Building Squares with Optimal State Complexity in Restricted Active Self-Assembly

Robert M. Alaniz ✉
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

David Caballero ✉
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

Sonya C. Cirlos ✉
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

Timothy Gomez ✉
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

Elise Grizzell ✉
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

Andrew Rodriguez ✉
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

Robert Schweller ✉
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

Armando Tenorio ✉
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

Tim Wylie ✉
Department of Computer Science,
University of Texas Rio Grande Valley, TX, USA

Abstract

Tile Automata is a recently defined model of self-assembly that borrows many concepts from cellular automata to create active self-assembling systems where changes may be occurring within an assembly without requiring attachment. This model has been shown to be powerful, but many fundamental questions have yet to be explored. Here, we study the state complexity of assembling $n \times n$ squares in seeded Tile Automata systems where growth starts from a seed and tiles may attach one at a time, similar to the abstract Tile Assembly Model. We provide optimal bounds for three classes of seeded Tile Automata systems (all without detachment), which vary in the amount of complexity allowed in the transition rules. We show that, in general, seeded Tile Automata systems require $\Theta(\log^{\frac{1}{4}} n)$ states. For Single-Transition systems, where only one state may change in a transition rule, we show a bound of $\Theta(\log^{\frac{1}{3}} n)$, and for deterministic systems, where each pair of states may only have one associated transition rule, a bound of $\Theta((\frac{\log n}{\log \log n})^{\frac{1}{2}})$.

2012 ACM Subject Classification Theory of computation → Computational geometry; Applied computing → Computational biology; Theory of computation → Self-organization

Keywords and phrases Active Self-Assembly, State Complexity, Tile Automata

Digital Object Identifier 10.4230/LIPIcs.SAND.2022.6

Supplementary Material *Software (Source Code)*: `https://github.com/asarg/AutoTile` archived at `swh:1:dir:fd83de54cc0e347b80c90911b19bd8e0266a5bc8`

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

Acknowledgements We would like to thank the reviewers for their comments, specifically for pointing us toward relevant Cellular Automata Literature.

1 Introduction

Self-assembly is the process by which simple elements in a system organize themselves into more complex structures based on a set of rules that govern their interactions. These types of systems occur naturally and can be easily constructed artificially to offer many advantages when building micro or nanoscale objects. One abstraction of these systems that has yielded interesting results is Tile Self-Assembly.

1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022).
Editors: James Aspnes and Othon Michail; Article No. 6; pp. 6:1–6:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the abstract Tile Assembly Model (aTAM) [35], the elements of a system are represented using labeled unit squares called tiles. A system is initialized with a seed (a tile or assembly) that grows as other single tiles attach until there are no more valid attachments. The behavior of a system can then be programmed, using the interactions of tiles, and is known to be capable of Turing Computation [35], is Intrinsically Universal [14], and can assemble general scaled shapes [33]. However, many of these results utilize a concept called *cooperative binding*, where a tile must attach to an assembly using the interaction from two other tiles. Unlike with cooperative binding, the non-cooperative aTAM is not Intrinsically Universal [25, 27] and more recent work has shown that it is not capable of Turing Computation [26]. Many extensions of this model increase the power of non-cooperative systems [4, 16, 18, 22, 23, 30].

One recent model of self-assembly is Tile Automata [8]. This model marries the concept of state changes from Cellular Automata [19, 28, 37] and the assembly process from the 2-Handed Assembly model (2HAM) [6]. Previous work [3, 7, 8] has explored Tile Automata as a unifying model for comparing the relative powers of the many different Tile Assembly models. The complexity of verifying the behavior of systems along with their computational power was studied in [5]. Many of these works impose additional experimentally motivated limitations on the Tile Automata model that help connect the model and its capabilities to potential molecular implementations, such as using DNA assemblies with sensors to assemble larger structures [21], building spacial localized circuits on DNA origami [10], or DNA walkers that sort cargo [34].

In this paper, we explore the aTAM generalized with state changes; we define our producible assemblies as what can be grown by attaching tiles one at a time to a seed tile or performing transition rules, which we refer to as seeded Tile Automata. This is a bounded version of Asynchronous Cellular Automata [15]. Reachability problems, which are similar to verification problems in self-assembly, have been studied with many completeness results [13]. Further, the freezing property used in this and previous work also exists in Cellular Automata [20, 29].[1] Freezing is defined differently in Cellular Automata by requiring that there exists an ordering to the states.

While Tile Automata has many possible metrics, we focus on the number of states needed to uniquely assemble $n \times n$ squares at the smallest constant temperature, $\tau = 1$. We achieve optimal bounds in three versions of the model with varying restrictions on the transition rules. Our results, along with previous results in the aTAM, are outlined in Table 1.

1.1 Previous Work

In the aTAM, the number of tile types needed, for nearly all n, to construct an $n \times n$ square is $\Theta(\frac{\log n}{\log n \log n})$ [1, 31] with temperature $\tau = 2$ (row 2 of Table 1). The same lower bounds hold for $\tau = 1$ (row 1 of Table 1). The run time of this system was also shown to be optimal $\Theta(n)$ [1]. Other bounds for building rectangles were shown in [2]. While no tighter bounds[2] have been shown for $n \times n$ squares at $\tau = 1$ in the aTAM, generalizations to the model that allow (just-barely) 3D growth have shown an upper bound of $\mathcal{O}(\log n)$ for tile types needed [11]. Recent work in [17] shows improved upper and lower bounds on building thin rectangles in the case of $\tau = 1$ and in (just-barely) 3D.

Other models of self-assembly have also been shown to have a smaller tile complexity, such as the staged assembly model [9, 12] and temperature programming [24]. Investigation into different active self-assembly models have also explored the run time of systems [32, 36].

[1] We would like to thank a reviewer for bringing these works to our attention.
[2] Other than trivial $\mathcal{O}(n)$ bounds.

Table 1 Bounds on the number of states for $n \times n$ squares in the Abstract Tile Assembly model, with and without cooperative binding, and the seeded Tile Automata model with our transition rules. ST stands for Single-Transition.

Model	τ	$n \times n$ Squares		
		Lower	Upper	Theorem
aTAM	1	$\Omega(\frac{\log n}{\log\log n})$	$\mathcal{O}(n)$	[31], [1]
aTAM	2	$\Theta(\frac{\log n}{\log\log n})$		[31], [1]
Flexible Glue aTAM	2	$\Theta(\log^{\frac{1}{2}} n)$		[2]
Seeded TA Det.	1	$\Theta((\frac{\log n}{\log\log n})^{\frac{1}{2}})$		Thm. 2, 12
Seeded TA ST	1	$\Theta(\log^{\frac{1}{3}} n)$		Thm. 4, 12
Seeded TA	1	$\Theta(\log^{\frac{1}{4}} n)$		Thm. 3, 12

1.2 Our Contributions

In this work, we explore building an important benchmark shape, squares, in non-cooperative seeded Tile Automata. We also consider only affinity-strengthening transition rules that remove the ability for an assembly to break apart. Our results are shown in Table 1.

We start in Section 3 by proving lower bounds for building $n \times n$ squares based on three different transition rule restrictions. The first is nondeterministic or general seeded Tile Automata, where there are no restrictions and a pair of states may have multiple transition rules. The second is Single-Transition rules where only one tile may change states in a transition rule, but we still allow multiple rules for each pair of states. The last restriction, Deterministic, is the most restrictive where each pair of states may only have one transition rule (for each direction).

In Section 4, we use Transition Rules to optimally encode strings in the various versions of the model. We use these encodings as gadgets to seed the future constructions. We show how to build optimal state complexity rectangles in Section 5, and finally optimal state complexity squares in Section 6. Future work is discussed in Section 7.

AutoTile. To test our constructions, we developed AutoTile, a seeded Tile Automata simulator. Each system discussed in the paper is currently available for simulation. AutoTile is available at `https://github.com/asarg/AutoTile`.

2 Definitions

The Tile Automata model differs quite a bit from normal self-assembly models since a *tile* may change *state*, which draws inspiration from Cellular Automata. Thus, there are two aspects of a TA system being: the self-assembling that may occur with tiles in a state and the changes to the states once they have attached to each other. To address these aspects, we define the building blocks and interactions, and then the definitions around the model and what it may assemble or output. Finally, since we are looking at a limited TA system, we also define specific limitations and variations of the model. For reference, an example system is shown in Figure 1.

2.1 Building Blocks

The basic definitions of all self-assembly models include the concepts of tiles, some method of attachment, and the concept of aggregation into larger assemblies. The Cellular Automata aspect also brings in the concept of transitions.

Tiles. Let Σ be a set of *states* or symbols. A tile $t = (\sigma, p)$ is a non-rotatable unit square placed at point $p \in \mathbb{Z}^2$ and has a state of $\sigma \in \Sigma$.

Affinity Function. An *affinity function* Π over a set of states Σ takes an ordered pair of states $(\sigma_1, \sigma_2) \in \Sigma \times \Sigma$ and an orientation $d \in D$, where $D = \{\perp, \vdash\}$, and outputs an element of $\mathbb{Z}^0$. The orientation d is the relative position to each other with $\vdash$ meaning horizontal and $\perp$ meaning vertical, with the σ_1 being the west or north state respectively. We refer to the output as the *Affinity Strength* between these two states.

Transition Rules. A *Transition Rule* consists of two ordered pairs of states $(\sigma_1, \sigma_2), (\sigma_3, \sigma_4)$ and an orientation $d \in D$, where $D = \{\perp, \vdash\}$. This denotes that if the states (σ_1, σ_2) are next to each other in orientation d (σ_1 as the west/north state) they may be replaced by the states (σ_3, σ_4).

Assembly. An assembly A is a set of tiles with states in Σ such that for every pair of tiles $t_1 = (\sigma_1, p_1), t_2 = (\sigma_2, p_2)$, $p_1 \neq p_2$. Informally, each position contains at most one tile. Further, we say assemblies are equal in regards to translation. Two assemblies A_1 and A_2 are equal if there exists a vector $\vec{v}$ such that $A_1 = A_2 + \vec{v}$.

Let $B_G(A)$ be the bond graph formed by taking a node for each tile in A and adding an edge between neighboring tiles $t_1 = (\sigma_1, p_1)$ and $t_2 = (\sigma_2, p_2)$ with a weight equal to $\Pi(\sigma_1, \sigma_2)$. We say an assembly A is τ-stable for some $\tau \in \mathbb{Z}^0$ if the minimum cut through $B_G(A)$ is greater than or equal to τ.

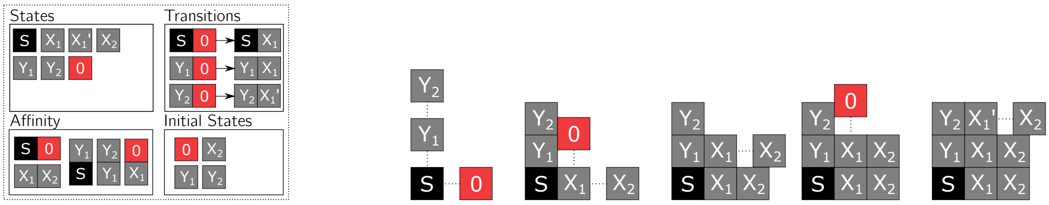
2.2 The Tile Automata Model

Here, we define and investigate the *Seeded Tile Automata* model, which differs by only allowing single tile attachments to a growing seed similar to the aTAM.

Seeded Tile Automata. A Seeded Tile Automata system is a 6-tuple $\Gamma = \{\Sigma, \Lambda, \Pi, \Delta, s, \tau\}$ where Σ is a set of states, $\Lambda \subseteq \Sigma$ a set of *initial states*, Π is an *affinity function*, Δ is a set of *transition rules*, s is a stable assembly called the *seed* assembly, and τ is the *temperature* (or threshold). Our results use the most restrictive version of this model where s is a single tile.

Attachment Step. A tile $t = (\sigma, p)$ may attach to an assembly A at temperature τ to build an assembly $A' = A \bigcup t$ if A' is τ-stable and $\sigma \in \Lambda$. We denote this as $A \rightarrow_{\Lambda,\tau} A'$.

Transition Step. An assembly A is transitionable to an assembly A' if there exists two neighboring tiles $t_1 = (\sigma_1, p_1), t_2 = (\sigma_2, p_2) \in A$ (where t_1 is the west or north tile) such that there exists a transition rule in Δ with the first pair being (σ_1, σ_2) and $A' = (A \setminus \{t_1, t_2\}) \bigcup \{t_3 = (\sigma_3, p_1), t_4 = (\sigma_4, p_2)\}$. We denote this as $A \rightarrow_{\Delta} A'$.



(a) **(b)**

Figure 1 (a) Example of a Tile Automata system, it should be noted that $\tau = 1$ and state S is our seed. (b) A walkthrough of our example Tile Automata system building the 3×3 square it uniquely produces. We use dotted lines throughout our paper to represent tiles attaching to one another.

Producibles. We refer to both attachment steps and transition steps as production steps, we define $A \to_* A'$ as the transitive closure of $A \to_{\Lambda,\tau} A'$ and $A \to_\Delta A'$. The set of *producible assemblies* for a Tile Automata system $\Gamma = \{\Sigma, \Lambda, \Pi, \Delta, s, \tau\}$ is written as $PROD(\Gamma)$. We define $PROD(\Gamma)$ recursively as follows,

- $s \in PROD(\Gamma)$
- $A' \in PROD(\Gamma)$ if $\exists A \in PROD(\Gamma)$ such that $A \to_{\Lambda,\tau} A'$.
- $A' \in PROD(\Gamma)$ if $\exists A \in PROD(\Gamma)$ such that $A \to_\Delta A'$.

Terminal Assemblies. The set of terminal assemblies for a Tile Automata system $\Gamma = \{\Sigma, \Lambda, \Pi, \Delta, \tau\}$ is written as $TERM(\Gamma)$. This is the set of assemblies that cannot grow or transition any further. Formally, an assembly $A \in TERM(\Gamma)$ if $A \in PROD(\Gamma)$ and there does not exists any assembly $A' \in PROD(\Gamma)$ such that $A \to_{\Lambda,\tau} A'$ or $A \to_\Delta A'$. A Tile Automata system $\Gamma = \{\Sigma, \Lambda, \Pi, \Delta, s, \tau\}$ *uniquely* assembles an assembly A if $A \in TERM(\Gamma)$, and for all $A' \in PROD(\Gamma), A' \to_* A$.

2.3 Limited Model Reference

We explore an extremely limited version of seeded TA that is affinity-strengthening, freezing, and may be a single-transition system. We investigate both deterministic and nondeterministic versions of this model.

Affinity Strengthening. We only consider transitions rules that are affinity strengthening, meaning for each transition rule $((\sigma_1, \sigma_2), (\sigma_3, \sigma_4), d)$, the bond between (σ_3, σ_4) must be at least the strength of (σ_1, σ_2). Formally, $\Pi(\sigma_3, \sigma_4, d) \geq \Pi(\sigma_1, \sigma_2, d)$. This ensures that transitions may not induce cuts in the bond graph.

In the case of non-cooperative systems ($\tau = 1$), the affinity strength between states is always 1 so we may refer to the affinity function as an affinity set Λ_s, where each affinity is a 3-pule (σ_1, σ_2, d).

Freezing. Freezing systems were introduced with Tile Automata. A freezing system simply means that a tile may transition to any state only once. Thus, if a tile is in state A and transitions to another state, it is not allowed to ever transition back to A.

Deterministic vs. Nondeterministic. For clarification, a deterministic system in TA has only one possible production step at a time, whether that be an attachment or a state transition. A nondeterministic system may have many possible production steps and any choice may be taken.

Single-Transition System. We restrict our TA system to only use single-transition rules. This means that for each transition rule one of the states may change, but not both. It should be noted that we still allow Nondeterminism in this system.

3 State Space Lower Bounds

Let $p(n)$ be a function from the positive integers to the set $\{0, 1\}$, informally termed a *proposition*, where 0 denotes the proposition being false and 1 denotes the proposition being true. We say a proposition $p(n)$ holds for *almost all* n if $\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} p(i) = 1$.

▶ **Lemma 1.** *Let U be a set of TA systems, b be a one-to-one function mapping each element of U to a string of bits, and ϵ a real number from $0 < \epsilon < 1$. Then for almost all integers n, any TA system $\Gamma \in U$ that uniquely assembles either an $n \times n$ square or a $1 \times n$ line has a bit-string of length $|b(\Gamma)| \geq (1-\epsilon)\log n$.*

Proof. For a given $i \geq 1$, let $M_i \in U$ denote the TA system in U with the minimum value $|b(M_i)|$ over all systems in U that uniquely assembly an $i \times i$ square or $1 \times i$ line, and let M_i be undefined if no such system in U builds such a shape. Let $p(i)$ be the proposition that $|b(M_i)| \geq (1-\epsilon)\log i$. We show that $\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} p(i) = 1$. Let $R_n = \{M_i | 1 \leq i \leq n, |b(M_i)| < (1-\epsilon)\log n\}$. Note that $n - |R_n| \leq \sum_{i=1}^{n} p(i)$. By the pigeon-hole principle, $|R_n| \leq 2^{(1-\epsilon)\log n} = n^{(1-\epsilon)}$. Therefore,

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} p(i) \geq \lim_{n\to\infty} \frac{1}{n}(n - |R_n|) \geq \lim_{n\to\infty} \frac{1}{n}(n - n^{1-\epsilon}) = 1.$$ ◀

▶ **Theorem 2** (Deterministic TA). *For almost all n, any Deterministic Tile Automata system that uniquely assembles either a $1 \times n$ line or an $n \times n$ square contains $\Omega(\frac{\log n}{\log\log n})^{\frac{1}{2}}$ states.*

Proof. We can create a one-to-one mapping $b(\Gamma)$ from any deterministic TA system to bit-strings in the following way. Let S denote the set of states in a given system. We encode the state set in $\mathcal{O}(\log |S|)$ bits, we encode the affinity function in a $|S| \times |S|$ table of strengths in $\mathcal{O}(|S|^2)$ bits (assuming a constant bound on bonding thresholds), and we encode the rules of the system in an $|S| \times |S|$ table mapping pairs of rules to their unique new pair of rules using $\mathcal{O}(|S|^2 \log |S|)$ bits, for a total of $\mathcal{O}(|S|^2 \log |S|)$ bits to encode any $|S|$ state system.

Let Γ_n denote the smallest state system that uniquely assembles an $n \times n$ square (or similarly a $1 \times n$ line), and let S_n denote the state set. By Lemma 1, $|b(\Gamma_n)| \geq (1-\epsilon)\log n$ for almost all n, and so $|S_n|^2 \log |S_n| = \Omega(\log n)$ for almost all n. We know that $|S_n| = \mathcal{O}(\log n)$, so for some constant c, $|S_n| \geq c(\frac{\log n}{\log\log n})^{\frac{1}{2}}$ for almost all n. ◀

▶ **Theorem 3** (Nondeterministic TA). *For almost all n, any Tile Automata system (in particular any Nondeterministic system) that uniquely assembles either a $1 \times n$ line or an $n \times n$ square contains $\Omega(\log^{\frac{1}{4}} n)$ states.*

▶ **Theorem 4** (Single-Transition TA). *For almost all n, any Single-Transition Tile Automata system that uniquely assembles either a $1 \times n$ line or an $n \times n$ square contains $\Omega(\log^{\frac{1}{3}} n)$ states.*

4 String Unpacking

A key tool in our constructions is the ability to build strings efficiently. We do so by encoding the string in the transition rules.

▶ **Definition 5** (String Representation). *An assembly A over states Σ represent a string S over a set of symbols U if there exists a mapping from the elements of U to the elements of Σ and a $1 \times |S|$ (or $|S| \times 1$) subassembly $A' \sqsubset A$, such that the state of the i^{th} tile of A' maps to the i^{th} symbol of S for all $0 \leq i \leq |S|$.*

4.1 Deterministic Transitions

We start by showing how to encode a binary string of length n in a set of (freezing) transition rules that take place on a $2 \times (n+2)$ rectangle that will print the string on its right side. We extend this construction to work for an arbitrary base string.

4.1.1 Overview

Consider a system that builds a length n string. First, we create a rectangle of index states that is two wide as seen on the left side of Figure 5c. Each row has a unique pair of index states so each bit of the string is uniquely indexed. We divide the index states into two groups based on which column they are in, and which "digit" they represent. Let $r = \lceil n^{\frac{1}{2}} \rceil$. Starting with index states A_0 and B_0, we build a counter pattern with base r. We use $\mathcal{O}(n^{\frac{1}{2}})$ states shown in Figure 2 to build this pattern. We encode each bit of the string in a transition rule between the two states that index that bit. A table with these transition rules can be seen in Figure 5b.

The pattern is built in r sections of size $2 \times r$ with the first section growing off of the seed. The tile in state S_A is the seed. There is also a state S_B that has affinity for the right side of S_A. The building process is defined in the following steps for each section.

1. The states $S_B, 0_B, 1_B, \ldots, (r-1_B)$ grow off of S_B, forming the right column of the section. The last B state allows for a' to attach on its west side. a tiles attach below a' and below itself. This places a states in a row south toward the state S_A, depicted in Figure 3b.
2. Once a section is built, the states begin to follow their transition rules shown in Figure 4a. The a state transitions with seed state S_A to begin indexing the A column by changing state a to state 0_A. For $1 \leq y \leq n-2$, state a vertically transitions with the other y'_A states, incrementing the index by changing from state a to state $(y+1)_A$.
3. This new index state z_A propagates up by transitioning the a tiles to the state z_A as well. Once the z_A state reaches a' at the top of the column, it transitions a' to the state z'_A. Figure 4b presents this process of indexing the A column.
4. If $z < n-1$, there is a horizontal transition rule from states $(z'_A, n-1_B)$ to states $(z'_A, n-1'_B)$. The state 0_B attaches to the north of $n-1_B$ and starts the next section. If $z = n$, there does not exist a transition.
5. This creates an assembly with a unique state pair in each row as seen in the first column of Figure 5c.

4.1.2 States

An example system with the states required to print a length-9 string are shown in Figure 2. The first states build the seed row of the assembly. The seed tile has the state S_A with initial tiles in state S_B. The index states are divided into two groups. The first set of index states, which we call the A index states, are used to build the left column. For each i, $0 \leq i < r$, we have the states i_A and i'_A. There are two states a and a', which exist as initial tiles and act as "blank" states that can transition to the other A states. The second set of index states are the B states. Again, we have r B states numbered from 0 to $r-1$, however, we do not

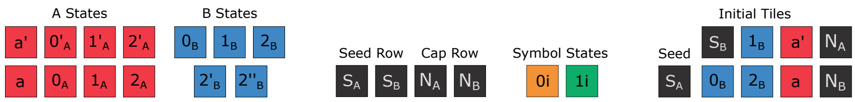
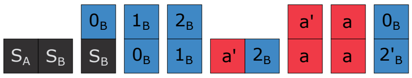
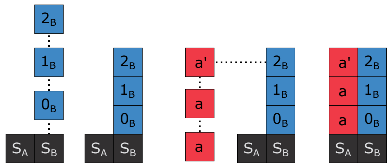


Figure 2 States to build a length-9 string in deterministic Tile Automata.



(a) Affinity Rules for Initial Tiles.



(b) Process of Building a section.

Figure 3 (a) Affinity rules to build each section. We only show affinity rules that are actually used in our system for initial tiles to attach, while our system would have more rules in order to meet the affinity strengthening restriction. (b) The B column attaches above the state S_B as shown by the dotted lines. The a' attaches to the left of 2_B and the other a states may attach below it until they reach S_A.

have a prime for each state. Instead, there are two states $r-1'_B$ and $r-1''_B$, that are used to control the growth of the next column and the printing of the strings. The last states are the symbol states 0_S and 1_S, the states that represent the string.

4.1.3 Affinity Rules/Placing Section

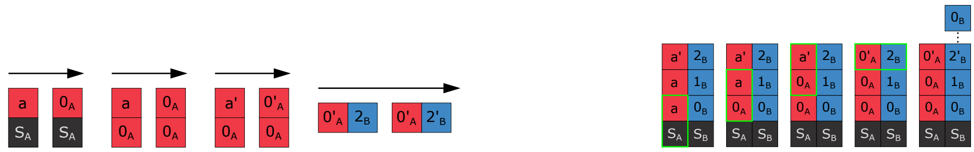
Here, we describe the affinity rules for building the first section. We later describe how this is generalized to the other $r-1$ sections. We walk through this process in Figure 3b. To begin, the B states attach in sequence above the tile S_B in the seed row. Assuming $r^2 = n$, n is a perfect square, the first state to attach is 0_B. 1_B attaches above this tile and so on. The last B state $r-1_B$ does not have affinity with 0_B, so the column stops growing. However, the state a' has affinity on the left of $r-1_B$ and can attach. a has affinity for the south side of a', so it attaches below. The a state also has a vertical affinity with itself. This grows the A column southward toward the seed row.

If n is not a perfect square, we start the index state pattern at a different value. We do so by finding the value $q = r^2 - n$. In general, the state i_B attaches above S_B for $i = q\%r$.

4.1.4 Transition Rules/Indexing A column

Once the A column is complete and the last A state is placed above the seed, it transitions with S_A to 0_A (assuming $r^2 = n$). A has a vertical transition rule with i_A ($0 \leq i < r$) changing the state A to state i_A. This can be seen in Figure 4a, where the 0_A state is propagated upward to the A' state. The A' state also transitions when 0_A is below it, going from state A' to state $0'_A$. If n is not a perfect square, then A transitions to i_A for $i = \lfloor q/r \rfloor$.

Once the transition rules have finished indexing the A column if $i < r-1$, the last state i'_A transitions with $r-1_B$ changing the state $r-1_B$ to $r-1'_B$. This transition can be seen in Figure 4b. The new state $r-1'_B$ has an affinity rule allowing 0_B to attach above it allowing the next section to be built. When the state A is above a state j'_A, $0 \leq j < r-1$, it transitions with that state changing from state A to $j+1_A$, which increments the A index.



(a) Transition Rules to Index the first section. **(b)** Process of Indexing A column.

Figure 4 (a) The first transition rule used is takes place between the seed S_A and the a state changing to 0_A. The state 0_A changes the states north of it to 0_A or $0'_A$. Finally, the state $0'_A$ transitions with 2_B (b) Once the a states reach the seed row they transition with the state S_A to go to 0_A. This state propagates upward to the top of the section.

4.1.5 Look up

After creating a $2 \times (n+2)$ rectangle, we can encode a length n string S into the transitions rules. Note that each row of our assembly consists of a unique pair of index states, which we call a *bit gadget*. Each bit gadget will *look up* a specific bit of our string and transition the B tile to a state representing the value of that bit.

Figure 5b shows how to encode a string S in a table with two columns using r digits to index each bit. From this encoding, we create our transition rules. Consider the k^{th} bit of S (where the 0^{th} bit is the least significant bit) for $k = ir + j$. Add transition rules between the states i_A and j_B, changing the state j_B to either 0_S or 1_S based on the k^{th} bit of S. This transition rule is slightly different for the northmost row of each section as the state in the A column is i'_A. Also, we do not want the state in the B column, $r-1_B$, to prematurely transition to a symbol state. Thus, we have the two states $r-1'_B$ and $r-1''_B$. As mentioned, once the A column finishes indexing, it changes the state $r-1_B$ to state $r-1'_B$, allowing for 0_B to attach above it, which starts the next column. Once the state 0_B (or a symbol state) is above $r-1'_B$, there are no longer any possible undesired attachments, so the state transitions to $r-1''_B$, which has the transition to the symbol state.

The last section has a slightly different process as $r-1_B$ state will never have a 0_B attach above it, so we have a different transition rule. This alternate process is shown in Figure 5a. The state $r-1'_A$ has a vertical affinity with the cap state N_A. This state allows N_B to attach on its right side. This state transitions with $r-1_B$ below it, changing it directly to $r-1''_B$, allowing the symbol state to print.

▶ **Theorem 6.** *For any binary string s with length $n > 0$, there exists a freezing tile automata system Γ_s with deterministic transition rules, that uniquely assembles an $2 \times (n+2)$ assembly A_S that represents S with $\mathcal{O}(n^{\frac{1}{2}})$ states.*

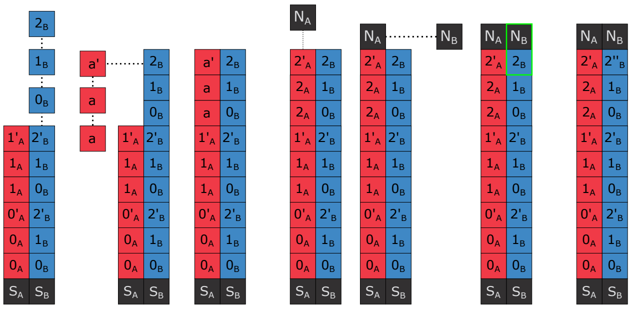
4.1.6 Arbitrary Base

In order to optimally build rectangles, we first print arbitrary base strings. Here, we show how to generalize Theorem 6 to print base-b strings.

▶ **Corollary 7.** *For any base-b string S with length $n > 0$, there exists a freezing tile automata system Γ with deterministic transition rules, that uniquely assembles an $(n+2) \times 2$ assembly which represents S with $\mathcal{O}(n^{\frac{1}{2}} + b)$ states.*

4.2 Nondeterministic Single-Transition Systems

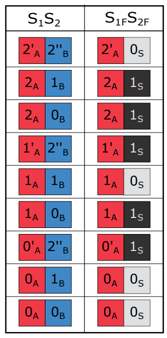
For the case of Single-Transition systems, we use the same method from above but instead building bit gadgets that are of size 3×2. Expanding to 3 columns allows for a third index digit to be used giving us an upper bound of $\mathcal{O}(n^{\frac{1}{3}})$. The second row will be used for error



(a) Attaching Cap Row.

A	**B**	S
2	2	0
2	1	1
2	0	1
1	2	1
1	1	0
1	0	1
0	2	1
0	1	0
0	0	0

(b) Encoding of S.



(c) Transition Rules.

Figure 5 (a) Once the last section finishes building the state N_A attaches above $2'_A$. N_B then attaches to the assembly and transitions with 2_B changing it directly to $2''_B$ so the string may begin printing. (b) A table indexing the string $S = 011101100$ using two columns and base $|S|^{\frac{1}{2}}$. (c) Transition Rules to print S. We build an assembly where each row has a unique pair of index states in ascending order.

checking which we will describe later in the section. This system utilizes Nondeterministic transitions, (two states may have multiple rules with the same orientation) and is non-freezing (a tile may repeat states). This system also contains cycles in its production graph, this implies the system may run indefinitely. We conjecture this system has a polynomial run time. Here, let $r = \lceil n^{\frac{1}{3}} \rceil$.

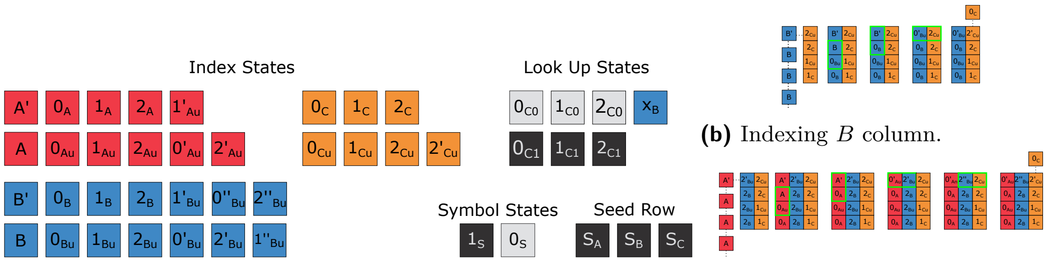
4.2.1 Index States and Look Up States

We generalize the method from above to start from a C column. The B column now behaves as the second index of the pattern and is built using B' and B as the A column was in the previous system. Once the B reaches the seed row, it is indexed with its starting value. This construction also requires bit gadgets of height 2, so we will use index states i_A, i_B, i_C and north index states i_{Au}, i_{Bu}, i_{Cu} for $0 \leq i < r$. This allows us to separate the two functions of the bit gadget into each row. The north row has transition rules to control the building of each section. The bottom row has transition rules that encode the represented bit.

In addition to the index states, we use $2r$ look up states, 0_{Ci} and 1_{Ci} for $0 \leq i < r$. These states are used as intermediate states during the look up. The first number (0 or 1) represents the value of the retrieved bit, while the second number represents the C index of the bit. The A and B indices of the bit will be represented by the other states in the transition rule.

In the same way as the previous construction, we build the rightmost column first. We include the C index states as initial states and allow 0_C to attach above S_C. We include affinity rules to build the column northwards as follows starting with the southmost state $0_C, 0_{Cu}, 1_C, 1_{Cu}, \ldots, r-2_{Cu}, r-1_C, r-1_{Cu}$.

To build the other columns, the state b' can attach on the left of $r-1_{Cu}$. The state b is an initial state and attaches below b' and itself to grow downward toward the seed row. The state b transitions with the seed row as in the previous construction to start the column. However, we alternate between C states and Cu states. The state b above i_C transitions b to i_{Cu}. If b is above i_{Cu} it transitions to i_C. The state b' above state i_B transitions to i'_{Bu}. If $i < r-1$, the state i'_B and $r-1_{Cu}$ transition horizontally changing $r-1'_{Cu}$, which allows 0_C to attach above it to repeat the process. This is shown in Figure 6b.



(a) States space for when $|S| = 27$.

(b) Indexing B column.

(c) Indexing the A column.

Figure 6 (a) States needed to construct a length 27 string where $r = 3$. (b) The index 0 propagates upward by transitioning the tiles in the column to 0_B and 0_{Bu} and transitions a' to $0'_{Bu}$. The state $0'_{Bu}$ transitions with the state 2_{Cu}, changing the state 2_{Cu} to $2'_{Cu}$, which has affinity with 0_C to build the next section. These rules also exist for the index 1. (c) When the index state 2_B reaches the top of the section, it transitions b' to $2'_{Bu}$. This state does not transition with the C column and instead has affinity with the state a', which builds the A column downward. The index propagates up the A column in the same way as the B column. When the index state 0_A reaches the top of the section, it transitions the state $2'_B$ to $2''_B$. This state transitions with 2_{Cu} changing it to $2'_{Cu}$ allowing the column to grow.

The state a' attaches on the left of $r - 1_{Cu}$. The A column is indexed just like the B column. For $0 \leq i < r - 1$, the state i'_{Au} and $r - 1'_{Bu}$ change the state $r - 1'_{Bu}$ to $r - 1''_{Bu}$. This state transitions with $r - 1_{Cu}$, changing it to $r - 1'_{Cu}$. See Figure 6c.

4.2.2 Bit Gadget Look Up

The bottom row of each bit gadget has a unique sequence of states, again we use these index states to represent the bit indexed by the digits of the states. However, since we can only transition between two tiles at a time, we must read all three states in multiple steps. These steps are outlined in Figure 7a. The first transition takes place between the states i_A and j_B. We refer to these transition rules as look up rules. We have r look up rules between these states for $0 \leq k < r$ of these states that changes the state j_B to that state k_{C0} if the bit indexed by i, j, and k is 0 or the state k_{C1} if the bit is 1.

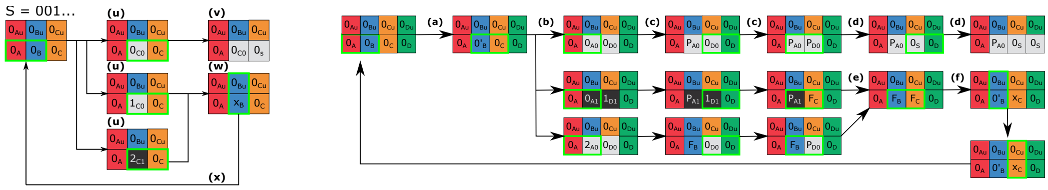
Our bit gadget has Nondeterministically looked up each bit indexed by it's A and B states, Now, we must compare the bit we just retrieved to the C index via the state in the C column. The states k_{C0} and k_C transition changing the state k_C to the $0i$ state only when they represent the same k. The same is true for the state k_{C1} except C_k transitions to $1i$.

If they both represent different k, then the state k_C goes to the state B_x. This is the error checking of our system. The B_x states transitions with the north state j_{Bu} above it transitioning B_x to j_B once again. This takes the bit gadget back to it's starting configuration and another look up can occur.

▶ **Theorem 8.** *For any binary string S with length $n > 0$, there exists a Single-Transition tile automata system Γ, that uniquely assembles an $(2n+2) \times 3$ assembly which represents S with $\mathcal{O}(n^{\frac{1}{3}})$ states.*

4.3 General Nondeterministic Transitions

Using a similar method to the previous sections, we build length n strings using $\mathcal{O}(n^{\frac{1}{4}})$ states. We start by building a pattern of index states with bit gadgets of height 2 and width 4.



(a) ST Bit Gadget look up. **(b)** Nondeterminstic Bit Gadget look up.

Figure 7 (a.u) For a string S, where the first 3 bits are 001, the states 0_A and 0_B have $|S|^{\frac{1}{3}}$ transition rules changing the state 0_B to a state representing one of the first $|S|^{\frac{1}{3}}$ bits. The state is i_{C0} if the i^{th} bit is 0 or i_{C1} if the i^{th} bit is 1 (a.v) The state 0_{C0} and the state 0_C both represent the same C index so the 0_C state transition to the 0_s. (a.w) For all states not matching the index of 0_C, they transition to x_B, which can be seen as a blank B state. (a.x) The state 0_{Bu} transitions with the state x_B changing to 0_B resetting the bit gadget. (b.a) Once the state A_0 appears in the bit gadget it transitions with 0_B changing 0_B to $0'_B$. (b.b) The states $0'_B$ and 0_C Nondeterminstically look up bits with matching B and C indices. The state $0'_B$ transitions to look up state representing the bit retrieved and the bit's A index. The state 0_C transitions to a look up state representing the D index of the retrieved bit. (b.c) The look-up states transition with the states 0_A and 0_D, respectively. As with the Single-Transition construction these may pass or fail. (b.d) When both tests pass, they transition the D look up state to a symbol state that propagates out. (b.e) If a test fails, the states both go to blank states. (b.f) The blank states then reset using the states to their north.

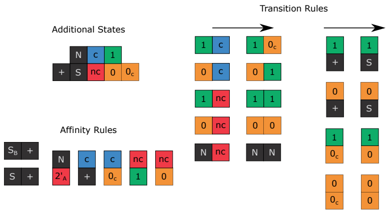
4.3.1 Overview

Here, let $r = \lceil n^{\frac{1}{4}} \rceil$. We build index states in the same way as the Single-Transition system but instead starting from the D column. We have 4 sets of index states, A, B, C, D. The same methods are used to control when the next section builds by transitioning the state $r-1_D$ to $r-1'_D$ when the current section is finished building.

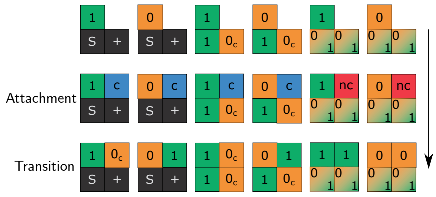
We use a similar look up method as the previous construction where we Nondeterministically retrieve a bit. However, since we are not restricting our rules to be a Single-Transition system, we may retrieve 2 indices in a single step. We include 2 sets of $\mathcal{O}(r)$ look up states, the A look up states and the D look up states. We also include Pass and Fail states $F_B, F_C, P_{A0}, P_{D0}, P_{A1}, P_{D1}$ along with the blank states B_x and C_x. We utilize the same method to build the north and south row.

Let $S(\alpha, \beta, \gamma, \delta)$ be the i^{th} bit of S where $i = \alpha r^3 + \beta r^2 + \gamma r + \delta$. The states β'_B and γ_C have r^2 transitions rules. The process of these transitions is outlined in Figure 7b. They transition from (β'_B, γ_C) to either $(\alpha_{A0}, \delta_{D0})$ if $S(\alpha, \beta, \gamma, \delta) = 0$, or $(\alpha_{A1}, \delta_{D1})$ if $S(\alpha, \beta, \gamma, \delta) = 1$. After both transitions have happened, we test if the indices match to the actual A and D indices. We include the transition rules (α_A, α_{A0}) to (α_A, P_{A0}) and (α_A, α_{A1}) to (α_A, P_{A1}). We refer to this as the bit gadget passing a test. The two states (P_{A0}, P_{D0}) horizontally transition to $(P_{A0}, 0s)$. The $0s$ state then transitions the state δ_D to $0s$ as well as propagating the state to the right side of the assembly. If the compared indices are not equal, then the test fails and the look up states will transition to the fail states F_B or F_C. These fail states will transition with the states above them, resetting the bit gadget as in the previous system.

▶ **Theorem 9.** *For any binary string S with length $n > 0$, there exists a tile automata system Γ, that uniquely assembles an $(2n+2) \times 4$ assembly which represents S with $\mathcal{O}(n^{\frac{1}{4}})$ states.*

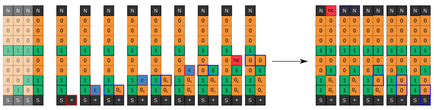


(a) New states and rules for a binary counter.

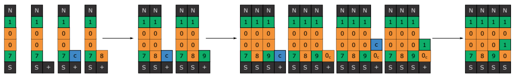


(b) Every case for the half adder.

Figure 8 (b) The 0/1 tile is not present in the system. It is used in the diagram to show that either a 0 tile or a 1 tile can take that place.



(a) Binary Counter.



(b) Base-10 Counter.

Figure 9 (a) The process of the binary counter. (b) A base-10 counter.

5 Rectangles

In this section, we will show how to use the previous constructions to build $\mathcal{O}(\log n) \times n$ rectangles. All of these constructions rely on using the previous results to encode and print a string then adding additional states and rules to build a counter.

5.1 States

We choose a string and construct a system that will create that string, using the techniques shown in the previous section. We then add states to implement a binary counter that will count up from the initial string. The states of the system, seen in Figure 8a, have two purposes. The north and south states (N and S) are the bounds of the assembly. The plus, carry, and no carry states (+, c, and nc) forward the counting. The 1, 0, and 0 with a carry state make up the number. The counting states and the number states work together as half adders to compute bits of the number.

5.2 Transition Rules / Single Tile Half Adder

As the column grows, in order to complete computing the number, each new tile attached in the current column along with its west neighbor are used in a half adder configuration to compute the next bit. Figure 8b shows the various cases for this half adder.

When a bit is going to be computed, the first step is an attachment of a carry tile or a no-carry tile (c or nc). A carry tile is attached if the previous bit has a carry, indicated by a tile with a state of plus or 0 with a carry (+ or 0c). A no-carry tile is placed if the previous bit has no-carry, indicated by a tile with a state of 0 or 1. Next, a transition needs to occur between the newly attached tile and its neighbor to the west. This transition step is the addition between the newly placed tile and the west neighbor. The neighbor does not change states, but the newly placed tile changes into a number state, 0 or 1, that either contains a carry or does not. This transition step completes the half adder cycle, and the next bit is ready to be computed.

5.3 Walls and Stopping

The computation of a column is complete when a no-carry tile is placed next to any tile with a north state. The transition rule changes the no-carry tile into a north state, preventing the column from growing any higher. The tiles in the column with a carry transition to remove the carry information, as it is no longer needed for computation. A tile with a carry changes states into a state without the carry. The next column can begin computation when the plus tile transitions into a south tile, thus allowing a new plus tile to be attached. The assembly stops growing to the right when the last column gets stuck in an unfinished state. This column, the stopping column, has carry information in every tile that is unable to transition. When a carry tile is placed next to a north tile, there is no transition rule to change the state of the carry tile, thus preventing any more growth to the right of the column.

▶ **Theorem 10.** *For all $n > 0$, there exists a Tile Automata system that uniquely assembles a $\mathcal{O}(\log n) \times n$ rectangle using,*
- *Deterministic Transition Rules and $\mathcal{O}(\log^{\frac{1}{2}} n)$ states.*
- *Single-Transition Transition Rules and $\Theta(\log^{\frac{1}{3}} n)$ states.*
- *Nondeterministic Transition Rules and $\Theta(\log^{\frac{1}{4}} n)$ states.*

5.4 Arbitrary Bases

Here, we generalize the binary counter process for arbitrary bases. The basic functionality remains the same. The digits of the number are computed one at a time going up the column. If a digit has a carry, then a carry tile attaches to the north, just like the binary counter. If a digit has no carry, then a no-carry tile is attached to the north. The half adder addition step still adds the newly placed carry or no-carry tile with the west neighbor to compute the next digit. This requires adding $\mathcal{O}(b)$ counter states to the system, where b is the base.

▶ **Theorem 11.** *For all $n > 0$, there exists a Deterministic Tile Automata system that uniquely assembles a $\mathcal{O}(\frac{\log n}{\log \log n}) \times n$ rectangle using $\Theta\left((\frac{\log n}{\log \log n})^{\frac{1}{2}}\right)$ states.*

6 Squares

In this section we utilize the rectangle constructions to build $n \times n$ squares using the optimal number of states.

Let $n' = n - 4\lceil\frac{\log n}{\log \log n}\rceil - 2$, and Γ_0 be a determinstic Tile Automata system that builds a $n' \times (4\lceil\frac{\log n}{\log \log n}\rceil + 2)$ rectangle using the process described in Theorem 11. Let Γ_1 be a copy of Γ_0 with the affinity and transition rules rotated 90 degrees clockwise, and the state labels appended with the symbol "*1". This system will have distinct states from Γ_0, and will build an equivalent rectangle rotated 90 degrees clockwise. We create two more copies of Γ_0 (Γ_2 and Γ_3), and rotate them 180 and 270 degrees, respectively. We append the state labels of Γ_2 and Γ_3 in a similar way.

We utilize the four systems described above to build a hollow border consisting of the four rectangles, and then adding additional initial states which fill in this border, creating the $n \times n$ square.

We create Γ_n, starting with system Γ_0, and adding all the states, initial states, affinity rules, and transition rules from the other systems $(\Gamma_1, \Gamma_2, \Gamma_3)$. The seed states of the other systems are added as initial states to Γ_n. We add a constant number of additional states and transition rules so that the completion of one rectangle allows for the "seeding" of the next.

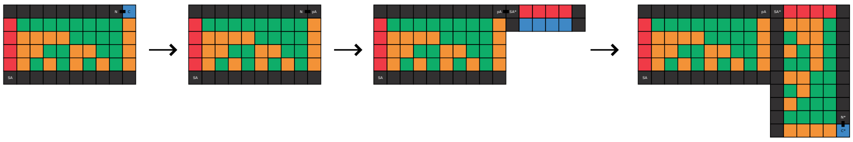


Figure 10 The transitions that take place after the first rectangle is built. The carry state transitions to a new state that allows a seed row for the second rectangle to begin growth.

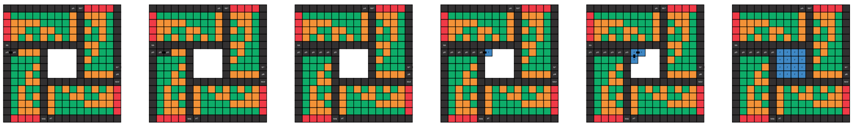


Figure 11 Once all 4 sides of the square build the pD state propagates to the center and allows the light blue tiles to fill in.

Reseeding the Next Rectangle. To Γ_n we add transition rules such that once the first rectangle (originally built by Γ_0) has built to its final width, a tile on the rightmost column of the rectangle will transition to a new state pA. pA has affinity with the state $S_A * 1$, which originally was the seed state of Γ_1. This allows state $S_A * 1$ to attach to the right side of the rectangle, "seeding" Γ_1 and allowing the next rectangle to assemble (Figure 10). The same technique is used to seed Γ_2 and Γ_3.

Filler Tiles. When the construction of the final rectangle (of Γ_3) completes, transition rules propagate a state pD towards the center of the square (Figure 11). Additionally, we add an initial state r, which has affinity with itself in every orientation, as will as with state pD on its west side. This allows the center of the square to be filled with tiles.

▶ **Theorem 12.** *For all $n > 0$, there exists a Tile Automata system that uniquely assembles an $n \times n$ square with,*

- *Deterministic transition rules and $\Theta\left((\frac{\log n}{\log\log n})^{\frac{1}{2}}\right)$ states.*
- *Single-Transition rules and $\Theta(\log^{\frac{1}{3}} n)$ states.*
- *Nondeterministic transition rules and $\Theta(\log^{\frac{1}{4}} n)$ states.*

7 Future Work

This paper showed optimal bounds for uniquely building $n \times n$ squares in three variants of seeded Tile Automata without cooperative binding. En route, we proved upper bounds for constructing strings and rectangles. Serving as a preliminary investigation into constructing shapes in this model. This leaves many open questions:

- As shown in [5], even 1D Tile Automata systems can perform Turing computation. This behavior may imply interesting results for constructing $1 \times n$ lines. We conjecture, it is possible to achieve the optimal bound of $\Theta((\frac{\log n}{\log\log n})^{\frac{1}{2}})$ with deterministic rules.
- Our rectangles had a height bounded by $\mathcal{O}(\frac{\log n}{\log\log n})$, and none fell below the $k < \frac{\log n}{\log\log n}$ [2] bound for a thin rectangle. In Tile Automata without cooperative binding, is it possible to optimally construct $k \times n$ thin rectangles?
- We allow transition rules between non-bonded tiles. Can the same results be achieved with the restriction that a transition rule can only exist between two tiles if they share an affinity in the same direction?

- While we show optimal bounds can be achieved without cooperative binding, can we simulate so-called zig-zag aTAM systems? These are a restricted version of the cooperative aTAM that is capable of Turing computation.
- We show efficient bounds for constructing strings in Tile Automata. Given the power of the model, it should be possible to build algorithmically defined shapes such as in [33] by printing Komolgorov optimal strings and inputting them to a Turing machine.

References

1 Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 740–748, 2001.

2 Gagan Aggarwal, Qi Cheng, Michael H Goldwasser, Ming-Yang Kao, Pablo Moisset De Espanes, and Robert T Schweller. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34(6):1493–1515, 2005.

3 John Calvin Alumbaugh, Joshua J. Daymude, Erik D. Demaine, Matthew J. Patitz, and Andréa W. Richa. Simulation of programmable matter systems using active tile-based self-assembly. In Chris Thachuk and Yan Liu, editors, *DNA Computing and Molecular Programming*, pages 140–158, Cham, 2019. Springer International Publishing.

4 Bahar Behsaz, Ján Maňuch, and Ladislav Stacho. Turing universality of step-wise and stage assembly at temperature 1. In Darko Stefanovic and Andrew Turberfield, editors, *DNA Computing and Molecular Programming*, pages 1–11, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

5 David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. Verification and Computation in Restricted Tile Automata. In Cody Geary and Matthew J. Patitz, editors, *26th International Conference on DNA Computing and Molecular Programming (DNA 26)*, volume 174 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.DNA.2020.10`.

6 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert T. Schweller, Scott M Summers, and Andrew Winslow. Two Hands Are Better Than One (up to constant factors): Self-Assembly In The 2HAM vs. aTAM. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 172–184. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.

7 Angel A Cantu, Austin Luchsinger, Robert Schweller, and Tim Wylie. Signal passing self-assembly simulates tile automata. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

8 Cameron Chalk, Austin Luchsinger, Eric Martinez, Robert Schweller, Andrew Winslow, and Tim Wylie. Freezing simulates non-freezing tile automata. In David Doty and Hendrik Dietz, editors, *DNA Computing and Molecular Programming*, pages 155–172, Cham, 2018. Springer International Publishing.

9 Cameron Chalk, Eric Martinez, Robert Schweller, Luis Vega, Andrew Winslow, and Tim Wylie. Optimal staged self-assembly of general shapes. *Algorithmica*, 80(4):1383–1409, 2018.

10 Gourab Chatterjee, Neil Dalchau, Richard A. Muscat, Andrew Phillips, and Georg Seelig. A spatially localized architecture for fast and modular DNA computing. *Nature Nanotechnology*, July 2017. URL: `https://www.microsoft.com/en-us/research/publication/spatially-localized-architecture-fast-modular-dna-computing/`.

11 Matthew Cook, Yunhui Fu, and Robert Schweller. Temperature 1 self-assembly: Deterministic assembly in 3d and probabilistic assembly in 2d. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 570–589. SIAM, 2011.

12 Erik D Demaine, Martin L Demaine, Sándor P Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T Schweller, and Diane L Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with o (1) glues. *Natural Computing*, 7(3):347–370, 2008.

13 Alberto Dennunzio, Enrico Formenti, Luca Manzoni, Giancarlo Mauri, and Antonio E Porreca. Computational complexity of finite asynchronous cellular automata. *Theoretical Computer Science*, 664:131–143, 2017.

14 David Doty, Jack H Lutz, Matthew J Patitz, Robert T Schweller, Scott M Summers, and Damien Woods. The tile assembly model is intrinsically universal. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 302–310. IEEE, 2012.

15 Nazim Fates. A guided tour of asynchronous cellular automata. In *International Workshop on Cellular Automata and Discrete Complex Systems*, pages 15–30. Springer, 2013.

16 Bin Fu, Matthew J Patitz, Robert T Schweller, and Robert Sheline. Self-assembly with geometric tiles. In *International Colloquium on Automata, Languages, and Programming*, pages 714–725. Springer, 2012.

17 David Furcy, Scott M. Summers, and Logan Withers. Improved Lower and Upper Bounds on the Tile Complexity of Uniquely Self-Assembling a Thin Rectangle Non-Cooperatively in 3D. In Matthew R. Lakin and Petr Šulc, editors, *27th International Conference on DNA Computing and Molecular Programming (DNA 27)*, volume 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.DNA.27.4`.

18 Oscar Gilbert, Jacob Hendricks, Matthew J Patitz, and Trent A Rogers. Computing in continuous space with self-assembling polygonal tiles. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 937–956. SIAM, 2016.

19 Eric Goles, P-E Meunier, Ivan Rapaport, and Guillaume Theyssier. Communication complexity and intrinsic universality in cellular automata. *Theoretical Computer Science*, 412(1-2):2–21, 2011.

20 Eric Goles, Nicolas Ollinger, and Guillaume Theyssier. Introducing freezing cellular automata. In *Cellular Automata and Discrete Complex Systems, 21st International Workshop (AUTOMATA 2015)*, volume 24, pages 65–73, 2015.

21 Leopold N Green, Hari KK Subramanian, Vahid Mardanlou, Jongmin Kim, Rizal F Hariadi, and Elisa Franco. Autonomous dynamic control of DNA nanostructure self-assembly. *Nature chemistry*, 11(6):510–520, 2019.

22 Daniel Hader and Matthew J Patitz. Geometric tiles and powers and limitations of geometric hindrance in self-assembly. *Natural Computing*, 20(2):243–258, 2021.

23 Jacob Hendricks, Matthew J. Patitz, Trent A. Rogers, and Scott M. Summers. The power of duples (in self-assembly): It's not so hip to be square. *Theoretical Computer Science*, 743:148–166, 2018. `doi:10.1016/j.tcs.2015.12.008`.

24 Ming-Yang Kao and Robert Schweller. Reducing tile complexity for self-assembly through temperature programming. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 571–580, USA, 2006. Society for Industrial and Applied Mathematics.

25 Pierre-Etienne Meunier, Matthew J. Patitz, Scott M. Summers, Guillaume Theyssier, Andrew Winslow, and Damien Woods. Intrinsic universality in tile self-assembly requires cooperation. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 752–771, 2014. `doi:10.1137/1.9781611973402.56`.

26 Pierre-Étienne Meunier and Damien Regnault. Directed Non-Cooperative Tile Assembly Is Decidable. In Matthew R. Lakin and Petr Šulc, editors, *27th International Conference on DNA Computing and Molecular Programming (DNA 27)*, volume 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:21, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.DNA.27.6`.

27 Pierre-Étienne Meunier and Damien Woods. The non-cooperative tile assembly model is not intrinsically universal or capable of bounded Turing machine simulation. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 328–341, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3055399.3055446.

28 Turlough Neary and Damien Woods. P-completeness of cellular automaton rule 110. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 132–143, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

29 Nicolas Ollinger and Guillaume Theyssier. Freezing, bounded-change and convergent cellular automata. *arXiv preprint*, 2019. arXiv:1908.06751.

30 Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Exact shapes and Turing universality at temperature 1 with a single negative glue. In *Proceedings of the 17th International Conference on DNA Computing and Molecular Programming*, DNA'11, pages 175–189, Berlin, Heidelberg, 2011. Springer-Verlag.

31 Paul WK Rothemund and Erik Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 459–468, 2000.

32 Nicholas Schiefer and Erik Winfree. Time complexity of computation and construction in the chemical reaction network-controlled tile assembly model. In Yannick Rondelez and Damien Woods, editors, *DNA Computing and Molecular Programming*, pages 165–182, Cham, 2016. Springer International Publishing.

33 David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007.

34 Anupama J. Thubagere, Wei Li, Robert F. Johnson, Zibo Chen, Shayan Doroudi, Yae Lim Lee, Gregory Izatt, Sarah Wittman, Niranjan Srinivas, Damien Woods, Erik Winfree, and Lulu Qian. A cargo-sorting DNA robot. *Science*, 357(6356):eaan6558, 2017. doi:10.1126/science.aan6558.

35 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

36 Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 353–354, 2013.

37 Thomas Worsch. Towards intrinsically universal asynchronous ca. *Natural Computing*, 12(4):539–550, 2013.