Randomness, information encoding, and shape replication in various models of DNA-inspired self-assembly

Eric M. Martinez
The University of Texas Rio Grande Valley

RANDOMNESS, INFORMATION ENCODING, AND SHAPE REPLICATION IN VARIOUS MODELS OF DNA-INSPIRED SELF-ASSEMBLY

A Thesis

by

ERIC M. MARTINEZ

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

MASTER OF SCIENCE

August 2016

Major Subject: Computer Science

RANDOMNESS, INFORMATION ENCODING, AND SHAPE REPLICATION IN VARIOUS MODELS OF DNA-INSPIRED SELF-ASSEMBLY

A Thesis
by
ERIC M. MARTINEZ

COMMITTEE MEMBERS

Dr. Robert Schweller
Chair of Committee

Dr. Timothy Wylie
Committee Member

Dr. Emmett Tomai
Committee Member

August 2016

ABSTRACT

Martinez, Eric M., Randomness, Information Encoding, and Shape Replication in Various Models of DNA-Inspired Self-Assembly. Master of Science (MS), August, 2016, 74 pp., 2 tables, 30 figures, 57 references, 90 titles.

Self-assembly is the process by which simple, unorganized components autonomously combine to form larger, more complex structures. Researchers are turning to self-assembly technology for the design of ever smaller, more complex, and precise nanoscale devices, and as an emerging fundamental tool for nanotechnology.

We introduce the *robust random number generation* problem, the problem of encoding a target string of bits in the form of a *bit string pad*, and the problem of *shape replication* in various models of tile-based self-assembly. Also included are preliminary results in each of these directions with discussion of possible future work directions.

Also described is VersaTILE, a cross-platform multi-model self-assembly simulator for various models of algorithmic self-assembly.

ACKNOWLEDGMENTS

My time as a graduate student has been a great period of my life and a wonderful experience. Foremost, I would like to thank my advisor, Dr. Robert Schweller, for the continuous support of my study and research. I took his class on algorithms during my Bachelor's, which sparked my interest in theoretical computer science. I would like to express my sincere gratitude for his immense knowledge, enthusiasm, and guidance. I could not have imagined having a better advisor and mentor for my Master's study.

I would also like to thank Dr. Tim Wylie for all of his help. Both Robert and Tim have had profound influence on my career. They were excellent collaborators, and real joys to work with. I would also like to thank Cameron Chalk being an excellent collaborator and friend during this time.

My collaborators have greatly influenced me, guiding and shaping my ideas, refining my work, teaching me new techniques, and being available to ping-pong ideas off of.

My co-authors include Cameron Chalk (UTRGV), Erik D. Demaine (MIT), Martin L. Demaine (MIT), Bin Fu (UTRGV), Alejandro Huerta (UTRGV), Mario Maldonado (IBM), Robert T. Schweller (UTRGV), Lucian Silcox, Emmett Tomai (UTRGV), Luis Vega, Andrew Winslow (ULB), and Tim Wylie (UTRGV).

Last but not the least, I would like to thank my family: my parents Alicia and Carlos Martinez, for giving birth to me in the first place and supporting me and my endeavors. I would also like to thank my brother Carlos Martinez for always supporting and pushing me to 'get it done'.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Algorithmic self-assembly is the process of designing molecules such that, through local interactions, they autonomously combine to form larger, more complex structures. Self-assembly occurs abundantly in nature and is an important mechanism during the construction and life-cycle of all biological organisms. Researchers are turning to self-assembly technology for the design of ever smaller, more complex and precise nanoscale devices [18], [25], [30], [32], [34], [41], [51], [52]. At the macroscale, industry relies on machinery for the ability to fabricate components from simple plastic to complex microchips. Devices this large, however, cannot operate on nanoscale molecules. Self-assembly has emerged as a fundamental tool in the field of nanotechnology to tackle these problems.

Algorithmic self-assembly allows for the simulation of complex computational processes which allows for small, compact systems to efficiently assemble large complex shapes and patterns. These systems also serve as models for the design of parallel molecular computers. Fully understanding the power of self-assembly systems, and how to control them, presents the possibility of designing molecular algorithms for precise and efficient manipulation of matter at the nanoscale. Models inspired by the ability DNA strands to predictibly bond with their complementary strands have been developed that may allow the theoretical portion of this line of research to be realized. In this work, we consider models of algorithmic self-assembly that are *tile-based*. Tile-based self-assembly systems have nonrotatable square *tiles* as the monomers of the system with glues on each of their edges. These glues, based on certain rules, can allow tiles to interact and bind with each other. DNA strands can be synthesized to simulate nonrotatable square tiles and their glues, which is the inspiration for these types of self-assembly models. In this work, I explore several fundamental problems and detail my contributions in these directions: *random number generation*, *encoding information*, *shape replication*, and *simulation*. Simulating various models of self-assembly is an essential tool to debugging and testing self-assembling systems, without the need for *wetlab* experiments.

Tile-based Self-Assembly

The simplest model, the abstract Tile Assembly Model, was first defined by [8] in which the monomers of the assembly system are square tiles, with glue types assigned to tile edges. Self-assembly within the tile assembly model is driven by a large (infinite) number of copies of a set of tile types floating about, bumping into one another in the plane, and potentially sticking together when glue affinities exceed some set threshold. This class of self-assembly model is motivated by the ability to fold DNA strands into compact molecular building blocks, or tiles [2], [9], [10], [11], [12], [26], [35], [42], and design a library of combinatorially distinct DNA strands [3], [4], [5], [6], [7], [17], [19], [22], [23], [24], [29] for implementation of glues.

While simple, tile assembly has been extensively studied [15], [16], [21], [27], [31], [33], [36], [37], [38], [39], [40], [43], [46], [48], [49], [50], [53], [54], [55], [56], [57], [60], [62], [64], [68], [69], [71], [72], [73], [74], [76], [77], [78], [79], [89] and shown to be capable of universal computation [8] to be intrinsically universal [63].

Summary of Contributions

Randomization and Fair Coins

Here I focus on the ability to generate uniformly random bits within the abstract Tile Assembly Model, defined in Section II. This allows for the creation of tile systems that implement randomized algorithms and models of computation. Previous research focused only on random string generation and was restrictive and relied on specific concentrations of tiles [44], [49].

An aTAM algorithm generates a “fair coin” *iff* it generates a finite set of terminals that can be partitioned into two subsets, and each subset is generated with probability 0.5.

This problem is trivial to solve with certain tile sets and tile type concentrations. If two tile types nondeterministically compete to bind at a single edge on a seed tile, using the same glue type and exposing no new glues of their own, then this system can generate a fair coin, if the relative concentrations of the competing tile types are equal. Each competing tile type has $1/2$ probabability of attaching. This solution, however, breaks down if the concentrations of the tile types are all different. Instead, a result is shown that works invariant of the concentrations of the tile types.

Encoding information

We then turn our attention to the Staged Tile Assembly Model, defined in Section II. A key primitive is the design of an x-bit string pad. A width-m gap-f x-bit string pad is a $m \times f(r-1)+1$ rectangular assembly with r glue types of value 0 or 1 exposed on the north face of the rectangle at intervals of length f, starting from the westmost northern edge. It can be used to build linear assemblies, as input for specifying an $n \times n$ rectangle, or to describe a desired shape to a Turing machine implemented in the staged tile assembly model. Two methods for building bit string pads are described. The first, describes how to build many different types of bit string pads, with each type in their own bin, showcasing a complexity tradeoff between tile types and bins. The second method, utilizes only tile types to encode a representation of a number in a higher base and then "unpack" that representation into a bit string pad representing the original number in binary. These constructions can be used to build "wings" that allow large assemblies to self-assembly linearly in some specified order.

Shape replication

Here, I describe a process in which the power of assembly can be used to build a system that first senses the shape of a given unknown object, then builds copies of that shape. This model is an extension of the 2-Handed Assembly Model (2-HAM), defined in Section II, that allows for positive as well as negative glue strengths. Prior work, has shown this problem to be solvable in an extension of the 2-Handed Assembly Model which includes an *RNAse enzyme*. In this model, there are two types of tiles (DNA or RNA) as well as an operation utilizing an *RNAse enzyme* which destroys all tiles of the RNA type. A constant set of tile types plus a global enzyme operation are combined with an input shape yield a replication system. This model is unsatisfactory as it requires multiple stages and a global “destroy” operation. In this work, the model is much more simple and only requires positive glue types (attractive forces) and negative glue types (repulsive forces). It receives a shape as input and with the addition of $O(1)$-tile types, yields an unbounded number of copies of the input shape.

Simulation of self-assembling systems

Here, I describe the simulation of self-assembling systems via software. Specifically, my contributions towards VersaTILE, a cross-platform multi-model self-assembly simulator. Simulation of self-assembling systems allows the user to debug, test, and experiment with simulations without the use of wetlab experiments. By creating an easy-to-use simulator, I made it easy for researchers to test and interact with their systems in real-time and also integrated many of the popular extensions to the standard model proposed in literature.

CHAPTER II

BACKGROUND

The various models used in the different chapters are described below. The Abstract Tile Assembly Model is the model used in Chapter III. The Staged Tile Assembly Model is the model used in Chapter IV. The 2-Handed Tile Assembly Model is the model used in Chapter V.

Abstract Tile Assembly Model (aTAM)

Consider some alphabet of glue types Π. A tile is a unit square with four edges each assigned some glue type from Π. Further, each glue type $g \in \Pi$ has some non-negative integer strength $str(g)$. Each tile may be assigned a finite length string label, e.g., “black",“white",“0", or “1". For simplicity, we assume each tile center is located at a pixel $p = (p_x, p_y) \in \mathbb{Z}^2$. For a given tile t, we denote the tile center of t as its position. As notation, we denote the set of all tiles that constitute all translations of the tiles in a set T as the set T^*. An *assembly* is a set of tiles each assigned unique coordinates in $\mathbb{Z}^2$. For a given assembly α, define the *bond graph* G_α to be the weighted graph in which each element of α is a vertex, and each edge weight between tiles is $str(g)$ if the tiles share an overlapping glue g, and 0 otherwise. An assembly α is said to be τ*-stable* for a positive integer τ if the bond graph G_α has min-cut at least τ, and τ*-unstable* otherwise. A *tile system* is an ordered triple $\Gamma = (T, \sigma, \tau)$ where T is a set of tiles called the *tile set* (we refer to elements of T as tile types), σ is an assembly called the *seed* and τ is a positive integer called the *temperature*. When considering a tile a that is some translation of an element of a tile set T, we will use the term *tile type* of a to reference the element of T that a is a translation from. Assembly proceeds by growing from assembly σ by any sequence of single tile attachments from T as long as each tile attachment connects with strength at least τ. Formally, we define what can be built in this fashion as the set of producible assemblies:

Definition 2.1 (Producibility)**.** For a given tile system $\Gamma = (T, \sigma, \tau)$, the set of **producible assemblies** for system Γ, $\texttt{PROD}_\Gamma$, is defined recursively:

- (Base) $\sigma \in \texttt{PROD}_\Gamma$
- (Recursion) For any $A \in \texttt{PROD}_\Gamma$ and $b \in T^*$ such that $C = A \cup \{b\}$ is τ-stable, then $C \in \texttt{PROD}_\Gamma$.

As additional notation, we say $A \rightarrow^\Gamma_1 B$ if A may grow into B through a single tile attachment, and we say $A \rightarrow^\Gamma B$ if A can grow into B through 0 or more tile attachments. An **assembly sequence** for a tile system Γ is a sequence (finite or infinite) $\vec{\alpha} = \langle \alpha_1, \alpha_2, \ldots \rangle$ in which $\alpha_1 = \sigma$, each α_{i+1} is a single-tile extension of α_i, and each α_i is τ-stable. The **frontier** of an assembly α, written as $F(\omega, \Gamma)$, is a partial function that maps an assembly ω and a tile system Γ to a set of tiles $\{t \in T^* | \omega \cup \{t\} \in \texttt{PROD}_\Gamma \land t \notin \omega\}$. We further define $\texttt{TERM}_\Gamma$ to be the subset of $\texttt{PROD}_\Gamma$ consisting only of assemblies for which no further tile in T may attach (i.e., the assembly has an empty frontier).

Definition 2.2 (Finiteness and Space)**.** For a given tile assembly system $\Gamma = (T, \sigma, \tau)$, we say Γ is **finite** iff $\forall \beta \in \texttt{PROD}_\Gamma, \exists \alpha \in \texttt{TERM}_\Gamma : \beta \rightarrow^\Gamma \alpha$. That is, each producible assembly has a growth path ending in a finite, terminal assembly. If Γ is not finite, we say it is **infinite**. Define the **space of an assembly** α as $|\alpha|$. Let the **space of a tile assembly system** be defined as the $\max_{\alpha \in \texttt{TERM}_\Gamma} |\alpha|$ iff Γ is finite. If Γ is infinite, let **space** remain undefined. Note that a finite system may have infinite/unbounded space.

Definition 2.3 (Extensibility)**.** Consider a tile assembly system $\Gamma = (T, \sigma, \tau)$, and assembly $\alpha \in \texttt{PROD}_\Gamma$. We denote the set of all locations at which a tile may stably attach to α as L_α. More formally, $L_\alpha = \{p_t | t \in F(\alpha, \Gamma)\}$. We say a tile system Γ is k**-extensible** iff $\forall \alpha \in \texttt{PROD}_\Gamma, |L_\alpha| \leq k$. Informally, a tile assembly system is k-extensible iff at any point in the assembly process, the assembly can only grow in at most k locations.

Probability in Tile Assembly

We use the definition of probabilistic assembly presented in [31], [39], [49], [55], [61]. Let P be a function denoting a **concentration distribution** over a tileset T representing the concentrations of each tile type with the restrictions $\forall t \in T, P(t) > 0$ and $\sum_{t \in T} P(t) = 1$. For a tile t, we sometimes refer to $P(t)$ as the **concentration** of t. Using a concentration distribution, we can consider probabilities for certain events in the system. To study probabilistic assembly, we can consider the assembly process as a Markov chain where each producible assembly is a state and transitions occur with non-zero probability from assembly A to each B whenever $A \to_1^\Gamma B$. For each B that satisfies $A \to_1^\Gamma B$, let $t_{A \to B}$ denote the tile in T whose translation is added to A to get B. The transition probability from A to B is defined to be

$$TRANS(A, B) = \frac{P(t_{A \to B})}{\sum_{\{C | A \to_1^\Gamma C\}} P(t_{A \to C})} \tag{2.1}$$

The probability that a tile system Γ terminally assembles an assembly A is defined to be the probability that the Markov chain ends in state A. For each $A \in \texttt{TERM}_\Gamma$, let $\texttt{PROB}^P_{\Gamma \to A}$ denote the probability that Γ terminally assembles A with respect to concentration distribution P.

Definition 2.4 (Expected Space)**.** For a given finite tile system $\Gamma = (T, \sigma, \tau)$, let the expected space of Γ relative to a concentration distribution P be defined as

$$\texttt{EXPECTEDSPACE}_\Gamma = \sum_{\alpha \in \texttt{TERM}_\Gamma} |\alpha| \cdot \texttt{PROB}^P_{\Gamma \to \alpha}$$

Definition 2.5 (Coin Flipping)**.** We consider a finite tile system Γ a **coin flip tile system with bias** b with respect to a concentration distribution P for some $b \in \mathbb{R}^+$ iff the set of terminal assemblies in $\texttt{PROD}_\Gamma$ can be partitioned into two sets X and Y such that $\left| \sum_{x \in X} \texttt{PROB}^P_{\Gamma \to x} - \sum_{y \in Y} \texttt{PROB}^P_{\Gamma \to y} \right| \leq 2b$. A **fair coin flip tile system** is a coin flip tile system with bias 0. We consider a finite tile system a **robust coin flip tile system with bias** b iff it is a coin flip tile system with bias b for all concentration distributions; i.e. $\left| \sum_{x \in X} \texttt{PROB}^C_{\Gamma \to x} - \sum_{y \in Y} \texttt{PROB}^C_{\Gamma \to y} \right| \leq 2b$ for all concentration distributions C. A **robust fair coin flip tile system** is a robust coin flip tile system with bias 0.

Staged Tile Assembly Model

Tiles. A *tile* is a non-rotatable unit square with each edge labeled with a *glue* from a set Σ. Each pair of glues $g_1, g_2 \in \Sigma$ has a non-negative integer *strength*, denoted $\text{str}(g_1, g_2)$. Every set Σ contains a special *null glue* whose strength with every other glue is 0. If the glue strengths do not obey $\text{str}(g_1, g_2) = 0$ for all $g_1 \neq g_2$, then the glues are *flexible*. Unless otherwise stated, we assume that glues are not flexible.

Configurations, assemblies, and shapes. A *configuration* is a partial function $A : \mathbb{Z}^2 \to T$ for some set of tiles T, i.e., an arrangement of tiles on a square grid. For a configuration A and vector $\vec{u} = \langle u_x, u_y \rangle \in \mathbb{Z}^2$, $A + \vec{u}$ denotes the configuration $f \circ A$, where $f(x, y) = (x + u_x, y + u_y)$. For two configurations A and B, B is a *translation* of A, written $B \simeq A$, provided that $B = A + \vec{u}$ for some vector $\vec{u}$. For a configuration A, the *assembly* of A is the set $\tilde{A} = \{B : B \simeq A\}$. An assembly $\tilde{A}$ is a *subassembly* of an assembly $\tilde{B}$, denoted $\tilde{A} \sqsubseteq \tilde{B}$, provided that there exists an $A \in \tilde{A}$ and $B \in \tilde{B}$ such that $A \subseteq B$. The *shape* of an assembly $\tilde{A}$ is $\{\text{dom}(A) : A \in \tilde{A}\}$ where dom() is the domain of a configuration. A shape S' is a *scaled* version of shape S provided that for some $k \in \mathbb{N}$ and $D \in S$, $\bigcup_{(x,y) \in D} \bigcup_{(i,j) \in \{0,1,\ldots,k-1\}^2} (kx + i, ky + j) \in S'$.

Bond graphs and stability. For a configuration A, define the *bond graph* G_A to be the weighted grid graph in which each element of $\text{dom}(A)$ is a vertex, and the weight of the edge between a pair of tiles is equal to the strength of the coincident glue pair. A configuration is *τ-stable* for $\tau \in \mathbb{N}$ if every edge cut of G_A has strength at least τ, and is *τ-unstable* otherwise. Similarly, an assembly is *τ-stable* provided the configurations it contains are τ-stable. Assemblies $\tilde{A}$ and $\tilde{B}$ are *τ-combinable* into an assembly $\tilde{C}$ provided there exist $A \in \tilde{A}$, $B \in \tilde{B}$, and $C \in \tilde{C}$ such that $A \bigcup B = C$, $\text{dom(A)} \bigcap \text{dom(B)} = \emptyset$, and $\tilde{C}$ is τ-stable.

Two-handed assembly and bins. We define the assembly process via bins. A bin is an ordered tuple (S, τ) where S is a set of *initial* assemblies and $\tau \in \mathbb{N}$ is the *temperature*. In this work, τ is always equal to 2 for upper bounds, and at most some constant for lower bounds. For a bin (S, τ), the set of *produced* assemblies $P'_{(S,\tau)}$ is defined recursively as follows:

1. $S \subseteq P'_{(S,\tau)}$.

2. If $A, B \in P'_{(S,\tau)}$ are τ-combinable into C, then $C \in P'_{(S,\tau)}$.

A produced assembly is *terminal* provided it is not τ-combinable with any other producible assembly, and the set of all terminal assemblies of a bin (S, τ) is denoted $P_{(S,\tau)}$. That is, $P'_{(S,\tau)}$ represents the set of all possible assemblies that can assemble from the initial set S, whereas $P_{(S,\tau)}$ represents only the set of assemblies that cannot grow any further.

The assemblies in $P_{(S,\tau)}$ are *uniquely produced* iff for each $x \in P'_{(S,\tau)}$ there exists a corresponding $y \in P_{(S,\tau)}$ such that $x \sqsubseteq y$. Unique production implies that every producible assembly can be repeatedly combined with others to form an assembly in $P_{(S,\tau)}$.

Staged assembly systems. An *r-stage b-bin mix graph M* is an acyclic r-partite digraph consisting of rb vertices $m_{i,j}$ for $1 \leq i \leq r$ and $1 \leq j \leq b$, and edges of the form $(m_{i,j}, m_{i+1,j'})$ for some i, j, j'. A *staged assembly system* is a 3-tuple $\langle M_{r,b}, \{T_1, T_2, \ldots, T_b\}, \tau \rangle$ where $M_{r,b}$ is an r-stage b-bin mix graph, T_i is a set of tile types, and $\tau \in \mathbb{N}$ is the temperature. Given a staged assembly system, for each $1 \leq i \leq r$, $1 \leq j \leq b$, a corresponding bin $(R_{i,j}, \tau)$ is defined as follows:

1. $R_{1,j} = T_j$ (this is a bin in the first stage);

2. For $i \geq 2$, $R_{i,j} = \left(\bigcup_{k\colon\, (m_{i-1,k}, m_{i,j}) \in M_{r,b}} P_{(R_{(i-1,k)}, \tau_{i-1,k})} \right)$.

Thus, bins in stage 1 are tile sets T_j, and each bin in any subsequent stage receives an initial set of assemblies consisting of the terminally produced assemblies from a subset of the bins in the previous stage as dictated by the edges of the mix graph.[1] The *output* of a staged system is the union of the set of terminal assemblies of the bins in the final stage.[2] The output of a staged

[1]The original staged model [38] only considered $O(1)$ distinct tile types, and thus for simplicity allowed tiles to be added at any stage (since $O(1)$ extra bins could hold the individual tile types to mix at any stage). Because systems here may have super-constant tile complexity, we restrict tiles to only be added at the initial stage.

[2]This is a slight modification of the original staged model [38] in that there is no requirement of a final stage with a single output bin. It may be easier in general to solve problems in this variant of the model, so it is considered for lower bound purposes. However, all results herein apply to both variants of the model.

system is *uniquely produced* provided each bin in the staged system uniquely produces its terminal assemblies.

2-Handed Tile Assembly Model (2HAM)

Here we define the two-handed tile self-assembly model (2HAM) with both negative and positive strength glue types, as well as formulate the problem of designing a tile assembly system that replicates any input shape. The 2HAM model was first presented in [68]. The 2HAM is a variant of the abstract tile self-assembly model (aTAM) first presented in [20] in which there is no seed tile, and large assemblies may combine together.

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

For a given positioned assembly Υ, define the *bond graph* G_Υ to be the weighted grid graph in which each element of Υ is a vertex and the weight of an edge between tiles is the strength of the matching coincident glues or 0.[3] A positioned assembly C is said to be τ*-stable* for positive integer τ provided the bond graph G_C has min-cut at least τ, and C is said to be *connected* if every pair of vertices in G_C has a connecting path using only positive strength edges.

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

[3]Note that only matching glues of the same type contribute a non-zero weight, whereas non-equal glues always contribute zero weight to the bond graph. Relaxing this restriction has been considered as well [28].

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

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

Note that A and B may be combinable into an assembly that is not stable. This is a key property that is leveraged throughout our constructions. See Figure 2.1 for an example.

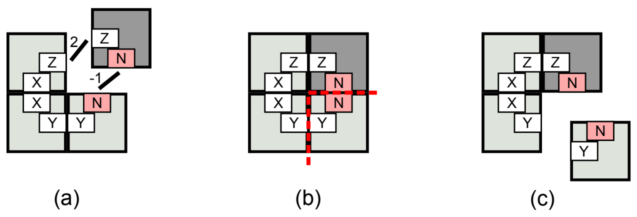


Figure 2.1: A simple example of attachment and detachment events and the notation for the shape replication construction. On each tile, the glue label is presented. Red (shaded) labels represent negative glues, and the relevant glue strengths for the tiles can be found in the captions. For caption brevity,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

States, Tile Systems, and Assembly Sequences. A *state* is a multiset of assemblies whose counts are in $Z^{+} \bigcup \infty$. We use notation $S(x)$ to denote the multiplicity of an assembly x in a state S. A state S_1 *transitions* to a state S_2 at temperature τ, and is written as $S_1 \rightarrow_1^{\tau} S_2$, if S_2 is obtained from S_1 by either replacing a pair of combinable assemblies in S_1 with their combination assembly, or by replacing a breakable assembly in S_1 by its pieces. We simplify this to $S_1 \rightarrow_1 S_2$ when τ is clear from context. We write $S \rightarrow^\tau S'$ to denote the transitive closure of $\rightarrow^\tau_1$, i.e., $S \rightarrow^\tau S'$ means that $S \rightarrow^\tau_1 S_1 \rightarrow^\tau_1 S_2 \rightarrow^\tau_1 \ldots S_k \rightarrow^\tau_1 S'$ for some sequence of states $\langle S_1, \ldots S_k \rangle$.

A *tile system* is an ordered tuple $\Gamma = (\sigma, \tau)$ where σ is a state called the *initial system state* and τ is a positive integer called the *temperature*. We refer to any sequence of states $\langle \sigma, S_1, S_2, \ldots, S_k \rangle$ such that $\sigma \rightarrow^\tau_1 S_1$, and $S_i \rightarrow^\tau_1 S_{i+1}$ for $i = \{1 \ldots k-1\}$, as a *valid assembly sequence* for Γ. We denote any state P at the end of of a valid assembly sequence (equivalently, $\sigma \rightarrow^\tau P$) as a *producible state* of Γ, and any assembly contained in a producible state is said to be a *producible assembly*. A producible state T is said to be terminal if T does not transition at temperature τ to any other state. A producible assembly is *terminal* if it is not breakable and not combinable with any other producible assembly.

CHAPTER III

RANDOM NUMBER GENERATION

Introduction

A promising new direction in self-assembly is the consideration of *randomized* self-assembly systems. In randomized self-assembly (a.k.a. nondeterministic self-assembly), assembly growth is dictated by nondeterministic, competing assembly paths yielding a probability distribution on a set of final, terminal assemblies. Through careful design of tile-sets, and the relative concentration distributions of these tiles, a number of new functionalities and efficiencies have been achieved that are provably impossible without this nondeterminism. For example, by precisely setting the concentration values of a generic set of tile species, arbitrarily complex strings of bits can be *programmed* into the system to achieve a specific shape with high probability [39], [49]. Alternately, if the concentration of the system is assumed to be fixed at a uniform distribution, randomization still provides for efficient expected growth of linear assemblies [61] and low-error computation at temperature-1 [55]. Even in the case where concentrations are unknown, randomized self-assembly can build certain classes of shapes without error in a more efficient manner than without randomization [67].

Motivated by the power of randomized self-assembly, along with the potential for even greater future impact, we focus on the development of the most fundamental randomization primitive: the *robust* generation of a uniform random bit. In particular, we introduce the problem of self-assembling a uniformly random bit that is guaranteed to work for all possible concentration distributions. We define a tile system to be a *coin flip* system, with respect to some tile concentration distribution, if the terminal assemblies of the system can be partitioned such that each partition has exactly probability 1/2 of assembling one of its terminals. We say a system is a *robust coin flip* system if such a partition exists that guarantees 1/2 probability for all possible tile concentration distributions. Through designing systems that flip a fair coin for all possible (adversarially chosen) concentration distributions, we achieve an intrinsically fair coin-flipping system that is robust to the experimental realities of imprecise quantity measurements. Such fair systems may allow for increased scalability of randomized self-assembly systems in scenarios where exact concentrations of species are either unknown or intractable to predict at successive assembly stages.

Unbounded Space, 1-Extensible, Robust Coin Flipping

Below, we describe a 1-extensible aTAM system capable of robust fair coin flips in unbounded space. In 1951, John von Neumann gave a simple method for extracting a fair coin from a biased one [1]. We show an algorithm based on the Von Neumann extractor. Algorithm 1 uses an unbounded number of *rounds* to extract a fair coin flip. We use Algorithm 1 to show that a *fair coin flip extractor* can be implemented in the aTAM to achieve an unbounded space, 1-extensible, robust coin flip tile system. Let Algorithm 2 denote an extension of this method in which we create a *bounded fair coin flip extractor* by adding a parameter k which controls the maximum number of rounds allowed. If after all k rounds have been exhausted the system has not returned a fair coin flip, the result of a single flip is returned. We implement this *bounded coin flip extractor* in the aTAM and achieve an s space, 1-extensible, and robust coin flip tile system with bounded bias, for some space constraint s.

We now describe our 1-extensible aTAM tile system that implements Algorithm 1. In Algorithm 1, a coin is a set of cardinality 2 with possible values *heads* and $tails$. *flip* is a function that selects and returns a *heads* or *tails* value based on the probabilities h and t, respectively, where $h, t \in (0, 1)$ and $h + t = 1$. In our construction, calls to the *flip* function are carried out by a nondeterministic competition for attachment between a *h tile* and a *t tile*. Aside from calls to the *flip* function, the rest of the algorithm can be implemented by deterministic tile placements. Figure 3.1 gives the tile set used in the construction. This construction yields Theorem 3.1, and an example is shown in Figure 3.2.

Algorithm 1: Unbounded

```
Input: $h, t \in \mathbb{R}$, where $0 < h < 1, t = 1 - h$
Output: heads or tails
1: procedure UnboundedFCFE($h, t$)
2:     $coin = \{heads, tails\}$
3:     $pdist = \{h, t\}$
4:     repeat
5:         $flip\_1 \leftarrow flip(coin, pdist)$
6:         $flip\_2 \leftarrow flip(coin, pdist)$
7:     until $flip\_1 \neq flip\_2$
8:     return $flip\_2$
9: end procedure
```

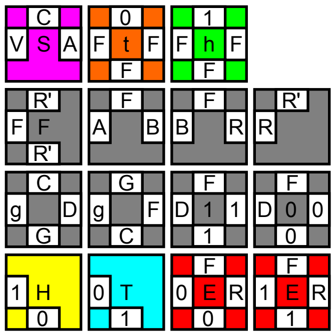
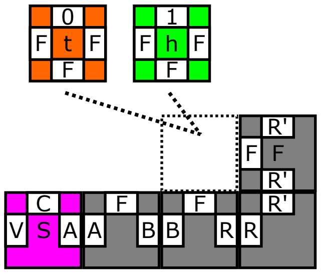


Figure 3.1: Here we have the algorithm for an unbounded fair coin flip extractor on the left and the tile set for the construction that implements that algorithm on the right. The relative concentrations of the *h tile* and *t tile* serve as parameters h and t, respectively. The tile labeled S is the seed of the tile assembly system and the temperature is 2. The strength of the glues are as follows: str(0)=1, str(1)=1, str(A)=2, str(B)=2, str(C)=1, str(D)=1, str(F)=1, str(G)=2, str(R)=2, and str(R')=2.

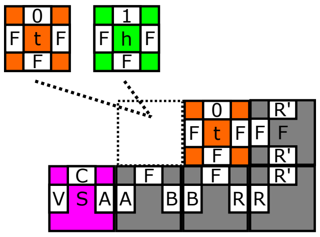
Theorem 3.1. There exists a 1-extensible tile system $\Gamma = (T, \sigma, 2)$ in the aTAM that implements a robust fair coin flip tile system (unbounded) and achieves $O\left(\frac{1}{pq}\right)$ expected space, where p and q denote the relative concentrations of the two tiles with the largest difference in concentration for a given concentration distribution.

Proof. Let the probability that flipping a single coin is heads be represented by u and tails be represented by $1 - u$. In our construction, we have a *h tile* and a *t tile* with concentrations C_h and C_t, respectively. Let $u = \frac{C_h}{C_h + C_t}$ and $v = 1 - u$. In each round, we flip two times. Let the probability of generating a bit each round be $g = 2uv$. Then, let the expected total number of flips be t. If we succeed in the first round, we have only flipped twice. Otherwise, we have to start over, so the expected remaining number of flips is still two. Therefore, $t = 2g + (1 - g)(2 + t) = \frac{2}{g}$.

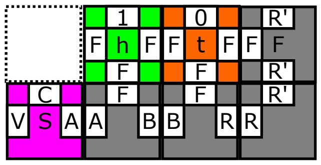
Using this strategy, each round requires two flips. Heads and tails each have a probability uv of being generated. Thus, each round can succeed with a probability $2uv$ and the average number of flips required to generate a bit is $\frac{2}{g} = \frac{2}{2uv} = \frac{1}{uv}$. Since each round utilizes two flips, the expected number of rounds is then $\frac{t}{2} = \frac{1}{2uv}$. In the best case, $u = v$ and the expected number of rounds would



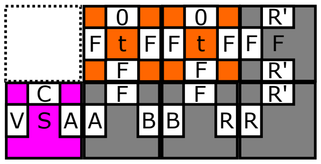
(a) An assembly with two possible choices for the next attachment corresponding to the first flip in the algorithm.



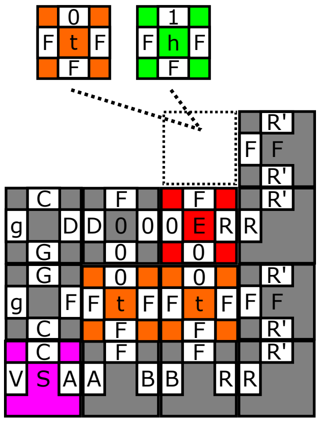
(b) Without loss of generality, this shows possible choices for the second flip of the algorithm after the first has been chosen.



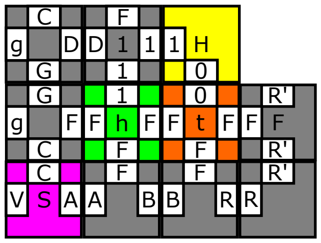
(c) A *t tile* and a *h tile* have been placed for the first and second flip, respectively. From Algorithm 1, this will return a *heads*.



(d) Two *t tiles* were placed for the first two flips. From Algorithm 1, the system must perform another round.



(e) An assembly where the first round of the algorithm failed to generate a bit and proceeds to start a new round.



(f) An assembly where the first round of the algorithm was a valid flip and it generates a *heads*.

Figure 3.2: A sample of producible assemblies for Round 1.

be $\frac{1}{2uv} = 2$. In the worst case, the two tiles with the largest difference in concentration are the *h tile* and *t tile* implying $\frac{1}{2uv} = \frac{1}{2pq}$. Each round places a constant number of tiles z, therefore the expected space of generating a coin flip is the expected number of rounds multiplied by the number of tiles per round, $\frac{z}{2pq} = O\left(\frac{1}{pq}\right)$. The placement of an *H tile* or *T tile* maps to the event that the algorithm returns *heads* or *tails*, respectively. □

Fixed Space, 1-Extensible, Robust Coin Flipping

We can also now limit the number of rounds, k, so that space of the system does not exceed some constraint s by using some additional tile types and modification to glue strengths. If after k rounds, the system has not returned a fair coin flip, the system returns the result of a single additional flip of the two tiles used in nondeterministic attachment. This *bounded fair coin flip extractor* can be implemented in the aTAM to achieve a fixed space, 1-extensible, robust coin flip tile system with bounded bias. The bounded k-rounds can be controlled by first constructing a column of height $O(k)$ with glues that allow the variant of the construction of Theorem 3.1 to grow along the right edge of the column. Note that this column can be built more efficiently, by allowing some width, using a 1-extensible version of the aTAM counter construction from [28] for a desired base, leading to a tradeoff in bias, space, and tile complexity.

Theorem 3.2. There exists an s space 1-extensible robust coin flip tile system in the aTAM with bias $p^{(s/10)}$, where p is the larger relative concentration from the pair of tiles with the largest difference in concentration for a given concentration distribution.

Proof. We need at most 7 tile placements to perform a single additional flip when we fail all allotted rounds and each round places 10 tiles. We design a system that can perform as many possible rounds, k, given $s - 7$ space, where

$$\frac{s}{10} - 1 < k = \lfloor (s-7)/10 \rfloor < \frac{s}{10}. \tag{3.1}$$

In the worst case, the two tile types with the largest difference in concentration, for a given concentration distribution, are the two tile types used in nondeterministic attachment. In our construction, those tiles are the *h tile* and a *t tile* with concentrations C_h and C_t, respectively. Without loss of generality, consider that $C_h > C_t$ and thus, $p = \frac{C_h}{C_h + C_t}$ and let $q = 1 - p$. Let $P(X = heads)$ denote the probability that this system returns a heads and $P(X = tails) = 1 - P(X = heads)$. Let F_k denote the probability that the system fails to return a coin flip after k rounds, that is $F_k = (1 - 2pq)^k$. Therefore, $P(X = heads) = pF_k + \frac{1 - F_k}{2}$ and

$$\frac{|P(X = heads) - P(X = tails)|}{2} = F_k\left(p - \frac{1}{2}\right). \tag{3.2}$$

And, since $\frac{1}{2} < p < 1$ and $p^k \geq F_k$,

$$\begin{aligned} F_k\left(p - \frac{1}{2}\right) &\leq p^k\left(p - \frac{1}{2}\right) \\ &\leq p^{k+1} \\ &< p^{\frac{s}{10}}. \end{aligned} \tag{3.3}$$

Therefore,

$$2p^{\frac{s}{10}} \geq |P(X = heads) - P(X = tails)| \tag{3.4}$$

which implies this system has bias $p^{\frac{s}{10}}$. □

The Big Picture

The 1-extensible results detailed above show that a robust fair coin flip can be generated in unbounded space, but when the amount of space is bounded, the construction incurs bounded bias. A natural question then would be, is there an aTAM construction that constitutes a robust fair coin flip system which completes in a guaranteed $O(1)$ space even at temperature one? This and other results, are included in the published version [77] of this body of work. The results detailed above showcase my primary contributions to [77].

In [77], our primary result is an aTAM construction that constitutes a robust fair coin flip system which completes in a guaranteed $O(1)$ space even at temperature one. We apply our robust coin-flip construction to the result of Chandran, Gopalkrishnan, and Reif [61] to show that for any positive integer n, there exists a $O(\log n)$ tile system that assembles a constant width-4 linear assembly of expected length n that works for all concentration assignments. This result is for temperature two; at temperature one it must be a width-6 linear assembly. We accompany this result with a proof that such a concentration independent assembly of width-1 assemblies is not possible with fewer than n tile types. We further accompany our main coin-flip construction with variant constructions that provide trade-offs among standard aTAM metrics such as space, tile complexity, and temperature, as well as new metrics such as coin bias, and the *extensibility* of the system, which is the maximum number of distinct locations a tile can be added to a single producible assembly of the system.

We utilize the coin-flip construction as a fair random bit generator for implementation of some classical random number generation algorithms. We show that 1-extensible systems, while computationally universal, cannot robustly coin-flip in bounded space without incurring a bias, but can robustly coin-flip in bounded expected space. We also consider the more extreme model in which concentrations may change adversarially at each assembly step. We show that the aTAM cannot robustly coin flip in bounded space within this model, but a number of more exotic extensions of the aTAM from the literature are able to robustly coin flip in $O(1)$ space. The problem of self-assembling random bits has been considered before [45], but their technique, and almost all randomized techniques to date, do not work when arbitrary concentrations are considered. Further, we utilize the self-assembly of uniform random bits to implement algorithms for uniform random number generation for any n, one construction achieving an unbiased generator with unbounded space and the other imposing a space constraint while incurring some bias.

CHAPTER IV

INFORMATION ENCODING

Introduction

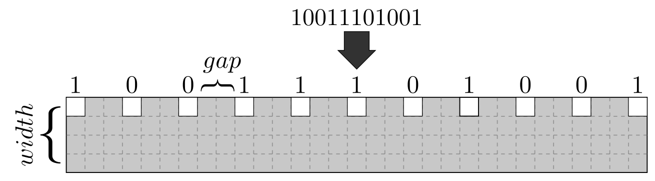
The *staged (tile assembly) model* is a generalization of the *two-handed (2HAM)* [38], [48], [68], [69] or *hierarchical* [62], [75] *tile assembly model*. In the aTAM, single tiles attach to a growing multi-tile seed assembly. The 2HAM additionally permits multi-tile assemblies attach to each other, yielding a growth process strictly more general than that of the aTAM [68].

The staged assembly model extends the 2HAM by including the ability to simultaneously carry out separate assembly processes in multiple *bins*. Such separation is motivated by similar experimental practices wherein many reactions are carried out simultaneously in distinct test tubes, often automated via liquid-handling robots [65].

In the staged model, a bin begins with a set of *input assemblies* previously assembled in other bins. These assemblies are repeatedly attached pairwise to yield a growing set of *producible assemblies* until all possibilities are exhausted. The producible assemblies not attachable to any other producible assemblies during this process are the *output assemblies* of the bin, and may be used as input assemblies for other bins.

An instance of the staged model, called a *staged system*, consists of many bins stratified into stages, and a *mix graph* that specifies which bins in each stage supply input assemblies for bins in the following stage. Input assemblies for bins in the first stage are sets of individual tiles, and the output assemblies of bins in the final stage are considered the *output* of the system.

Complexity in staged assembly. A common goal in the design of self-assembling systems is production terminal assemblies that meets some criteria. Here we consider the design of "efficient"



(a)

Figure 4.1: Example of a bit string pad. For compactness, glues are denoted by their subscripts in figures for the remainder of this chapter (e.g. g_0 denoted as 0). An example bit string $r = 10011101001$ encoded as a width-4 gap-2 11-*bit string pad* whose north-facing glues correspond to the bits in r.

systems with minimum complexity that assemble our target assemblies. Three metrics exist for staged systems:

- *Tile type complexity:* the number of distinct tile types used in the system.
- *Bin complexity:* the maximum number of bins used in a stage.
- *Stage complexity:* the number of stages.

Below is a description of the efficient assembly of *bit string pads*, shown in Figure 4.1: assemblies that expose a sequence of north-facing glues that encode a bit string.

Definition 4.1 (Bit string pad)**.** A *width-k gap-f r-bit string pad* is a $k \times (f(r-1)+1)$ rectangular assembly with r glues from a set of two glue types $\{g_0, g_1\}$ exposed on the north face of the rectangle at intervals of length f, starting from the westmost northern edge. All remaining exposed glues on the north tile edges have some common label g_F. The remaining exposed south, east, and west tile edges have glues g_S, g_E, and g_W. A bit string pad *represents* a given string of r bits if the exposed g_0 and g_1 glues from west to east, on the north facing edges, are equal to the given bit string.

The ability to reliably encode bit strings on assemblies is a fundamental primitive for many self-assembly algorithms. Two upper-bound constructions are detailed in the proceeding sections of this chapter. In the “Big Picture” section, lower bounds are stated that motivate these constructions as well how these constructions play a role in the published version [85] of this work.

Unique Bit String Pads in Separate Bins

In this section, a b bin system constructs all possible $\lfloor \log b \rfloor$-bit string pads, each contained in a distinct bin.

Lemma 4.2. There exists a constant c such that for any $b, t \in \mathbb{N}$ with $b, t > c$, there exists a $\tau = 2$ staged assembly system with b bins, t tile types, and $O(\frac{\log \log b}{\log t})$ stages whose uniquely produced output is all width-2 gap-1 $\lfloor \log(b) \rfloor$-bit string pads, each placed in a distinct bin.

Proof. Let $\gamma = \lfloor \frac{t-4}{2} \rfloor + 1$. The construction has three cases, depending upon the relative values of γ (determined by t) and b.

Case 1: $\gamma < 2 \log b$ **and** $b = \gamma^n$ **for** $n \in \mathbb{N}$**.** All three cases use width-2, gap-0 1-bit string pads called *binary gadgets*: $O(1)$-sized assemblies representing 0 and 1. The remaining tile types are used to assemble $2(\gamma - 1)$ *connectors* that bind to the west and east ends of binary gadgets. Connectors come in *west* and *east* varieties and connector has a positive integer *index* such that two connectors can bind to one another if and only if their indices are equal.

For each $k \in \{1, 2, \ldots, \gamma - 1\}$, assemble west and east connectors with index k. Then repeat the following 2-stage "round" seen in Figure 4.3. The ith round begins with $2^{\gamma^{i-1}}$ binary gadgets, each in their own bin, and each of the west and east connectors stored in their own bins. In the first stage of the round, mix each binary gadget with west and east connectors with indices $h - 1$ and h, respectively, for all $h \in \{1, 2, \ldots, \gamma - 1\}$, In the special case that $h = 1$ or $h = \gamma - 1$, omit the west or east connector, respectively. The result is $\gamma 2^{\gamma^i}$ distinct assemblies, with each assembly in its own bin. The tile set for bit string pads and connectors can be seen in Figure 4.2.

In the second stage of the round, selectively mix the γ-size subsets of the $\gamma 2^{\gamma^{i-1}}$ bins, choosing one binary gadget with each pair of connector indices, to assemble all width-2 γ^i-bit string pads in separate bins. The number of bins used in round i is thus $2^{\gamma^i} + \gamma$: enough for all γ^i-bit string pads and the γ (reusable) connectors.

Perform sufficient rounds to assemble all $\lfloor \log(b) \rfloor$-bit string pads, i.e., the smallest integer r such that $\gamma^r \geq \log(b)$ and thus $r = \lfloor \log_\gamma \log(b) \rfloor$. In the last stage of round r, the number of bins

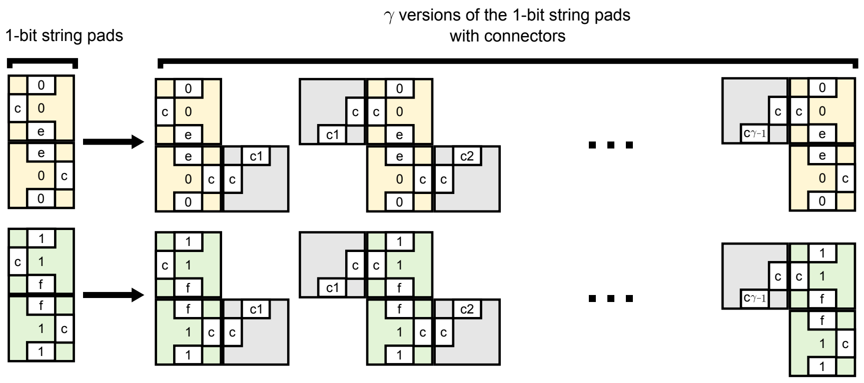


Figure 4.2: Example of bit string pads and winged bit string pads. Left: 2 width-2 1-bit string pads. Right: 2γ versions of each bit string pad with pairs of west and east connectors attached, as done in the first stage of each round.

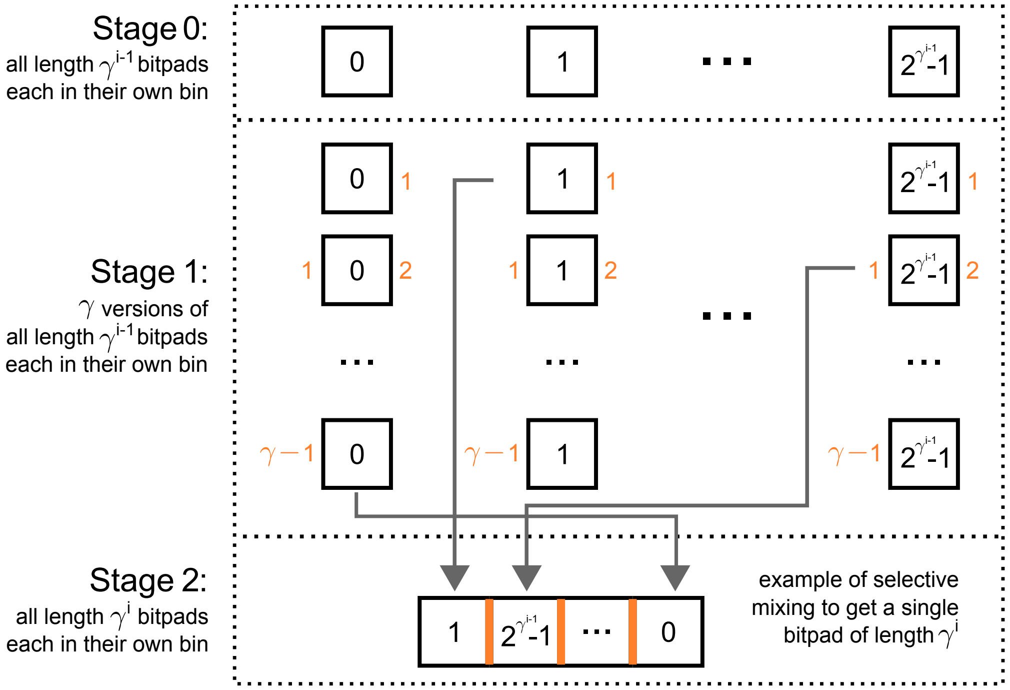
used can be reduced to 2^{γ^r}, dropping the γ additional bins containing connectors.

The number of bins used then does not exceed b since $2^{\gamma^i} + \gamma \leq 2^{\gamma^r} \leq b$ for all $i < r$. Moreover, the number of stages used is $O(r) = O(\frac{\log\log b}{\log \gamma}) = O(\frac{\log\log b}{\log t})$. This construction requires $b, t > c$ for some constant c large enough to ensure $\gamma \geq 1$ and b can accommodate at least 1 of each of the bin types.

Case 2: $\gamma < 2\log(b)$ **and** $b \neq \gamma^n$ **for** $n \in \mathbb{N}$**.** Note that this case is identical to Case 1, except that b is not a power of γ. Thus the desired length of the assembled bit string pads are between γ^{r-1} and γ^r for some r; as a solution, bit string pads are assembled from collections of shorter bit string pads of power-of-γ lengths.

Let $d_1 d_2 \ldots d_{\lfloor \log_\gamma(b) \rfloor}$ be the base-γ expansion of b, with D_i representing $d_{\lfloor \log_\gamma(b) \rfloor - i + 1}$. New *special* connectors with index 0 are used, in addition to the *standard* connectors from Case 1. Each round also uses *growth* bins, *output* bins, and *west* and *east incubator* bins. West and east incubator bins contain growing $\sum_{j=1}^{i} D_j \gamma^{j-1}$-bit string pads with special west or east connectors, respectively.

As in Case 1, use $\log_\gamma(b)$ rounds to assemble growing sets of longer bit string pads; the ith round begins with all γ^{i-1}-bit string pads in separate growth bins. In the first stage of the ith round, mix standard connectors with these pads to assemble γ versions of all γ^{i-1}-bit string pads in separate growth bins. Also, the output bins are carried down from the previous stage. In the second stage of the ith round, if $D_i > 0$, selectively mix γ-sized subsets of bit string pads with



Stage 0:
all length γ^{i-1} bitpads each in their own bin
Stage 1:
γ versions of all length γ^{i-1} bitpads each in their own bin
Stage 2:
all length γ^{i} bitpads each in their own bin
0
1
$2^{\gamma^{i-1}}-1$
$\gamma-1$
example of selective mixing to get a single bitpad of length γ^{i}

Figure 4.3: A 2-stage round used in Case 1 of Lemma 4.2. Top: at Round i Stage 0, every γ^{i-1}-bit string pad is in a separate bin (labeled according to index). Middle: in Stage 1, each of the γ^{i-1}-bit string pads is mixed with γ different pairs of west and east connectors with consecutive values, yielding γ versions of each γ^{i-1}-bit string pads. Bottom: in Stage 2, bit string pads from Stage 1 are mixed in size-γ groups, assembling every possible γ^{i}-bit string pad.

standard connectors (stored in growth bins) to assemble all $D_i\gamma^{i-1}$-bit string pads in separate bins. When $i = 1$ or the output bins are empty, these bit string pads are stored in their own output bins. Otherwise, the output bins are not empty; and they are also mixed with west connectors and stored in west incubator bins. In the same stage, mix every bit string pad, from the output bins, with a special east connector and store them in east incubator bins. Then, in a third stage mix all pairs of west and east incubator bins into separate output bins, replacing any previous output bins.

For all but the last round, mix γ-sized subsets of bit string pads with standard connectors (stored in growth bins) to assemble all γ^{i}-bit string pads in separate growth bins, replacing the previous growth bins. Since each round concatenates $D_i\gamma^{i-1}$ bits to the bit string pads in output bins, the bit string pads assembled in the output bins of the final round represent $\sum_{j=1}^{\lfloor\log_\gamma(b)\rfloor} D_j\gamma^{j-1} = \lfloor\log(b)\rfloor$

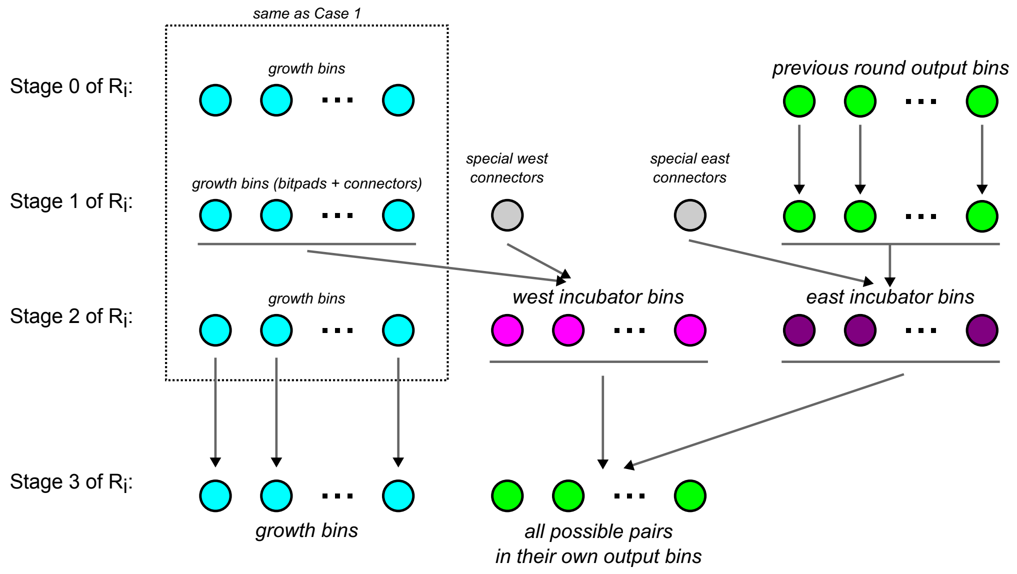


Figure 4.4: A 3-stage round used in Case 2 of Lemma 4.2. The bins labeled growth bins follow the same process as Case 1. Special west connectors are mixed with D_i of the bitpads (from the growth bins of Stage 1 of Round i) to produce the bitpads stored in the west incubator bins. Special east connectors are mixed with the bitpads stored in the outputs bins (produced in the previous round) to produce the bitpads stored in the east incubator bins. In Stage 3, all possible pairs of bitpads from west and east incubator bins are mixed to produce the bitpads stored in the output bins for Round i. If it is not the final round, the growth bins are also carried down.

bits. The high-level idea for the construction can be seen in Figure 4.4.

This construction requires $b, t > c$ for some constant c large enough to ensure that $\gamma \geq 1$ and at least one of each bin type (growth, output, etc.) is available. The number of stages remains $O(\frac{\log \log b}{\log t})$.

Case 3: $\gamma \geq 2 \log(b)$**.** Let $\beta = \lfloor \log(b) \rfloor$. Begin with β bins, each containing a binary gadget representing 0 with a distinct pair of west/east glues g_1 and g_2, g_2 and g_3, etc. Similarly, create a second set of identical β bins, but with binary gadgets representing 1. In the second stage, combine every β-sized subset of bins that contain binary gadgets with distinct west/east glue pairs to assemble all β-bit string pads. □

Fattening

Roughly speaking, "fattening" replaces a single tile type with a set of 5 corresponding tile types and a generic *filler* assembly of length $k-2$ to yield a $2 \times k$ assembly which exposes the same glues as the original tile (see Figure 4.5). Fattening is used as a subroutine for assembling decompression pads and can also further be used to increase the gap between the bitpads generated.

A *filler assembly* is an assembly of length $k-2$ used to fatten a tile type to some length k. The technique in Section IV for assembling bit string pads can be used to assemble length-$\Theta(\log b)$ filler assemblies. Lemma 4.2 yields all length-$\log(b)$ bit string pads in $O(\frac{\log\log b}{\log t})$ stages; small modifications yields a filler assembly of the same length in the same complexity.

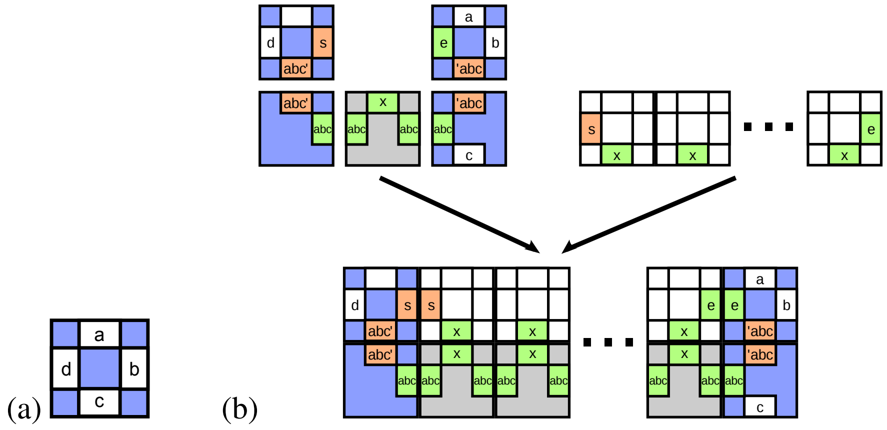


Figure 4.5: "Fattening" a single tile type from width 1 to width k. (a) The tile to be fattened is shown. (b) 5 tile types, based on the tile to be fattened, are mixed with a length-$(k-2)$ filler assembly which "fattens" the assembly from width 1 to width k. The four glues of the original are exposed in the same directions.

Encoding Via Tile Types

Here the goal is to design a collection of t tile types that assembles a target string of $\Theta(t \log t)$ bits in bit string pad form. The solution is to utilize the base conversion approach of [15], [28], [37], [81]. In this approach, tile types optimally encode integer values in a higher base (than binary) and then are "decompressed" into a binary representation. In total, t tile types are used to encode a value in a high base and decompress this representation into a string of $\Theta(t \log t)$ bits.

Definition 4.3 (Decompression pad)**.** For $k, d, x \in \mathbb{N}$ and $u = 2^x$, a width-k d-digit base-u decompression pad is a $k \times dx$ rectangular assembly with d glues from a set of $u-1$ glue types $\{g_0, g_1, ..., g_{u-1}\}$ exposed on the north face of the rectangle at intervals of length $x-1$ and starting from the westmost northern edge. All remaining glues on the north surface have a common type g_N. The remaining exposed south, east, and west tile edges have glues g_S, g_E, and g_W. The exposed glues on the northern edge, disregarding glues of type g_N, $g_{a_1}, g_{a_2}, \ldots, g_{a_d}$ represent string $a_1, a_2, \ldots, a_d$.

Consider the following example, also seen in Figure 4.6. Let $S = 010100000$ ($S = 240$ in base 8) be a bit string, with the goal of constructing a width-3 9-bit string pad representing S. First, build a decompression pad representing S in base 8 by combining 3 different $3 \times \log_2(8)$ blocks. Then convert the decompression pad into a bit string pad representing S using $O(u)$ tile types, where $u = 8$.

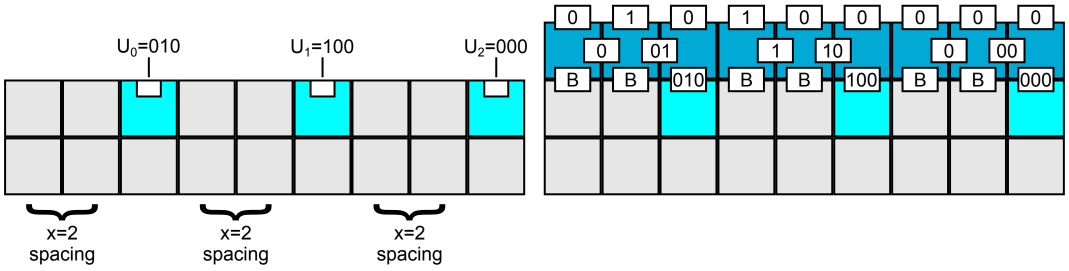
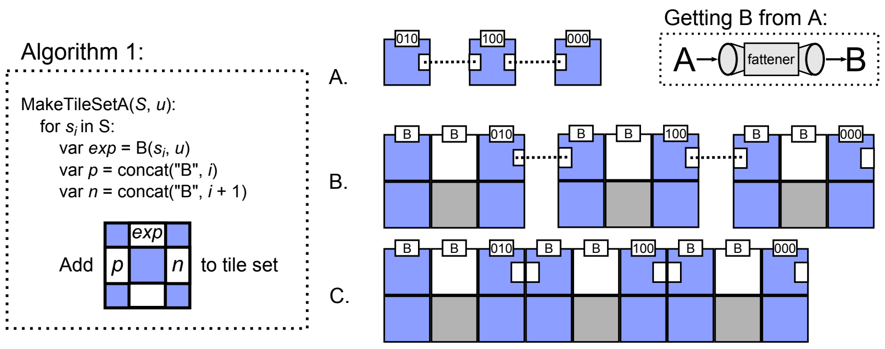


Figure 4.6: Example of "decompressing" a decompression pad into bit string pad. Left: a width-2 gap-($\log(u)-1$) decompression pad representing a bit string $S = 010100000$ in base $u = 8$. Right: $O(u)$ decompression tiles interact with the north glues of the decompression pad to combine into a width-3 bit string pad representing S in base 2.

Lemma 4.4. Given integers $x \geq 3$, $d \geq 1$ and $u = 2^x$, there exists a 1-stage, 1-bin staged $\tau = 2$ self-assembly system whose uniquely produced output is a d-digit decompression pad of width-2 and base-u, using at most $5d + \log(u) - 2$ tile types.

Proof. Consider a string of d digits, $S = s_0 s_1 \ldots s_{d-1}$ in a base u. We want to build a *decompression pad* that has glues on the north facing side representing each of the d digits with $\log(u) - 1$ spacing between the d digits (to have room to unpack into base-2) and with g_B glues on the north surface of those $\log(u) - 1$ regions. We can use the algorithm in Figure 4.7 to create the tiles that build the



```
MakeTileSetA(S, u):
  for s_i in S:
    var exp = B(s_i, u)
    var p = concat("B", i)
    var n = concat("B", i + 1)
```

Figure 4.7: Generating the tile sets for decompression pads. Using the algorithm on the left, we can generate a tile for each digit in the string, $S = 240$ with base $u = 8$, to get the tiles in A. We can fatten the tiles in A to length $\log(u)$ to get the tile set in B. We modify the fattening process described in Section IV such that all glues on the north surface, other than the eastmost one, are of type g_B. Finally, C shows us how the tile set of B can self-assemble into a decompression pad for string S. The algorithm takes as input a string of digits in base u, $S = s_0 s_1 \ldots s_{d-1}$. For every digit in the string, it creates a new tile type with the digit, converted to binary, as its north glue. The east glues and west glues bind to the next and previous digits in the sequence, respectively.

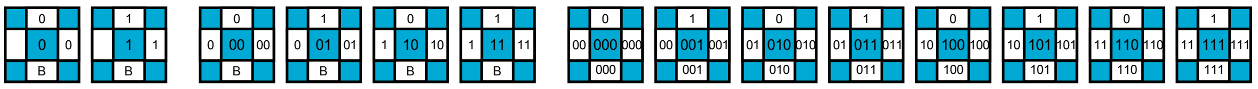


Figure 4.8: The set of tiles used to decompress a decompression pad representing a string in base 8. It requires $2u - 2$ tile types, where base $u = 8$, in this example. Similar tile sets can be made for any base u, using $2u - 2$ tiles. These tiles interact with glues exposed on a decompression pad and extract bits along the north surface.

decompression pad by first considering a tile for each of the d digits, then fattening it so that it has length $\log(u)$. This algorithm creates a series of length-$\log(u)$ decompression pads for each digit in S. These length-$\log(u)$ decompression pads self-assemble into a length-$d\log(u)$ decompression pad representing S. The length-$\log(u)$ decompression pads are created via fattening singleton tiles that chain together so that each singleton becomes a $2 \times \log(u)$ block. For an overview of fattening, see Section IV. Each block uses a shared filler of length $\log(u) - 2$ built with $\log(u) - 2$ tile types. Each digit requires 5 tile types to fatten. Therefore, the total tile complexity is $5d + \log(u) - 2$. □

Lemma 4.5. Given integers $d \geq 3$, $x \geq 3$, $u = 2^x$, and bit string S of length $d\log(u)$, there exists a

$\tau = 2$ staged assembly system with 1 bin, $5d + 2u + \log(u) - 4$ tile types, and 1 stage whose uniquely produced output is a width-3 gap-0 $d\log(u)$-bit string pad representing S.

Proof. Consider a string of d digits, $S = s_0 s_1 \ldots s_{d-1}$ that represent the same number as S but in base u. We use the construction of Lemma 4.4 to build a width-2 decompression pad for each of the d digits in base u. We build decompression tiles for a base u using $2u - 2$ tile types (see Figure 4.8).

Decompression tile types representing bit strings of length $\log(u)$ bind via a single glue to the decompression pad and expose a bit glue. Other decompression tiles then attach to these initial types interact with those tiles and the north surface to "unpack" the remaining bits. Each base u digit unpacks into $\log(u)$ bits. It follows that a string of d digits in a base u can be unpacked into $d\log(u)$ bits. The total tile complexity is $t = 5d + 2u + \log(u) - 4$: $5d + \log(u) - 2$ tupes to build the decompression pad and $2u - 2$ decompression tile types. □

Lemma 4.6. There exists a constant c such that for any $t \in \mathbb{N}$ with $t > c$, there exists an $x = \theta(t \log t)$ such that for any bit string S of length x, there exists a $\tau = 2$ staged assembly system with 1 bin, t tile types, and 1 stage whose uniquely produced output is a width-3 gap-0 x-bit string pad representing S.

Proof. Here we choose the parameters that maximize the number of bits encoded using Lemma 4.5. Given t tile types, let the base be $u = 2^{\lfloor \log \frac{t}{3} \rfloor}$. For a string of digits S in base u, Lemma 4.5 constructs length-$\log(u)$ decompression pads for every digit in the string. Decompression pads are constructed using 5 tile types plus some additional number of tile types called *shared spacing* that allow decompression pads to grow to any length. If $\lfloor \frac{t}{2} \rfloor$ tiles are used for building decompression pads without shared spacing, then we can build $\lfloor \frac{t}{10} \rfloor$ decompression pads, since each one requires 5 tile types. This allows us to represent $\lfloor \frac{t}{10} \rfloor$ digits in base u. We allocate another $\lfloor \frac{t}{2} \rfloor$ tile types to build the tiles used to decompress the decompression pads and for the shared spacing that makes them the appropriate length. Lemma 4.5 requires $2u + \log(u)$ types to do this.

The number of bits achieved is $y = d\log(u) = \left\lfloor \frac{t}{10} \right\rfloor \log 2^{\lfloor \log \frac{t}{3} \rfloor} = \left\lfloor \frac{t}{10} \right\rfloor \left\lfloor \log \frac{t}{3} \right\rfloor = \Theta(t \log t)$. Using Lemma 4.5 and base u, we can build any width-3 $\lfloor \frac{t}{10} \rfloor \lfloor \log \frac{t}{3} \rfloor$-bit string pad using only 1 stage and 1 bin. Due to constraints building the decompression pad, we need to build at least 3 bits with a minimum of base 8, for encoding into a higher base. For building the decompression pad, minus the shared spacing, we need at least $\lfloor \frac{t}{2} \rfloor \geq 5d = 15$ tile types. For the decompression tiles and shared spacing we need $\lfloor \frac{t}{2} \rfloor \geq 2u + \log(u) - 4 = 15$. For selecting base 8, we also need $u = 2^{\lfloor \log \frac{t}{3} \rfloor} = 8$. Thus, to implement this method, we need $\frac{t}{3} \geq 8$. At minimum, we need $\frac{t}{2} \geq 15$ and $\frac{t}{3} \geq 8$, so this method works for all cases where $t \geq c = 30$. □

Winged Bit String Pads

Bit string pads can be mixed with $O(1)$ tile types to assemble *wings*. Wings are rectangular assemblies with geometric bumps, or *teeth*, that encode a positive integer *index* in binary. A wing gadget has index i and m bits provided it geometrically encodes an m-bit binary string representing i. Wings come in two varieties, *west* and *east*, shown in Figure IV.

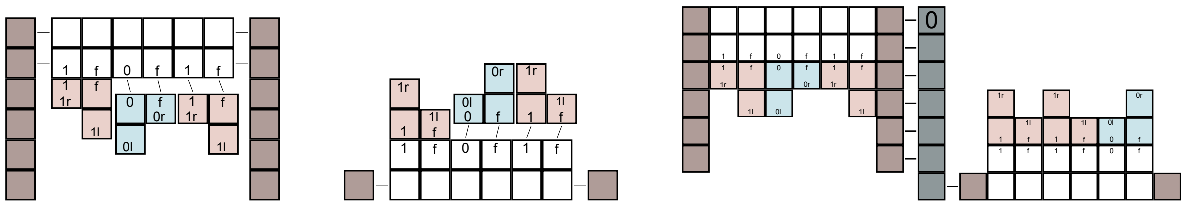
Wings encode their indices on either the north surface (west wings) or south surface (east wings). Roughly speaking, a pair of west and east wings can attach if their teeth match (perfectly interlock), equivalent to having the same index. Assemblies can brought together to self-assemble in order by attaching the approriate wings on the west and east edges of the assembly. All *winged* assemblies can then be mixed to self-assemble, linearly, in the order chosen by the designer.

The Big Picture

The constructions detailed above showcase two methods of creating bit string pads and are my primary contributions towards the published version of this body of work [85] describing an optimal algorithm for building bit string pads.

In [85], a lower bound on the number of stages required to build a shape containing a specified quantity of information.

Lemma 4.7. A staged system of fixed temperature τ with b bins, s stages, and t tile types can be specified using $O(t \log t + sb^2 + tb)$ bits. Such a system with flexible glues can be specified using $O(t^2 + sb^2 + tb)$ bits.


(a) (b) (c)

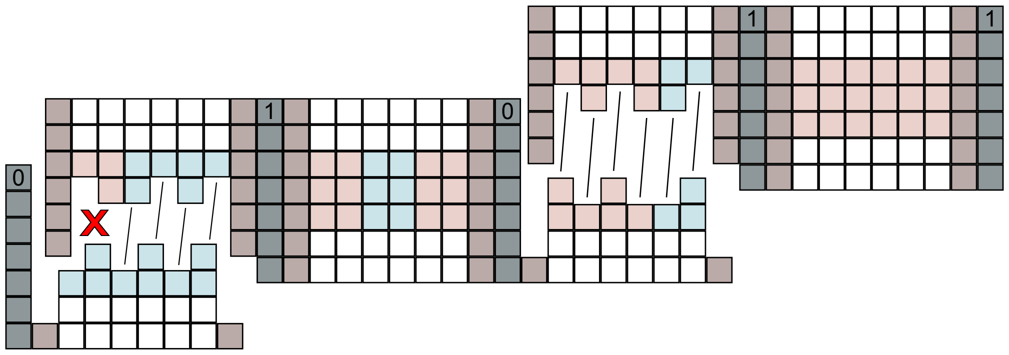

(d)

Figure 4.9: Assembling wings from bit string pads and extra $O(1)$-size assemblies. (a) Assembly of a west wing. (b) Assembly of an east wing. (c) The attachment of chosen west and east wings to a $O(1)$-sized assembly (dark shaded) (d) Directing the assembly of $O(1)$-sized assemblies (dark shaded) with attached wings. Left: mismatched wings prevented from attaching via matched glues due to mismatched geometry. Right: matching wings have no such geometric mismatch and so attach.

It immediately follows from Lemma 4.7 that for almost all bit strings, any staged system with b bins and t tiles that encodes the bit string must have $\Omega(\frac{x-tb-t\log t}{b^2})$ stages with standard glues and $\Omega(\frac{x-tb-t^2}{b^2})$ stages with flexible glues, where x is the length of the bit string.

The methods described above are used as subconstructions in an upper bound that nearly matches the lower bound.

Lemma 4.8. There exists a constant c such that for any $b, t \in \mathbb{N}$ with $b, t > c$ and any bit string S of length x, there exists a $\tau = 2$ staged assembly system with b bins, t tiles, and $O(\frac{x-tb-t\log t}{b^2} + \frac{\log\log b}{\log t})$ stages whose uniquely produced output is a width-7 gap-$\Theta(\log b)$ x-bit string pad representing S.

The additive gap between the upper and lower bounds implied by these lemmas is due to the

$O(\frac{\log \log b}{\log t})$ additional stages used to assemble the bit string pads from Section 4.2, needed to carry out the primary steps of Lemma 4.8.

Two classes of shapes have become the defacto standards [88] for measuring assembly efficiency: squares and and general shapes (with scaling permitted). Efficient assembly of these two classes was first considered in the aTAM, where matching upper and lower bounds on the tile complexity were obtained [14], [15], [37]. Squares and general scaled shapes can be assembled by combining a universal set of "computation" tiles with efficiently assembled "input" bit string pads.

For $n \times n$ squares, we prove the stage complexity is $O(\frac{\log n - tb - t \log t}{b^2} + \frac{\log \log b}{\log t})$ and, for almost all n, $\Omega(\frac{\log n - tb - t \log t}{b^2})$.[1] For shapes S with Kolmogorov complexity $K(S)$, we prove the stage complexity is $O(\frac{K(S) - tb - t \log t}{b^2} + \frac{\log \log b}{\log t})$ and $\Omega(\frac{K(S) - tb - t \log t}{b^2})$.

We obtain similar results when *flexible glues* [28], glues that can form bonds with non-matching glues, are permitted. In this case, the stage complexity for $n \times n$ squares is reduced to $O(\frac{\log n - t^2 - tb}{b^2} + \frac{\log \log b}{\log t})$ and, for almost all n, $\Omega(\frac{\log n - t^2 - tb}{b^2})$, and the stage complexity for general shapes is reduced to $O(\frac{K(S) - t^2 - tb}{b^2} + \frac{\log \log b}{\log t})$ and $\Omega(\frac{K(S) - t^2 - tb}{b^2})$.

Because our results are optimal across all choices of tile type and bin complexity, these results generalize and, in some cases, improve on prior results.

[1]The fraction of values for which the statement holds reaches 1 in the limit as $n \to \infty$.

CHAPTER V

SHAPE REPLICATION

Introduction

How can we harness the power of self-assembly to build a system that first senses the shape of a given unknown object, then builds copies of that shape, like the nanoscale equivalent of a photocopier? Although replicators are traditionally the subject of science fiction [90], biological reproduction illustrates that this is possible (at least approximately) in the real world, and recent biological engineering shows that information can be replicated using DNA crystal growth and scission [66]. In this paper, we investigate more complex replication of *geometric shape*, using biologically realistic models (taken to an extreme admittedly not yet practical). To do so, we need a sufficiently flexible algorithmic model of self-assembly, and a replication "algorithm" defined by particles that interact with each other and the given object.

The standard algorithmic abstraction of self-assembly is to model the self-assembling particles as *Wang tiles*, that is, unit squares with a specified "glue" on each side, which can translate but not rotate; each glue has a nonnegative integer *strength*. In the *2-Handed Assembly Model* (2HAM), two assemblies (eventually) join together if they can be translated so as to match up glues of total strength at least τ, the *temperature* of the system. The resulting *terminal* assemblies are those that do not join into any other assemblies.

We can define the *replication problem* in this model: given an unknown initial assembly of tiles, design a collection of tiles or small assemblies to add to the self-assembly system such that the resulting terminal assemblies consist of copies of the given shape (plus possibly some small "trash" assemblies). However, this goal is impossible to achieve in just the 2HAM model, or any model with just a mechanism to join assemblies together: to sense a shape without destroying it, we need to be also able to split assemblies back apart.

The first extension of the 2HAM shown to enable a solution to the replication problem is the *RNAse enzyme* staged assembly model [47]. In this model, tiles can be of two types (DNA or RNA), the given shape is all one type (DNA), and there is an operation that destroys all tiles of the other type (RNA). Abel et al. [47] show that this model enables making a desired number k of copies of a given unknown hole-free shape or, through a complicated construction, infinitely many copies of the shape using just a constant number of tiles and stages. This result requires that the shape has a *feature size* (minimum distance between two nonincident edges) of $\Omega(\lg n)$ where n is the (unknown) number of tiles in the shape. The model is unsatisfying, however, in the way that it requires multiple stages and a global operation that modifies all tiles.

Another extension of the 2HAM shown to enable a partial solution to the replication problem is the *Signal Tile Assembly Model* (STAM) [83]. This powerful model allows tiles to change their glues and trigger assembly/disassembly events when tiles attach to other tiles. Keenan et al. [83] show that this model enables replicating a given unknown *pattern* of tiles in a rectangular shape, but not an arbitrary shape. Hendricks et al. [82] show that this model enables infinitely replicating a given unknown hole-free shape of feature size at least 2. The model is unsatisfying, however, in the way that it allows tiles to have arbitrarily complex behaviors. (Recent results show how to simulate part of STAM using 2HAM (in 3D) [80], but this simulation necessarily cannot simulate the necessary aspect of breaking assemblies apart.)

In this paper, we study a simple extension of 2HAM to allow glues of negative strength, that is, repulsive forces in addition to standard attractive forces. This extension is practical, as biology implements both types of forces [13], and well-studied theoretically: negative glues have already been shown to enable fuel-efficient computation [73], space-efficient computation [70], and computation even at temperature $\tau = 1$ [58], [87]. The complexity of combinatorial optimization problems with negative glues has also been studied [59]. We show that shape replication is possible in this model: adding a fixed constant number of constant-size assemblies to a given unknown hole-free shape results in terminal assemblies of infinitely many copies of that shape, plus constant-size trash assemblies.

Shape Replication Systems

A system $\Gamma = (\sigma, \tau)$ is a *universal shape replicator* for a class of shapes if for any shape X in the class, there exists an assembly Υ of shape X such that system $\Gamma' = (\sigma \bigcup \Upsilon, \tau)$ (i.e., Γ with a single copy of Υ added to the initial state) produces an unbounded number of assemblies of shape X, and essentially nothing else. We formalize this in the following definition, and then add some discussion of additional desirable properties a replicator might have.

Definition 5.1. [Universal Shape Replicator] A system $\Gamma = (\sigma, \tau)$ is a *universal shape replicator* for a class of shapes U if for any shape $X \in U$, there exists an assembly Υ of shape X such that system $\Gamma' = (\sigma \bigcup \Upsilon, \tau)$ has the following properties:

- For any positive integer n and producible state S, there exists a producible state S' containing at least n terminal assemblies of shape X such that $S \rightarrow^{\tau} S'$.
- All terminal assemblies of super-constant size have shape X.

In addition to the above replicator properties, there are some additional desirable properties a universal replicator might have. For example, the producible assemblies of the system should be limited in size as much as possible ($O(|X|)$ in the best possible case). Additionally, it may be desirable to place substantial limitations on the complexity of the initial input assembly Υ, e.g., require its surface to expose essentially a single type of glue for each edge orientation. Our universal replicator achieves the following *bonus* constraints.

Definition 5.2 (Bonus properties!)**.** A universal replicator is said to be *sleek* if it has the following properties:

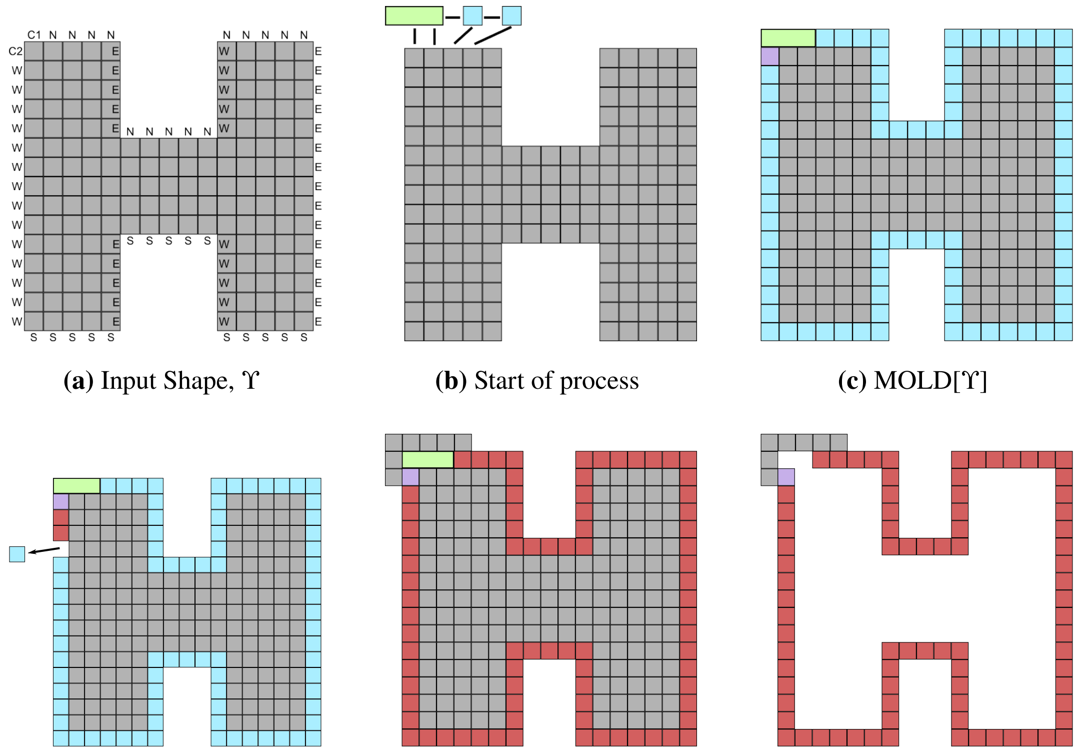
- (bonus!) All producible assemblies have size $O(|X|)$.
- (bonus!) The input assembly Υ is rather simple: infinite internal bonds, and a single glue type for each edge orientation (e.g. *North*, *East*, *South*, and *West*) for exposed surface glues, with at most $O(1)$ *special* tiles violating this convention.

The class of shapes which can be replicated by the system described in this paper are hole-free polyominoes with *feature size* of at least 9. The feature size of a shape is defined as follows (we use the same definition as [47]): for two points a, b in the shape, let $d(a, b) = \max(|a_x - b_x|, |a_y - b_y|)$. Then the feature size is the minimum $d(a, b)$ such that a, b are on two non-adjacent edges of the shape. The feature size ensures that the replication gadgets ($O(1)$ sized assemblies used to replicate the shape) function properly (e.g. the gadgets can encompass the shape without "bumping into" each other). Formally, we prove the following:

Theorem 5.3. There exists a sleek universal shape replicator $\Gamma = (\sigma, 10)$ for genus-0 (hole-free) shapes with feature size of at least 9.

Overview of Replication Process

The high-level process for replicating a given input assembly Υ is described here. We assume that Υ has special glues along its perimeter such that all North, East, South, and West facing edges have glues of N, E, S, and W, respectively, with the exception of the Northernmost-Westernmost unit of the shape having a North glue of $C1$ and West glue of $C2$ (Fig. 5.1a). Intuitively, the process is to assemble an outline of Υ that is filled to have the same shape as Υ. In Phase 1, *mold gadgets* attach clockwise along the outside perimeter of Υ, to detect the edges of the shape. In Phase 2, the mold gadgets are replaced, counterclockwise gadget-by-gadget, by *drill gadgets*. These drill gadgets follow the path laid out by the mold gadgets and do not attach to the shape. The drill gadgets use negative glues to destabilize adjacent mold gadgets. This technique is used to create an assembly called the FRAME which outlines Υ but whose bounding box is too large to simply fill to get a copy of the shape of Υ. In Phase 3, *inner mold gadgets* detect the outline of the inside perimeter of the FRAME. In Phase 4, the inner mold gadgets are replaced, counterclockwise gadget-by-gadget, by *inner drill gadgets* in order to begin destabilizing the FRAME from the assembly. With the help of an additional *inner post-drill gadget*, the assembly destabilizes into the FRAME and HOLLOW[Υ]. Using *fill gadgets*, HOLLOW[Υ] is filled to create COPY[Υ], which is a copy of the original shape.



(a) Input Shape, Υ **(b)** Start of process **(c)** MOLD[Υ]

(d) Drilling **(e)** FRAME[Υ] **(f)** FRAME

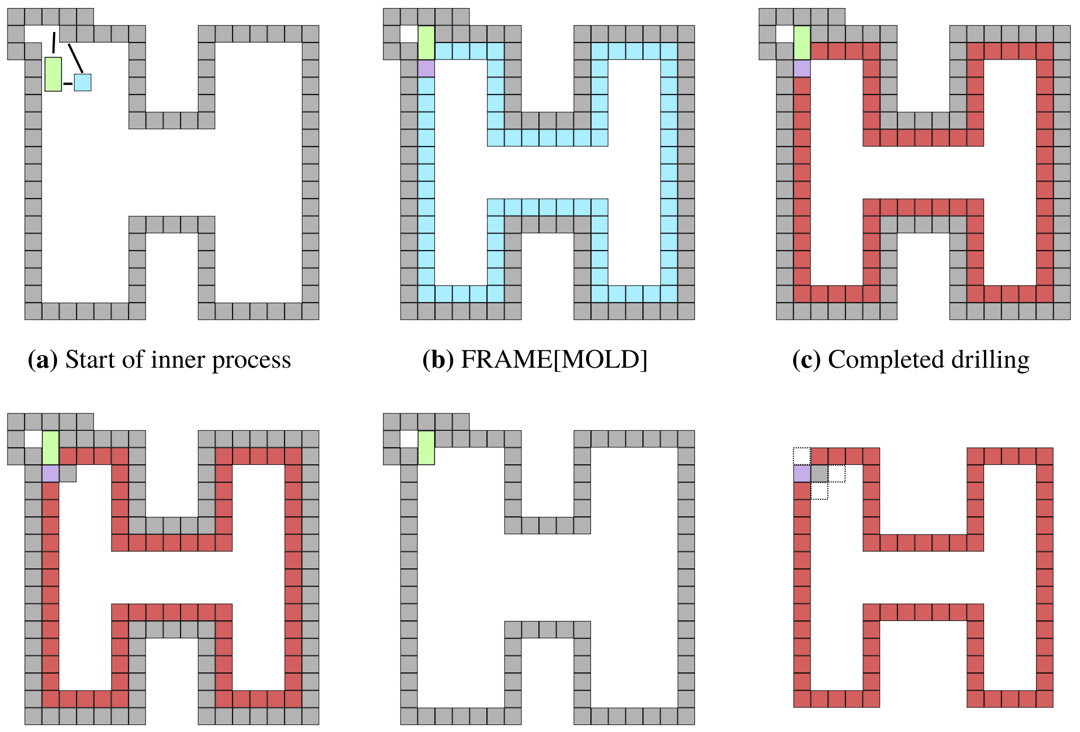
Figure 5.1: High level process for making the FRAME from input shape Υ.

Phase 1: Shape detection via mold gadgets (Section V)

1. Place an *outer start gadget* at the Northernmost-Westernmost corner utilizing glues $C1$ and N (Fig. 5.1b).
2. From the East edge of the outer start gadget, moving clockwise, trace the outside perimeter of the shape with a layer of *mold gadgets*. Call the result MOLD[Υ] (Figs. 5.1b, 5.1c).

Phase 2: Drilling to create a FRAME (Section V)

1. Place a *pre-drill* gadget at the Northernmost-Westernmost corner of MOLD[Υ].
2. Starting from the pre-drill gadget, and moving counter-clockwise, replace the mold gadgets in the surrounding layer with *drill gadgets* that do not have affinity to the shape (Fig. 5.1d).
3. When done drilling, place a *post-drill gadget*, call the result FRAME[Υ] (Fig. 5.1e).
4. FRAME[Υ] is unstable and separates into two assemblies, FRAME (Fig. 5.1f) and START[Υ]. START[Υ] is an assembly that contains only the input shape with the start gadget attached.



(a) Start of inner process **(b)** FRAME[MOLD] **(c)** Completed drilling

(d) FRAME[HOLLOW[Υ]] **(e)** FRAME[START] **(f)** HOLLOW[Υ]

Figure 5.2: High level process for making a copy of an input shape Υ from a *frame* of Υ.

Phase 3: Detecting the inside of the FRAME (Section V)

1. Place an *inner start gadget* at the Northernmost-Westernmost corner on the inside perimeter of FRAME, call the result FRAME[START] (Fig. 5.2a).
2. Starting from the inner start gadget, and moving clockwise, trace the inside perimeter of FRAME[START] with a layer of mold gadgets. Call the result FRAME[MOLD] (Fig. 5.2b).

Phase 4: Drilling and post-drilling to create HOLLOW[Υ] (Section V)

1. Starting from the Northernmost-Westernmost inside corner, replace the inner mold gadgets on the inside perimeter with inner drill gadgets that do not have affinity to FRAME (Fig. 5.2c).
2. Place an *inner post-drill gadget*. Call the result FRAME[HOLLOW[Υ]] (Fig. 5.2d).
3. FRAME[HOLLOW[Υ]] is unstable and separates into FRAME[START] and HOLLOW[Υ] (Fig. 5.2e and Fig. 5.2f).

Label	Strength	Label	Strength
C1	2	C2	9
N,E,S,W,n,e,s,w	9	T*,t*,O*,o*,L	1
B*,b*	5	K,k	3
X*,Y*,Z*,x*,y*,z*	9	D	-7
F*,f*	-2	V*,U*,H*,J*,S*,v*,u*,h*,j*,s*	9
R,r	2	A*	10
g*	9	M*	5
Q	-2	q	-5

Table 5.1: The glue strengths of each glue label in the shape replication system.

Phase 5: Filling and Repeating (Section V)

1. HOLLOW[ϒ] is filled to become COPY[ϒ], which is an assembly with the same shape as ϒ.
2. START[ϒ], repeats the process over again, starting at the second step of Phase 1.
3. FRAME[START] also repeats creating copies of the shape, starting at the second step of Phase 3.

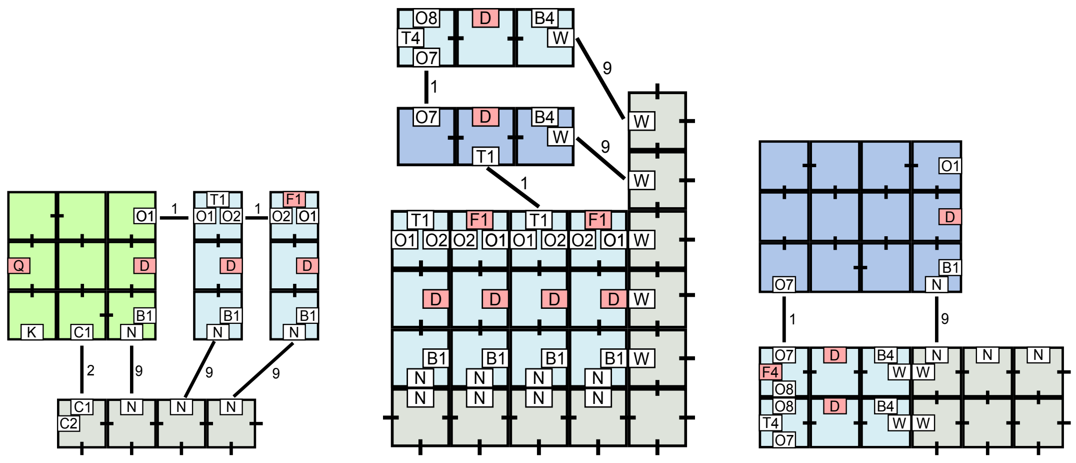
Replication Gadgets

In this section, we describe the assemblies, or gadgets, which are constructed from the tiles in the initial state of the replication system. These gadgets are designed to work in a temperature $\tau = 10$ system. There is a constant number of these distinct gadget types which are used to replicate arbitrarily sized input assemblies. In our figures, a black line perpendicular and in the middle of the edge of two adjacent tiles indicates a unique infinite strength bond (i.e. the strength of the glue is $\gg \tau$ such that no detachment events can occur in which these tiles are separated). Although each glue strength can be found in the figures and their captions, there is a full table of glue strengths in Table 5.1. First, we describe the gadgets used in the process of creating the layer of mold gadgets around the outer perimeter of the input assembly.

Phase 1 Gadgets for the Outer Mold

Below we describe the gadgets used to implement Phase 1 and create MOLD[ϒ].

Start gadget. The start gadget is an assembly, designed to attach using the $C1$ glue and N glue, which designates the Northernmost-Westernmost corner of the input assembly. Once the start



(a) (b) (c)

Figure 5.3: Phase 1: Starting the outer mold process and handling type-1 corners. (a) The start gadget attaches. $C1 + N = 2 + 9 \geq \tau = 10.$, The first mold gadget attaches, cooperatively, to the start gadget and the input shape. $O1 + N = 1 + 9 \geq \tau$. The second mold gadget attaches, cooperatively, to the previous mold gadget and the input shape. $O2 + N = 1 + 9 \geq \tau$. (b) A concave corner of the assembly. The North→West corner gadget attaches. $T1 + W = 1 + 9 \geq \tau$. A mold gadget attaches. $O7 + W = 1 + 9 \geq \tau$. (c) A convex corner of the assembly. The West→North corner gadget attaches. $O7 + N = 1 + 9 \geq \tau$.

gadget attaches, a North mold gadget may attach (Fig. 5.3a).

Mold gadgets. The mold gadgets, beginning at the start gadget as shown in Figure 5.3a, walk along the input assembly in a clockwise manner to create MOLD[Υ]. The North (South) mold gadgets are 1×3 assemblies designed to walk from West to East (East to West) along the North (South) edges of the input assembly. The West (East) mold gadgets are 3×1 assemblies designed to walk from South to North (North to South) along the West (East) edges of the input assembly. The mold gadgets expose either a positive or negative glue on their unused edge (e.g. North mold gadgets expose either a $T1$ or $F1$) used for detecting corners and drilling at corners, respectively.

Corner mold gadgets. The *corner mold gadgets* attach at corners of the input assembly once a mold gadget has reached the corner. Two sets of corner mold gadgets are used; one for concave corners and one for convex corners. There are two types of concave and convex corner mold gadgets. This is due to the edge length being even or odd. If the mold gadget that places adjacent to a corner of the shape has two negative glues, we denote the concave corner as a *type-1* concave corner and otherwise as a *type-2* concave corner. A type-1 concave corner gadget attachment can be seen in Figure 5.3b, and convex in Figure 5.3c. Type-2 corner gadget details can be seen in Figure 5.4. The North→West (South→East) corner gadget attaches cooperatively to a North (South) mold gadget and the input shape. The East→North (West→South) corner gadget attaches cooperatively to a East (West) mold gadget and the input shape.

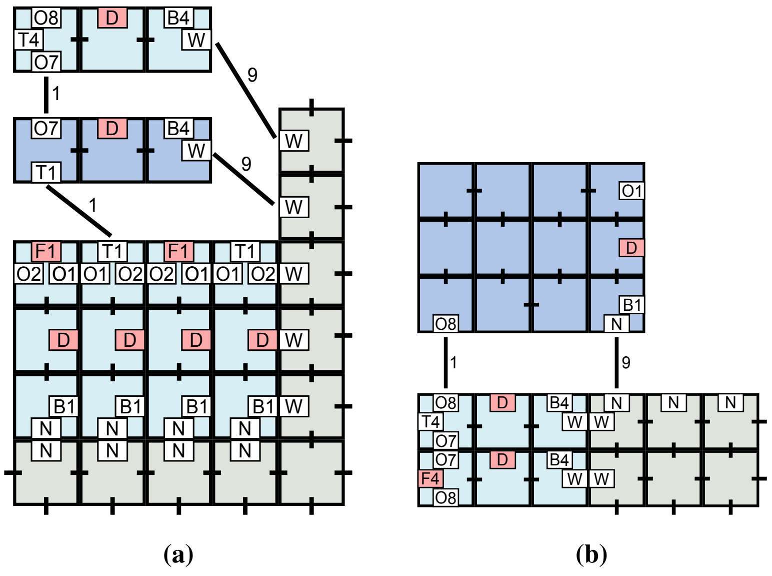


Figure 5.4: Phase 1: Type-2 concave and convex corners of the assembly. (a) A type-2 concave corner of the assembly. The type-2 North→West corner gadget attaches. $T1 + W = 1 + 9 \geq \tau$. An mold gadget attaches. $O7 + W = 1 + 9 \geq \tau$. (b) An convex corner of the assembly. The West→North corner gadget attaches. $O8 + N = 1 + 9 \geq \tau$.

Phase 2 Gadgets for Making the FRAME

Below we describe the gadgets used to implement Phase 2 and create FRAME and START[Υ].

Pre-drill gadget. A gadget that binds to the start gadget, a mold gadget (in some cases) , and the input shape that is designed to allow the placement of a *West drill helper* to its South. It allows the drilling process to begin once mold gadgets have finished encircling the input shape (Fig. 5.5).

Drill gadgets. The drill gadgets (Fig. 5.6a), along with drill helpers, walk along the boundary of the input assembly in a counter-clockwise manner destabilizing mold gadgets and

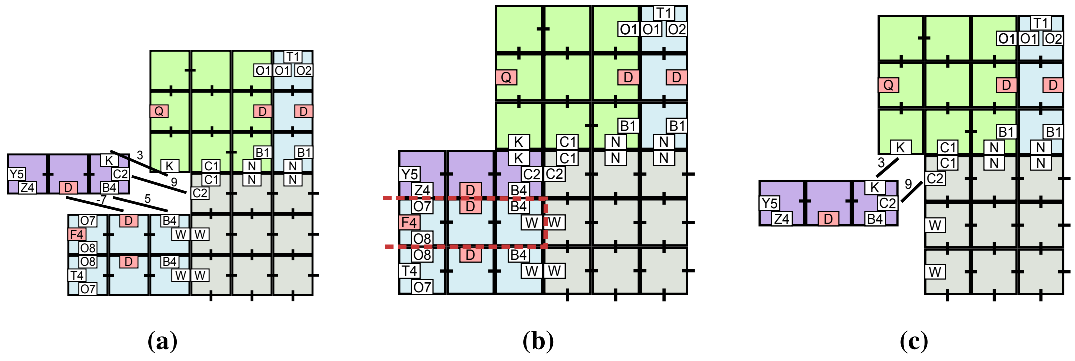


Figure 5.5: Phase 2: The pre-drill process. (a) Case 1: The mold process has completed. The pre-drill gadget attaches cooperatively to the start gadget, input shape, and a mold gadget. $K + C2 + B4 + D = 3 + 9 + 5 - 7 \geq \tau$. (b) The pre-drill gadget destabilizes the adjacent mold gadget. $B4 + W + O8 + D = 5 + 9 + 1 - 7 < \tau$. (c) Case 2: The mold process has not completed, the pre-drill gadget may still attach. After attachment, the negative 'D' glue prevents a mold gadget from occupying the space directly below. $K + C2 = 3 + 9 \geq \tau$.

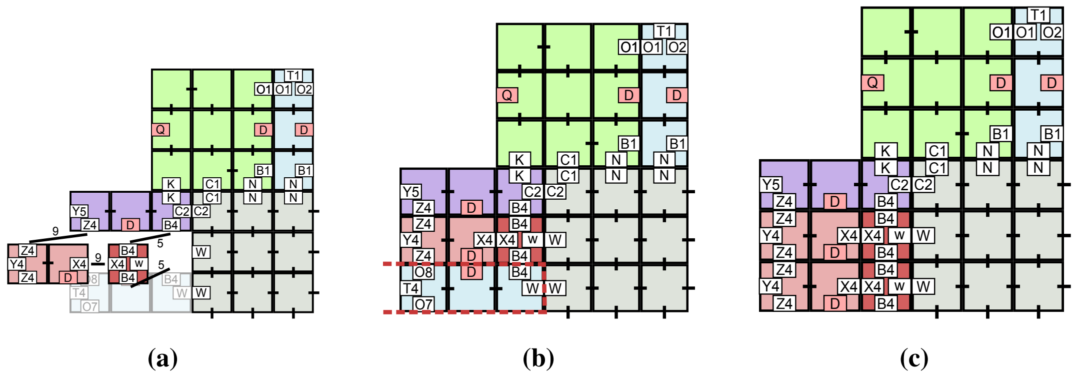


Figure 5.6: Phase 2: The drilling process begins. (a) A West drill helper attaches cooperatively to the pre-drill gadget and a mold gadget. $B4 + B4 = 5 + 5 \geq \tau$. A drill gadget attaches cooperatively to the pre-drill gadget and drill helper. $Z4 + X4 + D = 9 + 9 - 7 \geq \tau$. (b) The drill gadget destabilizes the mold gadget underneath. $B4 + W + O7 + D = 5 + 9 + 1 - 7 < \tau$. (c) The removal of the mold gadget allows the process shown in (a) and (b) to repeat. The process continues along the face of the shape.

replacing them to create the FRAME. The drill gadgets do not attach to the input shape Υ, so when the drills have removed all of the mold gadgets, the FRAME will not be attached to the input shape.

The drill gadgets are assemblies that attach to the North (South) drill helpers and destabilize mold gadgets to their East (West). The West (East) drill gadgets are assemblies that attach to the West (East) drill helpers and destabilize mold gadgets to their North (South).

Drill helpers. The drill helpers (Fig. 5.6a) are 1×1 assemblies that, beginning at the pre-drill gadget, walk along the boundary of the input assembly in a counter-clockwise manner exposing glues that allow drill gadgets to destabilize adjacent mold gadgets. These gadgets walk along the boundary by following glues exposed by mold gadgets, rather than using glues on the input assembly.

Drill corner gadgets. Drill corner gadgets bind cooperatively at type-1 and type-2 corners to drill helpers and mold gadgets in order for the drilling process to turn corners. Details can be seen in Figures 5.7,5.8,and 5.9.

Post-drill gadget. *Post-drilling* refers to the the process of removing the input shape to create a FRAME. It is needed because the pre-drill gadget is still attached to the input shape. The post-drill gadget may attach once the drill gadgets have encircled the input shape. It attaches to the pre-drill gadget and last-placed drill gadget. Once attached, it destabilizes the assembly with the input shape and start gadget to create the FRAME. (See Fig. 5.10.)

Phase 3 Gadgets for the Inner Mold

Here we describe the gadgets used to implement Phase 3 and create FRAME[MOLD]. The *inner mold gadgets* are similar to the mold gadgets used in Section V, with changes to allow them to encircle the inside perimeter of the FRAME, rather than the outside perimeter of the input shape Υ. The Phase 3 gadgets (shown in Figure 5.11) include an *inner start gadget* which attaches to the FRAME to allow the attachment of inner mold gadgets which attach along the inside perimeter of the FRAME in a clockwise manner. The inner mold gadgets reflect the form and function of the mold gadgets shown in Section V, handling corners in a similar way. Type-2 inner mold corner gadgets can be seen in Figure 5.12.

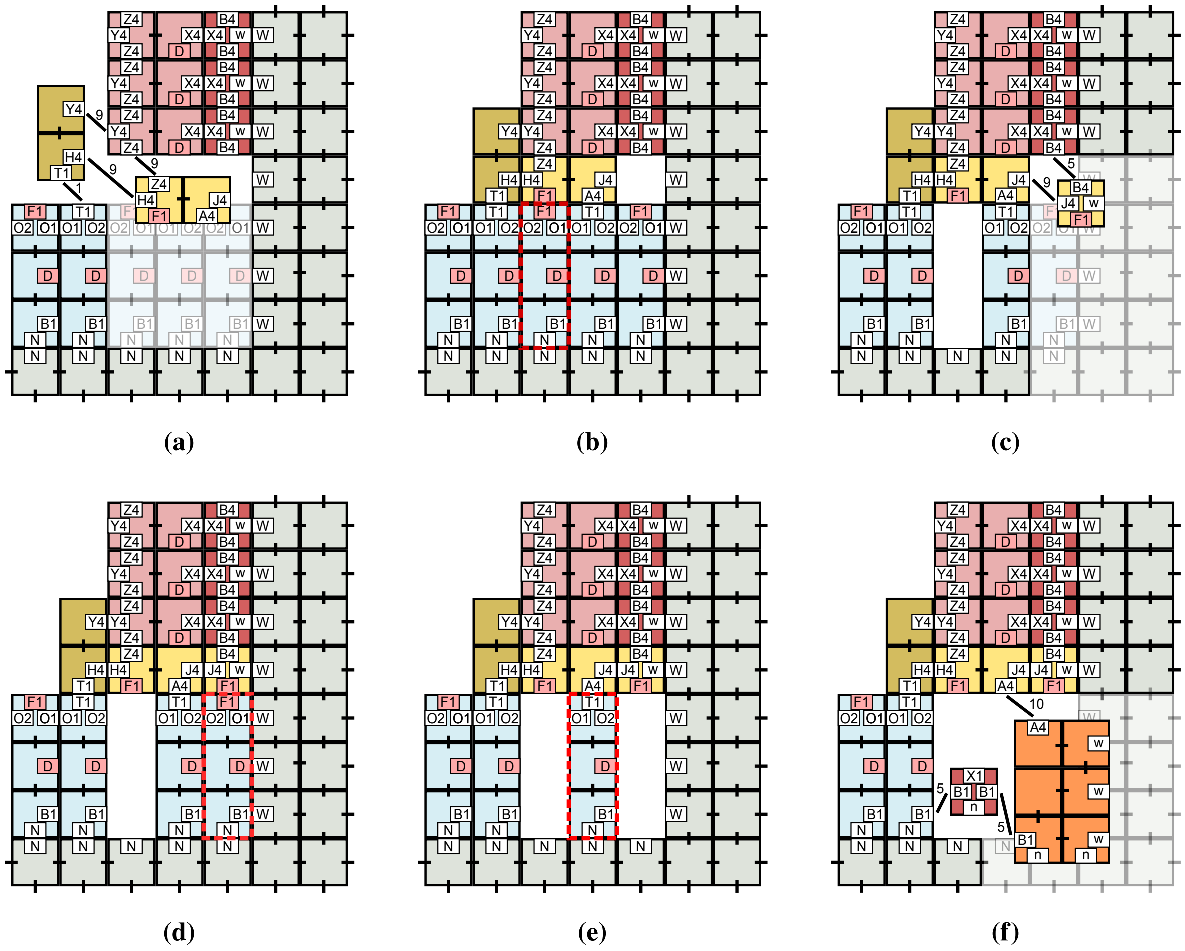


Figure 5.7: Phase 2: The drilling process encounters a type-1 concave corner of the assembly. (a) A drill gadget destabilizes the North→West corner gadget. $B4+W+T1+D = 5+9+1-7 < \tau$. A West→North drill corner helper attaches to a mold gadget and a drill gadget. $Y4+T1 = 9+1 \geq \tau$. A West→North drill corner gadget attaches, cooperatively. $H4+Z4+F1 = 9+9-2 \geq \tau$. (b) The West→North drill corner gadget destabilizes a mold gadget to its South. $O1+N+O2+F1 = 1+9+1-2 < \tau$. (c) A mold gadget has destabilized and has broken off the assembly. Another West→drill gadget attaches. (d) destabilizes a mold gadget to its South. $N+O2+F1 = 9+1-2 < \tau$ (e) Without any adjacent mold gadgets to its East or West, a mold gadget becomes unstable. $N = 9 < \tau$. (f) A drill corner gadget fills in the space in the concave corner and exposes a glue that will allow drilling to continue to the West. $A4 = 10 \geq \tau$.

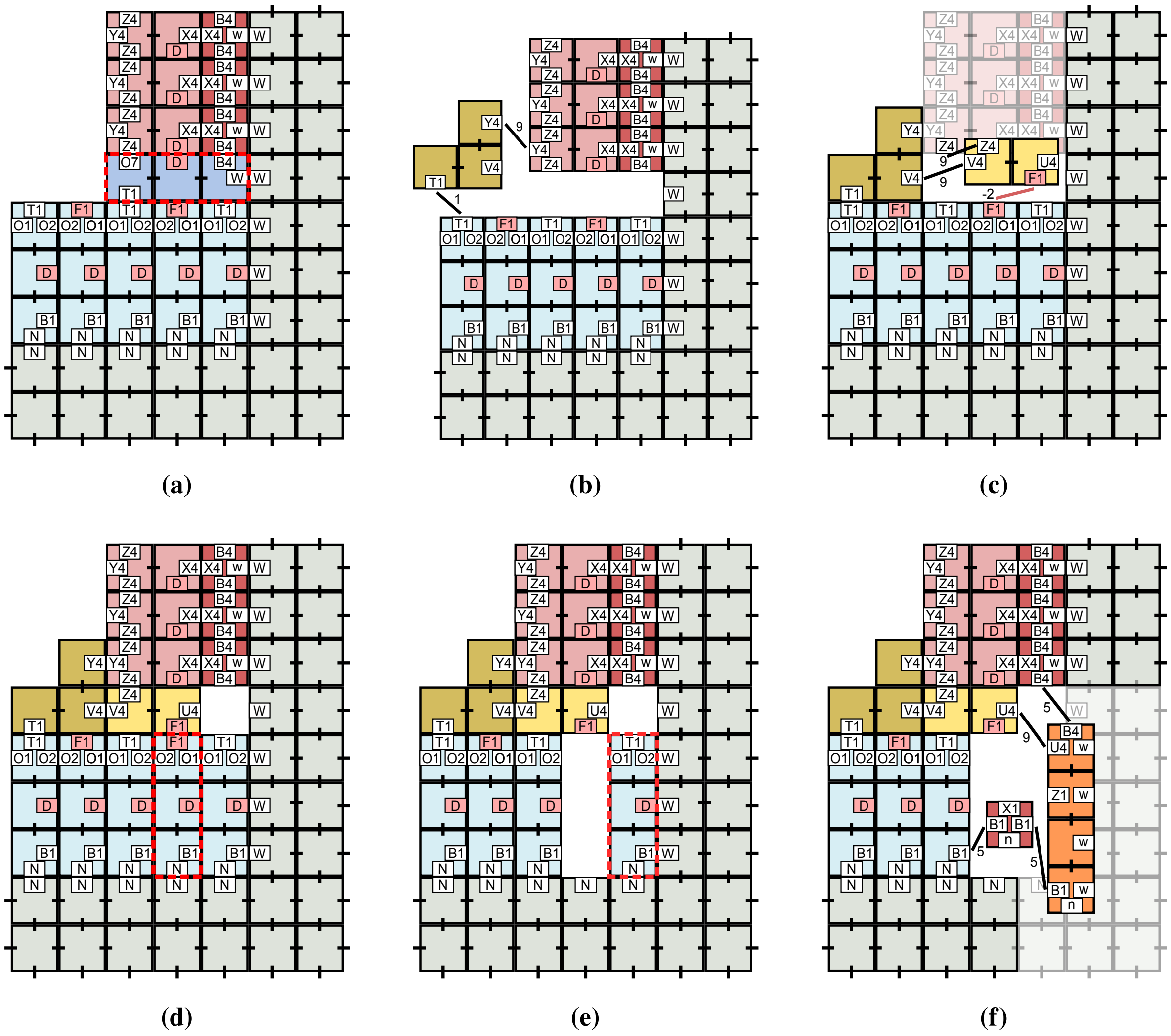


Figure 5.8: Phase 2: The drilling process encounters a type-2 concave corner of the assembly. (a) A drill gadget destabilizes the North→West corner gadget. $B4 + W + T1 + D = 5 + 9 + 1 - 7 < \tau$ (b) A West→North drill corner helper attaches to a mold gadget and a drill gadget. $Y4 + T1 = 9 + 1 \geq \tau$. (c) A West→North drill corner gadget attaches, cooperatively. $V4 + Z4 + F1 = 9 + 9 - 2 \geq \tau$. (d) The West→North drill corner gadget destabilizes a mold gadget to its South. $O1 + N + O2 + F1 = 1 + 9 + 1 - 2 < \tau$. (e) Without a mold gadget to its West, the mold gadget in the concave corner is unstable. $N = 9 < \tau$. (f) A drill corner gadget fills in the space in the concave corner. $U4 + B4 = 9 + 5 \geq \tau$. This gadget exposes a glue on its West side that allows drilling to continue to the West. $B1 + B1 = 5 + 5 \geq \tau$.

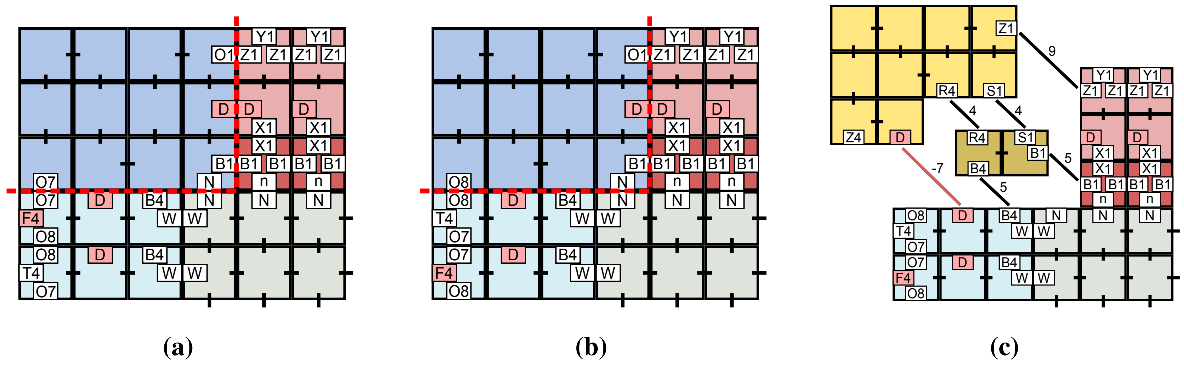


Figure 5.9: Phase 2: The drilling process encounters convex corners of the assembly. (a) Coming from the East, drill gadgets encounter a type-1 convex corner of the assembly. The mold corner gadget becomes unstable and will break from the assembly. $O7 + N + B1 + D = 1 + 9 + 5 - 7 \geq \tau$. (b) The drill gadgets encounter a type-2 convex corner gadget. It becomes unstable and will break from the assembly. $O8 + N + B1 + D = 1 + 9 + 5 - 7 < \tau$. (c) Convex corner gadgets bind and allow the drilling process to continue. $B4 + B1 = 5 + 5 \geq \tau$. $Z1 + R4 + S1 + D = 9 + 4 + 4 - 7 \geq \tau$.

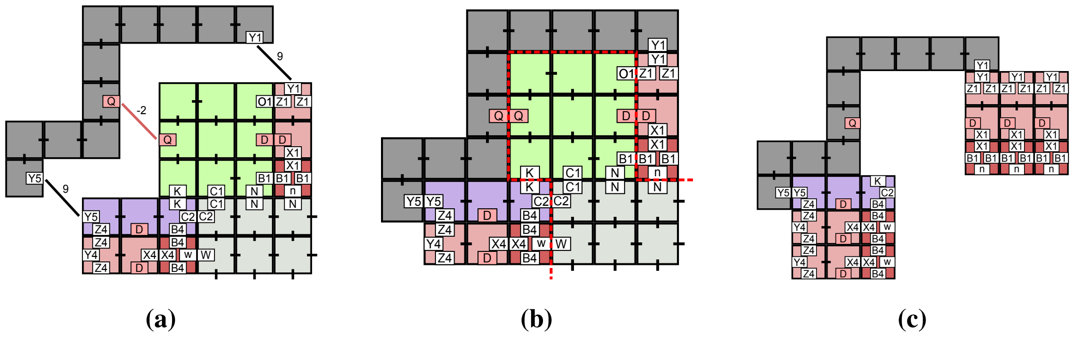


Figure 5.10: Phase 2: The post-drill process begins. (a) The post-drill gadget binds at the Northernmost-Westernmost corner of the assembly that contains the input shape. It binds cooperatively to the last placed drill block and the pre-drill gadget. $Y5 + Y1 + Q = 9 + 9 - 2 \geq \tau$. (b) Once placed, the pre-drill gadget destabilizes a cut which includes the input shape and the start block. $B1 + C2 + K + Q + D = 5 + 9 + 3 - 2 - 7 < \tau$. (c) Once the assembly with the input shape and the start block detached, the FRAME remains.

Phase 4 Gadgets for Making the Hollow Outline

Below we describe the gadgets used to implement Phase 4 and create FRAME[START] and HOLLOW[ϒ]. The *inner pre-drill gadget* can attach to the FRAME after the inner start gadget has

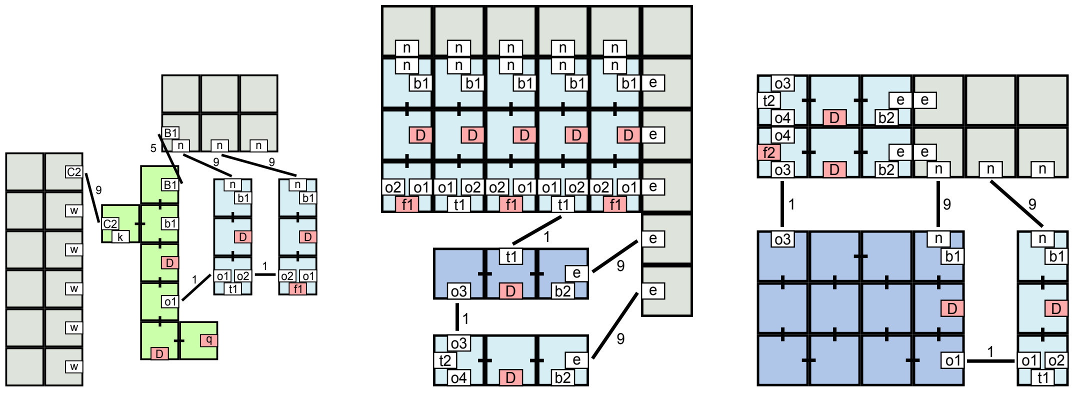

(a) (b) (c)

Figure 5.11: Phase 3: Starting the inner mold process and handling type-1 corners. (a) The inner start gadget attaches to the inside perimeter of the FRAME. $C2 + B1 = 9 + 5 \geq \tau$. An inner mold gadget attaches cooperatively to the start gadget and the FRAME. $o1 + n = 1 + 9 \geq \tau$. Another inner mold gadget attaches cooperatively to the FRAME and the previous inner mold gadget. $o2 + n = 1 + 9 \geq \tau$. (b) A concave corner of the assembly. A South→West concave corner gadget attaches to the concave corner. $t1 + e = 1 + 9 \geq \tau$. An inner mold gadget attaches cooperatively to the FRAME and the South→West concave corner gadget. $o3 + e = 1 + 9 \geq \tau$. (c) A convex corner of the assembly. A West→South convex corner gadget attaches at the convex corner, allowing the mold process to continue around the convex corner. $o3 + n = 1 + 9 \geq \tau$. $o1 + n = 1 + 9 \geq \tau$.

placed, similar to the pre-drill gadget shown in Section V. The inner pre-drill gadget can be seen in Figure 5.13. Once the inner pre-drilling gadget has attached, the inner drill gadgets attach along the inside edge of the frame, removing the inner mold gadgets one by one counterclockwise. These gadgets reflect the form and function of the drill gadgets shown in Section V. The inner drilling process reflects that of the outer drilling, and can be seen in Figure 5.14a. Corners are handled similarly to the *outer* drill gadgets, from Phase 2 (Section V) An. When the drills completely attach to the inner perimeter of the FRAME, the *inner post-drill* gadget can attach, as seen in Figure 5.15a, to create FRAME[HOLLOW[ϒ]]. After the inner post-drill gadget attaches, the assembly destabilizes into FRAME[START] and HOLLOW[ϒ], as shown in Figure 5.15b.

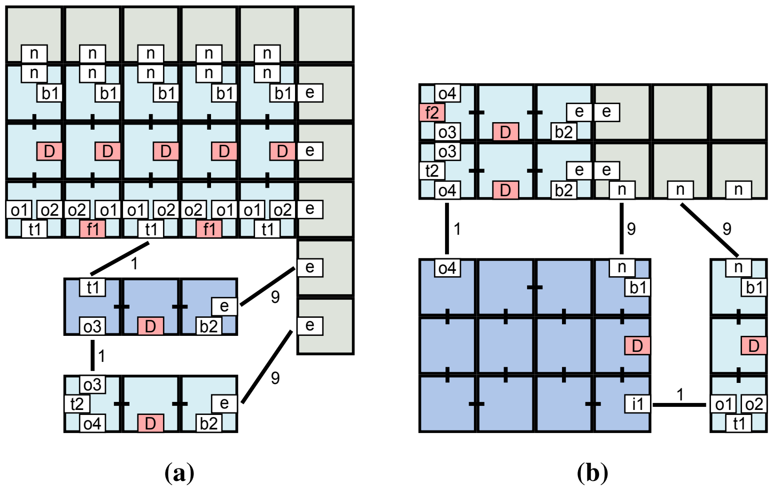


Figure 5.12: Phase 3: Type-2 concave and convex corners of the assembly. (a) A type-2 South→West concave corner gadget binds, which allows the mold process to continue around the concave corner. $t1 + e = 1 + 9 \geq \tau$. $o3 + e = 1 + 9 \geq \tau$.(b) A type-2 West→South corner gadget binds allowing the mold process to continue around the convex corner. $o4 + n = 1 + 9 \geq \tau$. $o1 + n = 1 + 9 \geq \tau$.

Phase 5 Gadgets to Fill the Hollow Outline

Below we describe the gadgets used to implement Phase 5 to fill HOLLOW[ϒ] to create COPY[ϒ]. The *filler gadgets* begin the process of filling the shape, shown in Figure 5.16. Two regions of HOLLOW[ϒ] need to be filled to become COPY[ϒ], the Northernmost-Westernmost corner (Figure 5.16a) and the interior (Figures 5.16b,5.16c).

Universal Shape Replication

In this section we formally state the results, including the class of shapes which can be replicated by the replication gadgets discussed in the previous section. The input shape must be sufficiently large for the replication gadgets to assemble in the intended manner, so we define the *feature size* of a shape as follows (we use the same definition as [47]): for two points a, b in the shape, let $d(a, b) = \max(|a_x - b_x|, |a_y - b_y|)$. Then the feature size is the minimum $d(a, b)$ such that a, b are on two non-adjacent edges of the shape.

Theorem 5.3. There exists a sleek universal shape replicator $\Gamma = (\sigma, 10)$ for genus-0 (hole-free)

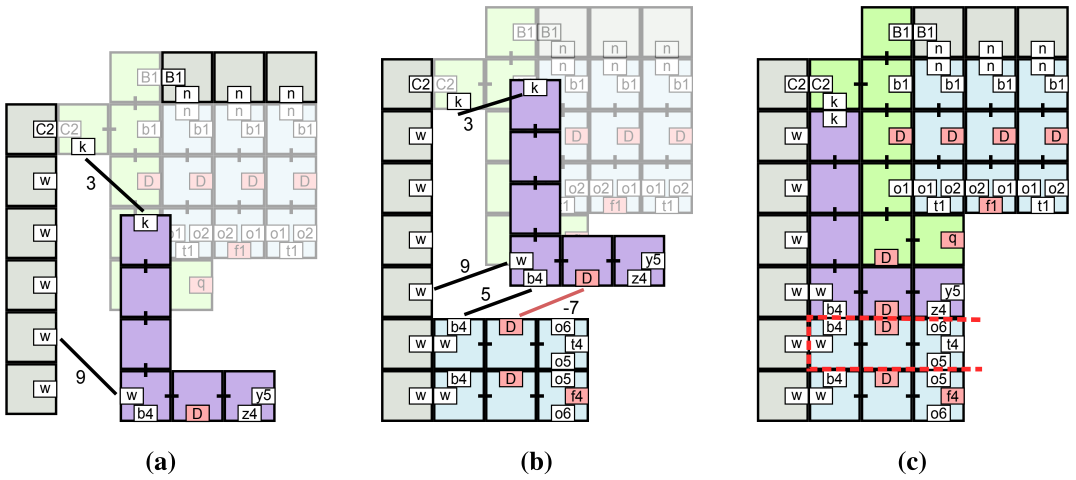


Figure 5.13: Phase 4: The inner pre-drilling process begins. (a) Case 1: The inner mold process has not completed and the inner pre-drill gadget binds cooperatively to the FRAME and the inner start gadget. $w + k = 9 + 3 \geq \tau$. (b) Case 2: The inner mold process has completed and the inner pre-drill gadget binds cooperatively to the FRAME, inner start gadget, and an inner mold gadget. $w + k + b4 + D = 9 + 3 + 5 - 7 \geq \tau$. (c) The inner pre-drill gadget prevents any inner mold gadget from attaching to its south and destabilizes any inner mold gadget that may have been in that space. $b4 + o5 + w + D = 5 + 1 + 9 - 7 < \tau$.

shapes with feature size of at least 9.

Proof. The proof follows by constructing an unbounded shape replication system $\Gamma = (\sigma, 10)$. Consider an initial assembly state σ consisting of infinite counts of the tiles which construct the replication gadgets shown in Section V. For any shape X of genus-0 and minimum feature size 9, consider an assembly Υ of shape X with the properties discussed at the beginning of Section V (i.e., the exposed glues on Υ are N, E, S, and W based on edge orientation, and there is a special glue in the Northernmost-Westernmost position of Υ which exposes glues $C1$ to the North and $C2$ to the West). Then we prove inductively that $\Gamma' = (\sigma \bigcup \Upsilon, 10)$ has the following property: for any $n \in \mathbb{N}$ and producible state S of Γ', there exists a producible state S' containing at least n terminal assemblies of shape X such that $S \rightarrow S'$.

For the base, note that the Γ' follows the process described in Section V. When the FRAME detaches from HOLLOW[Υ], HOLLOW[Υ] is filled to generate 1 terminal assembly of shape X.

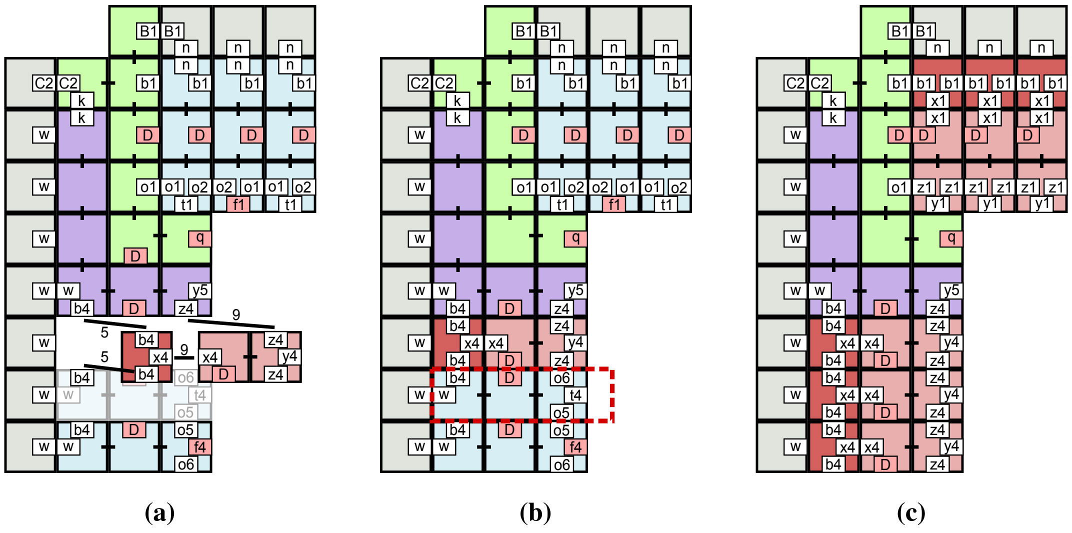


Figure 5.14: Phase 4: The inner drilling process begins. (a) A drill helper binds to the inner pre-drill gadget and an inner mold gadget. $b4 + b4 = 5 + 5 \geq \tau$. A drill block binds cooperatively to the drill helper and pre-drill gadget. $x4 + z4 + D = 9 + 9 - 7 \geq \tau$. (b) The drill gadgets destabilize an inner mold gadget. $b4 + o5 + w + D = 5 + 1 + 9 - 7 < \tau$. (c) The drilling process continues counter-clockwise along the inside perimeter of the FRAME until the drill gadgets reach the inner start gadget.

Then, for any producible state S, $\sigma \bigcup \Upsilon \to S$ by definition, so S contains at least 1 terminal of shape X or $S \to \hat{S}$ such that $\hat{S}$ contains at least 1 terminal of shape X.

Now, consider any producible state A such that A has at least $n \in \mathbb{N}$ terminal assemblies of shape X. For A to have at least one such terminal, HOLLOW[Υ] must have detached from the FRAME. When this occurs, the FRAME is still attached to the inner start gadget, so the inner mold/drill process repeats, generating HOLLOW[Υ]. Again, HOLLOW[Υ] detaches from the FRAME, and then HOLLOW[Υ] is filled, resulting in 1 more terminal assembly of shape X. Therefore $A \to A'$ such that A' contains at least $n + 1$ terminal assemblies of shape X. Then, since S or $\hat{S}$ contains at least 1 terminal of shape X, $S \to S'$ such that S' has at least n terminal assemblies of shape X for any n. This satisfies the first property of Definition 5.1.

Further, any terminal assembly B in the system that does not have shape X, $|B| = O(1)$, satisfying the second property of Definition 5.1. Note that the only assemblies which are larger

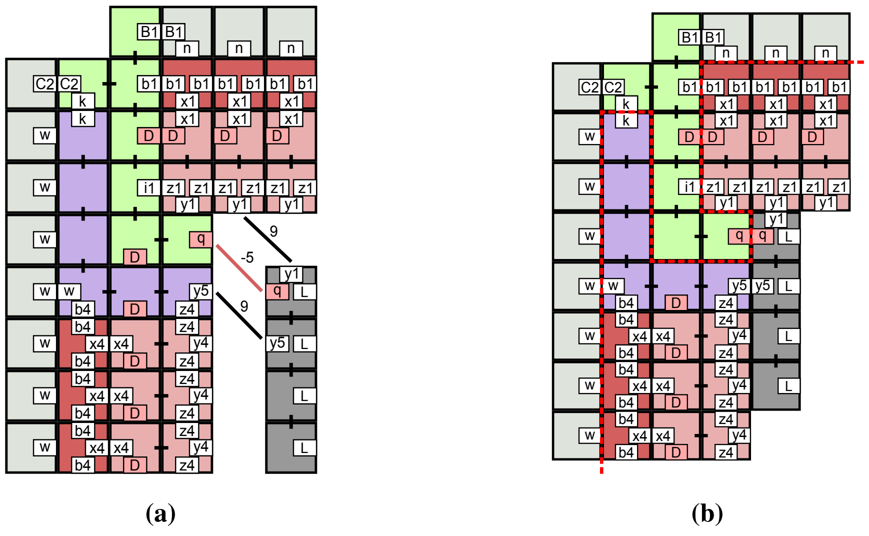


Figure 5.15: Phase 4: The inner post-drilling process. (a) The inner post-drill gadget binds cooperatively to the inner pre-drill gadget and the second-to-last placed drill gadget. $y5 + y1 + q = 9 + 9 - 5 \geq \tau$. (b) The assembly with the FRAME and the start gadget becomes unstable and breaks off the assembly. $b1 + w + k + q + D = 5 + 9 + 3 - 5 - 7 < \tau$.

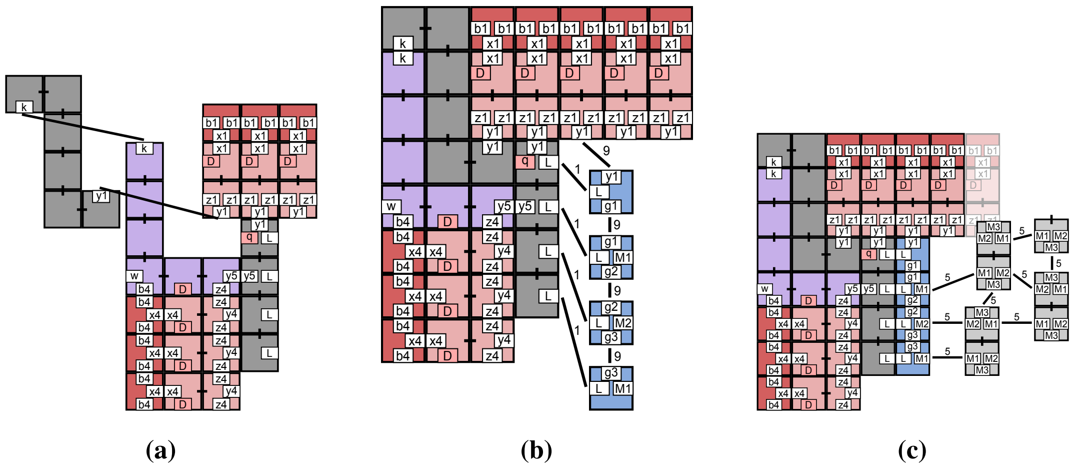


Figure 5.16: Phase 5: The inner filling process. (a) The first filler gadget comes in to fill in the Northernmost-Westernmost corner of HOLLOW[Υ]. $k + y1 = 3 + 9 \geq \tau$. (b) $L = 1, g* = 9$. (c) $M1 = M2 = M3 = 5$. First, the 4 tiles in (b) attach one by one to the hollow shape. The filler gadget in (c) can then attach and flood the inside of the shape.

than $O(1)$ size are assemblies related to the input shape, the FRAME, and HOLLOW[Υ]. The input shape and the intermediate assemblies produced while creating the FRAME are not terminal since the process repeats once the FRAME is detached. The FRAME and the intermediate assemblies produced while creating HOLLOW[Υ] are not terminal since the process repeats once HOLLOW[Υ] detaches. HOLLOW[Υ] fills to create a terminal assembly of shape X, so none of the intermediate assemblies are terminal.

Then, $\Gamma = (\sigma, 10)$ is a universal shape replicator for genus-0 (hole-free) shapes with minimum feature size 9. To show that it is a sleek universal shape replicator (i.e. the bonus properties discussed in the definitions apply), note that the input shape considered here has the same properties as described in that definition, and that the input shape, the FRAME, HOLLOW[Υ], and the intermediate steps of each of these assemblies, do not grow larger than $O(|X|)$ in size. $\square$

The Big Picture

The above described work reflects my contribution to a larger body of work [84]. In this work we formally introduced the problem of shape replication in the 2-handed tile assembly model and provided a universal replication system for all genus-0 shapes with at least a constant minimum feature size. Shape replication has been studied in more powerful self-assembly models such as the staged self-assembly model and the signal tile model. However, our result constitutes the first example of general shape replication in a passive model of self-assembly where no outside experimenter intervention is required, and where the system monomers are state-less, static pieces that interact based purely on the attraction and repulsion of surface chemistry.

Phase 1	
$C + N \geq \tau$	$O + N \geq \tau$
$T + W \geq \tau$	
Phase 2: Pre-drilling	
$K + C2 + B + D \geq \tau$	$K + C2 \geq \tau$
$B + W + O + D < \tau$	
Phase 2: Drilling	
$B + B \geq \tau$	$Z + X + D \geq \tau$
$B + X \geq \tau$	
Phase 2: Drilling Type-1 Corners	
$B + W + T + D < \tau$	$Y + T \geq \tau$
$Z + V + F \geq \tau$	$Y + Z + T + F \geq \tau$
$Z + B + V + Y + F \geq \tau$	$Z + X + V + Y + F \geq \tau$
$T + Z + B + F \geq \tau$	$O + N + O + F < \tau$
$B + B \geq \tau$	$B + U \geq \tau$
Phase 2: Drilling Type-2 Corners	
$Z + H + F \geq \tau$	$H + Y + Z + B + F \geq \tau$
$H + Y + Z + X + F \geq \tau$	$T + Z + B + F \geq \tau$
$O + N + O + F < \tau$	$N < \tau$
$A \geq \tau$	$B + B \geq \tau$
Phase 2: Drilling Obtuse Corners	
$B + N + O + D < \tau$	$B + B \geq \tau$
$Z + R + S + D \geq \tau$	$O + W + B + D < \tau$
Phase 2: Post-Drilling	
$B + N + C1 + K + D \geq \tau$	$Y + Y + Q \geq \tau$
$Y + Z + B + N + C1 + K + Q \geq \tau$	$B + N + C1 + C2 + B + Z + Y + Q + D \geq \tau$
$Y + Z + B + C2 + K + Q \geq \tau$	$B + N + C1 + K + D + Q \geq \tau$
$C2 + C1 + N \geq \tau$	$K + B + C2 + Q + D < \tau$
Phase 3	
$C2 + B \geq \tau$	$o + n \geq \tau$
$t + e \geq \tau$	
Phase 4 - Pre-drilling	
$k + w + b + D \geq \tau$	$b + w + i + D < \tau$
$k + w \geq \tau$	
Phase 4 - Drilling	
$b + b \geq \tau$	$z + x + D \geq \tau$
$b + w + i + D < \tau$	$b + x \geq \tau$
Phase 4 - Post-drilling	
$B + b + k + C2 + D \geq \tau$	$y + y + Q \geq \tau$
$B + b + k + C2 + D + Q \geq \tau$	$b + k + w + D + q < \tau$
$B + C2 \geq \tau$	$B + C2 + C1 + N + B1 + O + D \geq \tau$
$Z + B \geq \tau$	$B + W + W + O + D \geq \tau$
$k + y \geq \tau$	
Phase 5	
$L + g* \geq \tau$	$M + M \geq \tau$

Table 5.2: Shown are the constraints, in the form of inequalities, which must be satisfied for the replication gadgets shown in Section V to function in the way required to prove Theorem 5.3. All single glue labels except A must have strength $< \tau$. Unless otherwise stated, in these inequalities, a glue label G represents all glues $G*$ (e.g., G1, G2, etc.).

CHAPTER VI

SIMULATION

VersaTILE is a graphical simulator (Figure 6.1) and tile editor (Figure 6.2) designed to be able to simulate many models of tile-based algorithmic self-assembly. VersaTILE is cross-platform (Java) and currently supports the simulation of the abstract Tile Assembly Model, the dupled Tile Assembly Model, the hexagonal Tile Assembly Model, and the polyomino Tile Assembly Model. It also supports generalized versions of many of the extensions to these models such as probabilistic attachment, temperature programming, concentration programming, negative interactions, a run-time model, and flexible glues. VersaTILE has clean and consistent user-interface that makes it very easy for a user to identify how to use the software. To make things easier, VersaTILE automatically keeps up-to-date and looks the same across all platforms. Users are able to specify custom colors for tiles so that they can quickly get a high-level overview of the state of the tile assembly system. VersaTILE was inspired by other tile-based self-assembly simulators that exist, such as Xgrow and the ISU Tile Assembly Simulator. Simulators for this line of research are important tools for the rapid prototyping and debugging of tile assembly systems. The primary contribution of VersaTILE to this space is that VersaTILE generalizes the self-assembly process and uses polymorphism to allow a developer to make new models by simply constraining or modifying small pieces of the base classes through inheritance. Not only is VersaTILE open-source, but it uses an XML file format to make it simple for others to support or write to the VersaTILE format. Besides being a tool for rapid prototyping and development, VersaTILE is a useful tool for interactive demonstrations as well as education. This poster reviews the major features and functionality of VersaTILE.

Simulated Models

VersaTILE is able to simulate various models of DNA tile self-assembly such as the abstract Tile Assembly Model (aTAM), the polyomino Tile Assembly Model (polyTAM), the dupled Tile Assembly Model (DaTAM), hexagonal Tile Assembly Model (hTAM), and other models that extend on the use of unit square DNA tiles.

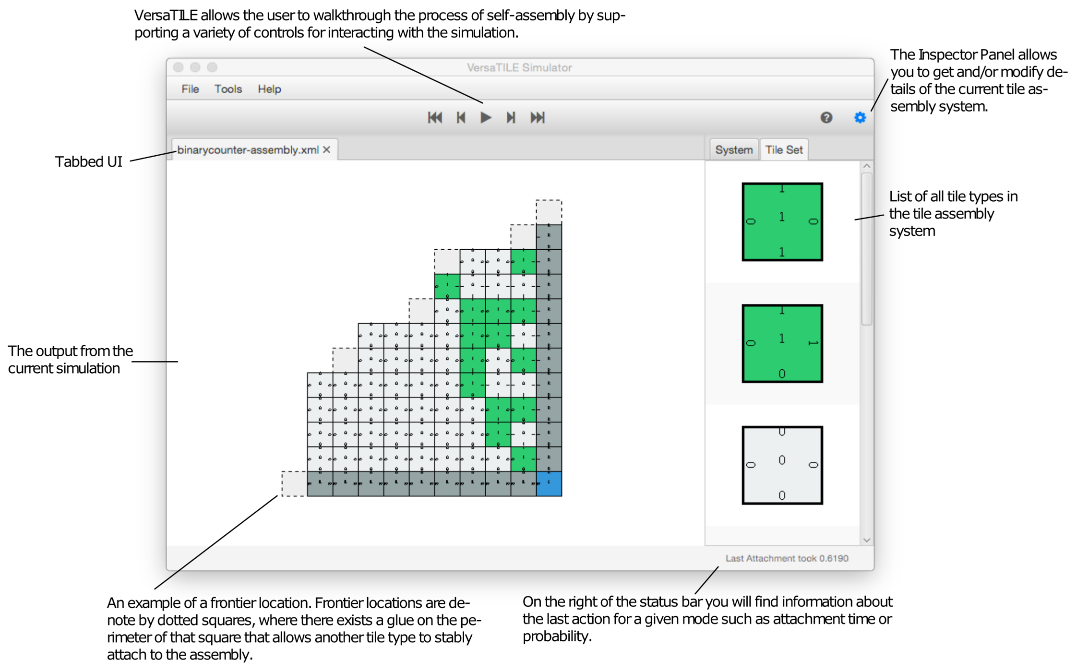
Features

VersaTILE supports generalized extensions to these models such as temperature programming, concentration programming, negative interactions (attachments only), a run-time model, flexible glues, and probabilistic assembly.

VersaTILE supports auto-updating, importing of ISU TAS files, XML output, visualization and debugging of the *frontier*, and other features. VersaTILE is easy to extend. Developers can add new models by simply extending existing Java classes and overriding default behavior.

The Big Picture

VersaTILE attempts to provide a solution for the problem of simulating self-assembling systems. As the lead developer my primary contributions to VersaTILE were the high-level design, the implementation of many of the base classes, the GUI interface, the tile editor, the auto-updating mechanism, the development of all the extensions to base models, and more. This work was presented at a poster session at the *21st International Conference on DNA Computing and Molecular Programming (DNA '15)* titled "Multi-Model Cross-Platform Self-Assembly Simulation with VersaTILE" to a large portion of the self-assembly community.



VersaTILE allows the user to walkthrough the process of self-assembly by supporting a variety of controls for interacting with the simulation.

The Inspector Panel allows you to get and/or modify details of the current tile assembly system.

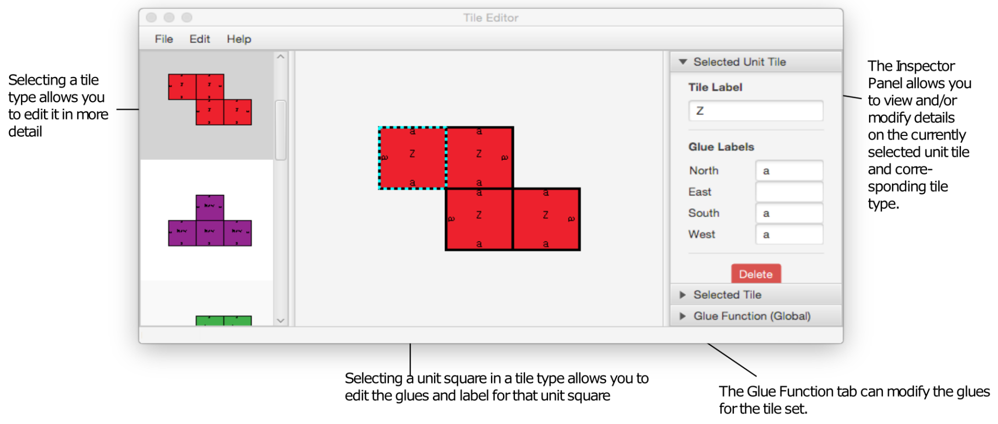
List of all tile types in the tile assembly system

The output from the current simulation

An example of a frontier location. Frontier locations are denote by dotted squares, where there exists a glue on the perimeter of that square that allows another tile type to stably attach to the assembly.

On the right of the status bar you will find information about the last action for a given mode such as attachment time or probability.

Figure 6.1: VersaTILE: A breakdown of the primary components on the main window.



Selecting a tile type allows you to edit it in more detail

The Inspector Panel allows you to view and/or modify details on the currently selected unit tile and corresponding tile type.

Selecting a unit square in a tile type allows you to edit the glues and label for that unit square

The Glue Function tab can modify the glues for the tile set.

Figure 6.2: VersaTILE: A breakdown of the primary components on the tile editor window.

CHAPTER VII

CONCLUSION

During my time as a graduate student, I researched the area of algorithmic self-assembly. Investigating different models is essential because without having a model it is almost impossible to design a real-world experimental system. Knowing the capabilities of different models, helps to identify which problems can or can't be solved by certain systems and helps to establish a hierarchy of models in terms of complexity. I was able to make original contributions to the areas of random number generation, information encoding, shape replication, and simulation.

Random Number Generation

In regards to random number generation, I was able to construct a 1-extensible robust coin flip system inspired by Von Neumann's randomness extractors that uses unbounded space. I was also able to rigorously analyze and prove a bound on the bias of this system when its growth is fixed within a space constraint. When paired with other results from the published version [77], this upper bound highlights a gap in the power between 1-extensible and 2-extensible systems, which can generate robust coin flips in $O(1)$-space.

A direction for future work is the consideration of generalizations of the coin flip problem. Our partition definition for coin flip systems extends naturally to distributions with more than two outcomes, as well as non-uniform distributions. What general probability distributions can be assembled in $O(1)$ space, and with what efficiency? We have also introduced the online variant of concentration robustness in which species concentrations may change at each step of the self-assembly process. We have shown in [77] that when such changes are completely arbitrary, coin flipping is not possible in the aTAM. A relaxed version of this robustness constraint could permit concentration changes to be bounded by some fixed rate. In such a model, how close to a fair coin flip can a system guarantee in terms of the given rate bound? As an additional relaxation, one could consider the problem in which an initial concentration assignment may be *approximately* set by the system designer, thereby modeling the limited precision an experimenter can obtain with a pipette.

A final line of future work focuses on applying randomization in self-assembly to computing functions. The parallelization within the abstract tile assembly model allows for substantially faster arithmetic than what is possible in non-parallel computational models [86]. Can randomization be applied to solve these problems even faster? Moreover, there are a number of potentially interesting problems that might be helped by randomization, such as primality testing, sorting, or general simulation of randomized boolean circuits.

Information Encoding

In regards to information encoding, I was able to utilize the base conversion approach of [15], [28], [37], [81] to reduce the tile complexity building bit string pads, when limited to $O(1)$ bins and stages. I was also able to describe how given some constraint of tile types t and constraint on bins b, $O(\frac{\log\log b}{\log t})$ stages can be used to uniquely produced all width-2 gap-1 $\lfloor \log(b) \rfloor$-bit string pads, each placed in a distinct bin. These constructions correspond to terms in the upper bound for the nearly optimal construction of bit string pads described in the published version of this work [85], which nearly matches the lower bound.

The obvious technical question that remains is whether the additive $O(\frac{\log\log b}{\log t})$ gap between the upper and lower bounds can be removed. This gap in the optimal bit pad constructions is induced by the wings built with the method from Section IV, and seems difficult to eliminate, as the wings serve as our generic solution to assembly labeling and coordinated attachment. As such, the wings subconstruction might be useful for improving the efficiency of staged assembly for other shape classes.

Shape Replication

In regards to shape replication, I was able to generate a constant set of tiles that can replicate all hole-free shapes with feature size of at least 9 in a passive model of self-assembly. Previous work in similar models required the use of stages and global "destroy" operations by an "RNAse enzyme" molecule. Our work opens up a number of directions for future work. One direction includes analyzing and improving the *rate* of replication. Under a reasonable model of replication time, our construction should costitute a *quadratic* replicator− in that after time t, $\Omega(t^2)$ copies of the input shape are expected. Designing a faster replicator is an open question. In particular, achieving an *exponential* replicator is an important goal. Further, to properly consider these questions requires a formal modelling of replication rates for this model.

Another direction is to further generalize the class of shapes that can be replicated. For example, can shapes with holes be replicated? This is likely difficult, but might be achievable by drilling into the input assembly in a carefully engineered way. Achieving such general replication would be the first example of shape replication that extends beyond genus-0. This seems to require a shape with more complex glues, which leads to another area of research.

The consideration of variations of the *sleek* requirements may be of interest. For example, removing the need for any special tiles from the replication system might be achievable. Or, allowing for more complex input assemblies could allow for high genus replication, as discussed above. Finally, determining the lowest necessary temperature and glue strengths needed for replication is an open question. We use temperature value 10 to maximize clarity of the construction and have not attempted to optimize this value. To help such optimization, we have included a compiled table (Table 5.2) showing the inequality specifications induced by our construction for each gadget to help guide where a modification to the replication algorithm might reduce the temperature needed.

Finally, extending replication to work in a planar fashion is an open question, as the current construction requires a large assembly to "pop" out of an encased frame. Planar replication systems might provide insight into extending replication into 3D, while maintaining a *spatial* construction.

Simulation

In regards to simulation, I contributed as the lead developer of VersaTILE. Future directions for this work would include the support for the kinetic Tile Assembly Model (kTAM), the two-Handed Tile Assembly Model, and the Staged Assembly Model. There are also plans to incorporate a 3D extension of all supported models in the simulator. With the decision to use JavaFX, a mobile version could be created for Android and iOS devices.

Closing Remarks

Each of these results relates to fundamental primitives for nanotechnology: the ability to use randomized algorithms, the ability to solve problems invariant of the relative concentrations of molecules, the ability to encode information into tile assembly systems in the form of bits, and the ability to “sense” information from unknown objects in the form of shape and to be able to replicate that shape.

This research is important because it provides constructions, algorithms for designing tile assembly systems, for solving these interesting problems. There are still many questions left to answer for the motivated researcher or upcoming graduate students.

BIBLIOGRAPHY

[1] J. von Neumann, “Various Techniques Used in Connection with Random Digits,” *Journal of Research of the National Bureau of Standards*, vol. 12, pp. 36–38, 1951.

[2] N. C. Seeman, “Nucleic-acid junctions and lattices,” *Journal of Theoretical Biology*, vol. 99, pp. 237–247, 1982.

[3] R. Deaton, M. Garzon, R. Murphy, D. Franceschetti, and S. Stevens, “Genetic Search of Reliable Encodings for DNA Based Computation,” in *Proceedings of the* 1^{st} *Annual Conference on Genetic Programming*, 1996, pp. 9–15.

[4] D. D. Shoemaker, D. A. Lashkari, D. Morris, M. Mittmann, and R. W. Davis, “Quantitative Phenotypic Analysis of Yeast Deletion Mutants Using a Highly Parallel Molecular Bar-coding Strategy,” *Nature*, vol. 16, pp. 450–456, Dec. 1996.

[5] S. Brenner, *Methods for Sorting Polynucleotides using Oligonucleotide Tags*, US Patent Number 5,604,097, Feb. 1997.

[6] A. G. Frutos, Q. Liu, A. J. Thiel, A. M. W. Sanner, A. E. Condon, L. M. Smith, and R. M. Corn, “Demonstration of a Word Design Strategy for DNA Computing on Surfaces,” *Nucleic Acids Research*, vol. 25, pp. 4748–4757, Dec. 1997.

[7] M. Garzon, R. Deaton, P. Neathery, D. Franceschetti, and R. Murphy, “A New Metric for DNA Computing,” in *Proceedings of the* 2^{nd} *Genetic Programming Conference*, 1997, pp. 472–278.

[8] E. Winfree, “Algorithmic self-assembly of DNA,” PhD thesis, California Institute of Technology, Jun. 1998.

[9] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman, "Design and self-assembly of two-dimensional DNA crystals.," *Nature*, vol. 394, no. 6693, pp. 539–44, 1998.

[10] F. Liu, R. Sha, and N. C. Seeman, "Modifying the surface features of two-dimensional DNA crystals.," *Journal of the American Chemical Society*, vol. 121, no. 5, pp. 917–922, 1999.

[11] C. Mao, W. Sun, and N. C. Seeman, "Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy.," *Journal of the American Chemical Society*, vol. 121, no. 23, pp. 5437–5443, 1999.

[12] C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman, "Logical computation using algorithmic self-assembly of DNA triple-crossover molecules.," *Nature*, vol. 407, no. 6803, pp. 493–6, 2000.

[13] P. W. K. Rothemund, "Using lateral capillary forces to compute by self-assembly," *Proceedings of the National Academy of Sciences*, vol. 97, no. 3, pp. 984–989, 2000. DOI: `10.1073/pnas.97.3.984`. eprint: `http://www.pnas.org/content/97/3/984.full.pdf`. [Online]. Available: `http://www.pnas.org/content/97/3/984.abstract`.

[14] P. W. K. Rothemund and E. Winfree, "The program-size complexity of self-assembled squares (extended abstract)," in *STOC 2000: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, 2000, pp. 459–468, ISBN: 1-58113-184-4. DOI: `http://doi.acm.org/10.1145/335305.335358`.

[15] L. M. Adleman, Q. Cheng, A. Goel, and M.-D. Huang, "Running time and program size for self-assembled squares," in *STOC 2001: Proceedings of the thirty-third annual ACM Symposium on Theory of Computing*, Hersonissos, Greece: ACM, 2001, pp. 740–748, ISBN: 1-58113-349-9. DOI: `http://doi.acm.org/10.1145/380752.380881`.

[16] L. M. Adleman, Q. Cheng, A. Goel, M.-D. Huang, and H. Wasserman, "Linear self-assemblies: Equilibria, entropy and convergence rates," in *In Sixth International Conference on Difference Equations and Applications*, Taylor and Francis, 2001.

[17] A. Brenneman and A. E. Condon, "Strand Design for Bio-Molecular Computation," *Theoretical Computer Science*, vol. 287, no. 1, pp. 39–58, 2001.

[18] J. D. Hartgerink, E. Beniash, and S. I. Stupp, "Self-Assembly and Mineralization of Peptide-Amphiphile Nanofibers," *Science*, vol. 294, no. 5547, pp. 1684–1688, 2001. DOI: `10.1126/science.1063187`. eprint: `http://www.sciencemag.org/cgi/reprint/294/5547/1684.pdf`. [Online]. Available: `http://www.sciencemag.org/cgi/content/abstract/294/5547/1684`.

[19] A. Marathe, A. Condon, and R. M. Corn, "On Combinatorial DNA Word Design," *Journal of Computational Biology*, vol. 8, no. 3, pp. 201–219, 2001.

[20] P. W. K. Rothemund, "Theory and experiments in algorithmic self-assembly," PhD thesis, University of Southern California, Dec. 2001.

[21] L. M. Adleman, Q. Cheng, A. Goel, M.-D. A. Huang, D. Kempe, P. M. de Espanés, and P. W. K. Rothemund, "Combinatorial optimization problems in self-assembly," in *STOC 2002: Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, 2002, pp. 23–32.

[22] O. D. King, "Bounds for DNA Codes with Constant GC-content," *Electronic Journal of Combinatorics*, vol. 10, no. 1, #R33 13pp, 2003.

[23] D. C. Tulpan and H. H. Hoos, "Hybrid Randomised Neighbourhoods Improve Stochastic Local Search for DNA Code Design," in *Lecture Notes in Computer Science 2671: Proceedings of the 16th Conference of the Canadian Society for Computational Studies of Intelligence*, Y. Xiang and B. Chaib-draa, Eds., New York, NY: Springer-Verlag, 2003, pp. 418–433.

[24] D. C. Tulpan, H. H. Hoos, and A. Condon, "Stochastic Local Search Algorithms for DNA Word Design," in *Lecture Notes in Computer Science 2568: Proceedings of the 8th International Workshop on DNA-Based Computers*, M. Hagiya and A. Ohuchi, Eds., New York, NY: Springer-Verlag, 2003, pp. 229–241.

[25] H. Yan, S. H. Park, G. Finkelstein, J. H. Reif, and T. H. LaBean, "DNA-Templated Self-Assembly of Protein Arrays and Highly Conductive Nanowires," *Science*, vol. 301, no. 5641, pp. 1882–1884, 2003. DOI: `10.1126/science.1089389`. eprint: `http://www.sciencemag.org/cgi/reprint/301/5641/1882.pdf`. [Online]. Available: `http://www.sciencemag.org/cgi/content/abstract/301/5641/1882`.

[26] P. W. Rothemund, N. Papadakis, and E. Winfree, "Algorithmic self-assembly of DNA Sierpinski triangles," *PLoS Biology*, vol. 2, no. 12, pp. 2041–2053, 2004.

[27] G. Aggarwal, Q. Cheng, M. H. Goldwasser, M.-Y. Kao, P. M. de Espanés, and R. T. Schweller, "Complexities for generalized models of self-assembly," *SIAM Journal on Computing*, vol. 34, pp. 1493–1515, 2005, Preliminary version appeared in SODA 2004.

[28] Q. Cheng, G. Aggarwal, M. H. Goldwasser, M.-Y. Kao, R. T. Schweller, and P. M. de Espanés, "Complexities for generalized models of self-assembly," *SIAM Journal on Computing*, vol. 34, pp. 1493–1515, 2005.

[29] P. Gaborit and O. D. King, "Linear constructions for DNA codes," *Theoretical Computer Science*, vol. 334, pp. 99–113, 2005.

[30] P. W. K. Rothemund, "Design of dna origami," in *ICCAD'05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, San Jose, CA: IEEE Computer Society, 2005, pp. 471–478, ISBN: 0-7803-9254-X.

[31] F. Becker, I. Rapaport, and E. Rémila, "Self-assembling classes of shapes with a minimum number of tiles, and in optimal time," in *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, 2006, pp. 45–56.

[32] A. M. Kalsin, M. Fialkowski, M. Paszewski, S. K. Smoukov, K. J. M. Bishop, and B. A. Grzybowski, "Electrostatic Self-Assembly of Binary Nanoparticle Crystals with a Diamond-Like Lattice," *Science*, vol. 312, no. 5772, pp. 420–424, 2006. DOI: `10.1126/science.1125124`. eprint: `http://www.sciencemag.org/cgi/reprint/312/5772/420.pdf`.

[Online]. Available: http://www.sciencemag.org/cgi/content/abstract/312/5772/420.

[33] M.-Y. Kao and R. T. Schweller, “Reducing tile complexity for self-assembly through temperature programming,” in *SODA 2006: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2006, pp. 571–580.

[34] Z. Tang, Z. Zhang, Y. Wang, S. C. Glotzer, and N. A. Kotov, “Self-Assembly of CdTe Nanocrystals into Free-Floating Sheets,” *Science*, vol. 314, no. 5797, pp. 274–278, 2006. DOI: 10.1126/science.1128045. eprint: http://www.sciencemag.org/cgi/reprint/314/5797/274.pdf. [Online]. Available: http://www.sciencemag.org/cgi/content/abstract/314/5797/274.

[35] H.-L. Chen, R. Schulman, A. Goel, and E. Winfree, “Reducing facet nucleation during algorithmic self-assembly,” *Nano Letters*, vol. 7, no. 9, pp. 2913–2919, Sep. 2007. DOI: 10.1021/nl070793o. [Online]. Available: http://dx.doi.org/10.1021/nl070793o.

[36] U. Majumder, T. H. LaBean, and J. H. Reif, “Activatable tiles for compact error-resilient directional assembly,” in *DNA 13: 13th International Meeting on DNA Computing, Memphis, Tennessee, June 4-8, 2007.*, 2007.

[37] D. Soloveichik and E. Winfree, “Complexity of self-assembled shapes,” *SIAM Journal on Computing*, vol. 36, no. 6, pp. 1544–1569, 2007, Preliminary version appeared in DNA 10.

[38] E. D. Demaine, M. L. Demaine, S. P. Fekete, M. Ishaque, E. Rafalin, R. T. Schweller, and D. L. Souvaine, “Staged self-assembly: Nanomanufacture of arbitrary shapes with $o(1)$ glues,” *Natural Computing*, vol. 7, no. 3, pp. 347–370, 2008.

[39] M.-Y. Kao and R. T. Schweller, “Randomized self-assembly for approximate shapes,” in *Inter. Coll. on Automata, Languages, and Programming*, ser. Lecture Notes in Computer Science, vol. 5125, 2008, pp. 370–384.

[40] L. M. Adleman, J. Kari, L. Kari, D. Reishus, and P. Sosík, "The undecidability of the infinite ribbon problem: Implications for computing by self-assembly," *SIAM Journal on Computing*, vol. 38, no. 6, pp. 2356–2381, 2009, Preliminary version appeared in FOCS 2002. [Online]. Available: `http://dx.doi.org/10.1137/080723971`.

[41] E. S. Andersen, M. Dong, M. M. Nielsen, K. Jahn, R. Subramani, W. Mamdouh, M. M. Golas, B. Sander, H. Stark, C. L. P. Oliveira, J. S. Pedersen, V. Birkedal, F. Besenbacher, K. V. Gothelf, and J. Kjems, "Self-assembly of a nanoscale dna box with a controllable lid," *Nature*, vol. 459, no. 7243, pp. 73–76, May 2009, ISSN: 0028-0836. DOI: `10.1038/nature07971`. [Online]. Available: `http://dx.doi.org/10.1038/nature07971`.

[42] R. D. Barish, R. Schulman, P. W. Rothemund, and E. Winfree, "An information-bearing seed for nucleating algorithmic self-assembly," *Proceedings of the National Academy of Sciences*, vol. 106, no. 15, pp. 6054–6059, Mar. 2009. DOI: `10.1073/pnas.0808736106`. [Online]. Available: `http://dx.doi.org/10.1073/pnas.0808736106`.

[43] H. Chandran, N. Gopalkrishnan, and J. H. Reif, "The tile complexity of linear assemblies," in *36th International Colloquium on Automata, Languages and Programming*, vol. 5555, 2009.

[44] D. Doty, J. H. Lutz, M. J. Patitz, S. M. Summers, and D. Woods, "Random number selection in self-assembly," in *UC*, 2009, pp. 143–157.

[45] ——, "Random number selection in self-assembly," in *UC 2009: Proceedings of The Eighth International Conference on Unconventional Computation*, ser. Lecture Notes in Computer Science, vol. 5715, Springer, 2009, pp. 143–157.

[46] J. Maňuch, L. Stacho, and C. Stoll, "Step-assembly with a constant number of tile types," in *ISAAC 2009: Proceedings of the 20th International Symposium on Algorithms and Computation*, Honolulu, Hawaii: Springer-Verlag, 2009, pp. 954–963, ISBN: 978-3-642-10630-9. DOI: `http://dx.doi.org/10.1007/978-3-642-10631-6_96`.

[47] Z. Abel, N. Benbernou, M. Damian, E. D. Demaine, M. L. Demaine, R. Flatland, S. D. Kominers, and R. Schwelle, “Shape replication through self-assembly and rnase enzymes,” in *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA ’10, Austin, Texas: Society for Industrial and Applied Mathematics, 2010, pp. 1045–1064, ISBN: 978-0-898716-98-6. [Online]. Available: `http://dl.acm.org/citation.cfm?id=1873601.1873686`.

[48] Z. Abel, N. Benbernou, M. Damian, E. Demaine, M. Demaine, R. Flatland, S. Kominers, and R. Schweller, “Shape replication through self-assembly and RNase enzymes,” in *SODA 2010: Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, Austin, Texas: Society for Industrial and Applied Mathematics, 2010.

[49] D. Doty, “Randomized self-assembly for exact shapes,” *SIAM Journal on Computing*, vol. 39, no. 8, pp. 3521–3552, 2010, Preliminary version appeared in FOCS 2009.

[50] D. Doty, M. J. Patitz, D. Reishus, R. T. Schweller, and S. M. Summers, “Strong fault-tolerance for self-assembly with fuzzy temperature,” in *FOCS 2010: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, IEEE, 2010, pp. 417–426.

[51] H. Gu, J. Chao, S.-J. Xiao, and N. C. Seeman, “A proximity-based programmable dna nanoscale assembly line,” *Nature*, vol. 465, no. 7295, pp. 202–205, May 2010, ISSN: 0028-0836. DOI: `10.1038/nature09026`. [Online]. Available: `http://dx.doi.org/10.1038/nature09026`.

[52] K. Lund, A. J. Manzo, N. Dabby, N. Michelotti, A. Johnson-Buck, J. Nangreave, S. Taylor, R. Pei, M. N. Stojanovic, N. G. Walter, E. Winfree, and H. Yan, “Molecular robots guided by prescriptive landscapes,” *Nature*, vol. 465, no. 7295, pp. 206–210, May 2010, ISSN: 0028-0836. DOI: `10.1038/nature09012`. [Online]. Available: `http://dx.doi.org/10.1038/nature09012`.

[53] N. Bryans, E. Chiniforooshan, D. Doty, L. Kari, and S. Seki, "The power of nondeterminism in self-assembly," in *SODA 2011: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2011, pp. 590–602.

[54] H.-L. Chen, D. Doty, and S. Seki, "Program size and temperature in self-assembly," in *ISAAC 2011: Proceedings of the 22nd International Symposium on Algorithms and Computation*, ser. Lecture Notes in Computer Science, vol. 7074, Springer-Verlag, 2011, pp. 445–453.

[55] M. Cook, Y. Fu, and R. T. Schweller, "Temperature 1 self-assembly: Deterministic assembly in 3D and probabilistic assembly in 2D," in *Proc. of the 22nd ACM-SIAM Sym. on Discrete Algorithms*, ser. SODA'11, 2011, pp. 570–589.

[56] E. D. Demaine, S. Eisenstat, M. Ishaque, and A. Winslow, "One-dimensional staged self-assembly," in *Proceedings of the 17th international conference on DNA computing and molecular programming*, ser. DNA'11, Pasadena, CA, 2011, pp. 100–114.

[57] E. D. Demaine, M. J. Patitz, R. T. Schweller, and S. M. Summers, "Self-assembly of arbitrary shapes using RNAse enzymes: Meeting the kolmogorov bound with small scale factor," in *STACS 2011: Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science*, 2011.

[58] M. Patitz, R. Schweller, and S. Summers, "Exact shapes and turing universality at temperature 1 with a single negative glue," in *DNA Computing and Molecular Programming*, ser. LNCS, vol. 6937, 2011, pp. 175–189, ISBN: 978-3-642-23637-2. DOI: 10.1007/978-3-642-23638-9_15. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23638-9_15.

[59] J. H. Reif, S. Sahu, and P. Yin, "Complexity of graph self-assembly in accretive systems and self-destructible systems," *Theoretical Computer Science*, vol. 412, no. 17, pp. 1592–1605, 2011, ISSN: 0304-3975. DOI: http://dx.doi.org/10.1016/j.tcs.2010.10.034. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397510005967.

[60] S. Cannon, E. D. Demaine, M. L. Demaine, S. Eisenstat, M. J. Patitz, R. Schweller, S. M. Summers, and A. Winslow, "Two hands are better than one (up to constant factors)," *Arxiv preprint arXiv:1201.1650*, 2012.

[61] H. Chandran, N. Gopalkrishnan, and J. Reif, "Tile complexity of linear assemblies," *SIAM Journal on Computing*, vol. 41, no. 4, pp. 1051–1073, 2012. DOI: 10.1137/110822487. eprint: http://dx.doi.org/10.1137/110822487. [Online]. Available: http://dx.doi.org/10.1137/110822487.

[62] H.-L. Chen and D. Doty, "Parallelism and time in hierarchical self-assembly," in *SODA 2012: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2012.

[63] D. Doty, J. H. Lutz, M. J. Patitz, R. Schweller, S. M. Summers, and D. Woods, "The tile assembly model is intrinsically universal," in *FOCS 2012: Proceedings of the 53rd IEEE Conference on Foundations of Computer Science*, 2012.

[64] B. Fu, M. Patitz, R. Schweller, and R. Sheline, "Self-assembly with geometric tiles," in *Automata, Languages, and Programming*, ser. LNCS, vol. 7391, 2012, pp. 714–725, ISBN: 978-3-642-31593-0. DOI: 10.1007/978-3-642-31594-7_60. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31594-7_60.

[65] Y. Ke, L. L. Ong, W. M. Shih, and P. Yin, "Three-dimensional structures self-assembled from dna bricks," *Science*, vol. 338, no. 6111, pp. 1177–1183, 2012.

[66] R. Schulman, B. Yurke, and E. Winfree, "Robust self-replication of combinatorial information via crystal growth and scission," *PNAS*, vol. 109, no. 17, pp. 6405–6410, 2012.

[67] N. Bryans, E. Chiniforooshan, D. Doty, L. Kari, and S. Seki, "The power of nondeterminism in self-assembly," *Theory of Computing*, vol. 9, no. 1, pp. 1–29, 2013, Preliminary version appeared in SODA 2011. DOI: 10.4086/toc.2013.v009a001. [Online]. Available: http://www.theoryofcomputing.org/articles/v009a001.

[68] S. Cannon, E. D. Demaine, M. L. Demaine, S. Eisenstat, M. J. Patitz, R. Schweller, S. M. Summers, and A. Winslow, "Two hands are better than one (up to constant factors): Self-assembly in the 2ham vs. atam," in *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, ser. LIPIcs, vol. 20, 2013, pp. 172–184, ISBN: 978-3-939897-50-7. [Online]. Available: `http://dblp.uni-trier.de/db/conf/stacs/stacs2013.html#CannonDDEPSSW13`.

[69] E. Demaine, M. Patitz, T. Rogers, R. Schweller, S. M. Summers, and D. Woods, "The two-handed tile assembly model is not intrinsically universal," in *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP 2013)*, Riga, Latvia, 2013.

[70] D. Doty, L. Kari, and B. Masson, "Negative interactions in irreversible self-assembly," *Algorithmica*, vol. 66, no. 1, pp. 153–172, 2013, ISSN: 1432-0541. DOI: `10.1007/s00453-012-9631-9`. [Online]. Available: `http://dx.doi.org/10.1007/s00453-012-9631-9`.

[71] A. Keenan, R. Schweller, and X. Zhong, "Exponential replication of patterns in the signal tile assembly model," *Proceedings of the 19th International Meeting on DNA Computing*, Sep. 2013.

[72] J. E. Padilla, M. J. Patitz, R. Pena, R. T. Schweller, N. C. Seeman, R. Sheline, S. M. Summers, and X. Zhong, "Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes," English, in *Unconventional Computation and Natural Computation*, ser. LNCS, vol. 7956, 2013, pp. 174–185, ISBN: 978-3-642-39073-9. DOI: `10.1007/978-3-642-39074-6_17`. [Online]. Available: `http://dx.doi.org/10.1007/978-3-642-39074-6_17`.

[73] R. Schweller and M. Sherman, "Fuel efficient computation in passive self-assembly," in *SODA 2013: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2013, pp. 1513–1525.

[74] E. Demaine, M. Demaine, S. Fekete, M. Patitz, R. Schweller, A. Winslow, and D. Woods, "One tile to rule them all: Simulating any tile assembly system with a single universal tile," in *Automata, Languages, and Programming*, ser. LNCS, vol. 8572, 2014, pp. 368–379, ISBN: 978-3-662-43947-0. DOI: 10.1007/978-3-662-43948-7_31. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-43948-7_31.

[75] D. Doty, "Producibility in hierarchical self-assembly," in *Proceedings of Unconventional Computation and Natural Computation (UCNC) 2014*, 2014, pp. 142–154.

[76] A. Keenan, R. Schweller, M. Sherman, and X. Zhong, "Fast arithmetic in algorithmic self-assembly," in *Unconventional Computation and Natural Computation*, ser. LNCS, vol. 8553, 2014, pp. 242–253.

[77] C. Chalk, B. Fu, A. Huerta, M. Maldonado, E. Martinez, R. Schweller, and T. Wylie, "Flipping tiles: Concentration independent coin flips in tile self-assembly," in *DNA Computing and Molecular Programming*, A. Phillips and P. Yin, Eds., ser. Lecture Notes in Computer Science, vol. 9211, Springer International Publishing, 2015, pp. 87–103.

[78] S. P. Fekete, J. Hendricks, M. J. Patitz, T. A. Rogers, and R. T. Schweller, "Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly," in *Proc. of the 25th ACM-SIAM Sym. on Discrete Algorithms*, ser. SODA'15, SIAM, 2015, pp. 148–167.

[79] S. Fekete, R. Schweller, and A. Winslow, "Size dependent tile self-assembly: Constant-height rectangles and instability," in *The 26th International Symposium on Algorithms and Computation*, 2015.

[80] T. Fochtman, J. Hendricks, J. E. Padilla, M. J. Patitz, and T. A. Rogers, "Signal transmission across tile assemblies: 3d static tiles simulate active self-assembly by 2d signal-passing tiles," *Natural Computing*, vol. 14, no. 2, pp. 251–264, 2015, ISSN: 1572-9796. DOI: 10.1007/s11047-014-9430-0. [Online]. Available: http://dx.doi.org/10.1007/s11047-014-9430-0.

[81] D. Furcy, S. Micka, and S. Summers, “Optimal program-size complexity for self-assembly at temperature 1 in 3d,” English, in *DNA Computing and Molecular Programming*, ser. Lecture Notes in Computer Science, A. Phillips and P. Yin, Eds., vol. 9211, Springer International Publishing, 2015, pp. 71–86, ISBN: 978-3-319-21998-1. DOI: 10.1007/978-3-319-21999-8_5. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-21999-8_5.

[82] J. Hendricks, M. J. Patitz, and T. A. Rogers, “Replication of arbitrary hole-free shapes via self-assembly with signal-passing tiles,” in *Unconventional Computation and Natural Computation: 14th International Conference, UCNC 2015, Auckland, New Zealand, August 30 – September 3, 2015, Proceedings*. Cham: Springer International Publishing, 2015, pp. 202–214, ISBN: 978-3-319-21819-9. DOI: 10.1007/978-3-319-21819-9_15. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-21819-9_15.

[83] A. Keenan, R. Schweller, and X. Zhong, “Exponential replication of patterns in the signal tile assembly model,” *Natural Computing*, vol. 14, no. 2, pp. 265–278, Jun. 2015. DOI: 10.1007/s11047-014-9431-z.

[84] C. Chalk, E. D. Demaine, M. L. Demaine, E. Martinez, R. Schweller, L. Vega, and T. Wylie, “Universal shape replicators via self-assembly with attractive and repulsive forces,” *ArXiv preprint arXiv:1608.00477*, 2016.

[85] C. Chalk, E. Martinez, R. Schweller, L. Vega, A. Winslow, and T. Wylie, “Optimal staged self-assembly of general shapes,” in *Proceedings of the 24th European Symposium on Algorithms (ESA)*, 2016.

[86] A. Keenan, R. Schweller, M. Sherman, and X. Zhong, “Fast arithmetic in algorithmic self-assembly,” *Natural Computing*, vol. 15, no. 1, pp. 115–128, 2016, ISSN: 1572-9796. DOI: 10.1007/s11047-015-9512-7. [Online]. Available: http://dx.doi.org/10.1007/s11047-015-9512-7.

[87] M. J. Patitz, T. A. Rogers, R. Schweller, S. M. Summers, and A. Winslow, “Resiliency to multiple nucleation in temperature-1 self-assembly,” in *DNA Computing and Molecular Programming*, Springer International Publishing, 2016.

[88] A. Winslow, “A brief tour of theoretical tile self-assembly,” in *Proceedings of the 22nd International Workshop on Cellular Automata and Discrete Complex Systems (AUTOMATA)*, ser. LNCS, vol. 9664, Springer, 2016, pp. 26–31.

[89] S. M. Summers, “Reducing tile complexity for the self-assembly of scaled shapes through temperature programming,” *Algorithmica*, to appear.

[90] M. A. Wikia, *Replicator*, `http://memory-alpha.wikia.com/wiki/Replicator`.

BIOGRAPHICAL SKETCH

Eric Martinez was born in Harlingen, TX on March 8, 1989. He attended the University of Texas-Pan American and later the University of Texas-Rio Grande Valley earning his Bachelor of Science in Computer Science degree and Master of Science in Computer Science degree in 2016. He was a research assistant and member of the Algorithmic Self-Assembly Research Group (led by Dr. Robert Schweller) and has co-authored several publications in the area of algorithmic self-assembly. Prior to graduation, he worked as a freelance web developer, marketing consultant, and advertising consultant for startups and businesses across the US.

Permanent Address: 1917 River Oaks Dr., Harlingen, TX 78552

Permanent E-mail Address: eric.michael.mtz@gmail.com