2019

Full Tilt: Universal Constructors for General Shapes with Uniform External Forces

Jose Balanza-Martinez
The University of Texas Rio Grande Valley

David Caballero
The University of Texas Rio Grande Valley

Angel A. Cantu
The University of Texas Rio Grande Valley

Luis Angel Garcia
The University of Texas Rio Grande Valley

Austin Luchsinger
The University of Texas Rio Grande Valley

See next page for additional authors

Full Tilt: Universal Constructors for General Shapes with Uniform External Forces*,†

Jose Balanza-Martinez David Caballero Angel A. Cantu Luis Angel Garcia
Austin Luchsinger Rene Reyes Robert Schweller Tim Wylie

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 weakly 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 deciding if a particle within a configuration may be relocated to another position, and deciding if a given configuration may be transformed into a second given configuration. We show both problems to be PSPACE-complete even when no particles stick to one another and movable particles are restricted to 1×1 tiles and a single 2×2 polyomino.

*This research was supported in part by National Science Foundation Grant CCF-1817602.

†Department of Computer Science, University of Texas - Rio Grande Valley

1 Introduction

The "tilt" model of self-assembly is an elegant and simple model of robotic motion planning and assembly proposed by Becker et. al. [4] 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 tilt term 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 [11,17,21], 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 focuses on 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? This problem has been considered in the restricted case of 1×1 polyominoes that never stick together and was shown to be NP-hard [5] (but is not known to be in NP). A related but distinct problem of minimizing the number of tilts to reconfigure between two configurations has also been considered and shown to be PSPACE-complete. In this paper, we add to this growing understanding of reconfiguration complexity by showing that reconfiguration is PSPACE-complete with 1×1 movable tiles and a single 2×2 movable polyomino. We also show PSPACE-completeness for a modified version of the reconfiguration problem in which we ask whether a particular polyomino may be relocated to a specific position.

1.1 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 [3]. 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 [13]. 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 [20] and the marble based Labyrinth [9].

Micro-scale. It has been shown how to control magneto-tactic bacterium via global signals to move bacteria around complex vascular networks via magnets [8], and how the same type of bacteria can be moved magnetically through a maze [14]. This has a plethora of medical uses, such as minimal invasive surgery, targeted drug payloads, subdermal microconstructions, "targeted delivery of hemotherapeutic agents or therapeutic genes into malignant cells while sparing healthy cells" [19], and "medical intervention in targeted zones inaccessible to catheterization" [15].

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 [12, 23]. Such walkers may be augmented with activation signals to drive the walker forward one step by way of a DNA strand-displacement reaction [22]. 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.

1.2 Our Contributions in Detail. We first show the existence of a (weakly) universal configuration for building any $h \times w$ bounded shape or pattern. The result utilizes simple geometry (all *concrete* tiles are connected), only a single type of bonding particle, and is quadratically smaller in size than the corresponding non-universal construction from previous work [7]. Moreover, this is the first result in the literature that is capable of building any connected shape. However, 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 [7]. A summary of universal shape construction results, along with closely related prior work, is provided in Table 1.

Result	Shape Class	Universal	Tilts	Size	Bonding Complexity	Geometry Complexity	Theorem
Fixed Shape	DROP	No	$O(hw)$[1]	$O(h^3w^3)$	2 labels	Complex	Thm. 6 in [7]
Universal Patterns	All	Weakly	$O(hwk)$	$O(hwk)$	k labels	Simple	Thm. 3.1
Universal Shapes	All	Weakly	$O(hw)$	$O(hw)$	1 label	Simple	Thm. 3.2
Universal Shapes	DROP	Strongly	$O(h^2w)$	$O(h^2w)$	2 labels	Complex	Thm. 4.1

Table 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 type of nonmovable tiles needed in the constructor. A simple object has all nonmovable polyominoes as a single connected shape. A complex object has multiple stationary pieces that are not connected. The **Theorem** refers to where this information is from.

Problem	Shapes	Complexity	Theorem
Relocation	1×1	NP-hard	Thm. 1 in [5]
Tilt-minimization	1×1	PSPACE-complete	Thm. 10 in [5]
Relocation	1×1, 2×2	PSPACE-complete	Thm. 5.1
Reconfiguration	1×1, 2×2	PSPACE-complete	Thm. 6.1

Table 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 **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.

Our next set of results focus on the problem of deciding if a given polyomino may be relocated to another given location on a board, termed the *relocation* problem, and whether a given board configuration may be reconfigured into a second given configuration, termed the *reconfiguration* problem. We show both problems to be PSPACE-complete, even when polyominoes are restricted to be 1×1 and 2×2 pieces that never stick to each other. Our proof for both relies on a reduction from a 2-toggle gadget network game that was recently proven to be PSPACE-complete by Demaine, Grosof, Lynch, and Rudoy [10]. Previous work on relocation has shown the problem to be NP-hard [5] 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 [5]. A summary of our complexity results, along with closely related prior work, is provided in Table 2.

2 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." The geometry of a board is said to be *simple* if the locations in W are a connected set with respect to adjacency in the 2D square lattice, and *complex* otherwise.

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*

[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.

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 $G(a, b) = G(b, a) = 1$.

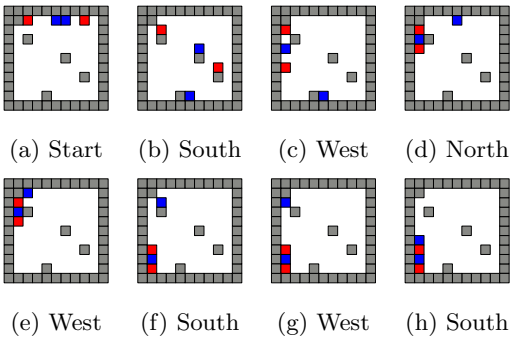
Polyomino. A *polyomino* is a finite set of tiles $P = \{t_1, \ldots t_k\}$ that is 1) connected with respect to the coordinates of the tiles in the polyomino and 2) *bonded* in that the graph of tiles in P with edges connecting bonded tiles is connected. A polyomino that consists of a single tile is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



(a) Start (b) South (c) West (d) North

(e) West (f) South (g) West (h) South

Figure 1: Tilt Example

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

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 3) 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.

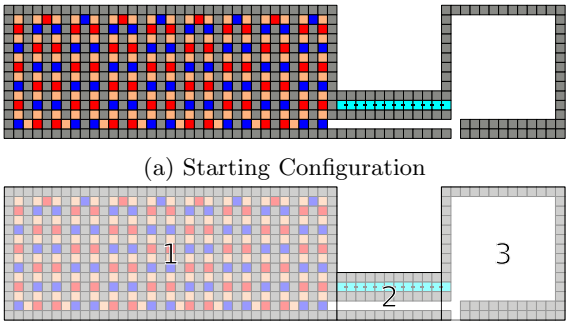
Universal Shape Builder. Given this representation, we say a configuration C' is *universal* for a set of shapes U if and only if there exists a set of configurations $\mathcal{C}$ such that 1) each $u \in U$ is represented by some $C \in \mathcal{C}$ and 2) C' is universal for $\mathcal{C}$. If each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

3 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.

3.1 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 2) which can assemble any binary-patterned rectangle of that size.

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. This chamber becomes split into two by a long horizontal polyomino which we call the *gate*.



(a) Starting Configuration

(b) Sections

Figure 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 *gate chamber*, and section 3 is the *construction chamber*. We call the light-blue polyomino in section 2 the *gate*.

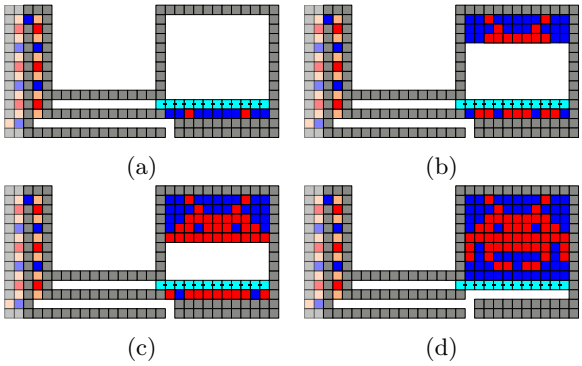
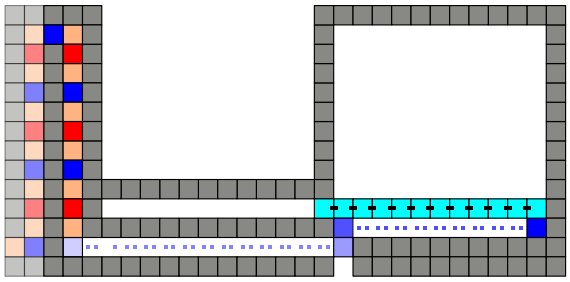


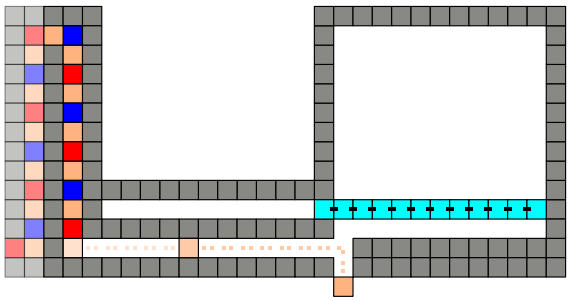
Figure 3: The rectangle construction process at different points. Patterns are built row by row and then added to the final shape. The gate ensures the finished portion will not be changed except when adding a new row. (a) depicts an assembled row of the pattern. (b) and (c) show subsequent configurations after more rows have been added and (d) shows the final assembled pattern.

The upper portion (with dimensions $h \times w$) is where the binary-patterned rectangle will be constructed. The lower portion (with dimensions $1 \times w$) is used to pre-assemble the lines (rows) which are used for that construction.

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



(a) Add Tile



(b) Remove Tile

Figure 4: Basic sequences used in the constructor. The cyclic sequence to advance the fuel is $\langle S, E, N, E, S, E, \dots \rangle$. (a) The sequence to add a tile to the line is $\langle E, N, E \rangle$. (b) The sequence to remove a tile from the system is $\langle E, S \rangle$.

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 $\langle E, S, E, N \rangle$ is required.

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.

Rectangle Construction. Once the line (row) is complete, the next step is to add it to the construction chamber. To do this, the user must open the gate, add the line to the construction chamber, and then close the gate. The sequences for all moves are in Table 3.

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

Table 3: Tilt sequences used for general shape and pattern construction. Some key tilt effects are: any S tilt "loads" a tile into the entrance of the gate chamber, any W tilt opens the gate, and any E tilt closes the gate. Also, 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.

3.2 Patterned Rectangles and General Shapes: Formal Results

THEOREM 3.1. *Given two positive integers $h, w \in \mathbb{Z}^+$, there exists a configuration C which is* weakly 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 weakly represents any pattern $u \in U$.*

Proof. We show this by constructing a configuration $C = (B, P)$, similar to that shown in Figure 2a, where B is the board and P is the 1×1 polyominoes in the fuel chamber in their starting configuration.
The three chambers of the board (fuel, gate, and construction) with scale based on h and w are:

- *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 $h + 5$ and width $w + 2$.
- *Gate chamber.* The gate must be $w+1$ tiles long, which dictates the length of the gate chamber, and a height of only 5 is needed.
- *Fuel chamber.* Let the fuel chamber of this configuration be of height $h + 5$ and length

[2]The term *sand* is inspired by the game Minecraft [16] 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.

$8w + 2$. 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. (Note that only height $h + 2$ is required, but we match the construction chamber height for aesthetics).

By observing the dimensions of the chambers, the height of the board is $h + 5$ and the width of the board is $10w + 4$. So, the size of the board is $\mathcal{O}(hw)$.

Reconfiguration. Consider the set of configurations $\mathcal{C}'$, where each $c' \in \mathcal{C}'$ is similar to that shown in Figure 2a and *weakly* 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, $\forall\ c' \in \mathcal{C}', C \to_* c'$. □

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 3.1. *Given two positive integers $h, w \in \mathbb{Z}^+$, there exists a configuration C which is* weakly 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 configuration which weakly represents any pattern $u \in U$.*

Proof. We use the construction from Theorem 3.1 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 5 shows this process.

THEOREM 3.2. *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 3.1, 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. □

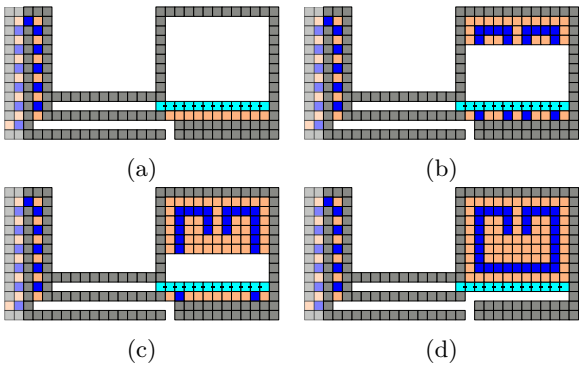


Figure 5: The shape construction process at different intervals. (a) The first row of the shape being built, however, the shape does not need this row and so only sand is added to the row. (b-c) Partially built shape with several disconnected sections of the shape being built simultaneously. (d) The 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 3.2. *Given two positive integers* $h, w \in \mathbb{Z}^+$, *there exists a configuration* C *which is* weakly universal *for the set of shapes* $U = \{u | u \subseteq \{1, \ldots, h\} \times \{1, \ldots, w\}$, *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 3.1, we can add k colors to our fuel chamber, causing the board size and number of tilts to scale with k. □

3.3 Additional Notes

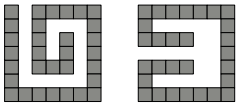
Keeping the Undesirable Tiles. One aspect of the constructor that may be undesirable is the idea of removing pieces from the board. Although not shown due to space constraints, the constructor can easily be modified to handle the tiles in numerous ways.

- Trash. The easiest solution is to have a chamber with another gate (where E opens it) that the tiles are thrown down into. The chamber must be large enough to accommodate pieces arbitrarily sticking together.
- 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. The top of the $h \times w$ construction chamber can be replaced with a width $w + 1$ gate that requires a unique tilt sequence to open and close. Thus, the gate can be opened, an N tilt releases the shape, and then the gate can be closed to begin the assembly of the next shape.

4 Drop Shapes

Next, we consider a class of shapes discussed in [7] 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 [7]. Figure 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 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.

4.1 Universal Drop Shape Builder: Construction. Figure 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 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.
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 10a and Table 4, respectively.

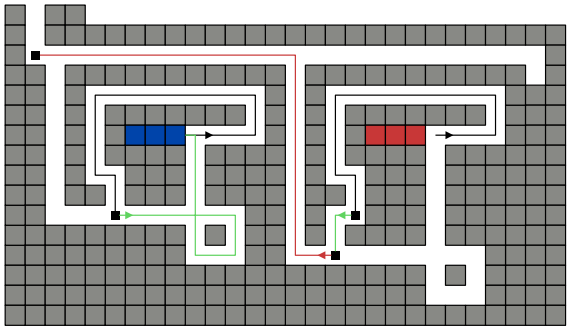


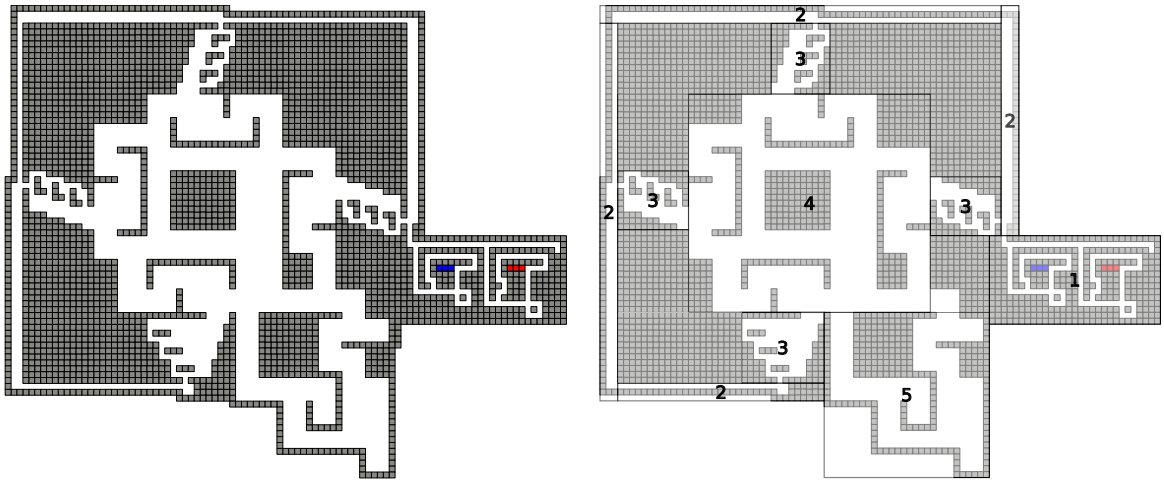
Figure 8: Tile selection gadget. Each tile is pulled out with the sequence $\langle E, N, W, S, E, S\rangle$, and stops at the first square. Then the left tile type (blue) is either pulled out of the gadget or put back in the storage area. This shows it being added back to the storage with $\langle E, S, W, N, W, S\rangle$. 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 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 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.



(a) Starting Configuration (b) Sections

Figure 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*.

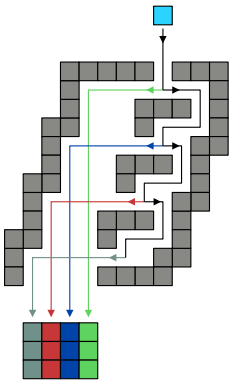


Figure 9: The column selection gadget for drop shapes. Assuming the shape to build is at a fixed location, this gadget allows any column to be selected to drop the new tile onto. The number of columns to drop from in this gadget determines the size of the shape we can build. Thus, this is for a drop 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.

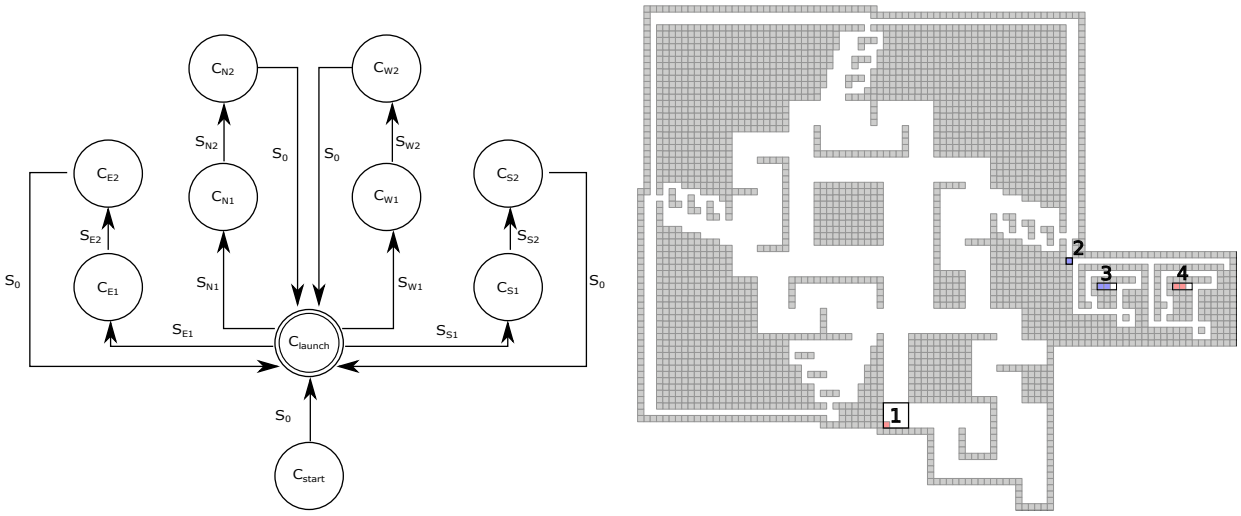
4.2 Universal Drop Shape Builder: Theorem and Proof.

THEOREM 4.1. *Given two positive integers h, w, 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 $u \in U$.*

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 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

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}	$\langle W,N\rangle^j + \langle E,N,W,S,W,N,E,S,E,S\rangle$

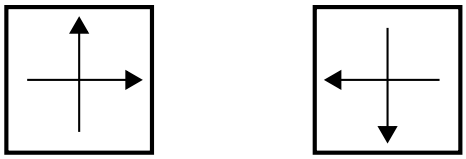
Table 4: The sequence of tilts used in the flowchart (Figure 10a). s_0 denotes the tile selection from $i+j+1$ different tile types and chambers. The sequence also places the current polyomino in the correct area for the next sequences (Figure 10b). For area 2 in Figure 7b, s_{E1}, s_{N1}, s_{W1}, s_{S1} 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 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. (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.



(a) Crossing 2-Toggle (C2T) (b) C2T State Change

Figure 11: (a) This gadget represents paths traversable by the robot. The robot can only move in the direction the arrow is pointing, and traversing a gadget will reverse the directions of both arrows. (b) Alternate state of a C2T gadget.

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, $C \rightarrow_* C'$, 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 10a. □

5 Relocation

Our next contribution is a continuation of the work done in [4–6]. In their papers, they discuss the relocation problem in a tilt-based system which asks whether a tile can be moved from one location to another specific location. They show this problem is NP-Hard. Additionally, they show that finding the *optimal* tilt sequence to relocate a tile to another location is PSPACE-complete. In this section, we show that deciding if a tilt sequence exists to move a polyomino in a given configuration to a given location is PSPACE-complete. Further, in line with many geometric problems, we show that deciding if a given configuration is reachable from a starting configuration is also PSPACE-complete. Our hardness proofs rely on a recent result in [10]. They show that the *puzzle solvability* problem, which asks whether or not a robot can traverse a system of specific types of gadgets, is PSPACE-Complete. We reduce from a gadget model proposed by them that is used to analyze the complexity of traversing particles. We first describe the puzzle solvability problem which we will reduce from to show PSPACE-Hardness.

First we summarize one of the gadget models introduced in [10], and define several of the key terms.

- 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.
- 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 11a shows how the gadget is represented with two arrows denoting the current traversable tunnels in the gadget. Figure 11b shows the gadgets other state with the tunnels both reversed.
- 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.

5.1 Relocation Gadget Model Preliminaries.

Now, we show a tilt gadget that behaves like the C2T gadget in order to simulate its complexity for our reduction. Figure 13 shows an overview of the gadget. Figure 13a shows the basic sections of the gadget which we describe later. The main idea is that a single 1×1 tile, referred to as the *state tile*, is confined to the gadget, and its possible paths represent what state the toggle is in. Figures 13b and 13c show the paths of the state tile and robot polyomino (the polyomino traversing the toggles),

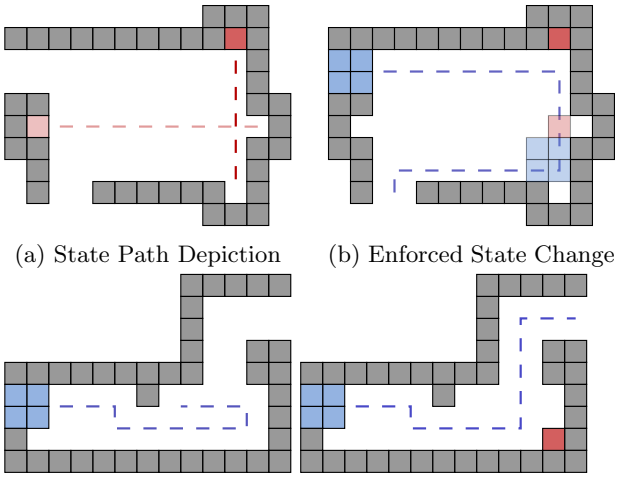
respectively. Figure 12 gives an overview of the two movable polyominoes in our system. As before, we will begin with a high-level overview of the gadget, and proceed to a more detail explanation.

(a) State Tile (b) Robot Polyomino

Figure 12: (a) The state tile is a 1×1 tile that exists within the gadget. The possible path of the state tile in the toggle gadget determines whether the robot polyomino can move up/right or down/left. Once it is used by the robot polyomino to pass through the gadget, it ends up on the alternate path meaning the gadget has flipped. (b) The robot polyomino is the 2×2 polyomino that we are attempting to move from one location in the board to another location.

In order to reduce from the puzzle solvability problem, we recreate the individual properties of a C2T gadget in our tilt assembly model. We must create a structure that can restrict traversal through itself to certain directions and allow for those directions to be changed after a traversal has been made. In our representation, a tunnel traversal through a C2T gadget is represented by relocating a 2×2 polyomino from an initial location to a final location in the gadget via a sequence of tilts. Our gadget's structure restricts traversal with specific geometry architecture that causes a 2×2 polyomino to become stuck if the gadget is traversed incorrectly. For a 2×2 polyomino to traverse through these otherwise untraversable gadgets, we require the assistance of a 1×1 helper tile confined within our gadgets. Figure 14 shows an example of a pathway not reachable by the 2×2 polyomino without the geometric assistance of a 1×1 tile. In Figure 14 we see an example of a 1×1 confined pathway that is changable into another with the assistance of the 2×2 polyomino. This depicts one of two different states that the 1×1 tile can be in. By combining these elements we create a complex gadget which can limit 2×2 polyominoes traversals in 2 perpendicular directions, and limits a 1×1 tile in a confined set of pathways within the gadget.

Robot Polyomino. The *robot polyomino* is a 2×2 polyomino that traverses through a puzzle. Figure 13c shows all paths of the robot in a gadget. The dotted lines show its path without the help of the state tile.



(a) State Path Depiction (b) Enforced State Change

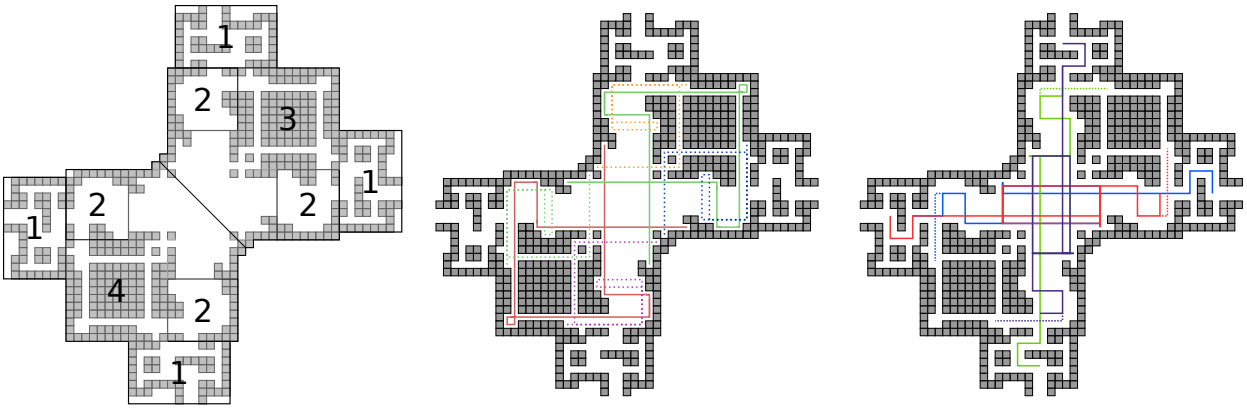
(c) Robot Limited Path (d) Robot Assisted Path

Figure 14: 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.

State Tile. The relocation gadget has a *state tile*, which is a single tile that is trapped inside the gadget. The state tile can freely move in one of the solid lines shown in Figure 13b, 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 architecture is diagonally symmetrical and allows 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 14.

Entrance/Exit Chambers. Marked as section 1 in 13a, 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



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

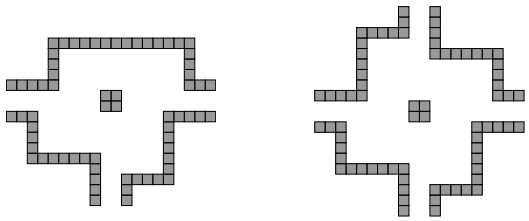
Figure 13: (a) Relocation sections where 1) represents entrance/exits locations, 2) represents areas where the robot polyomino becomes stuck if unassisted by the state tile, and 3) and 4) represent the NE and SE state tile areas. State and robot paths are shown in (b) and (c), where the state tile is 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, assisted by the state tile, or traverses the dotted lines if unassisted by the state tile

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 11.

Assistance Chamber. Marked as section 2 in 13a, an *assistance chamber* is the place where a robot polyomino and a state tile meet to assist the robot polyomino's traversal through the gadget. A robot polyomino can reach the assitance 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 15 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. Section 3 and 4 in 13a, *state chambers* store the state tiles and dictate the state of the gadget. These chambers allow the state tile to move freely through its north side to its east side and vice versa, or from its south side to its west side and vice versa. When the state tile moves to the opposite state chamber, the state of the gadget changes along with the positions of the entry and exit location points. See Figure 13b for the possible paths of the state tile in its two states.

Intersections. To allow robots to change directions in the paths between gadgets, we must have

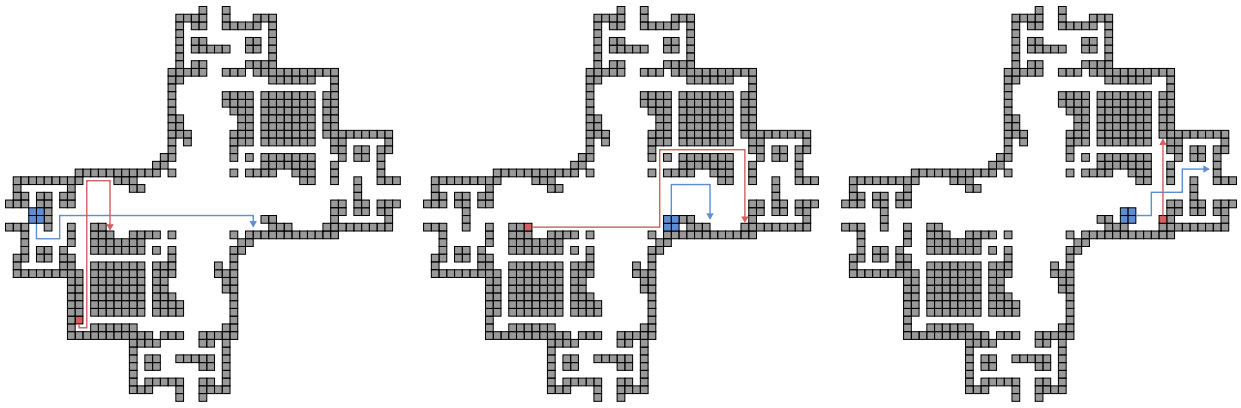


(a) 3-way intersection (b) 4-way intersection

Figure 16: 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.

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 16a and 16b, respectively.

Like a puzzle consisting of C2T gadgets, a robot polyomino traverses through a system of relocation gadgets via their directed tunnels and the wires that connect them. For a directed tunnel (a, b) to exist in our gadget means that a sequence of tilts exists such that we can relocate our robot polyomino from location a to location b. The location of the state



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

Figure 15: Example of a traversing sequence and state change when a gadget is in the state from Figure 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 13b to section 3's green path; Therefore, the gadget was toggled.

tile implies the state of our gadget by determining which directed tunnels exist in our gadget. In order for a robot polyomino to traverse a relocation gadget, the state tile must be in the correct path. Being in the NE path implies the existence of directed tunnels (N, S) and (E, W), while being in the SE path implies the reversal of those tunnels. When the robot polyomino enters the gadget correctly and gets assisted by the state tile to traverse safely, it forces the state tile to change its path and toggle the state of the gadget.

5.2 Relocation Gadget Results. The relocation gadget's architecture itself cannot prevent the robot polyomino from traversing a direction that does not comply with our tunnels. Thus, the gadget is designed such that the robot can not traverse the gadget incorrectly or it will get stuck within the gadget. We can list all the ways the robot can try to *cheat* in a relocation gadget, i.e., move in a way that does not comply with the rules of a C2T. Following, we give the ways the gadget could fail and show why this can not happen. The different types of ways to cheat or break the gadget are:

- The robot polyomino traverses the gadget in a way where a directed tunnel from its entrance to its exit did not exist prior to the entrance of the robot polyomino.
- The robot polyomino traverses the gadget without forcing the gadget to change states.
- The state tile leaves the gadget.

LEMMA 5.1. *The robot polyomino can traverse from location A to location B in a gadget if and only if an implied directed tunnel from location A to location B exists at the time of its entrance.*

Proof. We used a computer simulation and proved by exhaustion that the only way the robot polyomino can move through the system is in a way that complies with the directed tunnel in our gadgets. We define a graph $G = (V, E)$ where each vertex is a pair (r, s) where r and s are robot and state tiles positions in the gadget, respectively, and each edge is a directed 3-tuple (v, v', d) edge from vertex v to v' with label d where $d \in \{N, S, E, W\}$. We create a graph with a seed vertex v, where v is a possible starting configuration of the robot polyomino and state tile, and recursively perform tilts in all four directions from that vertex, adding new vertices that did not already exist in the graph. Adding a directed edge $e = (v, v', d)$ from v to v' with label d means you can get from configuration v to v' by tilting in direction d. We call a *valid starting configuration* a configuration such that an implied tunnel exists from the robots location to the opening on the opposite side of the

gadget, while an *invalid starting configuration* is a configuration such that no implied tunnel exists from the robot's location. The computer program generates a set of graphs showing that for every seed vertex which represents a valid starting configuration there is a path that allows us to traverse through the gadget. It also shows that for every invalid starting configuration there is no way to traverse the gadget at all (the robot and state polyominoes can not leave); the robot can only leave the same way it came in, or enter a "stuck" configuration, from which there is no way to exit the gadget. Information about the verification and sourcecode are available at https://github.com/asarg/TumbleTiles. □

LEMMA 5.2. *If a tunnel is traversed in the gadget, the state of the gadget is changed (the tunnels are reversed).*

Proof. The state of our gadget is inferred from the path that the state tile is in. This means we must ensure that if a robot polyomino traverses a tunnel, then the state tile must switch to the other path without switching back alone. Using the same method of proof by exhaustion, there is no sequence that will allow the robot to completely traverse a tunnel without switching the path the state tile is in. Sourcecode: https://github.com/asarg/TumbleTiles. □

LEMMA 5.3. *The relocation gadget correctly implements the behavior of the C2T gadget.*

Proof. Lemma 5.1 and 5.2 show that our gadget can simulate the main properties of a C2T gadget. Upon inspection we can see that in addition to these two properties, a state tile is never able to leave a gadget, as the entrances/exits are designed in a way that 1×1 tiles can not get through, and enter a hallway they can not exit. We have proven that the only way to traverse a gadget is through a directed tunnel, and that if a tunnel is fully traversed then the gadget's state must change, Therefore our gadget implements the properties of a C2T gadget. □

THEOREM 5.1. *Given an initial configuration C, a polyomino t at location a in C, and an empty location b in C, it is PSPACE-complete to decide whether there exists a sequence of tilts to reconfigure C into a configuration in which t has moved from location a to location b.*

Proof. First, we observe that this problem is in PSPACE. Given 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 puzzle solvability problem from [10]. A puzzle is a graph of connected gadgets including crossovers where the goal is to find a way through the maze. Thus, a puzzle of C2T gadgets has a solution if and only if there is a sequence of tilts that will relocate the robot from a to b through the puzzle of relocation gadgets.

If a C2T puzzle has a solution then any movement through a tunnel can be transformed to a sequence of tilts that allow traversal through the relocation gadget. Any movements through a single wire can be transformed to tilts through a hallway, while movements through an intersection of wires can be transformed to a sequence of tilts through the intersections mentioned above. Since we can convert all the moves in the solution to a C2T puzzle to moves in our relocation gadget system, a solution to a C2T puzzle implies a solution to our relocation problem.

Suppose there is a sequence of tilts that relocate the robot from a to b from starting configuration C. We can easily convert this sequence of tilts to a sequence of moves that a robot in a C2T puzzle can perform. We can map the sequence of tilts necessary to get through our relocation gadget to a simple N, E, S, W direction for the robot to move through the C2T gadget. A similar simplification is needed for each intersection. Thus, we can create directions to move through the C2T puzzle by simplifying the sequence of tilts and thus solve the C2T puzzle. □

6 Reconfiguration

In [6], 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, termed the *reconfiguration* problem, is PSPACE-complete. Our reduction is similar to the reduction for the relocation problem. But for the reconfiguration problem we must provide a unique final configuration for the problem input, 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 location of the state tile. We address this issue by expanding on the relocation gadget to allow for a final tilt sequence that positions the state tile for each gadget, regardless of the state, into a unique identical position, thereby providing a polynomial time computable target configuration for the reduction.

- Reconfiguration Gadget. The Reconfiguration gadget is a relocation gadget with additional architecture. Each state tile has additional pathways that lead to an inescapable chamber on the perimeter of the gadget. The robot polyomino maintains the same pathways it held 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 answer the reconfiguration problem as they convert all gadgets to one global unique configuration. To insert all state tiles to this ring, the user can move the state tile through the new architecture (depicted in Figure 19a) into the reconfiguration ring, and render all gadgets inoperable.

- State Tile Restarting Positions. We define *restarting positions* for all state tiles (depicted by the solid state tiles in Figure 19a). We initialize all state tiles of the same state to the same position in the gadget. This could be either of the two state positions in section 4 (Figure 18a). 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 17) is a small chamber gadget placed on wires and between every pair of reconfiguration gadgets to ensure the state tiles will not enter the perimeter hallway. The intermediate gadget gives the traversing robot room to make tilts in all directions. After every reconfiguration gadget traversal, the robot can move freely in the corresponding intermediate wire and reposition the state tiles to their restarting positions.

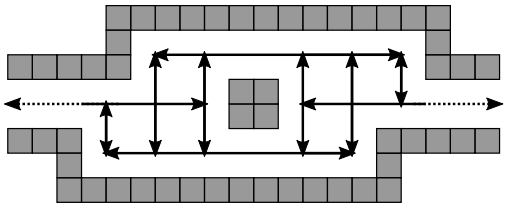
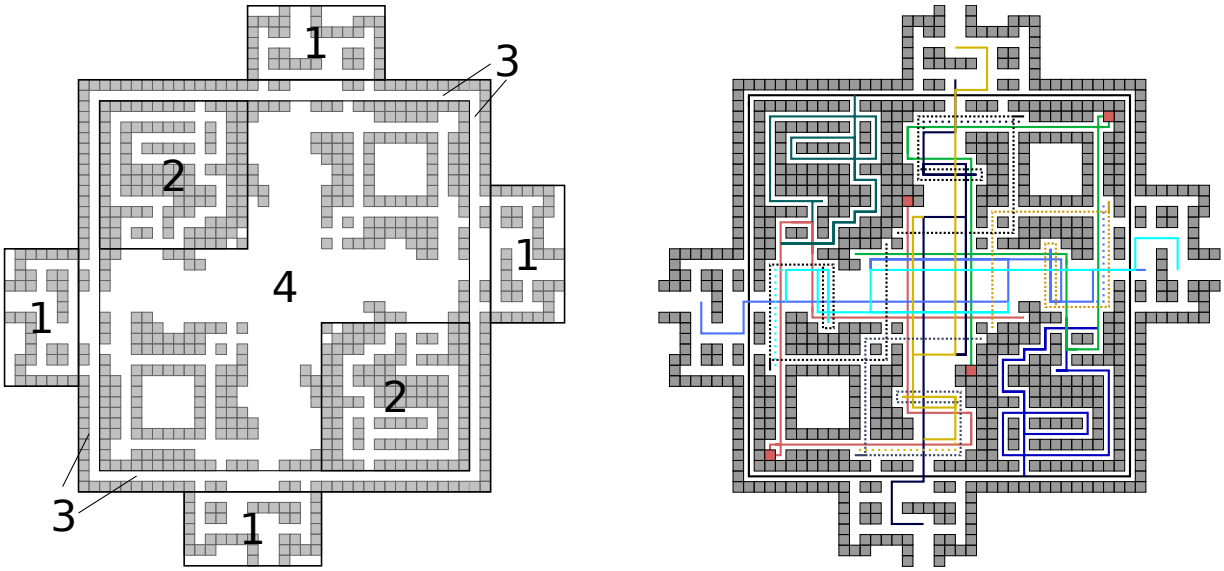


Figure 17: Intermediate Wire

6.1 Reconfiguration Theorems and Proofs. Using proof by computer to show that the relocation gadget is immune to cheaters, we show that in a system of reconfiguration gadgets the state tiles can be moved to their initial positions using the intermediate wire. 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 initial positions. We include this in our results to save the traversing robot from accidentaly placing a state tile in its reconfiguration ring before the user is ready to "freeze" all gadgets. Therefore, everytime the robot enters a gadget the state tiles will be in one of the two initial positions we depict in Figure 18b. The relocation gadget is a subset of the reconfiguration gadget. Before answering the reconfiguration problem, we must first answer the relocation problem by applying a sequence of tilts that relocates a robot polyomino to its goal location. When the relocation problem has been answered we then proceed to move all the state tiles through the added architecture hallways until they are in the gadget's reconfiguration ring. This converts all of the gadgets to one configuration type. The only cause of concern for this gadget is the possibility of unintentionally moving the state tile into the reconfiguration ring while the robot is traversing through the system of gadgets. However, with a specific tilt sequence performed after each gadget traversal, we can ensure this will never happen.

LEMMA 6.1. *For every system of gadgets there exists a sequence of tilts that allows the robot to traverse the gadget system without prematurely moving the state tile into the reconfiguration ring.*

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



(a) Reconfiguration Gadget Sections

(b) Reconfiguration Gadget Positions

Figure 18: (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 dotted lines are the accessible by assistance.

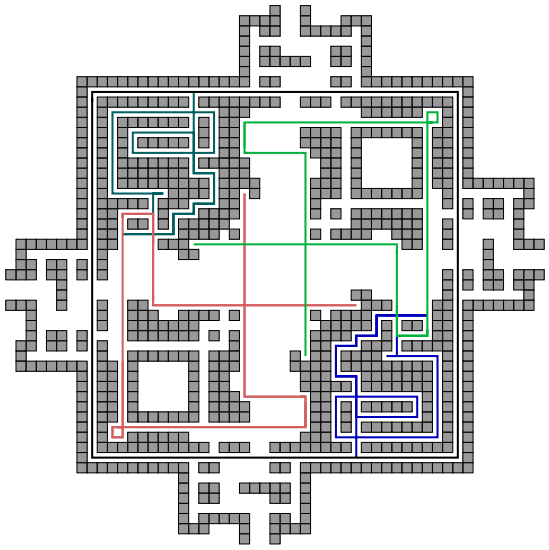
gadget, or trivially be one tilt away. This is true not only for the gadget being traversed, but also for every gadget in the system. This ensures that while traversing through the hallways no state tiles are forced into a reconfiguration ring. From the top-right and bottom-left state tile restarting 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 6.1. *Given a board B and configurations C and D, it is PSPACE-complete to decide whether C can be reconfigured into D.*

Proof. Since the reconfiguration gadget shares the same structure and properties as the relocation gadget, it also behaves as a C2T gadget in our reduction. We use an exhaustive computer simulation, as in the proof of the relocation gadget, to ensure the gadget works as described. This, in addition to Lemma(6.1) shows that answering the relocation problem is feasible using reconfiguration gadgets. After answering the relocation problem using reconfiguraton gadgets, we can proceed to move all state tiles into the 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 destination of the single polyomino. □

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 and reconfiguration problems are PSPACE-complete if both 1×1 and 2×2 blocks are considered. The complexity of this problem is only known to be NP-hard if restricted to 1×1 movable blocks [5]. In [5] the authors showed that 1×1 blocks have substantial



(a) Reconfiguration state tile paths.

Figure 19: 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.

limitations such as the impossibility of implementing a certain type of fanout gate. This might be evidence that this variant is not PSPACE-complete. On the other hand, the existence of necessarily exponentially long tilt sequences for reconfiguration shown in [5] provides some evidence that the problem does not lie within the class NP. An alternate modification is the consideration of simple (connected) geometry in the blocking structure of the board. Does the problem remain PSPACE-complete with this restriction, 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 [18], 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 for their thorough, careful, and constructive feedback.

References

[1] *Atomix*, Thalion Software, 1990.
[2] *Mega maze*, Phillips Media, 1995.
[3] Aaron T. Becker, *Parallel self-assembly of polyominoes under uniform control inputs*, `https://www.youtube.com/watch?v=G93H1Tecj-w`, 2018.
[4] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Golnaz Habibi, and James McLurkin,

Reconfiguring massive particle swarms with limited, global control, Algorithms for Sensor Systems (Berlin, Heidelberg), Springer Berlin Heidelberg, 2014, pp. 51–66.

[5] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Jarrett Lonsford, and Rose Morris-Wright, *Particle computation: complexity, algorithms, and logic*, Natural Computing (2017).

[6] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, and James McLurkin, *Particle computation: Designing worlds to control robot swarms with only global signals*, (2014), 6751–6756.

[7] Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, Christian Rieck, Christian Scheffer, and Arne Schmidt, *Tilt Assembly: Algorithms for Micro-Factories that Build Objects with Uniform External Forces*, 28th International Symposium on Algorithms and Computation (ISAAC 2017), Leibniz International Proceedings in Informatics (LIPIcs), vol. 92, 2017, pp. 11:1–11:13.

[8] Aaron T. Becker, Yan Ou, Paul Kim, Min J. Kim, and Agung Julius, *Feedback control of many magnetized: Tetrahymena pyriformis cells by exploiting phase inhomogeneity*, 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nov 2013, pp. 3317–3323.

[9] Brio, *Labyrinth game.*

[10] Erik D. Demaine, Isaac Grosof, Jayson Lynch, and Mikhail Rudoy, *Computational complexity of motion planning of a robot through simple gadgets*, 9th International Conference on Fun with Algorithms, FUN 2018, June 13-15, 2018, La Maddalena, Italy, 2018, pp. 18:1–18:21.

[11] Dave Doty, *Theory of algorithmic self-assembly*, Communications of the ACM **55** (2012), no. 12, 78–88.

[12] Cheulhee Jung, Peter B. Allen, and Andrew D. Ellington, *A simple, cleated dna walker that hangs on to surfaces*, ACS Nano **11** (2017), no. 8, 8047–8054, PMID: 28719175.

[13] Phillip Keldenich, Sheryl Manzoor, Li Huang, Dominik Krupke, Arne Schmidt, Sandor P. Fekete, and Aaron T. Becker, *On designing 2D discrete workspaces to sort or classify 2D polyominoes*, 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

[14] Islam S.M. Khalil, Hazem Abass, Mostafa Shoukry, Anke Klingner, Rasha M. El-Nashar, Mohamed Serry, and Sarthak Misra, *Robust and optimal control of magnetic microparticles inside fluidic channels with time-varying flow rates*, International Journal of Advanced Robotic Systems **13** (2016), no. 3, 123.

[15] Sylvain Martel, Ouajdi Felfoul, Jean-Baptiste Mathieu, Arnaud Chanu, Samer Tamaz, Mahmood Mohammadi, Martin Mankiewicz, and Nasr 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), no. 9, 1169–1182.

[16] Mojang, *Minecraft.*

[17] Matthew J. Patitz, *An introduction to tile-based self-assembly and a survey of recent results*, Natural Computing **13** (2014), no. 2, 195–224.

[18] Arne Schmidt, Sheryl Manzoor, Li Huang, Aaron T. Becker, and Sndor Fekete, *Efficient parallel self-assembly under uniform control inputs*, IEEE Robotics and Automation Letters (2018), 1–1.

[19] Rajni Sinha, Gloria J. Kim, Shuming Nie, and Dong M. Shin, *Nanotechnology in cancer therapeutics: bioconjugated nanoparticles for drug delivery*, Molecular Cancer Therapeutics **5** (2006), no. 8, 1909–1917.

[20] ThinkFun, *Tilt: Gravity fed logic maze.*

[21] Damien Woods, *Intrinsic universality and the computational power of self-assembly*, Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences **373** (2015), no. 2046.

[22] David Zhang and Georg Seelig, *Dynamic dna nanotechnology using strand-displacement reactions*, **3** (2011), 103–13.

[23] Chao Zhou, Xiaoyang Duan, and Na Liu, *A plasmonic nanorod that walks on dna origami*, Nature Communications **6** (2015), 8102–8102, 26303016[pmid].