Robotic Swarms: Assembly and Complexity

Angel Adrian Cantu Suarez
The University of Texas Rio Grande Valley

ROBOTIC SWARMS: ASSEMBLY AND COMPLEXITY

A Thesis

by

ANGEL ADRIAN CANTU SUAREZ

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

ROBOTIC SWARMS: ASSEMBLY AND COMPLEXITY

A Thesis
by
ANGEL ADRIAN CANTU SUAREZ

COMMITTEE MEMBERS

Dr. Tim Wylie
Chair of Committee

Dr. Robert Schweller
Committee Member

Dr. Bin Fu
Committee Member

Dr. Jingru Zhang
Committee Member

May 2020

ABSTRACT

Cantu Suarez, Angel A., Robotic Swarms: Assembly and Complexity . Master of Science (MS), May, 2020, 53 pp., 3 tables, 43 figures, 11 references.

This thesis focuses on the assembly of robotic swarms that move according to some global signal in a model called the “tilt” model. The model consists of a 2D board that contain open and blocked spaces, along with tiles or polyominoes that move toward a signaled cardinal direction. We look at two variations of this model called the *single-step* and *full-tilt* model, where the elements move single distances or maximally, respectively, when a signal is send. We show different methods of *shape* construction, defining board configurations that are *universal* for a set of shapes. Afterwards, we analyze different computational problems that arise from the tilt model in the latter chapters of this thesis. This thesis consists of published work in algorithmic and computational complexity conferences, along with new material that improve upon old contributions.

DEDICATION

For my dearest Camilo and Manola, who I long to see once more.

ACKNOWLEDGMENTS

I wish to express my gratitute towards Dr. Tim Wylie and Dr. Robert Schweller for giving me the opportunity to express myself creatively throughout my years as a student and the opportunity of authorship of important work. To Dr. Dongchul Kim, for treating my projects with seriousness and for always believing in me. To Mr. Peter-James Ehimika, whose help guided me through my years as a student, and who always sought to correct my frivolous point of views. To ASARG, the best research team anyone could wish for, my friends, and for shaping me as the person I am today. And to my family, for reasons that will not fit within the margins of this paper.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

When considering robots at the micro- and nano-scale, power and bandwidth limitations often make individual robot control infeasible. This notion has caused recent research to change its course to consider abstract models of motion planning that use global signals that control all robots uniformly and move all particles in a particular direction when given a movement command (unless movement is prevented by a blocked space). After gaining a better understanding of the power of this model, efforts were steered toward the problem of engineering particular environments to reconfigure particles into desired shapes. While some research have considered the reconfiguration of robots into particular "forms", others have had an emphasis on "building shapes" with these particles. Recent developments for "particle swarm shape builders" have focused on *universal constructors*, which are environments where movements can transform a particle swarm from a starting configuration into another from a given universe of configurations. Previous work have shown intuitive universal constructors, along with a notion of *weak* and *strong* representation of that shape. Shapes build weakly allows for a relaxed notion of shape construction where "extra" particles are allowed to exist in the environment. On the other hand, shape construction in which all particles in the environment must be considered in the final configuration is called a strong representation. The beauty of this 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 Becker et al. [1] have constructed a modular, reconfigurable board with both geometry

and sliding components, which they have demonstrated with video walkthroughs. 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 [7]. 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 and the marble based Labyrinth [4].

Micro-scale It has been shown how to control magneto-tactic bacterium via global signals to move bacteria around complex vascular networks via magnets [3], and how the same type of bacteria can be moved magnetically through a maze [8]. 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", and "medical intervention in targeted zones inaccessible to catheterization".

Nano-scale Promising potential applications for motion planning 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 [6, 11]. Such walkers may be augmented with activation signals to drive the walker forward one step by way of a DNA strand-displacement reaction [10]. 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 a motion planning model.

CHAPTER II

THE MODEL

The “tilt” model is an elegant and simple model of robotic motion planning and assembly proposed by Becker et. al. [2] 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 in the respective direction until reaching an obstacle. These mechanics allow us to define the two models that are used throughout this paper: the *single-step* and *full-tilt* model.

2.0.1 Definitions

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- often referred to as “concrete” or “walls.” We classify the different possible board geometries according to the following hierarchy:

- Connected: A *connected* board is one whose set of open spaces O is a connected shape.
- Simple: A connected board is *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.

Tiles. A *tile* (or “robot”) 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 $G : \Sigma \times \Sigma \rightarrow \{0,1\}$ 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.

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 $C \rightarrow_1 C'$) if applying one tilt in some direction d to C results in C'. We define the relation $\rightarrow_*$ to be the transitive closure of $\rightarrow_1$. Therefore, $C \rightarrow_* C'$ 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.

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 $C' \rightarrow_* C_i \ \forall \ C_i \in \mathscr{C}$.

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.

Reconfigurable Universal Shape Builder We say a set of configurations $\mathscr{C}$ are a *Reconfigurable Universal Shape Builder* for a set of shapes $\mathscr{S}$ if and only if 1) each $S \in \mathscr{S}$ is represented by a unique $C \in \mathscr{C}$, and 2) $\mathscr{C}$ is a reconfigurable set.

Reconfigurable Universal Shape Builder Given a universal configuration C, the worst-case step complexity is the maximum number of steps required to reconfigure C into some element from its universe set. Consider a universal configuration C over a set of configurations U. For each $u \in U$, let $d(C,u)$ denote the length of the smallest step-sequence from C to u. The worst-case step complexity of C over U is defined to be $\max\{d(C,u) | u \in U\}$. We extend this notion to reconfigurations performed under the tilt transformation, and refer to the maximum number of tilts required for reconfiguration as the worst-case tilt complexity for the universal configuration. The worst-case step complexity for a reconfiguration set is defined similarly, being the longest minimum step sequence between any pair of configurations within the set.

Reconfigurable Universal Shape Builder We define the worst-case step complexity for a reconfigurable set similarly. Consider a reconfigurable set $\mathscr{C}$. The worst-case step complexity for $\mathscr{C}$ is the maximum number of steps between two configurations in $\mathscr{C}$. For each $C_i, C_j \in C$, let $d(C_i,C_j)$ denote the length of the smallest step-sequence from C_i to C_j. The worst-case step complexity of $\mathscr{C}$ is defined to be $\max\{d(C_i,C_j) | C_i, C_j \in \mathscr{C}\}$. Again, we can extend this notion to reconfigurations performed under the tilt transformation to define the worst-case tilt complexity for reconfigurable sets. The worst-case step complexity for a reconfiguration set is defined similarly, being the longest minimum step sequence between any pair of configurations within the set.

2.0.2 Problems in the Tilt Model

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 step/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 step/tilt sequence 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 step/tilt sequence such that $C \rightarrow_* C'$?

CHAPTER III

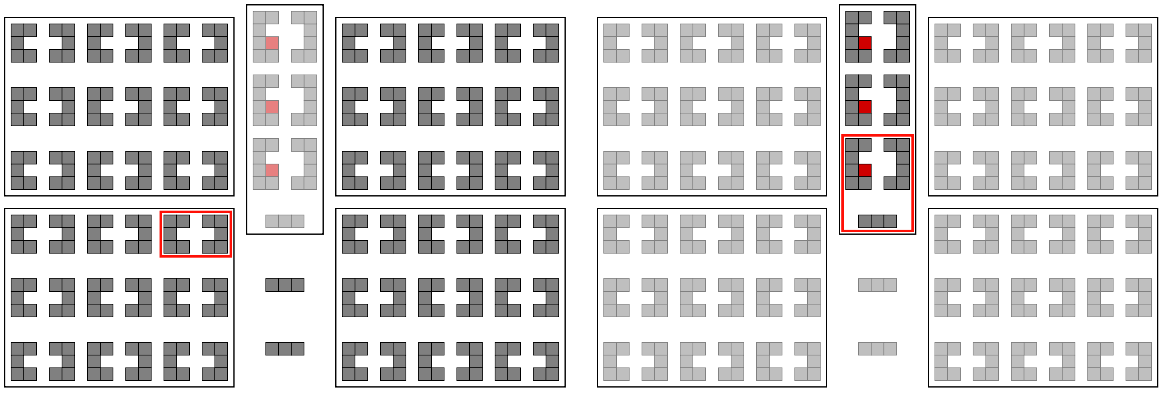
ASSEMBLY IN THE SINGLE-STEP MODEL

This chapter is based on the results for the construction of size-*n* shapes and patterned rectangles, along with a gadget used to coalesce a group of spacially separated robots.

3.1 Assembly of Size-*n* shapes

The mechanics of the constructor introduced here is designed to mimic an additive manufacturing method of construction where structures are gradually built by material depositors that move around a surface ejecting material and building the object piece-by-piece. An alternative interpretation of this method is to have material that is itself mobile that gets placed on the surface by a *fixed* material depositor, where structures are built when the ejected material moves itself around the depositor where the depositor would have otherwise moved itself if it were mobile.

The construction area consists of four $n \times n$ groups of *pixel* gadgets organized in a grid-like fashion, each occupying a 5×3 region on the board used to hold a single tile in place. The construction area is divided into a left and right $2n \times n$ region due to the allocation of the *fuel* gadgets, each a 5×4 region initialized with an output tile for a total of *n* gadgets, spaced out in the manner illustrated in Figure 3.1a. In this way, shapes are built when the output tiles from the fuel gadgets are held in pixel gadgets that are positioned relative to the tiles' position in the shape. All tiles are placed in the construction area by the fuel *depositor* gadget, the bottommost fuel gadget that utilizes the 1×3 line of concrete tiles beneath it to disalign the fuel with the one being deposited. Moreover, all tiles enter the construction area through the *head* pixel gadget, which is the pixel gadget diagonal to the fuel depositor. Given the dimensions of the construction area, a size-*n* shape can be translated such



(a) Pixel Gadgets (b) Fuel Gadgets

Figure 3.1: Illustration of the board. Outlined in red are the (a) head pixel gadget and the (b) fuel depositor gadget.

Name	**Sequence**
Insert	$\langle E, S^4, W^7, S^2, W, E, N, W, S \rangle$
Move Up	$\langle N^5, W, S, E, N, W, S \rangle$
Move Down	$\langle S^5, W, E, N, W, S \rangle$
Move Left	$\langle N^3, W^6, S^3, W^8, S^2, W, S, W, E, N, W, S \rangle$
Move Right	$\langle N^3, E^6, S^3, E^8, S^2, E, S, E, W^2, E, N, W, S \rangle$

Table 3.1: Step sequences used for the construction of size-*n* shapes.

that any position in the shape can be put into the head pixel gadget without the shape extending beyond the construction area. This is true so long as the step sequences, as they are defined in Table 3.1, are used in the shape construction process.

Tiles are inserted into the construction area by performing the *insert* sequences, causing the withdrawn tile to disalign itself from the rest of the tiles in the fuel gadgets and simultaneously bring them to the fuel gadget below it. This sequence also places the tile inside of the head pixel gadget while maintaining the tiles already in the construction area inside of their pixel gadgets. Tiles can be moved around the construction area using the other sequences, where the *move up* and *move down* sequences are straightforward, but the *move right* and *move left* sequences are more complex. These sequences will move tiles to the pixel gadgets in the given direction, while considering that some tiles need to travel across to the left or right region of the construction area. The first thing the

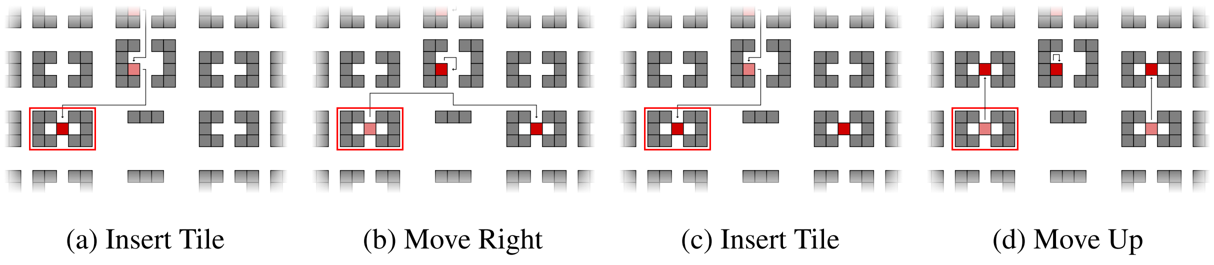

(a) Insert Tile (b) Move Right (c) Insert Tile (d) Move Up

Figure 3.2: A simple example of how the given step sequences can be used to insert and move tiles inside the pixel gadgets.

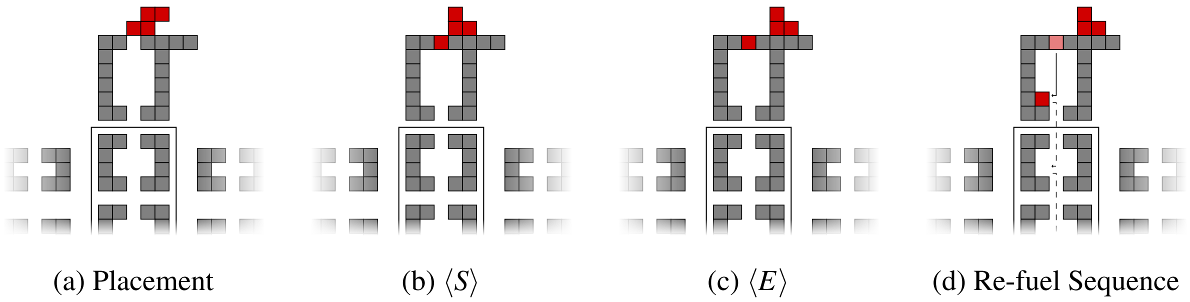

(a) Placement (b) $\langle S \rangle$ (c) $\langle E \rangle$ (d) Re-fuel Sequence

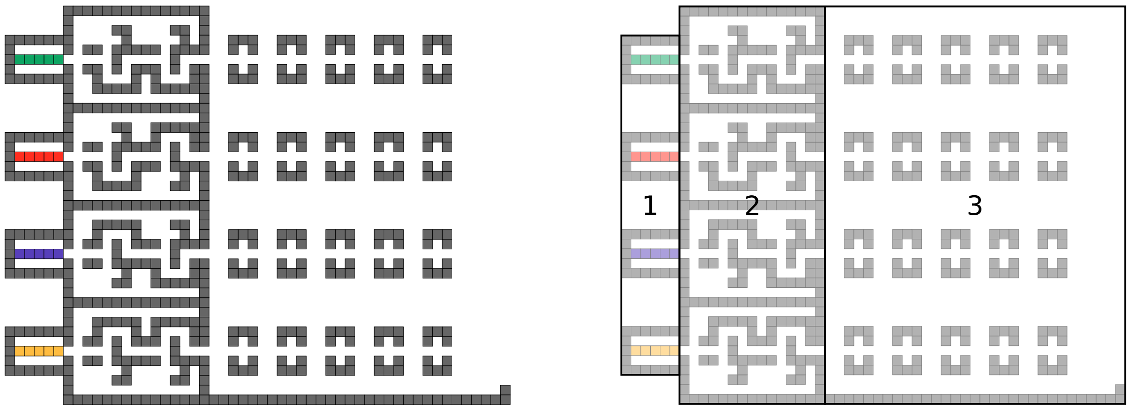
Figure 3.3: (a-c) Removing a single tile from the shape using the topmost opening of the gadget. (d) Performing the *re-fuel* sequence will move the tile to where the arrow points. The dotted arrows alternate paths that tiles would take if they resided in alternate locations.

sequences do is move the tiles that are in the left or right regions to their respective pixel gadgets, followed by the migration of the tiles that move from region to region into to the pixel gadgets of that region. The way these tiles move can be seen in Figure 3.2. Any construction of a size-n shape, using the board configuration presented in this section, is therefore describable as some combination of these step sequences.

3.1.1 Reconfigurable

The universal shape constructor can be further extended to a reconfigurable universal shape builder. The main idea behind reconfigurable configurations is the decomposition of a group of robots and placing them back into the fuel gadgets.

A shape can be decomposed with an additional gadget placed above the topmost fuel gadget as it is shown in Figure 3.3. This gadget consists of a similar structure as the fuel gadgets, though



(a) Constructor (b) Sections

Figure 3.4: (a) The linear patterns constructor and (b) the different sections. Section one consists of the fuel chambers. Section two consists of bit-selector gadgets. Section three consists of pixel holding chambers, as well as a concrete floor where the line will be assembled.

with more space inside the gadget and an additional length $n-2$ horizontal line of concrete tiles extending the top area of the gadget. Shapes moved to the top of this gadget can have their tiles removed one-by-one by placing a tile directly above the top opening and executing the sequence $\langle S, E \rangle$. By performing the *re-fuel* sequence $\langle S^4, W \rangle$, the tile just removed will be moved towards the bottom-left corner of this gadget. When this is done for the next tile, the tile inside the gadget will be moved simultaneously towards the topmost fuel gadget and be placed on the bottom-left corner of that fuel gadget, alike the second tile just placed in the gadget above it. When this is repeated for every tile in the shape, every tile that was once in the shape will have made its way back into the fuel gadget, and thus another size-n shape can be constructed.

3.2 Patterned Size-n Shapes

We now focus on the construction of patterned shapes, starting with the assembly of linear $1 \times n$ shapes over k colors. We construct a universal configuration with worst-case run time of $\mathcal{O}(n \log k + k)$, which is reasonably close to the lower bound of $\Omega(n \log k + \sqrt{k})$, and optimal in the case where $n \geq k$. The linear pattern constructor is made up of three sections: *fuel chambers*, *bit selectors*, and *holding chambers*.

Fuel Chambers. This section of the constructor consists of the fuel chambers, where each are $3 \times n$ open spaces surrounded by concrete with an opening on the center right, each chamber containing a $1 \times n$ line of robots of a particular color. Using the opening on the right side of each chamber we can "chop" off one robot at a time by placing them there and moving up or down to disalign the rest of the robots. By chopping off a robot from each chamber in parallel, we transmit a column of k differently colored robots into section two.

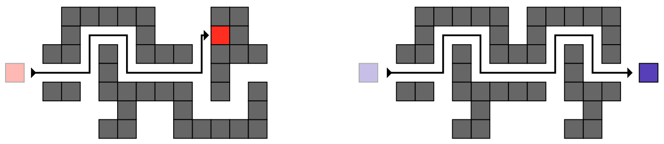

(a) Not Selected (b) Selected

Figure 3.5: Bit selectors usage of extracting one robot via execution of its unique bit string.

Bit Selectors. The *bit selectors* are gadgets used to assign a unique bit string to each colored robot entering the section. These bit strings are created by the unique combination of two smaller gadgets called the *up-select* and *down-select*. Each of these smaller gadgets has an open space path from one side to the other such that each of their paths are the opposite of the other, where the traversal of a robot through one gadget will get the other robot in the other gadget stuck in the notch structure. This idea is demonstrated in Figure 3.5. For all k robots entering this section from the fuel chamber, we can design a unique combination of up-select and down-select gadgets such that traversal of any robot through their respected gadgets will yield that robot on the other side, while all others stay within their gadgets. Therefore, bit selectors consisting of $\log k$ bits each are needed to yield a unique bit-string per robot, each of which creates an open space path from one side to the other of length $\mathcal{O}(\log k)$.

Holding Chambers. Each individual robot enters section three through the left side at possibly different heights since each colored robot comes from a different bit selector. There are nk holding chambers in the output area of the bit selectors, where each holding chamber is a 1×3 open space surrounded by concrete tiles, with an open space path from the center left to right. These chambers

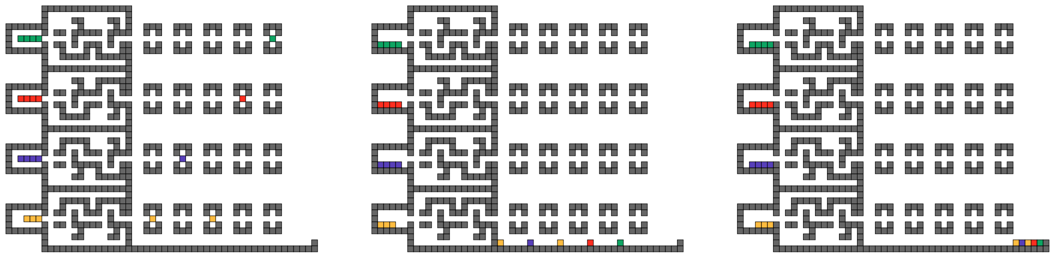

(a) Holding Chambers (b) Extraction (c) Line Building

Figure 3.6: Line building depicted.

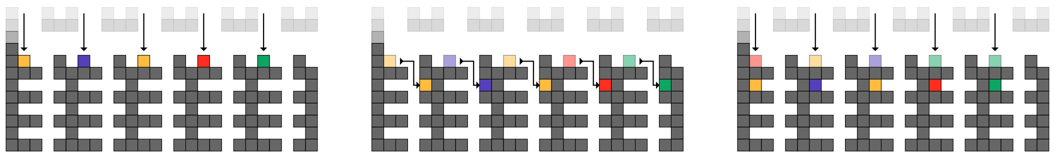

(a) Extraction (b) Placement (c) Extraction

Figure 3.7: Line holders depicted. After each line is built and extracted from the holding chambers, we place them in the first row of the line holder. This, in parallel, will move each line already within the line holder down into the next row.

hold the robots in place while other robots get extracted from the fuel chambers. After outputting a robot from a bit selector gadget, we place the robot in the closest holding chamber to the right. After placing the new robot in a holding chamber, we address the unselected robots that were blocked in the bit selectors. The unused robots are placed back into the fuel chambers by the sequence $\langle W^4, N^2, W^{\mathcal{O}(\log k)}, S^8, W^{\mathcal{O}(\log k)}, N^4, W^2, N, E^{\mathcal{O}(\log k)} \rangle$. After returning the unselected robots to their fuel chambers, we can continue the construction process. The sequence to extract robots and traverse the robot through the bit selector gadgets also moves all robots in a holding chamber to the next holding chamber on their right. After the n^{th} robot has been placed in section 3, we combine them by extracting them from the holding chambers and placing them all on the concrete floor. Then, we push them together using the single concrete tile on the right of this floor. Figure 3.6 shows this sequences.

Line Holders. To generalize the simple line pattern constructor to general shapes over k colors, we can simply add the *line holders* section to the configuration. For general shapes, we replace the south concrete floor of section 3 of the line pattern builder with the line holder depicted in Figure

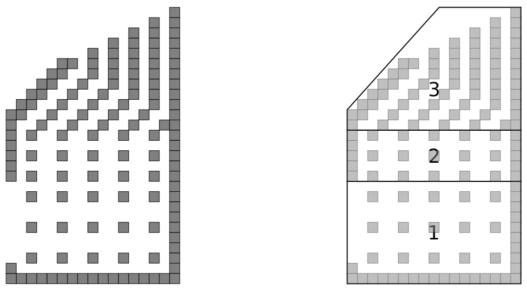


Figure 3.8: (a) An example funneling gadget for a 3×3 shape. (b) Basic functional sections of the funneling gadget.

3.7. As shown, each new line built can be moved into the line holder. If some lines are inside the line holder already, those lines will move in parallel to the next chamber below whenever a new line is added to the line holder. After all lines have been built, we can easily extract them, yielding essentially a general $w \times h$ pattern, but with a constant vertical and horizontal gap between pixels.

3.3 Funneling Gadget

One thing to notice about the constructors aforementioned is that the robots are placed in their appropriate location relative to one another, though spacially separated by some constant amount of spaces.

The *funneling* gadget is designed to take a group of robots separated by a constant amount of spaces and coalesce them into a desired shape. The architecture and sections of the funneling gadget are illustrated in Figure 3.8. Let α and β be constants equaling the largest number of vertical and horizontal spaces, respectively, that separates a robot from its neighbor in the shape. The first section of the funneling gadget reduces β so that the largest horizontal separation between two robots is two. Section two takes the group of robots from the former section and reduces α until it is one. The third section finally reduces α and β to zero, and outputs the group of robots as the desired shape outside the funneling gadget. However, this process skews the shape in one direction. Thus, our constructor builds a shape skewed in in the *opposite* direction, thus negating the skewing effect of the funneling gadget.

Section One. Section one consists of a grid-like organization of concrete tiles that are themselves

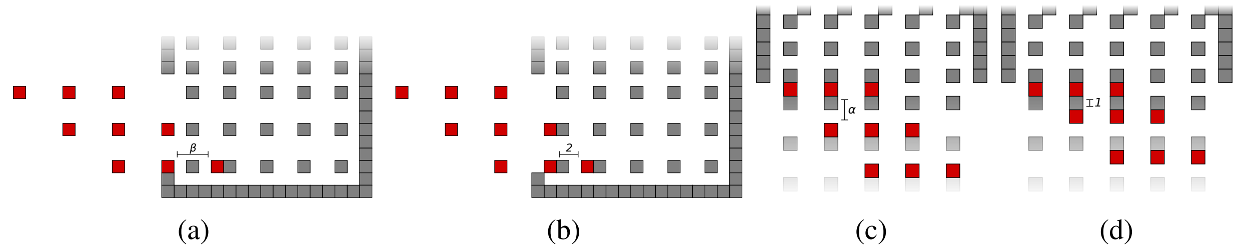


Figure 3.9: (a-b) Reducing the horizontal distance between the rightmost column of robots. (c-d) Reducing the vertical distance between the topmost row of robots.

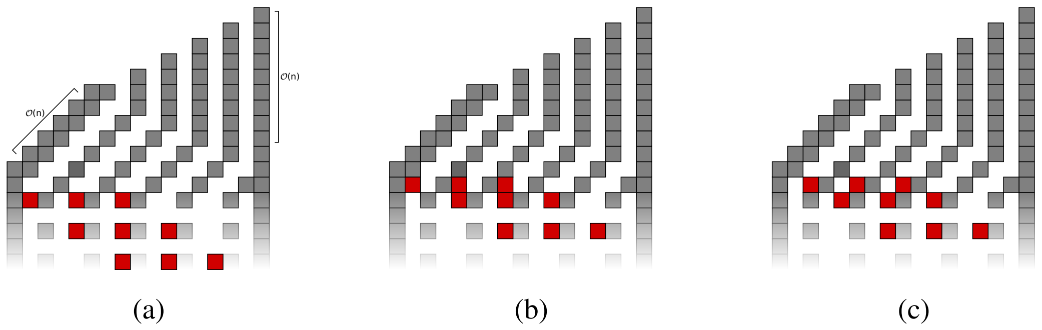


Figure 3.10: (a-b) Making the rows of robots adjacent by using section three.

spaced out vertically by α but horizontally by two, as shown in Figure 3.9. To reduce β to two, we place the rightmost column of the group of robots between the two leftmost columns of concrete tiles (Figure 3.9b). By stepping in the $\langle E \rangle$ direction enough times, the second-to-rightmost column of the group of robots will meet the leftmost column of section one, reducing the spaces between the two columns of robots. After repeating this process for every column of robots, section one will contain within itself the group of robots vertically separated by α and horizontally separated by a distance of two.

Section Two. Section two is a grid-like configuration of concrete tiles that are vertically separated by one space and horizontally separated by two spaces. The same basic process is applied here, but we instead place the rows of robots in between the rows of concrete tiles and perform sufficient steps in the $\langle N \rangle$ direction, as shown in Figure 3.10.

Section Three. By positioning the group of robots in the third section as depicted in Figure 3.10a, stepping twice in the $\langle N \rangle$ direction will cause the topmost rows of the group of tiles to meet. This is repeated for every row by first stepping in the $\langle E \rangle$ direction, followed by two steps in the $\langle N \rangle$

direction. After every row has been made adjacent, repeating the step sequence $\langle N, E \rangle$ will output the robots from the funneling gadget, bringing together each column of robots and outputting the desired shape at the top of the gadget (Figure 3.11).

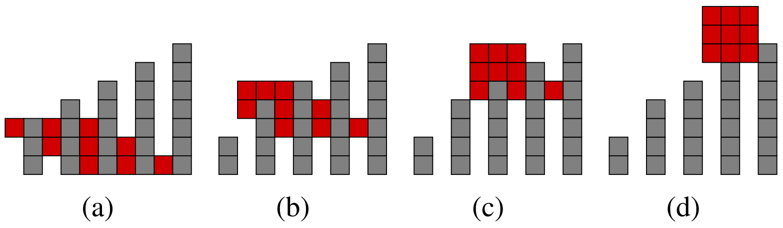


Figure 3.11: Repeating the sequence $\langle N, E \rangle$ will yield the shape on the outside of the funneling gadget.

CHAPTER IV

ASSEMBLY IN THE FULL TILT MODEL

This chapter is based on the results for the construction of size-n shapes (*weakly*), patterned rectangles, and any *drop shape*.

4.1 Assembly of Size-n Shapes

In this section we present a configuration that is a *weakly* reconfigurable universal shape builder for size-n shapes under tilt transformations. The universal constructor presented achieves optimal $\Theta(n)$ construction time to *weakly* build size-n shapes. The construction is based off the single step construction, but with added complexity to deal with the less precise tilt operation.

Tthe construction area consists of four groups of $n \times n$ pixel gadgets organized in a grid-like fashion separated by a column of n fuel gadgets located in between the left and right regions of the construction area. This particular constructor is initialized with an $n \times n$ bounding box of helper tiles inside the pixel gadgets in the position shown by the tile in Figure 4.3a. Shapes in this model are build by replacing the helper tiles of the bounding box with output tiles from the fuel gadgets, gradually placing them in their appropriate position in the bounding box, surrounding them with helper tiles.

Pixel gadgets here are significantly more complex than the single-step version, consisting of three sections that are each used at for different points in the construction process. Sections one and two of the pixel gadgets are made of concrete tiles that aid to send and receive tiles from adjacent pixel gadgets in any given directions. The tiles on the board can be moved around the construction area using the tilt sequences defined in Table 4.1 and shown in Figure 4.2, where the arrows depict where

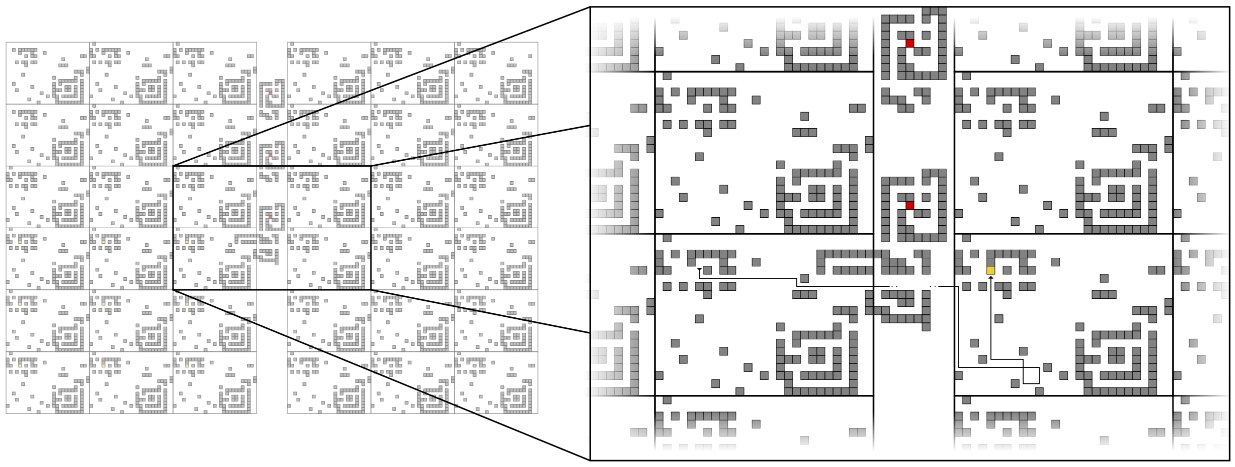


Figure 4.1: The construction area of the board. The pixel gadget boundaries are outline to show how they are placed on the board.

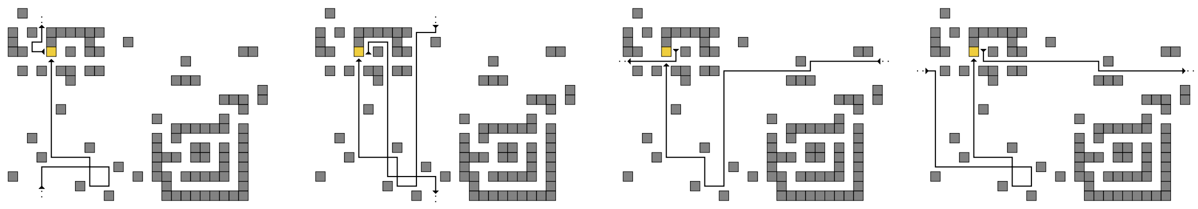

(a) North sequence (b) South sequence (c) West sequence (d) East sequence

Figure 4.2: Tilt sequences to send and receive robots.

the tile will exit a pixel gadget and enter the neighboring pixel gadget of a given direction. When output tiles are to be placed inside the construction area, tiles in the bounding box are placed first in the third section of their pixel gadget to prevent any decomposition of the bounding box.

This constructor includes some pre- and post-processing procedures shown in Figure 4.5 that are used during the construction process. The *prepare pixel* sequence is used before performing the *insert pixel* sequence in order to move all the tiles of the bounding box inside the third section of their respective pixel gadget. The *insert pixel* sequence for this constructor ejects output tiles from the fuel depositor gadgets onto the concrete structure below shown in Figure 4.3d which will lead the ejected tile inside the head pixel gadget, allowing it to interfere the helper tile, discarding the helper tile from the board. The *reposition* sequence is used after inserting the *insert pixel* sequence

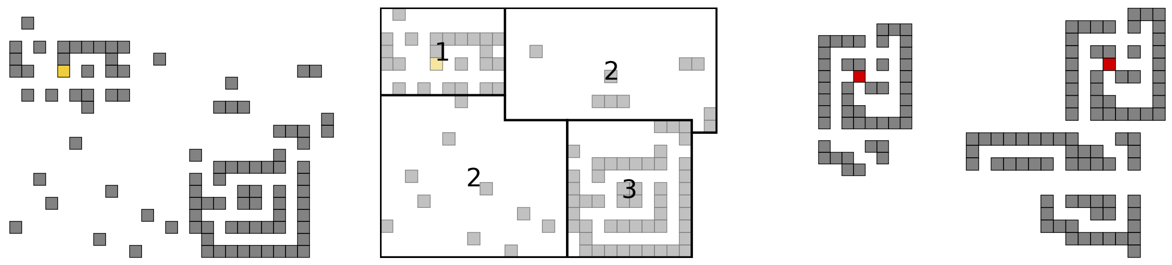


Figure 4.3: (a) Pixel Gadget. (b) Pixel gadget sections. (c) Fuel Gadget. (d) Fuel Depositor Gadget.

Name	Sequence
Insert Pixel	$\langle E,N,W,N,(W,S)^2,(E,N,E,S)^2\rangle$
Reposition	$\langle (W,N)^2,E,S,(E,N)^2,W,N,E,N,(W,S)^2,(W,N)^2\rangle$
Prepare Board	$\langle W,S,(E,S)^2,W,N\rangle$
Move South	$\langle E,N,(E,S)^2,W,S,(W,N)^2\rangle$
Move East	$\langle (E,S)^4,(W,N)^2\rangle$
Move West	$\langle E,S,(W,S)^2,(W,N)^2\rangle$
Move North	$\langle W,N,E,N,E,S,(W,N)^2\rangle$

Table 4.1: Tilt sequences used during the constrution process.

is used in order to withdraw the tiles of the bounding box from section three and allow them to continue moving around the construction area. After all of the output tiles have been placed in the construction area, the bounding box can be withdrawned from the pixel gadgets with the sequence $\langle W,N,W,N,E\rangle$ (Figure 4.4a), causing all of the tiles to be launched towards the right side of the board. Placing concrete tiles on the right side of the construction area that are aligned with the launching path of the pixel gadgets will cause all of the columns to combine in the manner shown in Figure 4.4b, which can be followed by a $\langle S\rangle$ tilt to combine all of the rows using the concrete shown in Figure 4.4c, thus yielding the shape.

4.1.1 Reconfigurable

Achieving reconfigurable universality is not so different from the single-step method, though this version requires more concrete tiles to accomplish it. First, instead of discarding the helper tile from the board as it was previously mentioned, we can instead make it take the path shown in Figure

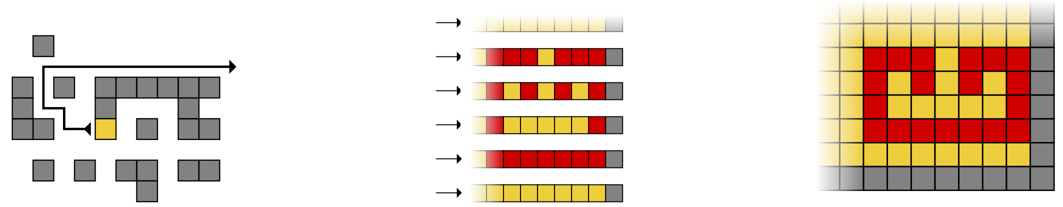

(a) Extract Shape (b) Combine Columns (c) Combine Rows

Figure 4.4: (a) Removing the elements from pixel gadgets. (b) The columns of the bounding box will combine via the collision with the concrete tiles after performing the sequence Extract Shape. (c) Performing a $\langle S \rangle$ tilt will combine all rows of the bounding box.

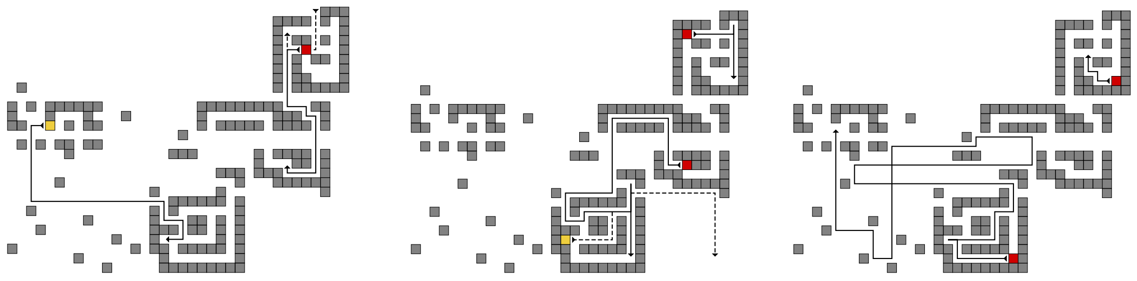

(a) Prepare Board sequence (b) Insert Pixel sequence (c) Reposition sequence

Figure 4.5: The dashed line in (a) shows the path the tile from the fuel gadget above will take. The dashed line in (b) depicts the path the helper tile will take when substituting the robot for it.

4.6a by including the concrete tile beneath the direction of the dotted arrow. As the *reposition* sequence is being performed, this helper tile will be sent to collide with the concrete tiles shown in the figure at the topmost fuel gadget, allowing the helper tile to enter the gadget. It follow that after all the output tiles have been placed on the board, the fuel gadgets will now contain n helper tiles. Moreover, after the rows and columns of the bounding box have been put together, we can move the bounding box into the *dispenser* gadget in order to begin removing one column at a time from the bounding box and placing them back into the pixel gadgets. We can place this dispenser gadget directly above the output region of the board, placing the bounding box inside with a simple $\langle N \rangle$ tilt. Simply repeating the *move west* sequence will cause each column, starting with the leftmost one, to be sliced off the bounding box and sent through the dispenser gadget. The dispenser gadget will separate the tiles and send them through a path that will collide with single concrete tiles that

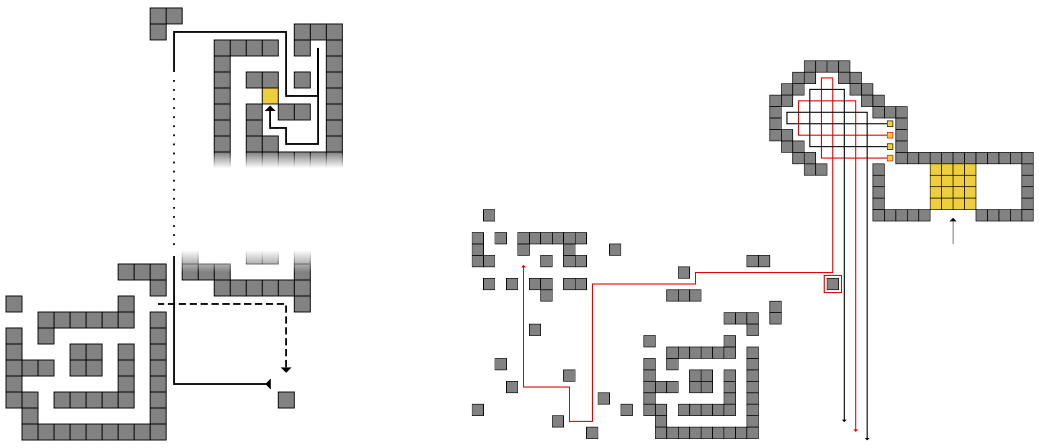

(a) Re-fuel (b) Dispensor Gadget

Figure 4.6: (a) Performing the *reposition* sequence will cause the disarded helper tile to enter the topmost fuel gadget. (b) The paths that the dispensor gadget sends the tiles of the bounding box.

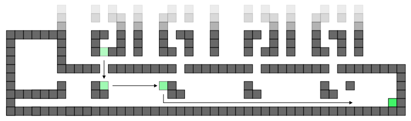
are aligned with the pixel gadgets on the east side of the board such that finishing the *move west* sequence will cause the tiles to enter the pixel gadgets. We repeat the *move west* sequence until all of the tiles are back into the construction area. To return the output tiles back into the fuel gadgets, we simply re-build the same shape (now in a different translation, but connected nonetheless), and repeat this process.

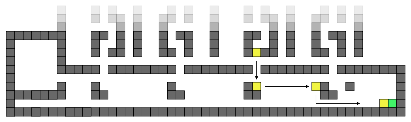
4.2 Patterned Rectangles

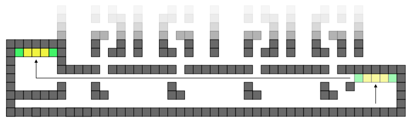
Next, we present the construction details for our optimal universal k-color size-n shape constructor. The high level idea of this construction is that it builds shapes patterns pixel-by-pixel, row-by-row.

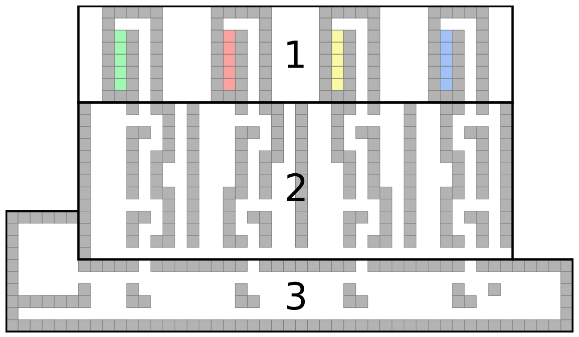
Fuel Chambers. Section one of the constructor (shown in Figure 4.7a) consists of *fuel chambers*, each consisting of concrete structures that hold a set of k-colored robots. Each chamber holds a total of n robots of a particular color $x \in \{1,2,\ldots,k\}$. We can send a group of k robots to the second section by executing $\langle N,E,S \rangle$ tilt sequence.

Bit String Module. Section two of the constructor (shown in Figure 4.7a) consists of *bit selector* gadgets which are attached to the fuel chambers and allow for a fast method to choose from the set of k-color robots. The bit string module is composed of single or many *bit selector* gadgets:


(b) Insert Robot


(c) Insert Robot


(d) Line Placement



(a) Sections

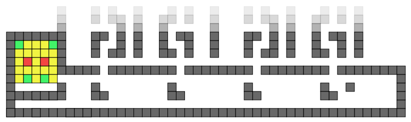
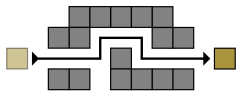
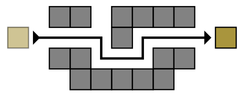
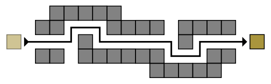

(e) Patterned Shape

Figure 4.7: An overview of the k-color constructor for $k = 4$. (a) This is an 4-color pattern constructor in its starting configuration. (b) We refer to various sections of the configuration by different names. Section one is the *fuel chamber*, section two has the bit string gadgets, and section thre is the *line* and *construction chamber*.


(a) Up Select


(b) Down Select


(c) Robot Kept

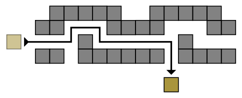

(d) Robot Discarded

Figure 4.8: (a-b) Bit selector gadgets. The entrance and exit locations are at the left and right ends of the gadget, respectively. (c-d) Bit Selector Gadget traversal example. (c) A successful traversal of the gadget. (d) A failed traversal of the gadget.

the *Up Select* and *Down Select*, as shown in Figures 4.8a and 4.8b, respectively. It is easy to see how the simultaneous traversal of robots in both gadgets will lead to one robot being discarded, depending on which robot is taken to exit the gadget through the eastern opening, as depicted in Figure 4.8. This traversal allows us to select one of the k colors in $\mathcal{O}(\log k)$ tilts. For the purpose of this construction we rotate the bit string module 90° clockwise.

Line and Construction Chamber. Section three of the constructor (shown in Figure 4.7a) consists of the *line chamber* and *construction chamber*. The robots that sucessfully traverse section two will

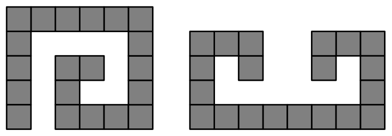


Figure 4.9: Drop Shapes examples. (a) This polyomino is buildable with the drop shape method, whereas (b) is a polyomino that is not a constructable drop shape. From a fixed single tile seed, it is not possible to build (b) by adding one tile at a time from a cardinal direction.

be placed in the line chamber by executing $\langle E,S,E\rangle$, where it is added to line construction. After a line is built, we can send the line to the construction chamber by executing the $\langle N,W,N\rangle$ sequence. Each row of a rectangle can therefore be constructed consecutively, and send to the construction chamber so that it fits inside its shape.

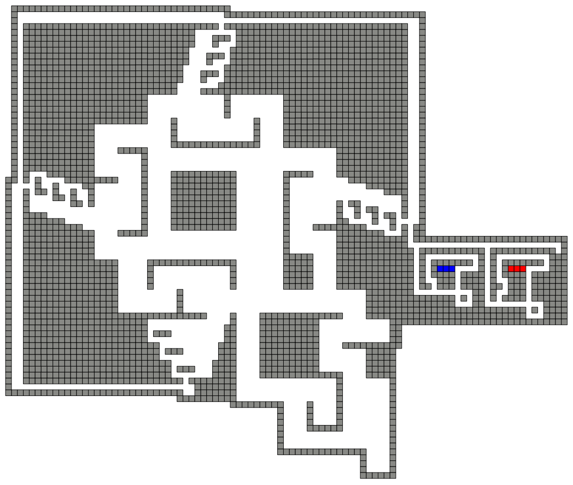
4.3 Drop Shapes

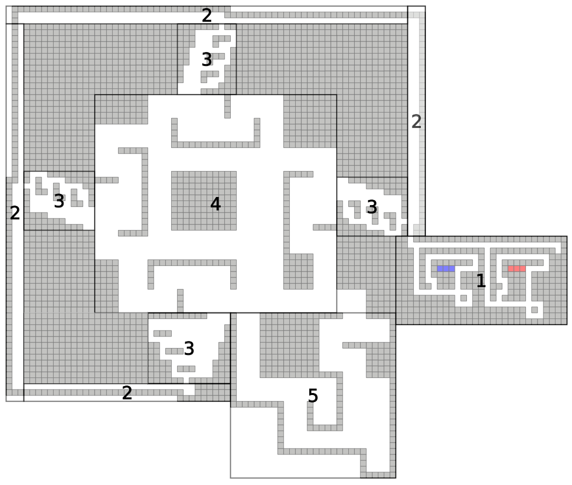
Next, we consider a class of shapes we refer to as *drop shapes*. A *drop shape* is any polyomino which is constructable by adding particles from any of the four cardinal directions *{N, E, S, W}* towards a fixed seed. Here, we present a shape builder which is strongly universal for the drop shapes set $U = \{u|u \subseteq \{1,\ldots,h\} \times \{1,\ldots,w\}\}$.
Figure 4.10 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 4.10b.

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


(a) Starting Configuration



(b) Sections

Figure 4.10: (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*.

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 4.13a and Table 4.2, respectively.

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 4.11 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

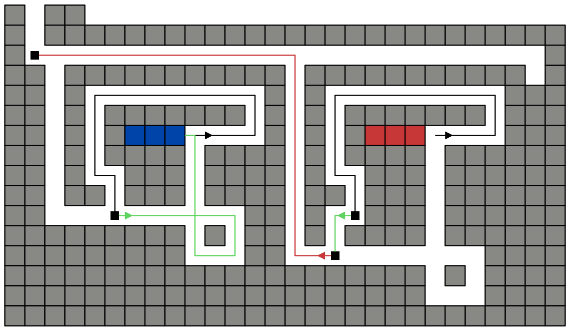


Figure 4.11: Tile selection gadget. Each tile is pulled out with the sequence $\langle E,N,W,S,E,S\rangle$, and stops at the first square. Then the left tile type (blue) is either pulled out of the gadget or put back in the storage area. This shows it being added back to the storage with $\langle E,S,W,N,W,S\rangle$. This sequence puts the next tile type (red) in a decision location. The red tile is selected with the sequence $\langle W,N,W,S,W,S\rangle$.

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 4.12 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.

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.

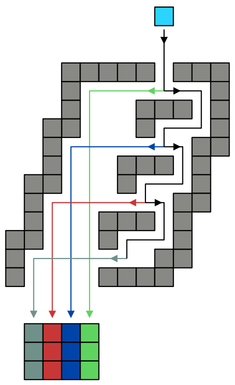
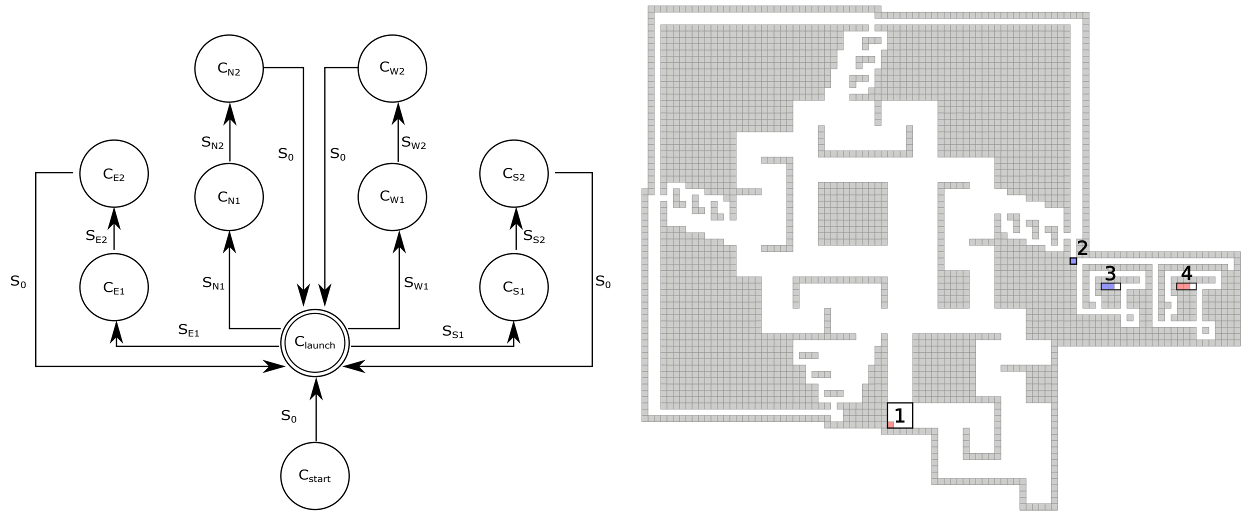


Figure 4.12: The column selection gadget for drop shapes. Assuming the shape to build is at a fixed location, this gadget allows any column to be selected to drop the new tile onto. The number of columns to drop from in this gadget determines the size of the shape we can build. Thus, this is for a drop 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).

This ensures that we can select a fuel type without repositioning the assembly.

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 4.2: The sequence of tilts used in the flowchart (Figure 4.13a). 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 4.13b). For area 2 in Figure 4.10b, 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 4.13: (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 4.2. (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.

CHAPTER V

COMPUTATIONAL COMPLEXITY IN THE SINGLE-STEP MODEL

This chapter focuses on the computational complexity of the Relocation and Reconfiguration problem when in the single-step model. We begin by first describing the *puzzle solvability* problem, which we often reduce from for our complexity results.

5.1 Puzzle Solvability

The model introduced in [5] 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.

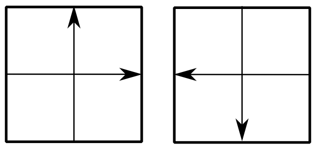
Gadgets and Puzzles:

- 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 5.1a 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 5.1b 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 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.


(a) Crossing 2-Toggle (C2T)

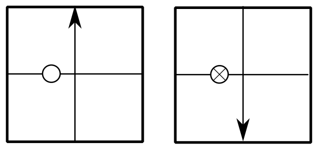

(b) Crossing Toggle-Lock (CTL)

Figure 5.1: (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).

5.2 Complexity with Rectangular Board Geometry and Dominoes

In this section we relax the restriction on tile size and show both the occupancy and relocation problems are both PSPACE-complete even when restricted to rectangular board geometry, and with particles of size at most 2. We show this by reducing from a simple gadget model proposed in [5]. The authors show that the problem of relocating a single agent in a connected system of these gadgets is PSPACE-complete.

Gadget Basics The gadgets used follow simple rules. They have two states, and contain tunnels that allow traversal through the gadgets. These tunnels exist in different types, such as the lock and toggle. A toggle tunnel can always be traversed in one direction, and on a state change that direction is reversed. The lock tunnel can be traversed in either direction when it is unlocked, and neither when it is locked. On a state change the lock tunnel will either lock or unlock. A gadget can contain multiple tunnels, each affected by the gadget's state changes. For our purpose we will use a crossing toggle lock gadget.

Crossing Toggle-Lock Domino Gadget The Crossing Toggle-Lock Domino Gadget, shown in Figure 5.2a, enforces the same rules for traversal with two dominoes. When in the unlocked state the horizontal tunnel contains only 1×1 tiles and allows for the space to travel through it unblocked. When in the locked state there is a domino blocking the horizontal path. When a space attempts to pass through that path it is blocked by the domino and cannot continue through the gadget.

The vertical tunnel only allows traversal in one direction based on the state of the gadget. When in

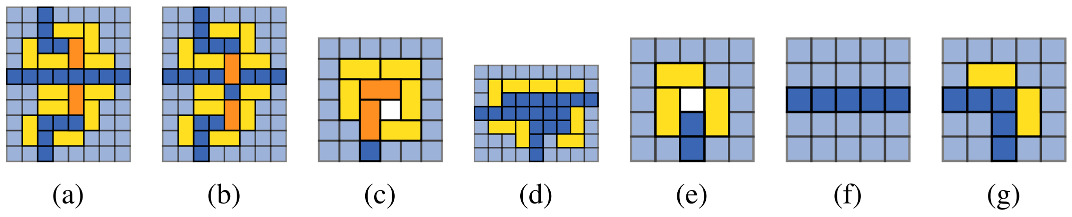


Figure 5.2: The crossing toggle-lock gadget implemented in the single-step tilt model. The left image (a) is in an open position, and the right (b) is the closed position. (c) The goal gadget for occupation. The space can only be covered by a polyomino if another space is in the gadget. (The light blue tiles are used to fill up the board and keep polyominoes in the gadgets from moving. These tiles will never move) (d) The 3-way branching gadget that allows the space to enter at any of the 3 locations and exit at any other. (e) The Start Gadget. Intially contains the space that acts as the agent and has dominos to enforce that the space can only exit at one location. (f) The wire gadget, a group of tiles act as a medium for the space to travel through. (g) Corner Gadget used to allow the space to change directions.

the unlocked state, traversal is allowed from south to north and if attempting to enter from the north location, it is blocked by a domino. When in the locked state traversal is only allowed from north to south. Any complete traversal through the vertical tunnel will change the location of the dominoes in the tunnel and the state of the gadget.

Other Gadgets In order to fully implement a CTL puzzle, we need a few other gadgets shown in Figures 5.2(d-g).

Branching Gadget The other gadget required in the motion planning problem is a 3-way branching gadget. The gadget is shown in Figure 5.2d and connects all three locations and allows for movement between them. The way the gadget is set up is when entering from any point the space will be able to cycle around the edges of the gadget. At certain positions in the gadget the space will be able to exit out one of the locations.

Wire Gadget The wire gadget shown in Figure 5.2f is just a group of single tiles. These tiles connect the other gadgets and allow the agent to travel through them.

Corner Gadget The puzzle solvabilty problem allows for wires that turn. Since the agent travels the maximum distance possible before reaching a domino or the edge of the board we create a corner gadget (Figure 5.2g) to allow the agent to stop and change direction.

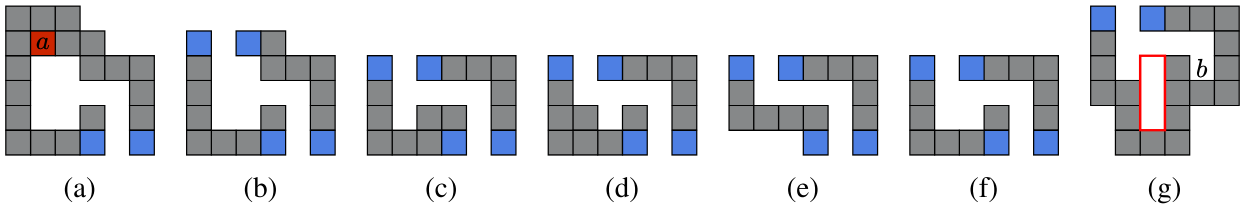


Figure 5.3: (a) Start gadget (b) Assignment gadget (c) Positive gadget (d) Negative gadget (e) Buffer gadget (f) Notch gadget (g) Goal gadget

Start Gadget The start gadget (Figure 5.2e) is where the agent starts. When constructing the reduction the gadget contains the space that acts as the agent. The open position is surrounded on three sides by dominoes so the space can only exit from one side.

Goal Gadget The goal gadget (Figure 5.2c) is the objective for the agent to reach. The gadget contains a second empty space that is surronded by dominoes. This space is the goal location. There is a horizontal domino that can be moved into this space if the agent reaches the goal gadget. The horizontal domino can only fill the goal location if the agent reaches the goal location.

5.3 Relocation with Limited Directions

In this section we detail a reduction from 3SAT to relocation in monotone boards using limited directions. We refer to the position the tile destined for relocation is initialized as location a and the goal position as location b. W.l.o.g, the directions used in the reduction are limited to *east* and *south*.

Gadgets All of the gadgets discussed in this section are show in Figure 5.3. We will later describe how to construct a board by attaching gadgets together via their north (head) and south (foot) *binding locations* (depicted in blue). The relocation tile is located at position a inside of the start gadget. The Assignment gadget is where tilts will correspond to variable assignments for our reduction. Every literal of a 3SAT formula has a corresponding Positive or Negative gadget, along with an associated tile that represents that literal. Location b resides inside the Goal gadget, a gadget whose structure grows with respect to the number of variables N. The area within the Goal gadget that grows based on N is outlined in Figure 5.3. The Buffer and Notch gadget are utility gadgets discussed when needed.

Notation Here we present the notation for describing the construction of a tilt board with the gadgets described above. We define $g_i \searrow g_j$ as gadget g_i's foot binding with gadget g_j's head. A *chain* is defined as a sequence of gadgets $S = \langle g_1, g_2, \ldots g_n \rangle$, s.t. $g_1 \searrow g_2$, $g_2 \searrow g_3$, $\ldots, g_{n-1} \searrow g_n$. It follows that a chain also has binding locations that allows it to bond with other chains or gadgets. We define $G = \langle S_1, S_2, \ldots, S_n \rangle$ as a *sequence of chains* s.t. $S_1 \searrow S_2$, $S_2 \searrow S_3$, $\ldots, S_{n-1} \searrow S_n$. Similarly, we say the binding of two sequences of chains $G = \langle S_1, S_2, \ldots, S_n \rangle$ and $G' = \langle S'_1, S'_2, \ldots, S'_m \rangle$ yields another sequence of chains $G'' = \langle S_1, S_2, \ldots, S'_{m-1}, S'_m \rangle$ s.t $S_1 \searrow S_2, \ldots, S_n \searrow S'_1, \ldots, S'_{m-1} \searrow S'_m$.

Board The board consists of three sections: the *assignment*, *formula*, and *validation* section. The tile initialized at location a within the assignment section is referred to as the *relocation* tile, the tiles inside the formula section are referred to as the *literal* tiles, and the tiles within the validation section are called the *validation* tiles. These board sections are all chains of the gadgets shown in Fig. 5.3.

Validation Section The validation section is the sequence of gadgets $S_V = \langle g_1, g_2, \ldots, g_N \rangle$ where:

- $g_N = \textit{Goal gadget}$ and $\forall i < N$, $g_i = \textit{Buffer gadget}$ with a validation tile allocated inbetween the head of every gadget.

Formula Section For the set of clauses C and set of variables $X = \{x_1, x_2, \ldots, x_N\}$, the clause chain of clause $c \in C$ is the sequence $G_c = \langle S_1, S_2, S_3, S_4, S_5 \rangle$ where:

- $S_1 = \langle g_1, g_2, \ldots, g_N \rangle$ where if the literal $l_p \in c$ is the variable $x_p \in X$, then $g_p = \textit{Positive gadget}$ if l_p is a positive literal or $g_p = \textit{Negative gadget}$ if l_p is a negative literal. The literal tile for literal l_p is allocated inbetween the head of g_1.

- $S_2 = \langle g_1, g_2, \ldots, g_N \rangle$ where if the literal $l_q \in c$ is the variable $x_q \in X$, then $g_q = \textit{Positive gadget}$ if l_q is a positive literal or $g_q = \textit{Negative gadget}$ if l_q is a negative literal. The literal tile for literal l_q is allocated inbetween the head of g_1.

- $S_3 = \langle g_1, g_2, \ldots, g_N \rangle$ where if the literal $l_r \in c$ is the variable $x_r \in X$, then $g_r = \textit{Positive gadget}$ if l_r is a positive literal or $g_r = \textit{Negative gadget}$ if l_r is a negative literal. The literal tile for literal l_r is allocated inbetween the head of g_1.

- $S_4 = \langle g_1, g_2, \ldots, g_{2N+1} \rangle$ where $\forall i < 2N+1$, $g_i = \textit{Buffer gadget}$, and $g_{2N+1} = \textit{Notch gadget}$.

- $S_5 = \langle g_1, g_2, \ldots, g_{2N+1} \rangle$ where $\forall i$, $g_i = \textit{Buffer gadget}$.

The formula section is therefore the sequence $G_F = \langle G_{c_1}, G_{c_2}, \ldots, G_{c_{|C|}} \rangle$ where every $G_{c_i} \in G_F$ is a clause chain and $G_{c_1} \searrow G_{c_2}$, $G_{c_2} \searrow G_{c_3}$, ..., $G_{c_{|C|-1}} \searrow G_{c_{|C|}}$

Assignment Section The assignment section is a sequence of chains $G_A = \langle S_1, S_2, S_3, S_4 \rangle$ where:

- $S_1 = \langle g_1, g_2, \ldots, g_N \rangle$ where $g_1 = \textit{Start gadget}$ and $\forall i > 1$, $g_i = \textit{Assignment gadget}$. The relocation tile is initialized within the Start gadget, as depicted in Figure 5.3.

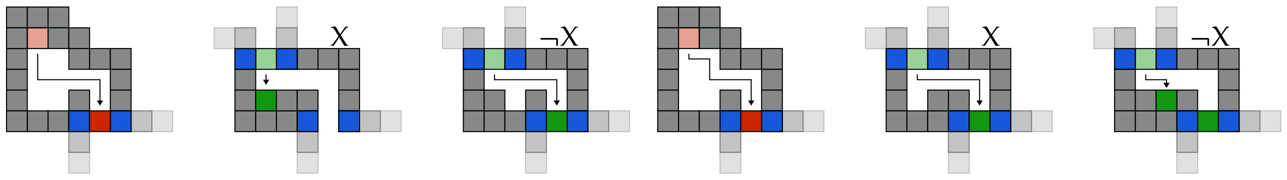
- $S_2 = \langle g_1, g_2, \ldots, g_{2N} \rangle$ where $\forall i$, $g_i = \textit{Notch gadget}$.

- $S_3 = \langle g_1, g_2, \ldots, g_{2N+1} \rangle$ where $g_1, g_{N+1}, g_{2N+1} = \textit{Assignmentgadget}$ and $\forall i$ s.t $g_i \neq \textit{Assignment gadget}$, $g_i = \textit{Notch gadget}$.

- $S_4 = \langle g_1, g_2, \ldots, g_v \rangle$ where $v = |S_V| + |G_F|$, and $\forall i \leq v$, $g_i = \textit{Notch gadget}$.

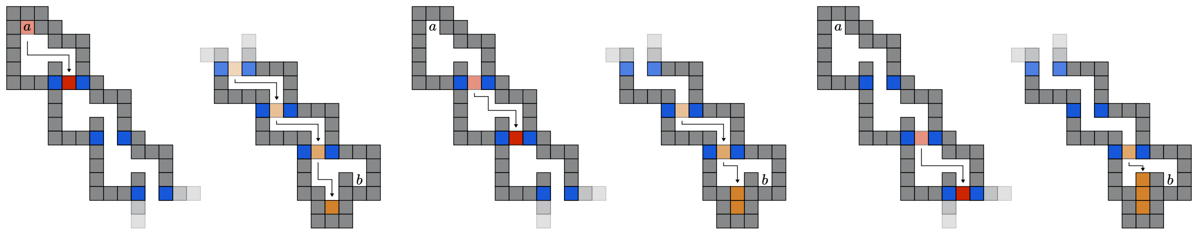
The board $B = \langle G_A, G_F, S_V \rangle$ is therefore the combination of the three sections where $G_A \searrow G_F$ and $G_F \searrow S_V$.

Reduction. The reduction can be understood as a two phase process. In the first phase variables are given a truth value one-by-one in ascending order for the set of variables $X = \{x_1, x_2, \ldots, x_N\}$. The second phase consists of verifying if the variable assignments from the first phase satisfied the clauses of the formula. Every clause chain is checked for any unconfined literal tiles, where if a clause chain confines at least one literal tile it is said to be satisfied, and unsatisfied otherwise. A



(a) $\langle S^2, E^3, S^2 \rangle$ (b) $\langle S, E, S, E^2, S^2 \rangle$

Figure 5.4: Assigning (a) *true* or (b) *false* to some variable x. Setting a literal to true is done by confining its corresponding literal tile within a Positive or Negative gadget.



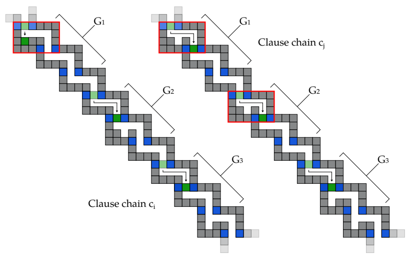
(a) (b) (c)

Figure 5.5: The leftmost chain belongs to the assignment section, and rightmost belongs to the validation Section. Validation tiles enter the Goal gadget for every assignment and Start gadget in the assignment section. This prevents any unnecessary step sequences from being performed.

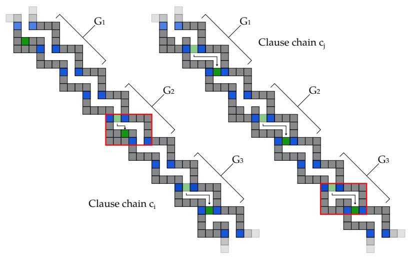
satisfied clause chain can have up to two remaining unconfined literal tiles, where an unsatisfied clause will have three unconfined literal tiles. In order to identify satisfied and unsatisfied clauses, the remaining literal tiles are ‘counted’ by attempting to occupy two open spaces using the literal tiles in each of their respective clause chains. Attempting to occupy the two spaces with three literal tiles (e.g, an unsatisfied clause chain) will leave one literal tile unconfined which will be used to prevent relocation.

First Phase. Starting with the board configuration described above, each variable in ascending order can be assigned some truth value by executing the step sequences depicted in Figure 5.4. During this phase, the relocation tile will visit every gadget inside the first chain of the assignment section if either one of the sequences depicted is executed for every variable truth assignment.

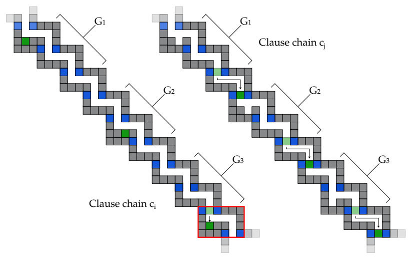
To ensure the step sequences depicted are the only ones executable during this phase, the validation tiles in the validation section will enter the Goal gadget one-by-one for every assignment in order to risk the unwanted occupancy of location b with one of the validation tiles. The validation tiles



(a) $x_1 \to 1$



(b) $x_2 \to 0$



(c) $x_3 \to 1$

Figure 5.6: Visualization of two clause chains $c_i = (x_1 \vee \neg x_2 \vee x_3)$, $c_j = (\neg x_1 \vee \neg x_1 \vee x_2)$ for $X = \{x_1, x_2, x_3\}$ during the first phase.

must solely experience the step sequences depicted above, unless the validation tiles will enter the notch hallway towards location b inexorably, as depicted in Figure 5.5. This relation between the relocation and validation tiles ensures that the two sequences depicted are the only ones performed during this phase.

All literal tiles visit every gadget of the chain it resides in for every clause chain (one of which is a Positive or Negative gadget) as the step sequence depicted above is performed. By design, the literal tiles reach a Positive or Negative gadget when its corresponding variable is next in the truth assignment procedure (Figure 5.6). After the last variable is assigned some truth value, every clause chain will confine at least one literal tile if the corresponding clause was satisfied, and zero literal tiles if it was not.

Second Phase Immediately after the first phase the relocation tile will reside inside the second chain of the assignment section, whose structure consists of only Notch gadgets. The Notch gadget serves two functions in this phase, one of which is to forbid the movement in the *south* direction when the relocation tile traverses through it. The first step in this phase is therefore to execute the step sequence $\langle S, W^3, S^2 \rangle$ until the relocation tile reaches the third chain, which simultaneously moves any unconfined literal tiles to the fourth chain of their respective clause chains.

Afterwards, the relocation and any unconfine literal tiles will reside inside the third and fourth chain, respectively, of their corresponding section/chain, as depicted in Figure 5.7. The last gadget of the fourth chain of every clause chain consists of a Notch gadget, whose function in this case

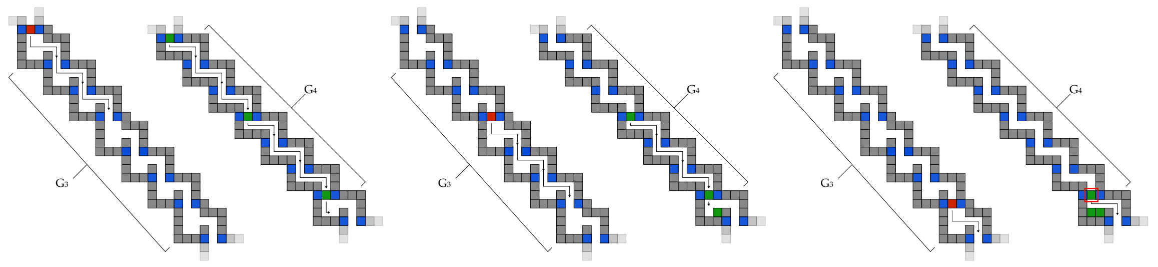
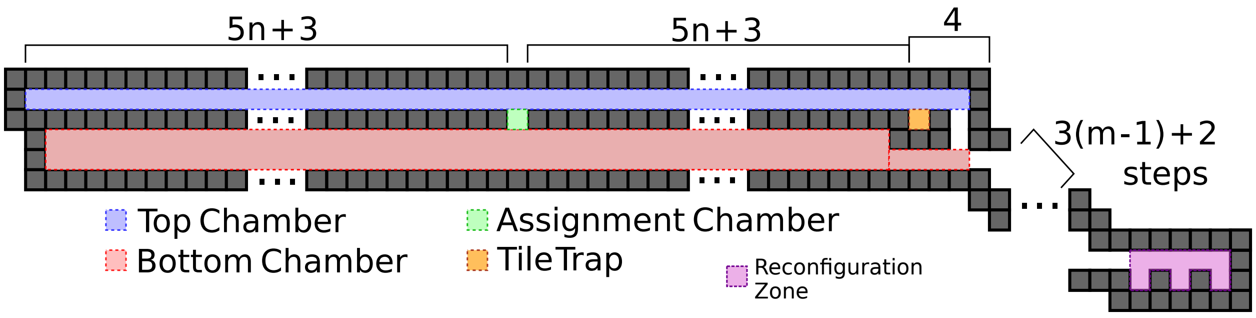


Figure 5.7: During the second phase, unconfined literal tiles are forced inside the Notch gadget of the fourth chain of their corresponding clause chain. Unsatisfied clause chains export one too many literal tiles, as shown in (c).

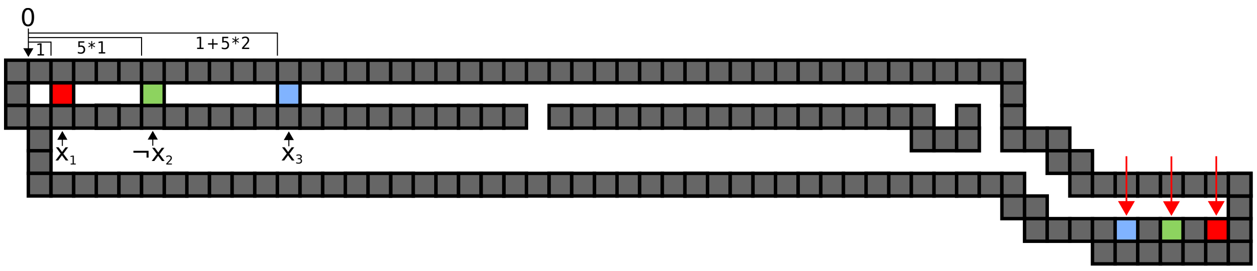
is to provide two open spaces for the unconfined literal tiles from the clause chains. Moving the relocation tile through the third chain of the assignment section, each remaining literal tile will reach the last Notch gadget of their corresponding clause chain and fit in one of the two open spaces made by the Notch gadget, provided that the clause chain was satisfied. If a clause chain was unsatisfied, then the third literal like will not fit within the two open spaces provided by the Notch gadget, leaving it free to continue through the board. At this point, the relocation tile would have reached the fourth chain of the assignment section, which is composed entirely of Notch gadgets and whose length is equal to that of the length of the formula section plus the length of the validation section. Traversing the tile through the fourth chain will move any free literal tiles towards the validation section, blocking location b. Therefore, if there exist at least one unsatisfied clause chain, there exists at least one unconfined literal tile inexorably pushed to occupy location b whilst the relocation tile traverses through the architecture of the fourth chain of the assignment section. If there are no unsatisfied clause chains, it follows that there will be a direct path from the relocation tile to location b on the board allowing its relocation there.

5.4 Reconfiguration with Limited Directions

We now will take a look at the problem of reconfiguration on a connected board when limited to two orthogonal directions. We show that with these constraints the problem is NP-Complete under the step transformation. Without loss of generality we will be limiting the directions to south and east.



(a)



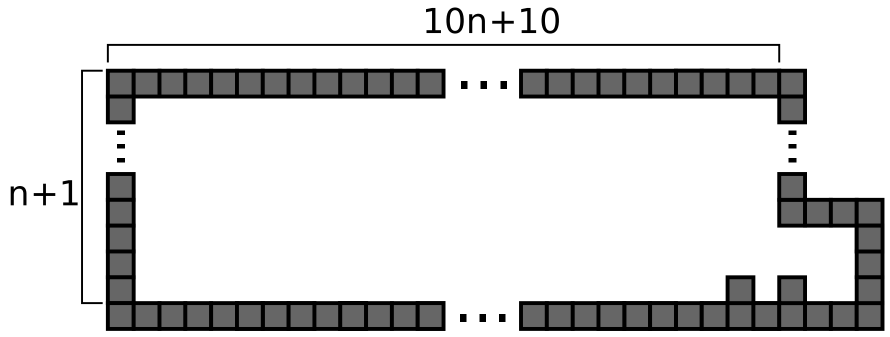
(b)

Figure 5.8: (a)3 SAT clause gadget, where n is the total number of unique variables, and m indicates that this clause is the mth clause in the formula. (b) Example of variable placement for clause $(x_1 \vee \neg x_2 \vee x_3)$ in a 5 variable formula. Goal locations indicated by red arrows.

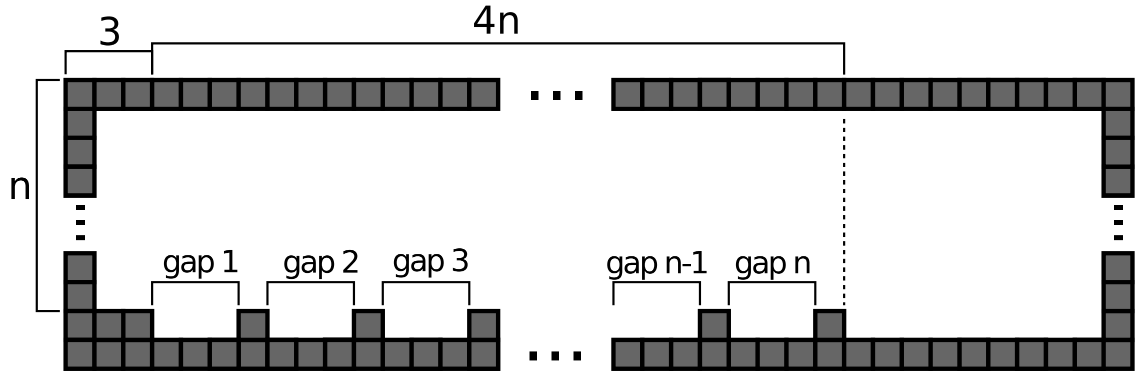
In this section, we prove hardness with a reduction from 3SAT. The following subsections describe the gadgets used when constructing a tilt configuration from a 3SAT formula with n variables and m clauses.

Clause Gadget Each clause of the 3SAT formula will be represented by a clause gadget (depicted in Figure 5.8). This gadget consists of a top chamber in which variable tiles will be placed and a 1-wide gap that connects that chamber to a second bottom chamber. There will be a staircase that connects the lower chamber to a final reconfiguration zone, which contains the goal locations for the three variable tiles in the clause. The number of steps of the staircase will be a function of the current clause; so, the m^{th} clause in the formula will have a staircase with $3(m-1)+2$ steps.

For every variable x_i, in every clause that it appears, we will place a corresponding variable tile in the top chamber of its respective clause gadget at position $5(i-1)+1$ for every positive variable and $5(i-1)$ for every negated variable. The goal locations of the tiles will be one in each of the



(a)



(b)

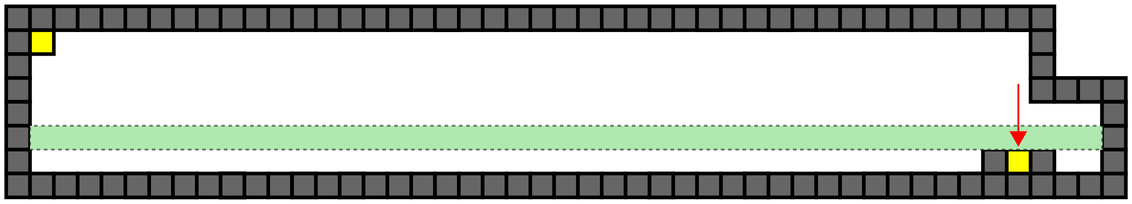
Figure 5.9: (a) South Limiter: Limits the amount of south steps made before all variables have been assigned. (b) South Forcer: Forces the user to make south steps at specific times

notches. Without loss of generality, the goal locations will be in the reverse order from west to east that they start at in the top chamber. Figure 5.12 shows an example tile placement for a clause gadget.

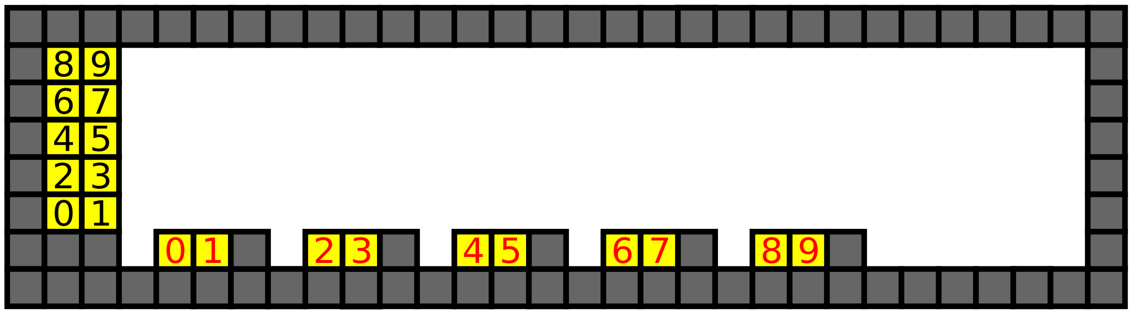
South Limiter Gadget Every instance of the reconfiguration problem obtained through this reduction will contain a single *south limiter gadget* (show in Figure 5.11a). The purpose of this gadget is to limit the number of *south* steps that can be made in the reconfiguration step sequence. The height and width of this gadget will scale linearly to the number of variables in the 3SAT formula.

In the South Limiter gadget we place a tile at the northwest corner, and the goal location is the notch at the other end of the gadget. Figure 5.13a depicts tile and goal placement for this gadget.

South Forcer Gadget Each instance will also have one *south forcer gadget* (shown in Figure 5.11b). The purpose of this gadget is to enforce the proper timing of any *south* step made in the reconfig-

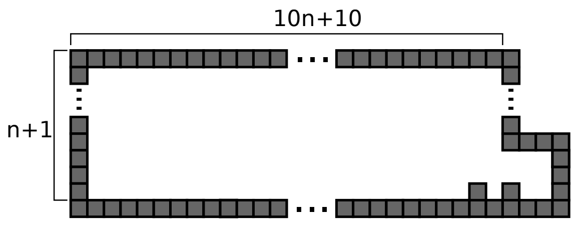


(a)

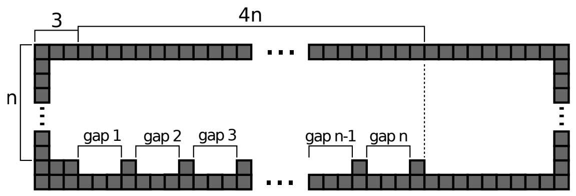


(b)

Figure 5.10: (a) Example of south limiter tile placement in a 5 variable formula. Goal location indicated by red arrow. Post assignment zone highlighted in green. (b) Example of south forcer tile placement in a 5 variable formula. Goal location indicated by red labels.



(a) South Limiter: Limits the amount of south steps made before all variables have been assigned



(b) South Forcer: Forces the user to make south steps at specific times

uration step sequence. The height and width of this gadget also scales linearly to the number of variables.

In the South Forcer gadget we place 2 tiles in a row for every distinct variable in the 3SAT formula on the west side of the gadget. The goal location of row n is the eastmost side of the n^{th} gap 5.11a, in the same position relative to each other that they started in. Figure 5.13b depicts tile and goal placement for this gadget.

Variable Assignment This reduction works by utilizing the uniform global signals to assign all variables x_i a truth value simultaneously. To start the process of reconfiguration we will begin

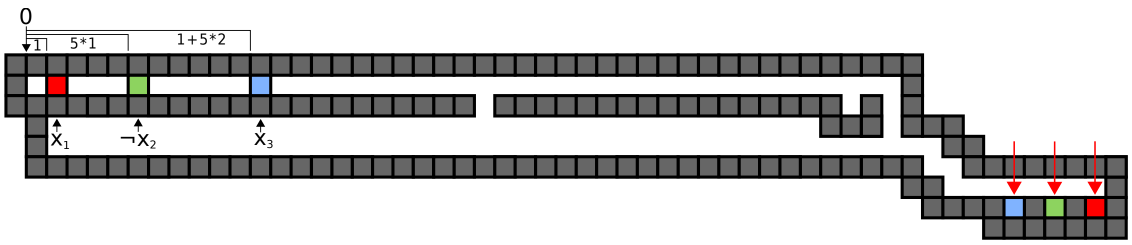
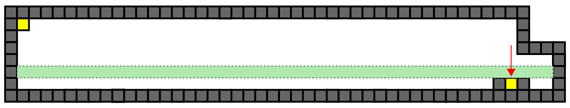
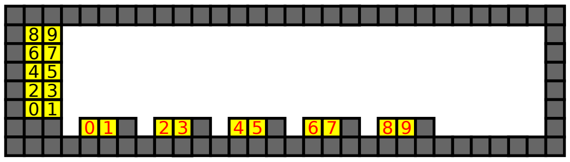


Figure 5.12: Example of variable placement for clause $(x_1 \vee \neg x_2 \vee x_3)$ in a 5 variable formula. Goal locations indicated by red arrows



(a) Example of south limiter tile placement in a 5 variable formula. Goal location indicated by red arrow. Post assignment zone highlighted in green.



(b) Example of south forcer tile placement in a 5 variable formula. Goal location indicated by red labels

stepping east, uniformly shifting all variable tiles along the top chamber of the clause gadgets. When the eastmost variable tile is located above the assignment chamber, the choice of assigning that variable true can be made by performing step sequence $s_t = \langle S, E \rangle$, or false with the step sequence $s_f = \langle E, S \rangle$. A variable tile evaluates to true if it enters the assignment chamber. Since all variable tiles t_i representing a particular variable x_i share the same x-coordinate, all of these tiles will receive the same assignment.

South Forcer/Limiter Gadgets The reconfiguration requirement of the *south forcer gadget* ensures that every variable must receive an assignment. The only way to place each tile-pair in their respective goal positions is to "assign" each of the n variables a value of either true of false. The *south limiter gadget* ensures that only n assignments can be made, since doing more would position the gadget's tile along the bottom edge of the gadget with no way of reaching its goal position. The combination of these gadgets prevents the possibility of assigning both a positive and negated variable tile, i.e. t_i and $\neg t_i$, values of "true". Once every variable tile has been assigned, the *south limiter gadget's* tile will be in the *post assignment zone* (see Figure 5.13a). In order to achieve the configuration goal inside that gadget, only east steps can be made until the tile is directly above its goal location. With global signals this causes all variables tiles to be pushed maximally to the east side of the top chamber of their respective clause gadget. Once the *south limiter's* tile is above its goal location, a south step must be performed, since we are limited to only south and east, and an east step would make the south limiter's tile pass its goal position.

Clause Verification A clause gadget is in its goal configuration when each of its variable tiles are in their goal locations at the bottom right of the gadget. A clause gadget will be unable to reach its goal configuration if no variable tiles make it through the assignment chamber (this is the equivalent to every literal of a 3SAT clause evaluating to false). If there are three variable tiles still in the top chamber of a clause gadget, a south step will place the west-most variable tile into the *tile trap*. This makes the board unreconfigurable. If at least 1 variable tile is passed into the assignment chamber, then the clause gadget will always be reconfigurable. From there the tiles can be moved to their respective staircase, where the varied number of steps allow for each tile to moved to its goal location individually.

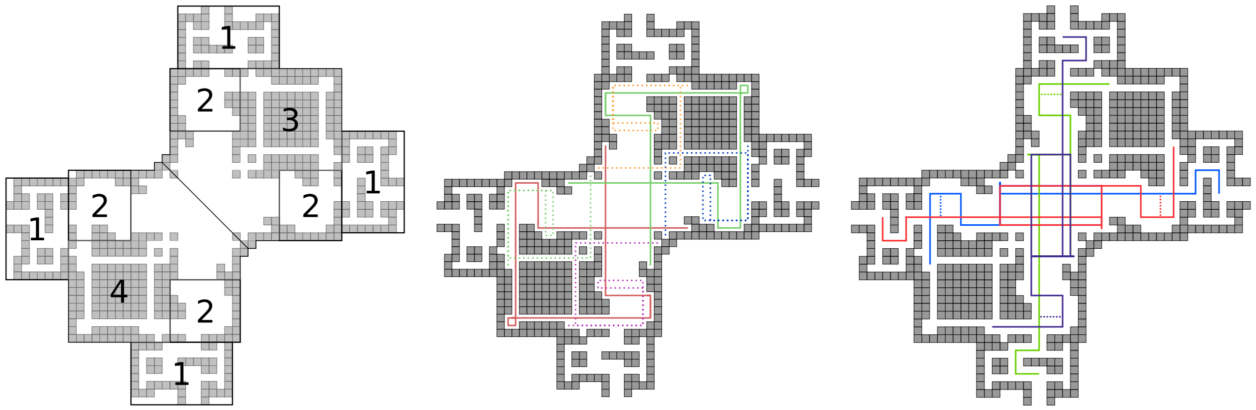
CHAPTER VI

COMPUTATIONAL COMPLEXITY IN THE FULL-TILT MODEL

This chapter focuses on the computational complexity of the Relocation and Reconfiguration problem when in the full-tilt model.

6.1 Relocation

Here we present a gadget in the tilt model that behaves like the C2T gadget described above. Figure 6.1 shows an overview of the gadget. Figure 6.1a 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 6.1b and 6.1c 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 6.2 shows an example of a pathway not reachable by the 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 6.2 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.



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

Figure 6.1: (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 The *robot polyomino* is a 2×2 polyomino that traverses through a puzzle. Figure 6.1c 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 6.1b, 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 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 6.2.

Entrance/Exit Chambers Marked as section 1 in 6.1a, 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.

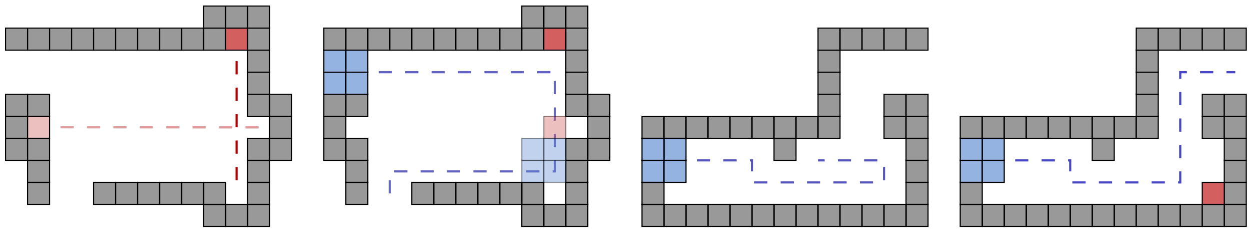

(a) State Path Depiction (b) State Change (c) Robot Limited Path (d) Robot Assisted Path

Figure 6.2: 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.

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 5.1.

Assistance Chamber Marked as section 2 in 6.1a, 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 6.3 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 6.1a 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 6.1b 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.

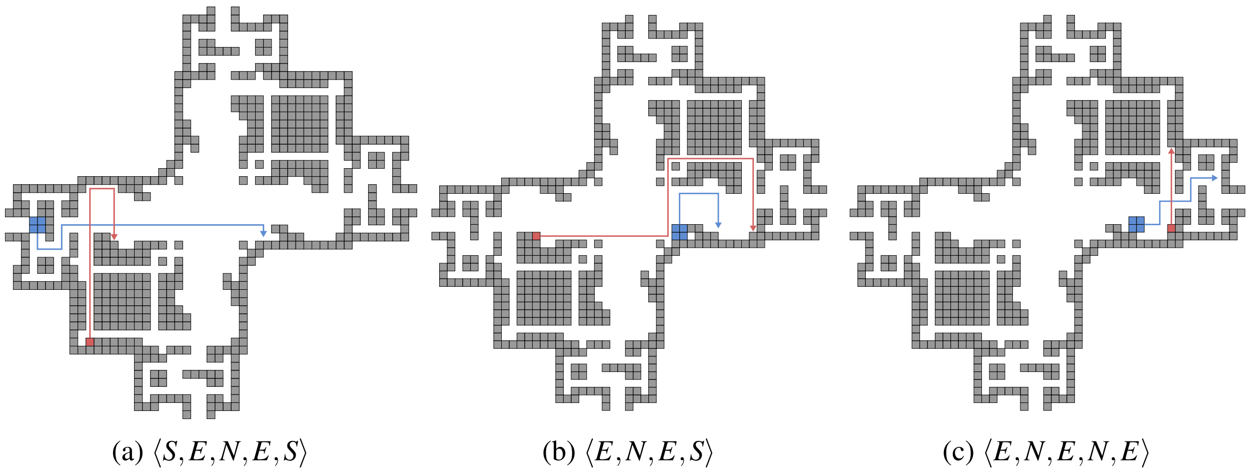

(a) $\langle S,E,N,E,S\rangle$ (b) $\langle E,N,E,S\rangle$ (c) $\langle E,N,E,N,E\rangle$

Figure 6.3: Example of a traversing sequence and state change when a gadget is in the state from Figure 5.1a. 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 6.1b to section 3's green path; Therefore, the gadget was toggled.

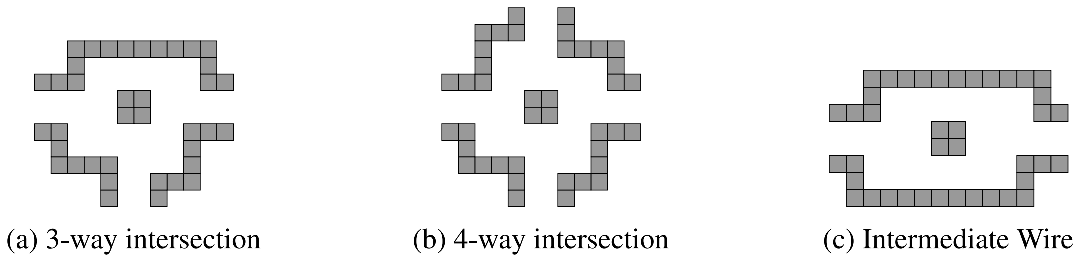

(a) 3-way intersection (b) 4-way intersection (c) Intermediate Wire

Figure 6.4: 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.

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 6.4a and 6.4b, respectively.

6.2 Reconfiguration

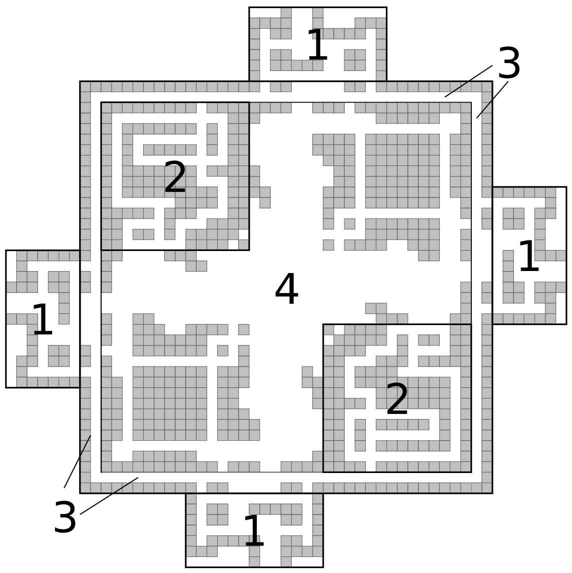
For the reconfiguration problem we must provide a unique final configuration, as opposed to just a final location of a single polyomino. The previous gadget is insufficient for this problem as there is generally not a unique final configuration for all successful traversals, and it would not be polynomial 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 6.6a) into the reconfiguration ring, and render all gadgets inoperable.

Restarting Positions We define *restarting positions* (depicted by the solid state tiles in Figure 6.6a) 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 6.4c) 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



(a) Reconfiguration Gadget Sections

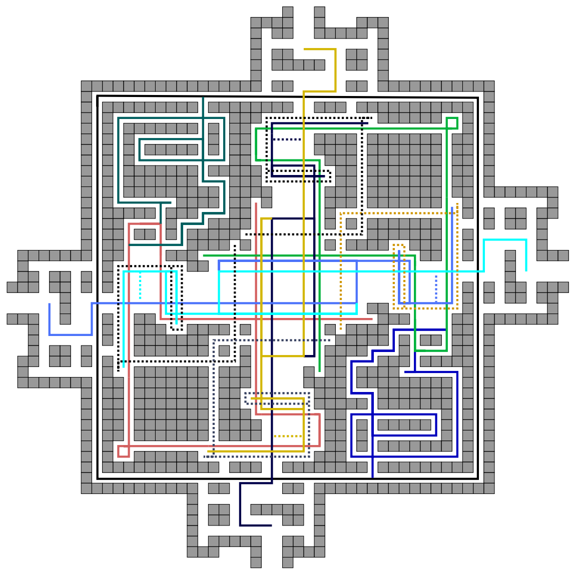

(b) Reconfiguration Gadget Positions

Figure 6.5: (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.

directions. After every reconfiguration gadget traversal, the robot can move freely in the intermediate wire and reposition the state tiles to their restarting positions.

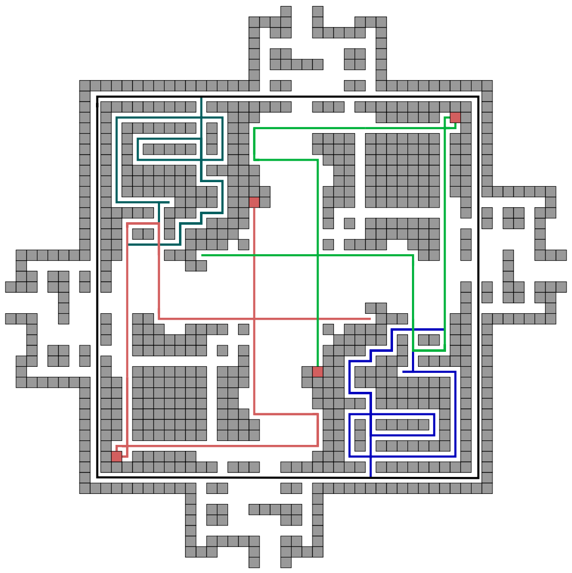

(a) Reconfiguration state tile paths.

Figure 6.6: 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.

CHAPTER VII

CONCLUSION

We have shown various methods of robotic swarm assembly, showing different universal configurations that build shapes of multiple sizes. In the single-step model, we showed a universal configuration that can build size-n shapes which can later be extended to become reconfigurable. Then, we demonstrated a universal configuration for patterned shapes, using similar ideas to the size-n shape constructor, placing robots respective to there position on the shape row-by-row. Since these constructors build shapes that are separated by some spaces, we introduced the *funneling* gadget that receives a group of robots spaced out and yields the shape that they correspond to. Thereafter, we showed in the full-tilt model a similar way to build size-n shapes using a *weakly* construction method. This constructor is able to place robots in their correct position in the construction area, which can then be withdrawn and put adjacent to form the shape. Then, the output of this constructor can be send through the dispenser gadget, which breaks down the columns one-by-one and places them back into the pixel gadgets. Moreover, we show a quick method to build patterned rectangles in the full-tilt model, improving upon earlier work done in this problem. We ended the assembly chapters talking about Drop Shapes, a massively interesting class of shapes that we show how to build strongly. The thesis ends with two chapters on the computational complexity of the Relocation and Reconfiguration problems on both of these models, showing reductions from a well used puzzle solvability problem.

BIBLIOGRAPHY

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

[2] 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, Berlin, Heidelberg, 2014, Springer Berlin Heidelberg, pp. 51–66.

[3] A. T. 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, Nov 2013, pp. 3317–3323.

[4] Brio, *Labyrinth game.*

[5] 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.

[6] 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.

[7] 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 2D polyominoes*, 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

[8] 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.

[9] 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.

[10] D. Zhang and G. Seelig, *Dynamic dna nanotechnology using strand-displacement reactions*, 3 (2011), pp. 103–13.

[11] C. Zhou, X. Duan, and N. Liu, *A plasmonic nanorod that walks on dna origami*, Nature Communications, 6 (2015), pp. 8102–8102. 26303016[pmid].

BIOGRAPHICAL SKETCH

Angel Adrian Cantu Suarez was born in McAllen, Texas, but spent his infant and adolescence years in Hidalgo, Texas. Graduating from Sharyland High School in 2013, Angel immediately enrolled as a Psychology student at The Univerity of Texas-Rio Grande Valley (formerly known as the University of Texas-Pan American). After his first three semesters, Angel decided to challenge himself by dominating both his interest in Psychology and Mathematics by changing his major to that of Computer Science, where he continued educating himself on Psychology by reading the literature on his own. As a Computer Science student, Angel would come to exercise his creative capabilities when he joined the ASARG Research group overseen by Dr. Robert Schweller and Dr. Tim Wylie, where he would conduct research in topics like Game Complexity and Algorithmic Self-Assembly. Around the same time, Angel would also join Dr. Dongchul Kim's Machine Learning research group, where he would put to the test various artificial intelligence algorithms for unsolved problems. During his undergraduate years, Angel earned The Engaged Scholar for his original undergraduate research. Upon receiving his Bachelor's in Computer Science, Angel joined the Master's program in Computer Science, where he would continue his research in Game Complexity, Machine Learning, and Algorithmic Self-Assembly, up until receiving his Master's in Computer Science in May 2020.

Contact Information: E-mail: aadcantu@gmail.com Address: 3025 Colorado Ave. McAllen,Texas 78504