" fashion. Another could be to employ a "sand sifter" along with this staged assembly as a part of the pattern builder. Related to this is the consideration of speeding up the assembly process by attempting to build distinct portions of a target shape in parallel. This approach has been recently explored [20], but has not been considered in the context of building universal configurations.

One interesting direction for future work involves relaxing the constraint in which polyominos slide *maximally* until stopped. Instead, polyominos could slide a fixed amount per tilt, or travel at some particular speed. This strengthens the model significantly, but is motivated by a number of practical proposed implementations. What interesting added capabilities does this modification allow, and how do complexity questions change for this model?

Acknowledgments

We would like to thank Andrew Winslow for pointing us to the literature on the C2T gadget, which greatly simplified our proof of the complexity of the reconfiguration and relocation problems. We would also like to thank Aaron Becker for providing feedback on the final presentation of this work. Finally, we would like to thank the anonymous reviewers of the preliminary conference version of this paper for their thorough, careful, and constructive feedback.

BIBLIOGRAPHY

[1] *Atomix*. Thalion Software, 1990.

[2] *Mega maze*. Phillips Media, 1995.

[3] *Tilt: Gravity fed logic maze.*, (2012).

[4] J. BALANZA-MARTINEZ, D. CABALLERO, A. CANTU, L. GARCIA, A. LUCHSINGER, R. REYES, R. SCHWELLER, AND T. WYLIE, *Full tilt: Universal constructors for general shapes with uniform external forces*, in 30^{th} Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'19, 2019.

[5] A. BECKER, Y. OU, P. KIM, M. J. KIM, AND A. JULIUS, *Feedback control of many magnetized: Tetrahymena pyriformis cells by exploiting phase inhomogeneity*, in 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2013, pp. 3317–3323.

[6] A. T. BECKER, *Parallel self-assembly of polyominoes under uniform control inputs*. https://www.youtube.com/watch?v=G93H1Tecj-w, 2018.

[7] A. T. BECKER, E. D. DEMAINE, S. P. FEKETE, G. HABIBI, AND J. MCLURKIN, *Reconfiguring massive particle swarms with limited, global control*, in Algorithms for Sensor Systems, 2014, pp. 51–66.

[8] A. T. BECKER, E. D. DEMAINE, S. P. FEKETE, J. LONSFORD, AND R. MORRIS-WRIGHT, *Particle computation: complexity, algorithms, and logic*, Natural Computing, (2017).

[9] A. T. BECKER, E. D. DEMAINE, S. P. FEKETE, AND J. MCLURKIN, *Particle computation: Designing worlds to control robot swarms with only global signals*, in 2014 IEEE Inter. Conf. on Robotics and Automation (ICRA), 2014, pp. 6751–6756.

[10] A. T. BECKER, S. P. FEKETE, P. KELDENICH, D. KRUPKE, C. RIECK, C. SCHEFFER, AND A. SCHMIDT, *Tilt Assembly: Algorithms for Micro-Factories that Build Objects with Uniform External Forces*, in 28th International Symposium on Algorithms and Computation (ISAAC 2017), Y. Okamoto and T. Tokuyama, eds., vol. 92 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2017, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 11:1–11:13.

[11] BRIO, *Labyrinth game.*, 1946.

[12] E. D. DEMAINE, I. GROSOF, J. LYNCH, AND M. RUDOY, *Computational complexity of motion planning of a robot through simple gadgets*, in 9th International Conference on Fun with Algorithms, FUN 2018, June 13-15, 2018, La Maddalena, Italy, 2018, pp. 18:1–18:21.

[13] D. DOTY, *Theory of algorithmic self-assembly*, Communications of the ACM, 55 (2012), pp. 78–88.

[14] C. JUNG, P. B. ALLEN, AND A. D. ELLINGTON, *A simple, cleated dna walker that hangs on to surfaces*, ACS Nano, 11 (2017), pp. 8047–8054. PMID: 28719175.

[15] P. KELDENICH, S. MANZOOR, L. HUANG, D. KRUPKE, A. SCHMIDT, S. P. FEKETE, AND A. T. BECKER, *On designing 2d discrete workspaces to sort or classify polynminoes*, in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct 2018, pp. 1–9.

[16] I. S. KHALIL, H. ABASS, M. SHOUKRY, A. KLINGNER, R. M. EL-NASHAR, M. SERRY, AND S. MISRA, *Robust and optimal control of magnetic microparticles inside fluidic channels with time-varying flow rates*, International journal of advanced robotic systems, 13 (2016), p. 123.

[17] S. MARTEL, O. FELFOUL, J.-B. MATHIEU, A. CHANU, S. TAMAZ, M. MOHAMMADI, M. MANKIEWICZ, AND N. TABATABAEI, *Mri-based medical nanorobotic platform for the control of magnetic nanoparticles and flagellated bacteria for target interventions in human capillaries*, The International journal of robotics research, 28 (2009), pp. 1169–1182.

[18] MOJANG, *Minecraft*, 2009.

[19] M. J. PATITZ, *An introduction to tile-based self-assembly and a survey of recent results*, Natural Computing, 13 (2014), pp. 195–224.

[20] A. SCHMIDT, S. MANZOOR, L. HUANG, A. T. BECKER, AND S. FEKETE, *Efficient parallel self-assembly under uniform control inputs*, IEEE Robotics and Automation Letters, (2018), pp. 1–1.

[21] R. SINHA, G. J. KIM, S. NIE, AND D. M. SHIN, *Nanotechnology in cancer therapeutics: bioconjugated nanoparticles for drug delivery*, Molecular cancer therapeutics, 5 (2006), pp. 1909–1917.

[22] D. WOODS, *Intrinsic universality and the computational power of self-assembly*, Philosophical Transaction of the Royal Society A, 373 (2015).

[23] D. ZHANG AND G. SEELIG, *Dynamic dna nanotechnology using strand-displacement reactions*, 3 (2011), pp. 103–13.

[24] C. ZHOU, X. DUAN, AND N. LIU, *A plasmonic nanorod that walks on dna origami*, Nature Communications, 6 (2015), pp. 8102–8102. 26303016[pmid].

CHAPTER V

HIERARCHICAL SHAPE CONSTRUCTION AND COMPLEXITY FOR SLIDABLE POLYOMINOES UNDER UNIFORM EXTERNAL FORCES

Collaborators: Jose Balanza-Martinez, David Caballero, Angel A. Cantu, Mauricio Flores, Timothy Gomez, Rene Reyes, Robert Schweller, and Tim Wylie **My contribution:** I was responsible for Lemma 9, Lemma 10, and Theorem 10: the algorithm for determining a polyomino's membership in our hierarchy. The text was written by all. **This chapter was published as:** Jose Balanza-Martinez, David Caballero, Angel A. Cantu, Mauricio Flores, Timothy Gomez, Austin Luchsinger, Rene Reyes, Robert Schweller, and Tim Wylie, "Self-Assembly of Any Shape with Constant Tile Types using High Temperature," In Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms, 2020.

5.1 Abstract

Advances in technology have given us the ability to create and manipulate robots for numerous applications at the molecular scale. At this size, fabrication tool limitations motivate the use of simple robots. The individual control of these simple objects can be infeasible. We investigate a model of robot motion planning, based on global external signals, known as the tilt model. Given a board and initial placement of polyominoes, the board may be tilted in any of the 4 cardinal directions, causing all slidable polyominoes to move maximally in the specified direction until blocked.

We propose a new hierarchy of shapes and design a single configuration that is *strongly universal* for any $w \times h$ bounded shape within this hierarchy (it can be reconfigured to construct any $w \times h$ bounded shape in the hierarchy). This class of shapes constitutes the most general set

of buildable shapes in the literature, with most previous work consisting of just the first-level of our hierarchy. We accompany this result with a $O(n^4 \log n)$-time algorithm for deciding if a given hole-free shape is a member of the hierarchy. For our second result, we resolve a long-standing open problem within the field: We show that deciding if a given position may be covered by a tile for a given initial board configuration is PSPACE-complete, even when all movable pieces are 1×1 tiles with no glues. We achieve this result by a reduction from Non-deterministic Constraint Logic for a one-player unbounded game.

5.2 Introduction

Advances in technology have given us the ability to create and manipulate robots for numerous applications at the molecular scale. At this size, fabrication tool limitations often make precise construction of the objects infeasible and thus self-assembly methods have been employed. This allows the creation of simple primitives that come together algorithmically to form the desired objects [11]. Similarly, the individual control of these simple nano-scale objects can be infeasible due to the cost or the desired simplicity of the building blocks. Thus, a growing area of interest is robot motion planning based on global external signals, known as the tilt model [4, 5]. This simple model gives all robots the same movement signal, which means all robots can be identical, and only their location and the geometry of the board need to be considered for fabrication.

The tilt self-assembly model consists of a 2D grid board with "open" and "blocked" spaces, as well as a set of slidable polyominoes placed on the board. The model uses a global external force to give all movable particles the same movement instruction. This may be done through any external force such as a magnetic field or gravity. This model has a history of assuming gravity as the external global force, hence the term *tilt*. In this model, a board may be *tilted* (an application of the global external force) in any of the four cardinal directions, causing all slidable polyominoes to slide maximally in the respective direction until reaching an obstacle. Various puzzle games exist which utilize the mechanics of this model; the maximal movement of [1] and the global movement signals in [2].

Shape Class	Result	Theorem
Hole-free Level-1 Drop Shapes ($\mathfrak{D}_1$)	$\mathcal{O}(n \log n)$	Thm. 5 in [8]
Level-1 Drop Shapes ($\mathfrak{D}_1$)	$\mathcal{O}(n^2 h!)$	Alg. 1 in [10]
Straight 2-Cuttable Hole-free Shape	$\mathcal{O}(n^3 \log n)$	Thm. 2 in [12]
Straight 2-Cuttable Shape with Convex Holes	$\mathcal{O}(n^4 \log n)$	Thm. 2 in [12]
Hole-free Drop Shapes ($\mathfrak{D}$)	$\mathcal{O}(n^4 \log n)$	**Thm. 10**

Table 5.1: Known results for deciding membership in the drop-shape hierarchy of a size-n shape (We refer to size as the number of tiles.). h denotes the number of holes in the shape. Previous results only decided shapes in $\mathfrak{D}_1$. With holes, no polynomial time algorithm is known even for $\mathfrak{D}_1$.

Class	Universal	Tilts	Board Size	Theorem
Drop Shapes ($\mathfrak{D}_1$)	No	$\mathcal{O}(hw)$	$\mathcal{O}(h^3 w^3)$	Thm. 6 in [8]
	Yes	$\mathcal{O}(h^2 w)$	$\mathcal{O}(h^2 w)$	Thm. 4.1 in [3]
Drop Shape Hierarchy ($\mathfrak{D}$)	No	$\mathcal{O}(hw)$	$\mathcal{O}(h^2 w^2)$	Thm. 3 in [12]
	Yes	$\mathcal{O}(h^3 w^2)$	$\mathcal{O}(h^2 w^2 \log^2(hw))$	**Thm. 13**

Table 5.2: Construction results for shapes with a $h \times w$ bounding box.

5.2.1 Related Work.

In [8], the authors introduced a class of shapes constructed by "dropping" pixels on a fixed seed from any of the four cardinal directions. Along with this new class of "drop-shapes", they presented a polynomial-time algorithm to check if a given hole-free shape is in this drop class (i.e., it can be constructed via this pixel dropping method). Given a hole-free shape, they showed how to construct a tilt model micro-factory which can build the shape. In [10] the authors provide an algorithm to determine if any given shape is in the class of drop shapes with the run time being exponential in the number of holes.

The efficient construction of drop-shapes in the tilt model was explored in [12]. The authors present a construction which efficiently builds a particular set of shapes in sublinear time through the use of parallelization. They introduce a hierarchical process whereby multi-tile subassemblies may be successively combined in a drop fashion. They define two classes of shapes 2-cuttable and straight 2-cuttable as any polyomino that can be decomposed into monotone subpolyominoes using valid 2-cuts and valid straight 2-cuts respectively. They also give a polynomial time algorithm for determining if a given simple polyomino, or one with convex shaped holes, is straight 2-cuttable.

Problem	Polyominoes	Result	Theorem
Occupancy/Relocation	1×1	NP-hard	Thm. 1 in [4]
	1×1, 2×2	PSPACE-Complete	Thm. 5.1 in [3]
	1×1	**PSPACE-Complete**	**Thms. 14, 4**
Reconfiguration Minimization	1×1	PSPACE-Complete	Thm. 4 in [6]
Reconfiguration	1×1, 2×2	PSPACE-Complete	Thm. 6.1 in [3]
	1×1	**PSPACE-Complete**	**Thm. 5**

Table 5.3: Known complexity results in the full tilt model. All current results for occupancy also imply hardness for relocation. Reconfiguration Minimization is finding the minimum length tilt sequence. Each of the results that use a 2×2 polyomino only uses a single one and multiple 1×1s.

They also have a polyomial time algorithm for finding if a valid 2-cut exists in the same two types of shapes. Given a shape in their buildable class, a tilt model configuration could be generated to assemble that shape through a sequence of tilts.

This drop-shape construction work was extended in [3]. Rather than focusing on designing configurations for specific drop-shapes, the authors create a general drop-shape constructor. They introduced the concept of *universal* constructors in the tilt model. These are constructors which, starting from one initial configuration, can construct any shape from a particular set. In this work, a configuration that is universal for the set of drop-shapes was created.

These construction results are usually paired with complexity results to decide what can be constructed. One of the most natural questions in the tilt model asks if a particular location on a board may ever be occupied by a particle from the starting configuration. This problem, known as the *occupancy problem*, was first introduced in [4]. The authors proved the problem was NP-hard, even when considering the restricted case of a configuration with only 1×1 tiles. The authors also showed that a dual-rail logic fanout was not possible with only 1×1 tiles, which seemed to serve as evidence that this problem was not PSPACE-complete. Following, in [3] the authors proved that the occupancy problem is PSPACE-complete if a single "large" polyomino is allowed (they used a single 2×2 polyomino along with 1×1 tiles).

5.2.2 Our Contributions.

In this paper, we formalize the notion of hierarchical construction (first started in [12]) by presenting a hierarchy of shape classes. In Section 5.4, we formally define a drop-shape hierarchy. We also present a $O(n^4 \log n)$-time algorithm for determining if a given hole-free shape is in this hierarchy. To accompany this result, we present the design for a configuration that is *strongly universal* for any shape within the hierarchy in Section 5.5. This result constitutes the most general set of buildable shapes in the literature, with previous work consisting of just the first level of our hierarchy. The second main result of this paper is in Section 5.6, where we resolve an open problem from [4] by showing that the occupancy problem, contrary to expectation, is PSPACE-complete even when restricting all polyominoes to be 1×1 tiles.

5.3 Model Preliminaries

Board. A *board* (or *workspace*) is a rectangular region of the 2D square lattice in which specific locations are marked as *blocked*. Formally, an $m \times n$ board is a partition $B = (O, W)$ of $\{(x,y) | x \in \{1,2,\ldots,m\}, y \in \{1,2,\ldots,n\}\}$ where O denotes a set of *open* locations, and W denotes a set of *blocked* locations- referred to as "concrete" or "walls." We classify the different possible board geometries according to the following hierarchy:

- Connected:[1] A board is said to have *connected* geometry if the set of open spaces O for a board is a connected shape.
- Simple:[2] A connected board is said to be *simple* if O has genus-0.
- Monotone: A simple board is *monotone* if O is either horizontally monotone or vertically monotone.
- Convex: A monotone board is *convex* if O is both horizontally monotone and vertically monotone.

[1] In [3], this hierarchy level was known as *complex*. We modify this class to allow for a proper hierarchy of shape classes.

[2] In [3], they define simple geometry based on the connectivity of W. We also modify the concept for hierarchical purposes.

- Rectangular: A convex board is *rectangular* if O is a rectangle.

Our board definitions have changed since [3] in order to create the hierarchy shown above.

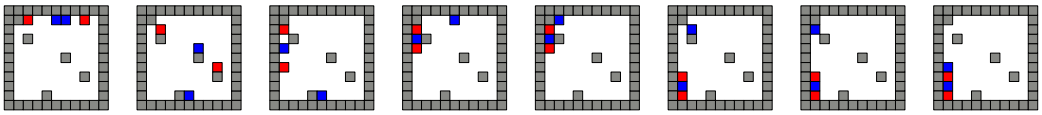
Tiles. A tile is a labeled unit square centered on a non-blocked point on a given board. Formally, a tile is an ordered pair (c,a) where c is a coordinate on the board, and a is an attachment label. Attachment labels specify which types of tiles will stick together when adjacent, and which have no affinity. For a given alphabet of labels Σ, and some *affinity* function $G : \Sigma \times \Sigma \to \{0,1\}$ which specifies which pairs of labels attract ($G(a,b) = 1$) and which do not ($G(a,b) = 0$), we say two adjacent tiles with labels a and b are *bonded* if $G(a,b) = G(b,a) = 1$.

Slidable Polyominoes. A *slidable polyomino* is a finite set of tiles $P = \{t_1, \ldots t_k\}$ that is 1) connected with respect to the coordinates of the tiles in the slidable polyomino and 2) *bonded* in that the graph of tiles in P with edges connecting bonded tiles is connected. When the context is clear, we often refer to these simply as polyominoes. A slidable polyomino that consists of a single tile is informally referred to as a "tile".

Configurations. A configuration is an arrangement of polyominoes on a board such that there are no overlaps among polyominoes, or with blocked board spaces. Formally, a configuration $C = (B, P = \{P_1 \ldots P_k\})$ consists of a board B, along with a set of non-overlapping polyominoes P that each do not overlap with the blocked locations of board B.

Step. A *step* is a way to turn one configuration into another by way of a global signal that moves all polyominoes in a configuration one unit in a direction $d \in \{N,E,S,W\}$ when possible without causing an overlap with a blocked location, or another polyomino. Formally, for a configuration $C = (B,P)$, consider the translation of all polyominoes in P by 1 unit in direction d. If no overlap with blocked board spaces occurs, then the new configuration is derived by first performing this translation, and then merging each pair of polyominoes that each contain one tile from a now (adjacently) bonded pair of tiles. If an overlap does occur, for each polyomino for which the translation causes an overlap with a blocked space, temporarily add these polyominoes to the set of blocked spaces and repeat. Once the translation induces no overlap with blocked

spaces, execute the translation and merge polyominoes based on newly bonded tiles to arrive at the new configuration. If all polyominoes are marked as blocked spaces, then the step transition does not change the initial configuration. If a configuration does not change under a step transition for direction d, we say the configuration is *d-terminal*. In the special case that a step causes a polyomino (or subpolyomino) to leave the board, we remove the polyomino from the configuration.



(a) Start (b) South (c) West (d) North (e) West (f) South (g) West (h) South

Figure 5.1: Tilt Example

Tilt. A *tilt* in direction $d \in \{N,E,S,W\}$ for a configuration is executed by repeatedly applying a step in direction $d \in \{N,E,S,W\}$ until a d-terminal configuration is reached. We say that a configuration C can be *reconfigured in one move* into configuration C' (denoted $C \to_1 C'$) if applying one tilt in some direction d to C results in C'. We define the relation $\to_*$ to be the transitive closure of $\to_1$. Therefore, $C \to_* C'$ means that C can be reconfigured into C' through a sequence of tilts.

Tilt Sequence. A *tilt sequence* is a series of tilts which can be inferred from a series of directions $D = \langle d_1, d_2, \ldots, d_k \rangle$; each $d_i \in D$ implies a tilt in that direction. For simplicity, when discussing a tilt sequence, we just refer to the series of directions from which that sequence was derived. Given a starting configuration, a tilt sequence corresponds to a sequence of configurations based on the tilt transformation. An example tilt sequence $\langle S,W,N,W,S,W,S \rangle$ and the corresponding sequence of configurations can be seen in Figure 5.1.

Universal Configuration. A configuration C' is universal to a set of configurations $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ if and only if $C' \to_* C_i \; \forall \; C_i \in \mathscr{C}$.

Configuration Representation. A configuration may be interpreted as having constructed a "shape" in a natural way. Define a shape to be a connected subset $S \subset \mathbb{Z}^2$. A configuration *strongly* represents S if the configuration consists of a single polyomino whose tile coordinates are

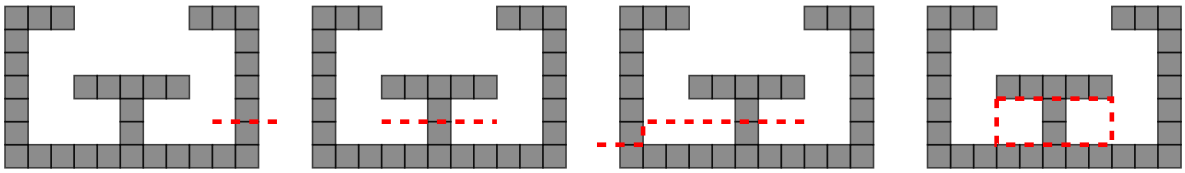
exactly the points of some translation of S. A weaker version allows for some "helper" polyominoes to exist in the configuration and not count towards the represented shape. In this case, we say a configuration *weakly* represents S.

Universal Shape Builder. Given this representation, we say a configuration C' is *universal* for a set of shapes U if and only if there exists a set of distinct configurations $\mathscr{C}$ such that 1) each $u \in U$ is represented by some $C \in \mathscr{C}$ and 2) C' is universal for $\mathscr{C}$. If each $u \in U$ is strongly represented by some $C \in \mathscr{C}$, we say C' is *strongly universal* for U. Alternately, if each $u \in U$ is weakly represented by some $C \in \mathscr{C}$, we say C' is *weakly universal* for U. In a similar way, a configuration can be universal for a set of patterns.

5.4 Drop-Shape Hierarchy

We utilize the same definitions for polyominoes and cuts as used in [12].

Polyomino. For a set $P \subset \mathbb{Z}^2$ of grid points in the plane, let the graph G_P be the induced grid graph in which two vertices $p_1, p_2 \in P$ are connected if they are unit distance apart. For any set P with connected grid graph G_P, a *polyomino* may be induced by replacing each point $p \in P$ with a unit square centered at p. A polyomino is said to be *hole-free* or *simple* if the induced graph $\mathbb{Z}^2 \setminus P$ is connected. We say that polyominoes which are equal up to translation have the same *shape*. We often use these terms interchangeably.



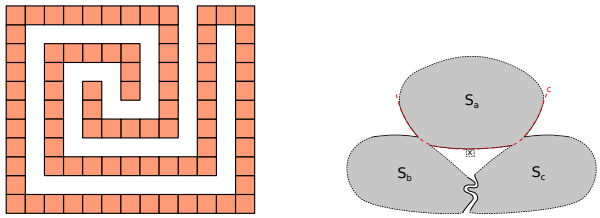
(a) Valid 2-cut (b) Invalid 2-cut (c) Valid 3-cut (d) Invalid 3-cut

Figure 5.2: Examples of valid and invalid cuts for a given polyomino. It is important to note that along with direct collision, we consider a cut to be invalid if the resulting polyominoes must slide past adjacent squares.

Cuts. A *cut* is an orthogonal curve moving between points of $\mathbb{Z}^2$. A p-cut is a cut that splits a polyomino P into p subpolyominoes. We say that a cut is *valid* if all of the induced

subpolyominoes may be partitioned into two nonempty groups which may be pulled apart in opposite directions without blocking each other. A polyomino A is *blocked* by another polyomino B in direction d if A cannot be pulled in direction d without colliding with B (note that A sliding adjacent to tiles in B is also considered as a collision). See Figure 5.2 for examples of valid and invalid cuts.

Tangled Polyomino. A *tangled* polyomino is one which does not have a valid 2-cut. That is, any cut either splits the polyomino into two subpolyominoes which cannot be pulled apart (invalid), or it splits the polyomino into more than two subpolyominoes (p-cut where $p > 2$). Figure 5.3a provides an example of a tangled polyomino.



(a) (b)

Figure 5.3: (a) A tangled polyomino for which no 2-cut exists. (b) An abstract representation of tangled polyomino S from Lemma 10 which depicts the required properties as stated in the proof.

Drop Construction Step. A drop construction step is described by a direction $\{N, E, S, W\}$ and a latitude or longitude l representing the column/row that a shape is placed. The shape arrives from (l, ∞) for north, $(l, -\infty)$ for south, (∞, l) for east, and $(-\infty, l)$ for west. The shape moves along the row/column until it reaches the first grid position adjacent to to an existing tile. We will refer to this as "dropping" a tile or shape.

Drop-Shape Hierarchy. Here we define the hierarchy of drop-shapes (denoted $\mathfrak{D}$). We use notation $\mathfrak{D}_h$ to denote level-h of the drop-shape hierarchy. Beginning with level-1, the levels of our hierarchy are defined recursively, containing the specified elements and nothing else:

- Level-0 ($\mathfrak{D}_0$). The single tile.
- Level-1 ($\mathfrak{D}_1$). The set of shapes in $\mathfrak{D}_1$ is defined recursively:

- (Base) $\mathfrak{D}_0 \subset \mathfrak{D}_1$
- (Drop Combinations) For any $A \in \mathfrak{D}_1$, any polyomino C that is produced by dropping a single tile onto A is also in $\mathfrak{D}_1$.

- Level-h ($\mathfrak{D}_h$). In general, the set of shapes in $\mathfrak{D}_h$ is defined recursively:
  - (Base) $\mathfrak{D}_{h-1} \subset \mathfrak{D}_h$
  - (Drop Combinations) For any $A \in \mathfrak{D}_{h-1}$ and $B \in \mathfrak{D}_h$, any polyomino C that is produced by dropping A onto B is also in $\mathfrak{D}_h$.
- Strict Level-h ($\widehat{\mathfrak{D}}_h$): The set of shapes that are in $\mathfrak{D}_h$, but not in $\mathfrak{D}_{h-1}$. Formally, $\widehat{\mathfrak{D}}_h = \mathfrak{D}_h \setminus \mathfrak{D}_{h-1}$.
- The Hierarchy ($\mathfrak{D}$): The hierarchy is defined as the union of all levels. Formally, $\mathfrak{D} = \bigcup_{i=0}^{\infty} \mathfrak{D}_i$

Since tangled polyominoes cannot be decomposed, no tangled polyomino is in the hierarchy.

Decomposition Tree. A decomposition tree for a given polyomino P is a rooted binary tree of polyominoes with root node being P, each leaf being either a single tile or a tangled polyomino, and all non-leaf nodes having two children consisting of two polyominoes that make a valid 2-cut for the polyomino at the given node. A decomposition tree is said to be *atomic* if all leaves are single tiles.

Strict Decomposition Tree. A decompostion tree is said to be a Strict Decomposition Tree if for every node p,either $p \in \mathfrak{D}_0$, or for some $h > 0$, $p \in \widehat{\mathfrak{D}}_h$ and has children $p_i \in \mathfrak{D}_h$ and $p_j \in \mathfrak{D}_{h-1}$.

Singly-Connected Tile. A tile t in polyomino P is said to be singly-connected w.r.t. P if and only if it is connected to $P - t$ on only one of its four edges.

5.4.1 Membership in the Drop-Shape Hierarchy.

A key contribution of this paper is a board configuration that is universal for all shapes in the drop-shape hierarchy of up to a given size n. This leads to the natural question of how efficiently can we decide if a given polyomino is a member of the drop-shape hierarchy $\mathfrak{D}$. $\mathfrak{D}$ is equivalent to the

set of 2-cuttable shapes defined in [12]. There, the authors present an algorithm for determining if a shape is in the set of *straight* 2-cuttable shapes, which is known to be a subset of $\mathfrak{D}$. It is currently unknown if the set of straight 2-cuttable shapes is equivalent to $\mathfrak{D}$. In this section we develop an efficient algorithm for deciding membership in $\mathfrak{D}$ for the case of genus-0 polyominoes. We first formally define the drop-shape hierarchy membership problem. We then prove two technical lemmas (Lemmas 9, 10) that will be used to establish the correctness of an efficient algorithm for deciding membership, provided in Theorem 10.

Drop-Shape Hierarchy Membership Problem. The drop-shape hierarchy membership problem asks: Given a polyomino P, is $P \in \mathfrak{D}$?

Lemma 9. *Given a hole-free polyomino P, $P \in \mathfrak{D}$ if and only if there exists an atomic decomposition tree for P.*

Proof. This proof builds off of Theorem 2 from [7], where they show that decomposition is the reverse of construction. Clearly, P is in the drop-shape hierarchy if such a decomposition tree exists, as the reverse drop construction step sequence can be performed to yield P. If no such decomposition tree exists, this means that all decomposition trees contain at least one leaf which is a tangled polyomino. Since a tangled polyomino cannot be in the hierarchy (as there are no valid 2-cuts), there must always exist a drop construction step which uses a polyomino that is not in the hierarchy. □

Lemma 10. *If there exists a decomposition tree for hole-free polyomino P which has a tangled polyomino S as a leaf, S must be a leaf in every decomposition tree of P.*

Proof. Given that there exists a decomposition tree T_A which has tangled shape S as a leaf, assume that some other decomposition tree T_B exists which does not have S as a leaf. This implies that T_B uses some decomposition sequence which involves a polyomino P' such that $S \subset P' \subseteq P$ and there exists a 2-cut for P' which also goes through subpolyomino S.

Let c be a valid 2-cut for P' which also cuts through S. Since we know that S by itself is tangled, one of the following must be true: 1) c was an invalid 2-cut through S which is now valid

because of the additional tiles introduced by $P' - S$. 2) c was a valid p-cut (with $p > 2$) through S which has now become a valid 2-cut because of the additional tiles introduced by $P' - S$. Since the addition of more tiles can never unblock a cut, we see that the first case cannot be true. The second case is not so straightforward, and requires more investigation.

First, consider the stand-alone polyomino S. We know c was a valid p-cut through S (with $p > 2$) which has now become a valid 2-cut through P'. For now, let it suffice to assume that c was a valid 3-cut through S. We will generalize to p later. This means that c cuts S into 3 subpolyominoes $\{S_a, S_b, S_c\}$ that can be partitioned into two sets which may be pulled apart in opposite directions without blocking each other. W.l.o.g., let that partition be $\{\{S_a\}, \{S_b, S_c\}\}$. Figure 5.3b shows a sketch that highlights the essential properties S must have. Since c is a 3-cut through S, we know that there must exist a path of empty spaces which separates S_b from S_c and begins at an empty space x along c which is adjacent to S_a. Furthermore, we know that subpolyominoes S_b and S_c must both block each other in the direction that $\{S_b, S_c\}$ is pulled apart from $\{S_a\}$; otherwise, a valid 2-cut would have existed in S (by pulling S_b or S_c off of S by itself).

Now, consider the polyomino P' with subpolyomino S. If location x contained a tile in polyomino P', we know that said tile would be blocked in each of the four directions by one of S's three subpolyominoes (there would be no way to remove the tile without also removing some portion of S). This, along with the fact that S is a node in the decomposition tree T_a, shows that location x must be empty in polyomino P'. In order for c to be a 3-cut through S and a 2-cut through P', the induced subpolyominoes S_b and S_c must be connected via a path of tiles in $P' - S$. Clearly, this path cannot exist on S_a's side of c, as it would make c an invalid 2-cut for P'. So, this path must connect S_b and S_c on their side of c. Since location x cannot contain a tile in P' we see that the connection between S_b and S_c must cut off x's connection to the outside plane. This cannot be the case, however, because P' is a hole-free polyomino.

So, c could not have simultaneously been a valid valid 2-cut through P' and valid 3-cut through S. It is easy to observe that any valid p-cut through S (which is also a valid 2-cut through P') must have at least one pair of induced polyominoes with the same properties as S_b and S_c, and

all others may be considered as S_a for the purposes of this proof. Thus, no valid 2-cut through P' may cut through S. This implies that decomposition tree T_B must not exist, as was originally assumed, and every decomposition tree must contain S as a leaf. □

Theorem 10. *A hole-free size-n polyomino P can be checked for membership in the drop-shape hierarchy in* $O(n^4 \log n)$ *time.*

Proof. This algorithm is straightforward. Given hole-free polyomino P, perform the following steps:

1. **Base Case:** If P is a single tile, return *yes*. If P is tangled, return *no*. Otherwise, continue to step 2.

2. Select a valid 2-cut which cuts P into subpolyominoes A and B. Continue to step 3.

3. Recurse on A and B by performing to step 1 for each of them. If both return *yes*, return *yes*; otherwise, return *no*.

We see that this algorithm creates a decomposition tree for polyomino P. The proof of correctness follows from the previous lemmas. If all leaves in the decomposition tree are single tiles, Lemma 9 tells us that P is in the hierarchy. If the decomposition tree contains a leaf which is a tangled polyomino, Lemmas 9 and 10 together show that P is not in the hierarchy.

The runtime is achieved as follows: Since a decomposition tree for a size-n hole-free polyomino P has at most n leaves, we know the tree has at most $2n-1$ nodes. Therefore, we can say that our algorithm is called at most $2n-1$ times. Step 1 of the algorithm checks if the polyomino is tangled (i.e., it checks if P has a valid 2-cut). Theorem 5 from [12] shows that a valid 2-cut can be found for a hole-free size-n polyomino in $O(n^3 \log n)$ time. So, our algorithm is called at most $2n-1$ times, with each call spending at most $O(n^3 \log n)$ time to find a valid 2-cut. Thus, we achieve a runtime of $O(n^4 \log n)$ to determine if P is in the drop-shape hierarchy. □

5.4.2 Hierarchy Characteristics.

In this section we derive some key properties of the drop-shape hierarchy. In particular, in Theorem 11 we show that any member of the hierarchy of size-n must be a member of level $\mathfrak{D}_{\lfloor \log |P| \rfloor}$. This result allows us to efficiently bound the size of the board needed to assembly size-n polyominoes in Section 5.5. Next, in Theorem 12 we establish that the drop-shape hierarchy is a true hierarchy, i.e. that each level h of the hierarchy contains shapes that are not members of any level lower than h.

Lemma 11. *For any polyomino $P \in \widehat{\mathfrak{D}}_H$ there exists a Strict decomposition Tree, for any $H \geq 0$.*

Proof. We will prove this by induction on the size of the shape. For our base case we know any single tile has a strict decomposition tree.

Now for our inductive step assume that there exists a strict decomposition tree for any polyomino p where $1 \leq |p| \leq n$ for some n. Consider a polyomino $P \in \widehat{\mathfrak{D}}_H$ where $|P| = (n+1)$. We know that since $P \in \widehat{\mathfrak{D}}_H$ there exists two polyominoes $P_a \in \mathfrak{D}_H$ and $P_b \in \mathfrak{D}_{H-1}$ that can be dropped onto each other to build P. We know that since both P_a and P_b are subpolyominoes of P, $n \geq |P_a|$ and $n \geq |P_b|$. From our inductive step we can assume that both P_a and P_b have strict decomposition trees and the only new node we add is P which has the properties required by a strict decomposition tree.

From here we see that every polyomino in a strict level $H \geq 0$ has a strict decomposition tree. □

Lemma 12. *For any polyomino $P \in \widehat{\mathfrak{D}}_H$ such that $H > 0$, any strict decomposition tree of P must have a node P' with two children P_l and P_r such that $P' \in \widehat{\mathfrak{D}}_H$ and both $P_l \in \widehat{\mathfrak{D}}_{H-1}$ and $P_r \in \widehat{\mathfrak{D}}_{H-1}$.*

Proof. First we will prove there must exist a node P' with two children P_l and P_r such that $P' \in \widehat{\mathfrak{D}}_H$ and both $P_l \in \mathfrak{D}_{H-1}$ and $P_r \in \mathfrak{D}_{H-1}$, then we will show why they both must be strict.

Let us first look at the root P. If both children of P are in $\mathfrak{D}_{H-1}$ than P is the node we are describing. Now consider the case where p is a child of P and $p \notin \mathfrak{D}_{H-1}$. Since we are looking at a strict decomposition tree we know that $p \in \widehat{\mathfrak{D}}_H$. Since the tree rooted at p is also a strict

decomposition tree we have the same two cases. However since a strict tree is also an atomic tree all the leaves of the tree must be single tiles so eventually the first case must be true.

Now we will show $P_l \notin \mathfrak{D}_{H-2}$ and $P_r \notin \mathfrak{D}_{H-2}$. Without loss of generality assume $P_l \in \mathfrak{D}_{H-2}$, this would mean that P' can be built by dropping a shape in $\mathfrak{D}_{H-2}$ onto a shape in $\mathfrak{D}_{H-1}$ which by definition would mean $P' \in \mathfrak{D}_{H-1}$ which we know is not true since $P' \in \widehat{\mathfrak{D}}_H$.

Since $P_l \in \mathfrak{D}_{H-1}$ and $P_r \in \mathfrak{D}_{H-1}$, and $P_l \notin \mathfrak{D}_{H-2}$ and $P_r \notin \mathfrak{D}_{H-2}$ we can see that $P_l \in \widehat{\mathfrak{D}}_{H-1}$ and $P_r \in \widehat{\mathfrak{D}}_{H-1}$. Therefore there must exist a node $P' \in \widehat{\mathfrak{D}}_H$ with both children in $\widehat{\mathfrak{D}}_{H-1}$. □

Lemma 13. *For any polyomino $P \in \widehat{\mathfrak{D}}_H$, $|P| \geq 2^H$*

Proof. We prove this by induction on H. For the base case we can see that the only strict level-0 shape is the single tile. We can see that $1 \geq 2^0$ is true.

For the inductive step assume for any $p \in \widehat{\mathfrak{D}}_H$, $|p| \geq 2^H$. Consider a polyomino $P \in \widehat{\mathfrak{D}}_{H+1}$. We know from lemma 11 there must exist a strict decomposition tree and from lemma 12 that there must exist a node in a strict decomposition tree P' that has two children $P_l, P_r \in \widehat{\mathfrak{D}}_H$. We can also see that since P_l and P_r are children of P',

$$|P'| = |P_r| + |P_l|$$

From our assumption since both $P_l, P_r \in \widehat{\mathfrak{D}}_H$, $|P_r| \geq 2^H$ and $|P_l| \geq 2^H$

$$|P'| \geq 2^H + 2^H$$

$$|P'| \geq 2^{H+1}$$

Finally, since P' is a subpolyomino of P, $|P| \geq |P'|$.

$$|P| \geq 2^{H+1}$$

□

Theorem 11. *For any polyomino $P \in \mathfrak{D}$, $P \in \mathfrak{D}_{\lfloor \log |P| \rfloor}$*

Proof. We can see from lemma 13 for any $P \in \widehat{\mathfrak{D}}_h$, $\log |P| \geq h$. We also know that for any $H \geq h, P \in \mathfrak{D}_H$. Let $H = \lfloor \log |P| \rfloor$, we can see that $P \in \mathfrak{D}_{\lfloor \log |P| \rfloor}$. □

Lemma 14. *For all positive integers h, given a hole-free polyomino $P \in \mathfrak{D}_h$, for any singly-connected, non-blocked tile $t \in P$, $P - t \in \mathfrak{D}_h$.*

Proof. We will prove this is true by induction on h. Lemma 3 of [8] showed that given a hole free polyomino $P_1 \in \mathfrak{D}_1$, for any tile $t_1 \in P_1$ that is non-cut ($P_1 - t_1$ is connected), non-blocked, and convex (there exists a 2×2 square that solely contains t_1), $P_1 - t_1 \in \mathfrak{D}_1$. For simplicity we will refer to a singly-connected, non-blocked tile as a *candidate* tile. A candidate tile is convex and non-cut. In general, we will say that $C(\mathfrak{D}_h)$ is true if and only if for all polyominoes $P_h \in \mathfrak{D}_h$, for any candidate tile $t \in P_h$, $P - t \in \mathfrak{D}_h$.

The base case $C(\mathfrak{D}_1)$ was shown to be true by Becker et al. in [8]. Assume $C(\mathfrak{D}_h)$ holds. We will prove by contrapositive that $C(\mathfrak{D}_h) \implies C(\mathfrak{D}_{h+1})$.

Assume $\neg C(\mathfrak{D}_{h+1})$. Consider a polyomino $P \in \mathfrak{D}_{h+1}$, a candidate tile $t \in P$, and a polyomino $P - t = P' \notin \mathfrak{D}_{h+1}$. Let T be a strict decomposition tree of P. The first cut in T is not solely removing t, since that would result in $P' \notin \mathfrak{D}_{h+1}$ (violating the strict decomposition tree), so we will also consider T', a decomposition tree for P' that initially makes the same cuts as T. Since T is a strict decomposition tree, for the children of its root, p_l and p_r, one is in $\mathfrak{D}_{h+1}$ and the other is in $\mathfrak{D}_h$. W.L.O.G., let t exist in the child node p_l. Now, consider the nodes derived from making equivalent cuts in T', p'_l and p'_r. Since t only existed in p_l, $p_r = p'_r$. There are two cases. First, if $p_l \in \mathfrak{D}_h$. We know $p'_l \notin \mathfrak{D}_h$ because $p'_r = p_r$ and $p_r \in \mathfrak{D}_{h+1}$, but $P' \notin \mathfrak{D}_{h+1}$. This means that $p_l \in \mathfrak{D}_h$ and $t \in p_l$ was a candidate tile, and $p_l - t = p'_l \notin \mathfrak{D}_h$, and therefore $\neg C(\mathfrak{D}_h)$. The second case is $p_l \notin \mathfrak{D}_h$, but since it is a strict decomposition tree it must then be in $\widehat{\mathfrak{D}}_{h+1}$. If this is the case, then consider p_l as the root polyomino. Repeat the same process to p_l and p'_l that we did to P and P'. Since all leaves in a strict decomposition tree are single tiles, we eventually arrive at a child node p in $\mathfrak{D}_h$, where $t \in p$ is a candidate tile, and a $p - t = p' \notin \mathfrak{D}_h$. Therefore, $\neg C(\mathfrak{D}_{h+1}) \implies \neg C(\mathfrak{D}_h)$, and by contrapositive: $C(\mathfrak{D}_h) \implies C(\mathfrak{D}_{h+1})$. □

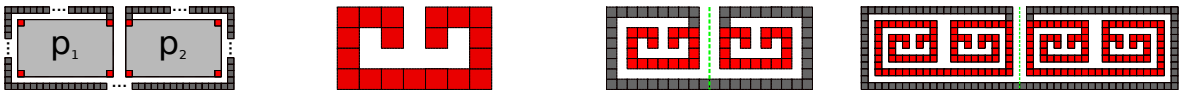
Theorem 12. *For all positive integers h, there exists a polyomino in* $\widehat{\mathfrak{D}}_h$.

Proof. Polyominoes in $\widehat{\mathfrak{D}}_1$ have previously been labeled as drop shapes. This is the set of shapes that can be built by dropping only single tiles, excluding the singleton tile itself. A polyomino in $\widehat{\mathfrak{D}}_2$ can be seen in Figure 5.4b. We will call this polyomino P_2. We can use a method we will refer to as *fractalization* to generate a polyomino in $\widehat{\mathfrak{D}}_3$ from P_2. See Figure 5.4c.

To show existence of a polyomino for every strict level of the hierarchy, we will show how a polyomino $p \in \widehat{\mathfrak{D}}_h$ that was generated through repeated fractalization to P_2 can be used to create a polyomino $p' \in \widehat{\mathfrak{D}}_{h+1}$. This method takes 2 copies of p, which we will label p_1 and p_2, and places them horizontally adjacent to each other 2 unit spaces apart (without loss of generality, assume p_1 is to the left of p_2). Then, two tiles are added; one above the north-most right-most tile of p_1 and the other above the north-most left-most tile of p_2. A string of single tiles is then wrapped around both polyominoes such that there is 1 unit space between the border and the bounding boxes of the inner polyominoes. This border polyomino connects the two tiles that were added from above. The border polyomino blocks both p_1 and p_2 in all directions, creating polyomino p', See Figure 5.4a.

We know that $p' \in \mathfrak{D}_{h+1}$, as it can clearly be 2-cut across the border resulting in two polyominoes in $\mathfrak{D}_h$ that can be built by dropping single tiles onto p. This cut is shown in Figure 5.4. We must now show that p' is not in $\mathfrak{D}_h$. If p' was in $\mathfrak{D}_h$, there would exist a valid 2-cut in which at least one of the resulting polyominoes derived from that cut was in $\mathfrak{D}_{h-1}$. This cut can not occur solely across the border connecting the two polyominoes, as the two resulting polyominoes derived from that cut are polyominoes that can be turned into p by repeatedly removing non-blocked, singly-connected tiles. Therefore by Lemma 14, since $p \in \widehat{\mathfrak{D}}_h$ we can see that these polyominoes can not be in $\mathfrak{D}_g$ for $g < h$. It is also clear that this cut can not occur solely within p_1 and/or p_2, as they are are now both blocked in all directions. It is also not possible for the cut to cross both the border and p_1 or p_2. First observe that due to the way these polyominoes are connected, there will never be 2×2 square fully occupied by tiles. Since there are also no holes, there is never a choice of which path to take to get from one tile to another. This means there is a single path of connectivity between any two tiles in p', and therefore a single path of connectivity between any tile in p_1 and

any tile in p_2. This path will always include the border that connects the two. If the cut occurs within the border, then this path is removed, meaning there are now 2 disconnected polyominoes. W.l.o.g., assume this cut also occurs within p_1, splitting it into p_l and p_r. Since there is only one path between any two tiles, there is now no path between p_l and p_r. There is also no path between p_2 and p_l or p_r. It follows that there are now 3 disconnected polyominoes as a result of this cut, leaving this cut an n-cut, where $n = 3$. If the cut continued through p_1 or into p_2, n could increase, but never decrease. This shows that the 2-cut in question does not exist, meaning $p' \notin \mathfrak{D}_h$. Since $p' \in \mathfrak{D}_{h+1}$ and $p' \notin \mathfrak{D}_h$, $p' \in \widehat{\mathfrak{D}}_{h+1}$. This shows that the fractalization method can be repeated to generate a polyomino in $\widehat{\mathfrak{D}}_h$ for all positive integers h. Figure 5.4 shows the fractalization method being used to create a polyomino in $\widehat{\mathfrak{D}}_3$ and a polyomino in $\widehat{\mathfrak{D}}_4$. □



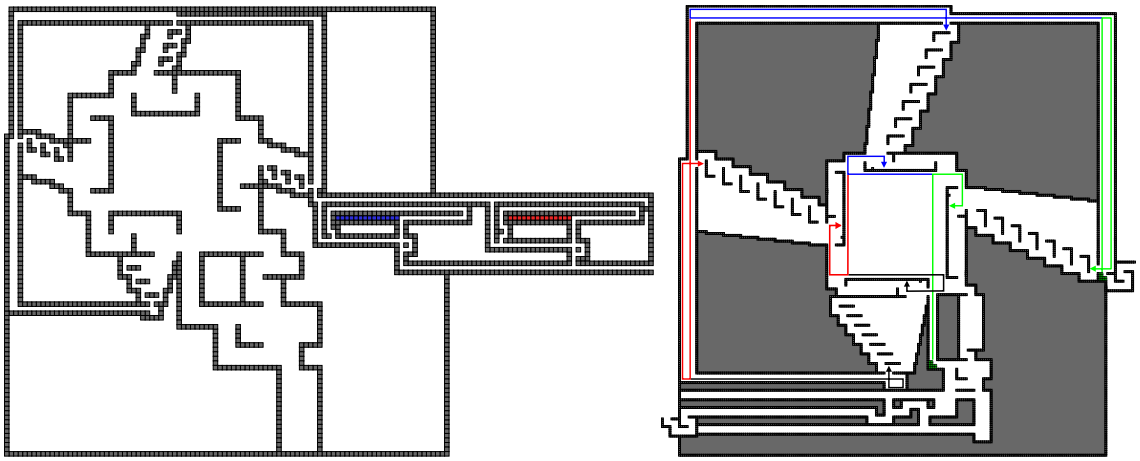
(a) Generalization of the fractalization method (b) Polyomino in $\widehat{\mathfrak{D}}_2$ (c) Polyomino in $\widehat{\mathfrak{D}}_3$ (d) Polyomino in $\widehat{\mathfrak{D}}_4$

Figure 5.4: Fractalization method used to reach higher strict levels of the hierarchy. The gray tiles represent the single tile border created, while the red tiles represent the polyominoes in $\widehat{\mathfrak{D}}_h$ used. The green dashed line represents the 2-cut that shows this shape is in $\mathfrak{D}_{h+1}$.

5.5 Drop-Shape Hierarchy Constructor

This section presents a construction that builds any shape in the drop-shape hierarchy. The construction is a direct extension of [3]. In that work, a universal constructor was given for drop-shapes ($\mathfrak{D}_1$ of our hierarchy) which fit in a $w \times h$ bounding box. At a high-level, we can extend the construction with larger chambers that are functionally identical to the $\mathfrak{D}_1$ constructor, but are scaled to allow the dropping of large polyominoes.

$\mathfrak{D}_1$ Constructor. This construction (shown in Figure 5.5a) was introduced in [3] and is capable of building any polyomino $P \in \mathfrak{D}_1$, provided P fits within a given $w \times h$ bounding box. The high level idea for this constructor is that tiles can be extracted from the fuel chambers and dropped onto any row or column of a shape in the center construction area. For the reader, we have included



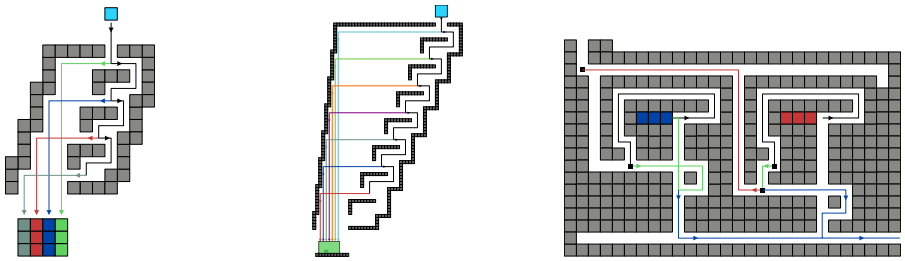
(a) $\mathfrak{D}_1$ Constructor (b) $\mathfrak{D}_i$ Constructor Gadget

Figure 5.5: (a) The preexisting universal drop-shape constructor which builds any shape in $\mathfrak{D}_1$. (b) The $\mathfrak{D}_i$ constructor gadget which allows the dropping of large polyominoes, it also depicts the sequence for selecting a direction to drop from. The paths from $\mathfrak{D}_i$ constructor gadget look the same in the $\mathfrak{D}_1$.

a descriptions of the tile selection, direction selection, and column selection process from [3] in Figures 5.6 and 5.5b, as well as the tilt sequences for these commands in Table 5.4. This constructor uses a modified tilt selection gadget which allows for the disposal of fuel tiles once the desired polyomino has been built.

$\mathfrak{D}_i$ Constructor Gadget. This gadget (shown in Figure 5.5b) is an extension to the $\mathfrak{D}_1$ constructor. Overall, this gadget is similar to the $\mathfrak{D}_1$ constructor, but scaled to allow the dropping of $w \times h$ polyominoes rather than single tiles. Because we are dropping two multi-tile polyominoes together, we need to double the dropping area to account for any drop that might be blocked by the adjacent wall in the dropping area. However, the dimensions of the polyomino built after the drop cannot exceed the $w \times h$ bounding box. The same tilt sequences used in the $\mathfrak{D}_1$ constructor are used in this constructor. This allows for a large polyomino to be dropped onto another in the same fashion as a tile is dropped in the other constructor. By attaching a series of these constructors onto a $\mathfrak{D}_1$ constructor, we can build polyominoes which are in higher levels of the hierarchy.

Bit String Tunnels. To connect the constructor chambers, we use bit string tunnels (Figure 5.8) which require a unique move sequence to move a polyomino from a constructor $\mathfrak{D}_i$ to a



(a) $\mathfrak{D}_1$ Column Selection (b) $\mathfrak{D}_i$ Column Selection (c) $\mathfrak{D}_i$ Tile Selection Gadget

Figure 5.6: (a) The column selection gadget for $\mathfrak{D}_1$ shapes. Assuming the shape to build is at a fixed location, this gadget allows any column to be selected to drop the new tile onto. The number of columns to drop from in this gadget determines the size of the shape we can build. Thus, this is for a drop shape within a 4×4 bounding box. This gadget is repeated on each of the four sides of the drop-shape constructor (with a slightly modified one on the south side in order to allow a non-conflicting move sequence.). (b) The column selection gadget for a 4×4 $\mathfrak{D}_i$ constructor. Notice that the tunnels are large enough to accommodate shapes that fit in a 4×4 bounding box, and the attachment area is twice as big as the $\mathfrak{D}_1$ version. This is to allow large polyominoes to be dropped onto any row/column of another large polyomino. (c) The fuel selection gadget for $\mathfrak{D}_1$. Each tile is pulled out with the sequence $\langle E,N,W,S,E,S\rangle$, and stops at the first square. Then the left tile type (blue) is either pulled out of the gadget or put back in the storage area. This shows it being added back to the storage with $\langle E,S,W,N,W,S\rangle$. This sequence puts the next tile type (red) in a decision location. The red tile is selected with the sequence $\langle W,N,W,S,W,S\rangle$. Once the desired shape has been built, one can remove the remaining fuel tiles off the board with $\langle E,N,W,S,E,S,W,S,E,S,W,S,E,W\rangle$. This sequence extracts both tile types off the storage area, and then removes them off the board.

constructor $\mathfrak{D}_{i+1}$. These tunnels require a sequence of up/down tilts, which can be thought of as 0 or 1 bits. It's easy to see that a construction which builds level h shapes requires $\log h$ many up/down selectors. At the end of every bit string tunnel there is a *reset gadget* as shown in Figure 5.7b section 3 which enforces all subpolyominoes to be in their launch configurations once a subpolyomino has traversed from constructor to constructor.

$\mathfrak{D}_4$ ***Constructor.*** Figure 5.9 shows a complete constructor which can build any polyomino P s.t. $P \in \mathfrak{D}_4$ and P fits in a 4×4 bounding box. Since P can have at most 16 tiles, Theorem 11 tells us that $P \in \mathfrak{D}_4$, i.e., the largest hierarchy level needed is four. Thus, the bit string tunnel only needs two up/down choice to encode all of the possible tunnel transitions.

Theorem 13. *Given positive integers $w,h \in \mathbb{Z}^+$, there exists a configuration which is* strongly *universal for the set of shapes $U = \{u \mid u \in \mathfrak{D}, u$ fits in a $w \times h$ bounding box$\}$. This configuration*

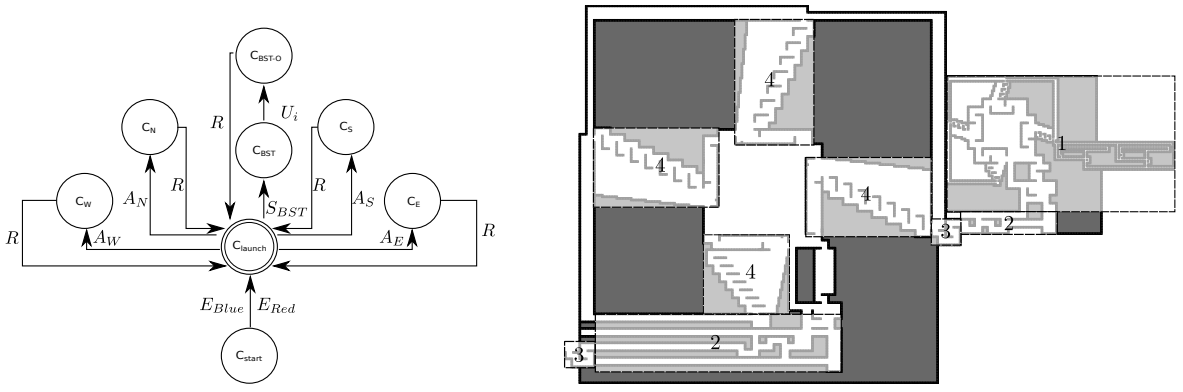
Command	Tilt Sequence
1. Extract blue tile (E_{Blue})	$\langle E,N,W,S,E,S,W,N,W,S,E,S,W,N,W\rangle$
2. Extract red tile (E_{Red})	$\langle E,N,W,S,E,S,E,S,W,N,E,S,W,S,W,N,W\rangle$
3. Add from east (A_E)	$\langle N,E,S,W,S\rangle + \langle S,W,N,W\rangle^j + \langle N,W,S,E,S,E,S\rangle$
4. Add from north (A_N)	$\langle N,W,N,E,S\rangle + \langle E,S,W,S\rangle^j + \langle W,S,E,N,E,S,E,S,W,\ S,E,S,E,S\rangle$
5. Add from west (A_W)	$\langle N,W,S,W,N,E\rangle + \langle N,E,S,E\rangle^j + \langle S,E,N,W,N,E,S,E,S,E,S,E,S,W,S,E,S,E,S,W,N\rangle$
6. Add from south (A_S)	$\langle N,W,S,E,S,W,N\rangle + \langle W,N\rangle^j + \langle E,N,W,S,W,N,E,S,E,S\rangle$
7. Send to BST (S_{BST})	$\langle N,W,S,E,S,E,S,W,S,E,S,W,S\rangle$
8. Traverse 1-bit (T_1)	$\langle W,N,W,S,W\rangle$
9. Traverse 0-bit (T_0)	$\langle W,S,W,N,W\rangle$
10. Reset to launch (R)	$\langle W,S,E,S,W,N,W\rangle$
11. Remove from board (D)	$\langle E,N,W,S,E,S,W,S,E,S,W,S,E,W\rangle$

Table 5.4: Commands E_{Blue}, E_{Red} move either a red or blue tile into launch configuration. Commands A_E, A_N, A_W, and A_S add a subpolyomino in the launch configuration into the shape being built. These said commands have been slightly modified from previous work to avoid conflicting with the remaining commands. Commands S_{BST}, T_0, and T_1 send the subpolyominoes to their respective bit string tunnels (BST's) and allow one of them to traverse through the tunnels. Command R returns back any polyominoes to their corresponding launch configuration. Finally, command D removes one red and blue tile from the board.

has size $\mathcal{O}(h^2w^2\log^2(hw))$ and uses $\mathcal{O}(h^3w^2)$ tilts to move into a configuration which strongly represents any shape $u \in U$.

Proof. This is a proof by construction. We begin with a configuration C where the chambers of all $\mathfrak{D}_i$ gadgets are empty except for the fuel chambers in $\mathfrak{D}_1$. We know from [3] that the $\mathfrak{D}_1$ gadget can build any level-1 drop shape. The building process follows the flowchart in Figure 5.7a using the tilt sequences in Table 5.4. This allows building any i-level shape bounded by h and w where $i \leq \log hw$ (by Theorem 11). While building the desired polyomino, all subpolyominoes follow the same sequence and thus reach the same position in their respective constructors simultaneously. Once a subpolyomino needs to move from a $\mathfrak{D}_i$ to a $\mathfrak{D}_{i+1}$ constructor, all subpolyominoes are sent to their corresponding bit string tunnel. The uniqueness of each tunnel guarantees only one subpolyomino successfully traverses towards the next constructor while the others are held back in the bit selector tunnel. The reset gadget at the end of each bit string tunnel enforces sequence R in Table 5.4 which sets every polyomino to its launch position.

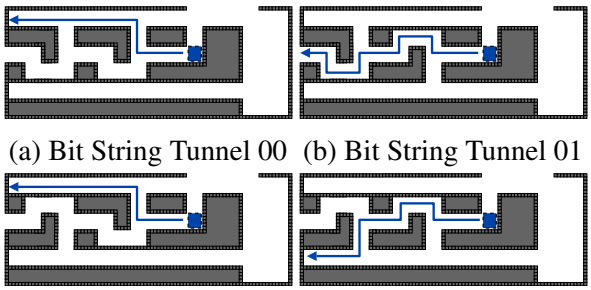
Once the desired polyomino has been built and is located in the $\mathfrak{D}_i$ constructor, we can perform sequence D until there is no more fuel pieces in the fuel chamber. We then have configura-



(a) (b)

Figure 5.7: (a) A flowchart where each state represents a set of configurations and the symbols represent sequences that can move from one state to another. The sequences for each of the symbols is shown in Table 5.4. The sequence marked as U_i is a unique combination of T_0 and T_1 tilt sequences for each ith constructor. Note that after performing the S_{BST}, U_i, R sequences one has successfully relocated the shape located in $\mathfrak{D}_i$ to $\mathfrak{D}_{i+1}$ and can add both shapes located in constructor $\mathfrak{D}_{i+1}$ with either of the A_N, A_E, A_S, A_W sequences. (b) This is an overview of the different sections of the hierarchy constructor. Section 1 is the $\mathfrak{D}_1$ shape constructor from previous work, section 2 is the bit string tunnels, section 3 is the reset gadget and section 4 is the column selection chambers for the $\mathfrak{D}_i$ constructor.

tion C' from C by performing a series of tilts. Consequentially, $C \to_* C'$. The overall process of transitions from the starting configuration to the configuration that represents shape $u \in U$, is shown in Figure 5.7a. □



(a) Bit String Tunnel 00 (b) Bit String Tunnel 01

(c) Bit String Tunnel 10 (d) Bit String Tunnel 11

Figure 5.8: This Figure demonstrates a 2-bit string tunnel. Notice that the tilt sequence $\langle W, N, W, S, W, S, W, N, W \rangle$ would only allow a shape to completely traverse bit string tunnel 01.

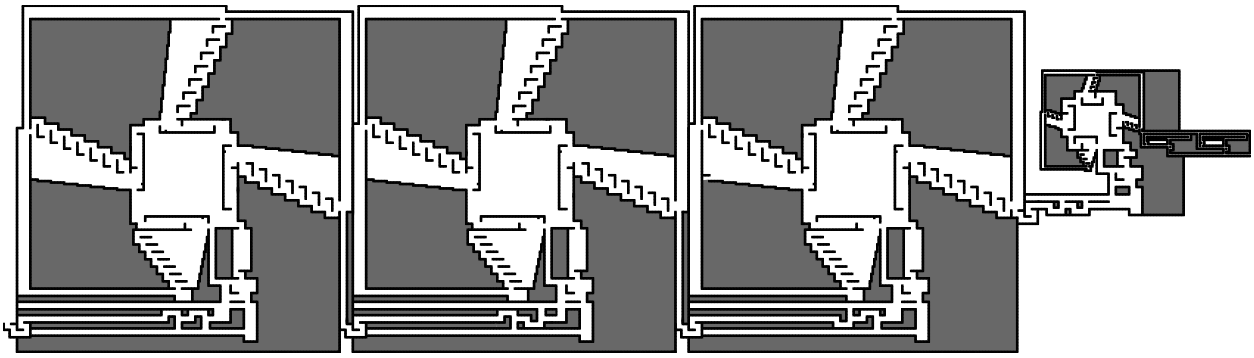


Figure 5.9: $\mathfrak{D}_4$ Constructor. This constructor is capable of building any 4×4-bounded polyomino in $\mathfrak{D}_4$.

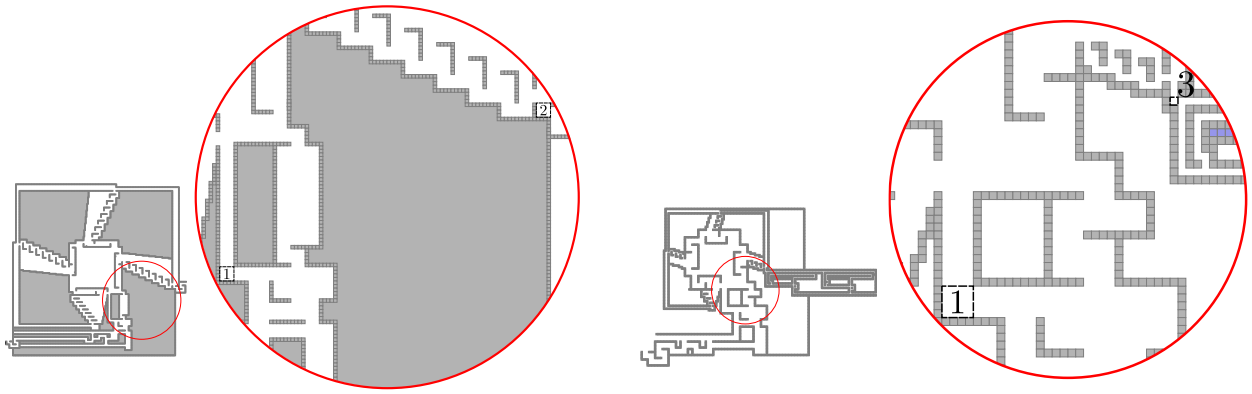


Figure 5.10: (a) $\mathfrak{D}_i$ Constructor in C_{launch} configuration. Section 1 is where shapes in all $\mathfrak{D}_i$ constructors will be located when in C_{launch} configuration. Section 2 indicates where a shape outputted from a reset gadget will be located when in C_{launch} configuration. (b) $\mathfrak{D}_1$ Constructor in C_{launch} configuration. Section 3 indicates where a single tile will be located after it is extracted from it's corresponding storage chamber. Similarly, once a tile is extracted from its storage chamber all shapes throughout the constructors will be located in Section 1.

5.6 Occupancy, Relocation, and Reconfiguration Complexity

Here, we prove that the occupancy, relocation, and reconfiguration problems are PSPACE-complete when limited to only 1×1 tiles by a polynomial time reduction from Non-Deterministic Constraint Logic. The occupancy problem asks whether a given position within a given board configuration can be occupied by a polyomino for some tilt sequence. Relocation asks whether a given position may be occupied by a specific given polyomino. And Reconfiguration asks if a given initial board configuration may be converted into a second given board configuration. The occupancy problem was originally defined by Becker et al. using only 1×1 tiles. They showed

it is NP-hard and the minimum move sequence for reconfiguration is PSPACE-complete [4, 6]. The authors also showed the impossibility of a fan out with dual rail logic which could be viewed as evidence against the problem being PSPACE-complete. However, recent work showed that with even a single additional 2×2 polyomino, the relocation and reconfiguration problems are PSPACE-complete [3]. In this section we definitively answer the question with only 1×1 tiles and show all three problems to be PSPACE-complete with only 1×1 tiles. To achieve this result we provide a polynomial time reduction from Non-Deterministic Constraint Logic [9]. The formal problem definitions are as follows.

Occupancy Problem. The occupancy problem asks whether or not a given location can be occupied by any tile on the board. Formally, given a configuration $C = (B, P)$ and a coordinate $e \in B$, does there exist a tilt sequence such that $C \rightarrow_* C'$ where $C' = (B, P')$ and $\exists p \in P'$ that contains a tile with coordinate e?

Relocation. The relocation problem asks whether a specified polyomino can be relocated to a particular position. That is, given a configuration, a polyomino within that configuration, and a translation of that polyomino, does there exist a sequence of tilts which moves the original polyomino to its translation?

Reconfiguration. The reconfiguration problem asks whether a configuration can be reconfigured into another. Formally, given two configurations $C = (B, P)$ and $C' = (B, P')$, does there exist a tilt sequence such that $C \rightarrow_* C'$?

5.6.1 Non-Deterministic Constraint Logic.

A constraint logic graph is a weighted directed graph with a constraint on each of the vertices [9]. The constraint specifies the minimum weight required from the edges directed in (the sum of the inflow) to any vertex. An example of two vertices can be seen in Figure 5.11e. When given a graph, the usual problem studied is whether a particular edge can be "flipped"- the direction of the edge changed, i.e., is there a sequence of edge flips that maintain the constraints on all vertices, and allows the target edge to be flipped? This is a one-player unbounded game. The problem is still

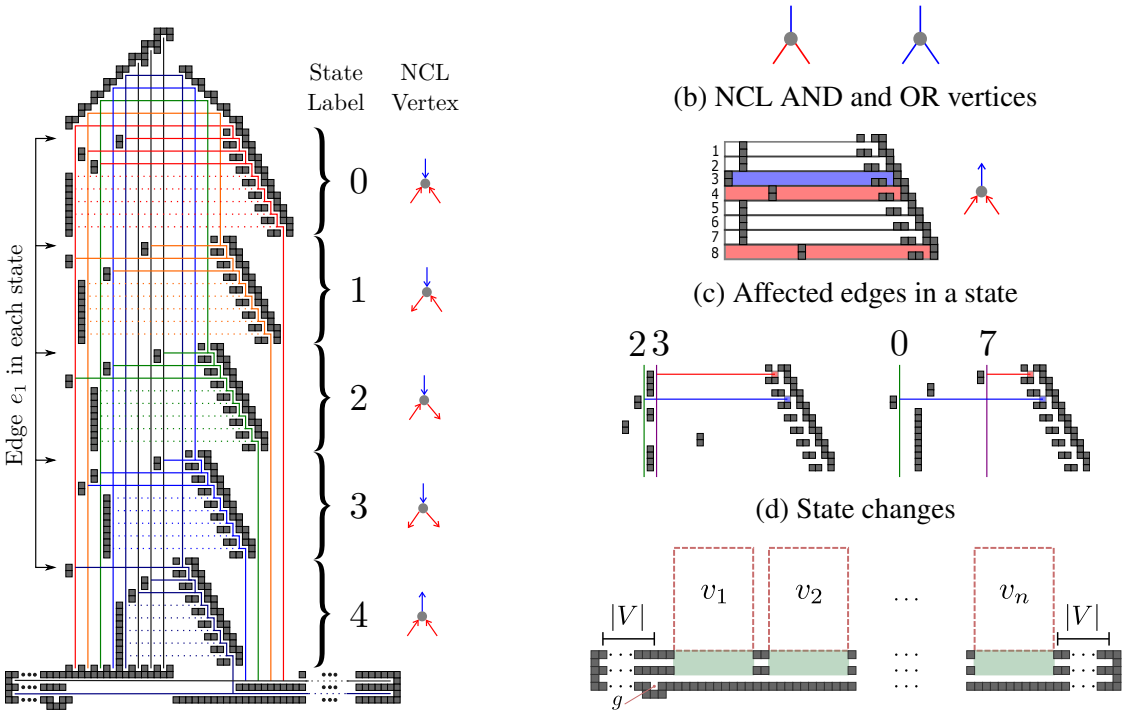
PSPACE-complete when the edge weights are all strength 1 or 2, and vertices have max degree 3. We address the following equivalent problem.

Configuration-to-Configuration Problem. Given two states of a constraint graph G and G', does there exist a sequence of edge flips starting with G that results in G' [9].

Vertex Gadget. We will assume a max degree of three for all vertices, which means there are 8 possible arrangements of in/out edges. Define the vertex *state* as a label from 0 to 7 determined by the directions of its incident edges. We label each edge of a vertex $\mathscr{E}_a, \mathscr{E}_b, \mathscr{E}_c$. The state is then the decimal value of a binary string of length three with each bit representing an edge ($\mathscr{E}_a\mathscr{E}_b\mathscr{E}_c$) where an edge directed inward is a 0, and an edge directed outward is a 1. Thus, the state values go from 000 to 111. We say a vertex is in a legal state if the weight of all edges pointed inward is greater than or equal to the constraint of the graph.

A vertex gadget contains a single 1×1 tile referred to as the *state tile*, a *transition area*, and a number of *state gadgets* equal to the number of legal states of that vertex. Since there are eight states, there are eight basic paths in the gadget that the state tile could be in representing the vertex's state. Figure 5.11e gives an example vertex gadget (a CL AND vertex) and the possible paths, and also shows the numbering of the states and the corresponding orientations of the original edges for that state. Table 5.5 gives the only move sequences needed for the system.

Flipping an edge is represented by a move sequence performed while in a valid state that moves the state tile from one state path to another, which happens simultaneously in two vertex gadgets since an edge connects two vertices. This edge flip happens in all vertex gadgets, but if the edge is not incident to that vertex, there is no effect on the path of the state tile.



(a) An AND vertex gadget

(e) Goal gadget overview

Figure 5.11: (a) A vertex gadget with the path/transition areas labeled with the corresponding state numbers and the representative edge orientations in a constraint graph. Each state is a different color. Paths that do not change the state of a tile are dotted. The grey lines are invalid states. (b) The necessary vertices for a one-player unbounded game are reversible AND and OR vertices in a constraint graph with constraint 2. The AND vertex has two red edges (weight 1) and a blue edge (weight 2). Directing the blue edge outward requires both red edges to be directed inward. The OR vertex has three blue edges. Only one edge needs to be directed inward. (c) An example of a state transition area the vertex it represents. White rows represent edges that do not change the state of the vertex (they are not incident). Red and blue rows represent the incident edges and the weights of those edges. (d) The state gadgets for two different vertices V_l and V_r. Both vertices share edge 3. V_l represents state 4 of an *OR* vertex. V_r represents state 3 of an *AND* vertex. If edge 1 is selected (the red tile and line), V_l will remain in state 3 while V_r will go to state 7. Selecting edge 3 changes the state of both gadgets. (e) An overview of the layout of the different components for the reduction. The dotted red lines represent the vertex gadgets (not to scale), the green boxes below denote the geometry specific to each vertex to force the state tile into the top row (if in the wrong state) or the bottom row (if in the correct state). The bottom row requires all $|V|$ state tiles in order for a tile to get into the goal location g.

State Transition Gadget. The state transition gadget is the transition area and the concrete that encode legal transitions from a given state. This will be unique to each vertex gadget. These are outlined in Figure 5.11e as states $0-4$. Figure 5.11c shows one of these areas with an explanation of the different paths. There are $|\mathscr{E}|$ levels on the right, where each level represents an edge in the graph. When a $\langle W \rangle$ command moves the state tile left, the tile stops at the blocked spot on that level. All edges not incident to the vertex have no effect on the path of the state tile (the dotted lines in

Flip edge $e_k \in \mathscr{E}$	$\langle\langle E,S\rangle^k,\langle W,N,E\rangle\rangle$
Extraction	$\langle\langle E,S\rangle^{\lvert\mathscr{E}\rvert+1},\langle W,S\rangle\rangle$

Table 5.5: Move sequences for the reduction. $\langle\cdot\rangle^K$ means repeat the sequence K times. $|\mathscr{E}|$ is the number of edges in the original constraint graph, as opposed to $\langle E\rangle$ which is the 'east' command.

Figure 5.11e and white rows in Figure 5.11c). The three edges that are incident will change the path of the state tile when a $\langle S\rangle$ command follows. If it is not a valid edge flip (due to the constraints), the state tile will be permanently stuck in a path representing an invalid state. If the new state is valid, the state tile will be in that state path. Note this will happen for both vertex gadgets representing a vertex incident to the edge chosen. Figure 5.11d shows an example of two state transition areas in two vertex gadgets representing two vertices that share an edge.

Goal Area. An overview of the reduction layout is in Figure 5.11e where the goal area is shown at the bottom of all the vertex gadgets. Once all the tiles are in positions that represent the target configuration, the tiles can be extracted into the goal area. An extraction is made the final level in a state gadget. After extraction the tiles enter the goal area. The goal area consists of two rows. The valid row and the invalid row. The invalid row (top row) traps any tiles that enter when a vertex was not in the specified (in the target configuration) state. The valid row (bottom row) contains the goal position (g in Figure 5.11e). The goal position is $|V|$ positions to the right of the left wall. Thus, $|V|$ tiles must be in this row in order to have the last tile be in this location. In order to have enough tiles to reach the goal position, each vertex must be in the correct state to output the tile to the goal area. This ensures all vertices, and thus the entire graph, is in the specified target configuration.

Lemma 15. *After performing a move sequence to flip an edge, only the two vertex gadgets representing vertices incident to that edge will have their state tile change state paths. All other vertex gadgets will have their state tile stay in the same state path.*

Proof. Since we enumerate all the edges and make each state gadget have an exit point for each edge, a move sequence to select and flip an edge moves all tiles out of their state gadget to the

transition area. We create the transition area for each vertex gadget based on the edges that are connected to that vertex. If flipping an edge causes a vertex to change states we place a concrete block to stop the tile in the state column of the new state. If flipping an edge does not affect a vertex then when that tile leaves the state gadget and goes to the transition area there will be a block of concrete to stop the tile in the state column of the state it was previously in. Since all tiles start at the top of the state gadget, they all move out of the same level, so only two vertex gadgets will have their state tile change state gadgets. We can see this in Figure 5.11d. Flipping the first edge will change the state of the right vertex but not the left. Flipping the third edge edge will change the state of both. □

Lemma 16. *If a vertex enters an illegal state, the representative vertex gadget's state tile will be trapped in an 'illegal' state path and cannot be extracted.*

Proof. If an edge flip would cause a vertex to enter an illegal state the tile will still be sent out of the state gadget into that states column. Since when we create the vertex gadget we block off the right of the top of any illegal state columns once a tile goes to the top or bottom of a state column it can only travel between the both of these. The only way this tile can be removed from this column is if a second tile enters this column. However, there is no way for another tile to enter the vertex gadget so there is no way to extract the tile once it becomes trapped in the column. □

Theorem 14. *Occupancy is PSPACE-complete with only* 1×1 *tiles in the full tilt model.*

Proof. We show Occupancy is PSPACE-hard with only 1×1 tiles by a reduction from Non-Deterministic Constraint Logic. Given an instance of a constraint graph G (with initial configuration C_i) and a goal configuration C_g of that graph, enumerate the edges and vertices of the configuration. For each vertex in the graph create a Vertex Gadget. We create the configuration in the tilt model, S_i, as described above with the goal location g and vertex gadgets laid out side by side above the goal area as shown in Figure 5.11e. The vertex gadgets will have a state transition gadget for each legal state of that vertex. The transition area is built based on the edges of the vertex. Then there exists a

sequence of edge flips to transition C_i to C_g if and only if there exists a sequence of tilt commands that transition board S_i to a board configuration with some tile at location g.

Given a sequence of edge flips to transition C_i to C_g in $G = (V, \mathscr{E})$, $F = \langle f_1, \ldots f_l \rangle$ where $f_i \in \mathscr{E}$, we can directly translate this into the move sequence necessary based on Table 5.5. Each vertex has its state tile start in the starting state of the vertex. Lemma 15 shows we can select any specific edge and perform a move sequence to select and flip that edge to change the state of two vertices and leave all other state tiles in the same state path. Since there exist move sequences to flip edges and change the states of vertex gadgets, a series of edge flips that are a solution to an NCL configuration-to-configuration problem can be turned into a move sequence that changes all vertex gadgets to their goal states which can then be extracted to solve the occupancy problem.

If given S_i and a sequence of tilts $T = \langle t_1, \ldots t_z \rangle$, where $t_i \in \{N, E, S, W\}$, that solved the occupancy problem, the edges to flip could be found from the sequences of Table 5.5. We know by Lemma 16 that any tilt sequence not corresponding to a legal edge flip would trap a state tile, and thus occupancy could not be solved. Thus, if our sequence solves the problem, only legal edge flips were made. Further, to solve occupancy, we need all $|V|$ state tiles in the goal area, meaning all vertex gadgets were in the correct state, as specified in C_g. □

Corollary 4. *Relocation is PSPACE-complete with only* 1×1 *tiles in the full tilt model.*

Proof. If we ask whether we can relocate the state tile in the vertex gadget for v_n to the goal location g, we have an equivalence of the Occupancy problem. □

Corollary 5. *Reconfiguration is PSPACE-complete with only* 1×1 *tiles in the full tilt model.*

Proof. Since state tiles remain in their vertex gadget, they remain in the same order when extracted. The goal configuration is all tiles extracted from the vertex gadgets in the valid row ordered by vertex number. In order to extract all the tiles into the valid row, all gadgets must be in the correct state when extracted. □

5.7 Conclusion

In this paper we presented a hierarchy of shapes that are buildable within the full tilt model. We proved several characteristics about the drop-shape class, then gave an algorithm to decide membership in the class for hole-free shapes. We then provided a universal constructor that strongly builds this class of shapes. We then answered an open question by proving that the Occupancy problem in full tilt is PSPACE-complete even with only 1×1 tiles.

We leave a number of open problems. When considering drop-shape membership, our algorithm does not consider shapes with holes. Does there exist an efficient algorithm to determine membership in $\mathfrak{D}$ for all shapes? Also in defining membership in levels of our hierarchy we only consider one tile type that sticks to itself in determining if a cut is valid. What shapes can be built in lower levels of the hierarchy if more tile types are allowed? Does there exist a tile type hierarchy and how does it relate to the drop shape hierarchy? In regards to Theorem 11, does there exist a tighter bound on the level of the hierarchy a shape must be in? Lastly, for complexity of the occupancy, relocation, and reconfiguration problems, we use a connected board to show PSPACE-completeness. Is the problem easier when limiting the board type to simple or monotone, or does it remain PSPACE-complete?

BIBLIOGRAPHY

[1] *Atomix*. Thalion Software, 1990.

[2] *Mega maze*. Phillips Media, 1995.

[3] J. BALANZA-MARTINEZ, D. CABALLERO, A. CANTU, L. GARCIA, A. LUCHSINGER, R. REYES, R. SCHWELLER, AND T. WYLIE, *Full tilt: Universal constructors for general shapes with uniform external forces*, in 30^{th} Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'19, 2019.

[4] A. T. BECKER, E. D. DEMAINE, S. P. FEKETE, G. HABIBI, AND J. MCLURKIN, *Reconfiguring massive particle swarms with limited, global control*, in Algorithms for Sensor Systems,

2014, pp. 51–66.

[5] A. T. BECKER, E. D. DEMAINE, S. P. FEKETE, J. LONSFORD, AND R. MORRIS-WRIGHT, *Particle computation: complexity, algorithms, and logic*, Natural Computing, 18 (2019), pp. 6751–6756.

[6] A. T. BECKER, E. D. DEMAINE, S. P. FEKETE, AND J. MCLURKIN, *Particle computation: Designing worlds to control robot swarms with only global signals*, in 2014 IEEE Inter. Conf. on Robotics and Automation (ICRA), 2014, pp. 6751–6756.

[7] A. T. BECKER, S. P. FEKETE, P. KELDENICH, D. KRUPKE, C. RIECK, C. SCHEFFER, AND A. SCHMIDT, *Tilt Assembly: Algorithms for Micro-Factories that Build Objects with Uniform External Forces*, in 28th International Symposium on Algorithms and Computation (ISAAC 2017), Y. Okamoto and T. Tokuyama, eds., vol. 92 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2017, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 11:1–11:13.

[8] A. T. BECKER, S. P. FEKETE, P. KELDENICH, D. KRUPKE, C. RIECK, C. SCHEFFER, AND A. SCHMIDT, *Tilt assembly: Algorithms for micro-factories that build objects with uniform external forces*, Algorithmica, (2018).

[9] R. A. HEARN AND E. D. DEMAINE, *The nondeterministic constraint logic model of computation: Reductions and applications*, in Proceedings of the 29th International Colloquium on Automata, Languages and Programming, ICALP '02, London, UK, UK, 2002, Springer-Verlag, pp. 401–413.

[10] S. MANZOOR, S. SHECKMAN, J. LONSFORD, H. KIM, M. J. KIM, AND A. T. BECKER, *Parallel self-assembly of polyominoes under uniform control inputs*, IEEE Robotics and Automation Letters, 2 (2017), pp. 2040–2047.

[11] M. J. PATITZ, *An introduction to tile-based self-assembly and a survey of recent results*, Natural Computing, 13 (2014), pp. 195–224.

[12] A. SCHMIDT, S. MANZOOR, L. HUANG, A. T. BECKER, AND S. FEKETE, *Efficient parallel self-assembly under uniform control inputs*, IEEE Robotics and Automation Letters, (2018), pp. 1–1.

BIOGRAPHICAL SKETCH

Austin Luchsinger was born in Rugby, North Dakota, but grew up primarily in Napa, California. It was there that he completed his primary and secondary schoolwork, graduating from Napa High School in 2008. Austin then enlisted in the United States Marine Corps in 2009, where he served as an avionics technician and operations specialist over the course of 5 years. While serving, Austin met his now-wife, Samantha, and upon his honorable discharge at the rank of Sergeant in 2014, the two of them moved to Samantha's hometown of Edinburg, Texas. Austin went on to attend the University of Texas Rio Grande Valley (UTRGV), where he graduated *magna cum laude* with a Bachelor of Science in computer science in 2018. Alongside his undergraduate studies, Austin began his involvement in academic research with the Algorithmic Self-Assembly Research Group under Dr. Robert Schweller and Dr. Tim Wylie. Motivated to progress in his research investigations, Austin continued his academic training at UTRGV by pursuing a master's degree in computer science. He was awarded the Presidential Graduate Research Assistantship, the university's most prestigious scholarship for graduate students, and graduated with a Master of Science in computer science in 2020. Austin may be reached by email at amluchsinger@gmail.com.