Simulation of Multiple Stages in Single Bin Active Tile Self-Assembly

Sonya C. Cirlos
University of Texas Rio Grande Valley
Timothy Gomez
Massachusetts Institute of Technology
Elise Grizzell
University of Texas Rio Grande Valley
Andrew Rodriguez
Texas State University
Robert Schweller
University of Texas Rio Grande Valley
Tim Wylie (timothy.wylie@utrgv.edu)
University of Texas Rio Grande Valley

Research Article

Keywords: Staged Self-assembly, Tile Automata, Context-Free Grammar, Freezing TA

Posted Date: December 21st, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3762430/v1

Additional Declarations: No competing interests reported.

Simulation of Multiple Stages in Single Bin Active Tile Self-Assembly

Sonya C. Cirlos[1], Timothy Gomez[2], Elise Grizzell[1], Andrew Rodriguez[3], Robert Schweller[1], Tim Wylie[1*†]

[1*]Department of Computer Science, University of Texas Rio Grande Valley, 1201 W. University Dr., Edinburg, TX, 78539, USA.
[2]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 32 Vassar Street, Cambridge, MA, 02139, USA.
[3]Department of Computer Science, Texas State University, San Marcos, TX, 78666, USA.

*Corresponding author(s). E-mail(s): timothy.wylie@utrgv.edu;
Contributing authors: sonya.cirlos01@utrgv.edu; tagomez7@mit.edu; elise.grizzell01@utrgv.edu; andrew.rodriguez@txstate.edu; robert.schweller@utrgv.edu;
†These authors contributed equally to this work.

Abstract

Two significant and often competing goals within the field of self-assembly are minimizing tile types and minimizing human-mediated experimental operations. The introduction of the Staged Assembly and Single Staged Assembly models, while successful in the former aim, necessitate an increase in mixing operations later. In this paper, we investigate building optimal lines as a standard benchmark shape and building primitive. We show that a restricted version of the 1D Staged Assembly Model can be simulated by the 1D Freezing Tile Automata model with the added benefits of the complete automation of stages and completion in a single bin while maintaining bin parallelism and a competitive number of states for lines, patterned lines, and context-free grammars.

Keywords: Staged Self-assembly, Tile Automata, Context-Free Grammar, Freezing TA

1 Introduction

Many molecular programmers dream of designing single-pot reactions in which system molecules do the entirety of the computational work without any necessary intervention by the experimenter. This is arguably *true* self-assembly. Yet the power of experimenter intervention, in the form of mixing and splitting pots over a sequence of stages, yields power and efficiency in both theory and practice [18] that is currently unmatched even with some of the most powerful models of active self-assembly. This paper aims to address this gap in the case of 1-dimensional (1D) assembly by showing how an abstract modeling of operations of experimental stages, termed the Staged Assembly Model (SAM) [12], can be efficiently simulated by an abstract model of single-pot *active* self-assembly, termed Tile Automata (TA) [9].

Tile Automata generalizes *passive* tile assembly models (such as the two-handed tile assembly model [7]) by giving tiles dynamic states that update based on local pairwise rules, thus making it a model of *active* self-assembly. The Staged Assembly Model (SAM) generalizes tile assembly models by the modeling of experimenter-mediated operations, including the ability to store different portions of the system particles in separate containers or *bins*, and the ability to combine separate bins or split the contents of a bin among multiple bins, over a sequence of distinct *stages*. Previous results show that both models have substantially increased power over the basic tile self-assembly models they generalize. In particular, by offloading some of the computation onto an experimenter responsible for performing the required mixing operations of the system between stages, SAM can build complex shapes and patterns in near-optimal complexity with respect to tile types, bin counts, and stage counts [10–13, 20].

In answer to the long-standing open question of whether the substantial power of the SAM could be efficiently encoded into the reaction rules of an active single-pot system, this paper shows that in the case of 1-dimensional systems, any staged system can be encoded into a single-pot TA system with a comparable state and rule space to the tiles, bins, and stages of the SAM system it simulates. This result provides a corresponding corollary in TA for any results in 1D staged self-assembly. Further, this provides a new approach for programming 1D TA systems since designing staged systems is relatively simple with strong timing guarantees based on separate bins and stages, whereas programming complex TA systems from scratch can be daunting as the single-pot nature of the system requires careful attention to race conditions. As evidence of the power of this new result, we show how several previous results in TA now become simple corollaries of this new result. Further, we show how a general linear pattern can be constructed in TA using a number of states linear in the size of the smallest context-free grammar that produces the target pattern.

1.1 Staged Self-Assembly and Tile Automata

Algorithmic self-assembly emerged from a formalization of Wang Tiles to explore self-assembling structures. Defined by Winfree in [19], this was partially motivated by new DNA techniques that allow for the creation of DNA-based 'tiles' that can assemble into lattice structures at the nanoscale [22]. Further experimental work has investigated active DNA-based components capable of complex tasks such as sorting molecules attached to a DNA origami surface [17].

The Staged Tile Assembly Model [12] generalizes the 2-Handed Assembly Model to allow growth to occur in multiple bins, mixing in a sequence described as stages, creating the capability to model experimental techniques, such as in [18] where 2D patterns are built with DNA origami tiles in multiple stages.

Tile Automata was introduced in [9] as a combination of hierarchical passive self-assembly systems and the active self-assembly of Cellular Automata systems where all *tiles* have a transitionable *state*. Affinity rules define which tiles can bond with each other based on their states and with how much strength. Starting from singleton tiles with states, any two producibles in the system may combine if there is enough affinity between adjacent tiles. Transition rules define state changes that may occur between two tiles once they are neighbors in an assembly.

Efficient line construction in Tile Automata was briefly studied in [5].

1.2 Related Work

Shape building was the first problem explored when the staged model was introduced [12]. In the staged model, a constant-sized set of glue types is sufficient to build any shape by encoding the description in the mix graph. The trade-off between the number of glues, bins, and stages was further investigated in later work with $1 \times n$, $\mathcal{O}(1) \times n$ [11], and general assemblies [10]. The complexity of verifying whether an assembly is uniquely produced is PSPACE-complete [6, 15].

A restricted class of systems in SAM, called Single Staged Assembly Systems (SSAS) in [13], requires each bin to only contain one terminal assembly built from two input assemblies. This restriction eliminates having multiple assemblies built in the same bin (*bin parallelism*). The size of the smallest SSAS that builds a 1D pattern $\mathcal{P}$ is equivalent (up to constant factors) to the size of the smallest Context-Free Grammar (CFG) that defines only $\mathcal{P}$. However, when bin parallelism is allowed, staged is more efficient than CFGs for a specific family of strings.

In [20], they built on previous results and define Polyomino Context-Free Grammars (PCFG), which generalize CFGs to 2D. The size of the smallest staged system that uniquely produces a patterned assembly is within a log factor of the smallest PCFG. In some cases, staged is much better.

One strength of Tile Automata is the possibility of being a "unifying" model, where multiple models can be connected through simulation results. The work that introduced the model [9] showed that the freezing model, where a tile may never repeat a state, simulates the non-freezing version of the model. Tile Automata was shown to simulate a model of programmable matter called Amoebots [2]. The chain of simulation was further extended in [8] where the Signal-Passing Tile Assembly Model (STAM) was shown to simulate Tile Automata. Work done in [3] shows how the 1D STAM can simulate a s stage 1D SSAS system using a single tile with $\mathcal{O}(s^4)$ glues types.

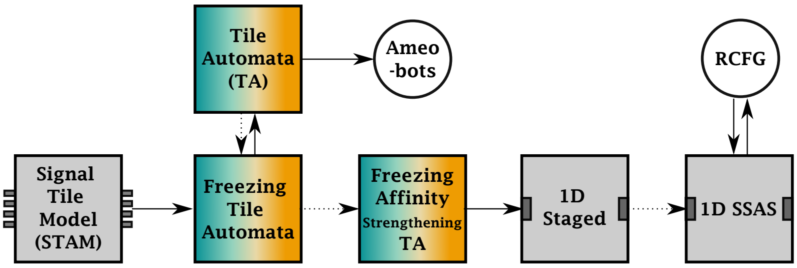


Fig. 1: Informal map of relations between models. Dotted line arrows indicate model is a special case of the previous. Solid lines indicate simulation results.

Tile Automata	**Scale**	**States**	**Theorem**
Freezing Strengthening	1	$\mathcal{O}(sbt)$	Thm.
Freezing Strengthening	2	$\mathcal{O}(sbg)$	Thm.
Strengthening	2	$\mathcal{O}(sg + bg)$	Open

Table 1: Restricted 1D Tile Automata can simulate 1D Staged model. We allow for 1D scaling. s is number of states, b is number of bins, g is the number of glues.

Aff Str	**Cycle**	**Frz.**	**Det.**	**Single**			**Double**		
Yes	Yes	No	ND	$\mathcal{O}(\|P\|^{\frac{1}{3}})$	2×3	[1]	$\mathcal{O}(\|P\|^{\frac{1}{4}})$	2×4	[1]
Yes	No	Yes	Det	$\mathcal{O}(\|P\|^{\frac{1}{2}})$	1×2	[1]	$\mathcal{O}(\|P\|^{\frac{1}{2}})$	1×1	[1]
No	Yes	Yes	Det	$\mathcal{O}(K_P)$	$\mathcal{O}(1) \times \mathcal{O}(1)$	[5, 8]	$\mathcal{O}(K_P)$	1×1	[5]
No	No	No	Det	$\mathcal{O}(K_P^{\frac{1}{2}})$	1×1	Thm. 4	$\mathcal{O}(K_P^{\frac{1}{2}})$	1×1	Thm. 4
Yes	No	No	Det	$\mathcal{O}(L_P^{\frac{1}{2}})$	$\mathcal{O}(1) \times 2$		$\mathcal{O}(L_P^{\frac{1}{2}})$	$\mathcal{O}(1) \times 1$	
Yes	No	Yes	ND	$\mathcal{O}(CF_P)$	1×1	Thm 2	$\mathcal{O}(CF_P)$	1×1	Thm. 2

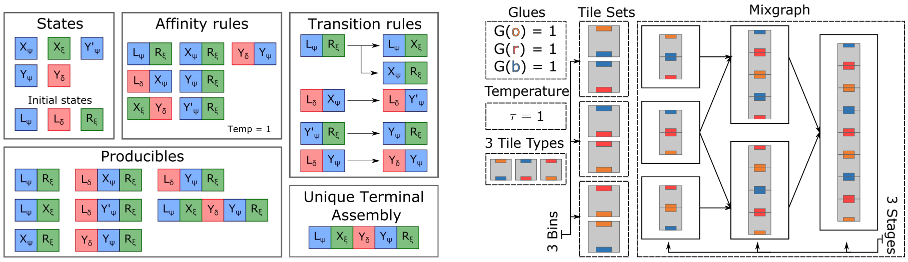
Table 2: Minimum number of states needed to construct a patterned rectangle over a constant number of colors representing the 1D pattern P in Affinity Strengthening Tile Automata with tiles not changing colors. K_P is the Kolmogorov complexity of the pattern P, CF_P is the size of the smallest Context Free Grammar that produces the singleton language $\{P\}$.

1.3 Our Contributions

We show that the 1D version of Freezing Affinity Strengthening Tile Automata can simulate the 1D staged assembly model, even with flexible glues (Section 3). The Tile Automata system uses $\mathcal{O}(sbt)$ states for a system with s stages, b bins, and t tile types.

Using this result we inherit the ability to simulate Context-Free Grammars from the staged model in [13] showing the same upper bound. For the line-building results, we inherit them from [12]. Additionally using results from [8], these results carry over to the STAM as well.

This is the full version of a paper presented at UCNC 2023. We include additional upper and lower bounds on pattern building in different versions of Tile Automata.



(a) A Tile Automata System (b) Staged Self-Assembly Example

Fig. 2: (a) An example of a Tile Automata system Γ. Recursively applying the transition rules and affinity functions to the initial assemblies of a system yields a set of producible assemblies. Any producibles that cannot combine with, break into, or transition to another assembly are considered terminal. Note that none of the transition rules allow states to change color. (b) A simple staged self-assembly example. The system has 3 bins, 3 stages, and 3 tile types, assigned to bins, as shown in the mix graph. Only terminal assemblies can pass to a successive stage. The result of this system is the assembly shown in the bin in stage 3.

The result in Section 4 is a direct version of a Context-Free Grammar simulation which works in a slighty stronger version of Tile Automata, i.e., Theorem 2 works even in the case of Deterministic Single-Transitions. We additionally include bounds on building patterns in relaxed versions of Tile Automata and these results are outlined in Table 2.

2 Model and Definitions

We provide simplified definitions for 1D Tile Automata, then define 1D Staged Assembly as a generalization. Refer to previous work [1] and [12] for full definitions of the models.

2.1 The 1D Tile Automata model (TA)

In this dimensionally restricted version of the model, a *Tile Automata system*[1] is a triple (Σ, Π, Δ) where Σ is an alphabet of state types, Π is an affinity function, and Δ is a set of transition rules for states in Σ. An example 1D Tile Automata system is shown in Figure 2.

[1]Typical TA models are defined with a temperature parameter τ however, with consideration of solely 1D, eliminating the possibility of cooperative binding, we assume $\tau = 1$.

Tile. Let Σ be a set of *states* or symbols. A tile $t = (\sigma, p)$ is a non-rotatable unit square placed at point $p \in \mathbb{Z}^1$ and has a state of $\sigma \in \Sigma$.

Assembly. An assembly A is a sequence of tiles $\{t_1, t_2, t_3, \ldots, t_{|A|}\}$. Let $A(i)$ and $A_\Sigma(i)$ represent the i^{th} tile and its state in assembly A, respectively. For a tile t in assembly A let $\rho_A(t)$ be the position of t in A.

Affinity Function. An *affinity function* Π takes an ordered pair in Σ^2 as input and outputs either 0 or 1. The *affinity strength* between two states for the ordered orientation is the binary output of the corresponding function. An assembly A is *stable* if, for every pair of tiles, $\Pi(A_\Sigma(i), A_\Sigma(i+1)) = 1$. Informally, if all adjacent tiles in assembly A have an affinity, A is stable. Two assembles, A and B are *combinable* if the concatenation of the two assemblies $AB = C$ is also a stable assembly.

Transition Rules. Transition rules allow states to change based on their neighbors. A *transition rule* is denoted $(\sigma_{1a}, \sigma_{2a}) \rightarrow (\sigma_{1b}, \sigma_{2b})$ with $\sigma_{1a}, \sigma_{2a}, \sigma_{1b}, \sigma_{2b} \in \Sigma$. If states σ_{1a} and σ_{2a} are adjacent to each other, they can transition to states σ_{1b} and σ_{2b}, respectively. An assembly A is *transitionable* to an assembly B if there exists two adjacent tiles $A(i), A(i+1) \in A$, two adjacent tiles $B(i), B(i+1) \in B$, a transition rule $(A_\Sigma(i), A_\Sigma(i+1)) \rightarrow (B_\Sigma(i), B_\Sigma(i+1)) \in \Delta$, and $A(j) = B(j)$ for all $j \neq i, i+1$.

Producibility. We define the set of producible assemblies starting from a set of initial assemblies Λ. For a given 1D Tile Automata system $\Gamma = (\Sigma, \Pi, \Delta)$ and initial assembly set Λ, the set of *producible assemblies* of Γ, denoted $\texttt{PROD}_\Gamma(\Lambda)$, is defined recursively:

- (Base) $\Lambda \subseteq \texttt{PROD}_\Gamma(\Lambda)$
- (Combinations) For any $A, B \in \texttt{PROD}_\Gamma(\Lambda)$ s.t. A and B are combinable into C, then $C \in \texttt{PROD}_\Gamma(\Lambda)$.
- (Transitions) For any $A \in \texttt{PROD}_\Gamma(\Lambda)$ s.t. A is transitionable into B using $\delta \in \Delta$, then $B \in \texttt{PROD}_\Gamma(\Lambda)$.

For a system Γ, we say $A \rightarrow_1^\Gamma B$ for assemblies A and B if A is combinable with some producible assembly to form B, if A is transitionable into B, or if $A = B$. Intuitively, this means that A may grow into assembly B through one or fewer combinations or transitions.
We define the relation $\rightarrow^\Gamma$ to be the transitive closure of $\rightarrow_1^\Gamma$, i.e., $A \rightarrow^\Gamma B$ means that A may grow into B through a sequence of combinations and transitions.

Terminal Assemblies. A producible assembly A of a Tile Automata system Γ is *terminal* provided A is not combinable with any producible assembly of Γ, and A is not transitionable to any producible assembly of Γ. Let $\texttt{TERM}_\Gamma(\Lambda) \subseteq \texttt{PROD}_\Gamma(\Lambda)$ denote the set of producible assemblies of Γ that are terminal.

Unique Assembly. A 1D TA system Γ, starting from initial assemblies Λ, *uniquely produces* a set of assemblies $\mathcal{A}$ if

- $\mathcal{A} = \texttt{TERM}_\Gamma(\Lambda)$,
- for all $B \in \texttt{PROD}_\Gamma(\Lambda)$, $B \rightarrow^\Gamma A$ for some $A \in \mathcal{A}$

2.2 Staged Assembly Model

Here, we define the Staged Assembly model using the definitions from above.

Tile Types and Glues. In the staged assembly model, tiles are defined by their glues. Let G be a set of glues. A *tile type* is an ordered pair of glues $(w, e) \in G^2$ where tile $t = (w, e)$ has west glue w and east glue e. The affinity function Π for the staged assembly model takes as input two tile types $t_1 = (a, b)$, $t_2 = (c, d)$ and outputs 1 if $b = c$ and 0 otherwise.

When allowing *Flexible Glues* we remove the restriction that Π outputs 0 when $b \neq c$ allowing for a general glue function. *Note this is equivalent to the affinity function of Tile Automata.*

Assembly. An assembly A in a staged assembly system is a sequence of tile types $\{t_1, t_2, t_3, \ldots, t_{|A|}\}$. Let $A(i)$ be the i^{th} tile type in assembly A.

Staged Assembly Systems. An *r-stage, b-bin mix-graph $M_{r,b}$*, 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 duple $\Upsilon = (M_{r,b}, T)$ where $M_{r,b}$ is an r-stage, b-bin mix-graph, $T \subset G^2$ is a set of tiles types labeled from the set of pairs of glues G.

Two-Handed Assembly and Bins We define the assembly process in terms of bins[2]. Each bin can be considered an instance of a Tile Automata system without transition rules where $\Delta = \emptyset$. However, each bin has a different set of initial assemblies denoted as $\Lambda_{i,j}$ where i is the stage and j is the bin. Let T_j be the set of initial tile types in bin j.

1. $\Lambda_{1,j} = \{T_j\}$ (this is a bin in the first stage);
2. For $i \geq 2$, $\Lambda_{i,j} = \Big(\bigcup_{k:\ (m_{i-1,k}, m_{i,j}) \in M_{r,b}} \texttt{TERM}_{\Upsilon}(\Lambda_{i-1,k}) \Big)$.

Thus, the j^{th} bin in stage 1 is provided with the initial tile set T_j. Each bin in any later stage receives an initial set of assemblies consisting of the terminally produced assemblies' bins in the previous stage indicated by the edges of the mix-graph. The *output* of the staged system is the union of all terminal assemblies from each bin in the final stage. We say this set of output assemblies is *uniquely produced* if each bin in the staged system uniquely produces its respective set of terminal assemblies.

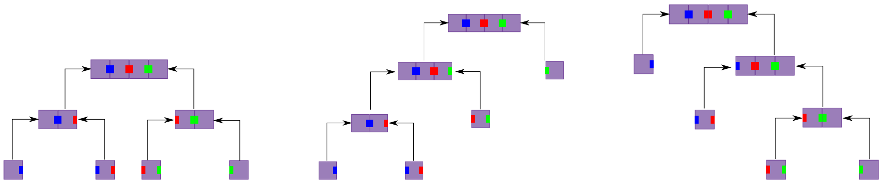
2.3 Assembly Trees

We may represent the assembly process in a single bin as an assembly tree in the staged model. An example tree can be seen in Figure 3a.

Definition 1 (Assembly Tree). *An assembly tree T_A^b, for a producible assembly A in a bin b, is a binary tree where each node represents a subassembly of A. The root represents assembly A, and each leaf represents an initial assembly of b. Each node can be formed by combining the assemblies represented by the children.*

An assembly tree is a *Left-Handed* Assembly Tree if every assembly that attaches on the right side is an initial assembly. A *Right-Handed* Assembly Tree is the inverse

[2] Each bin may be seen as an instance of the 2-Handed Assembly Model.



(a) Assembly Tree (b) Left Handed (c) Right Handed

Fig. 3: Examples of assembly trees for the same assembly. (a) A balanced tree. (b) A left-handed assembly tree. (c) A right-handed assembly tree.

where every left assembly is an initial assembly. Examples of these two types of trees are in Figures 3b and 3c.

2.4 Colors and Patterns

In this section, we augment the Tile Automata model with the concept of a tile's color being based on the current state. Colors for Staged has been defined in [13]. For a set of color labels C, this is a partition of the states into $|C|$ sets. We only consider constant-sized C. Thus, the *color* of a tile t is the partition of the tile's state, denoted as $c(t)$.

Definition 2 (Pattern)**.** *A pattern P over a set of colors C is a partial mapping of $\mathbb{Z}$ to elements in C. Let $P(z)$ be the color at $z \in \mathbb{Z}$. A scaled pattern P^{hw} is a pattern replacing each pixel within a $1 \times w$ line of pixels.*

Definition 3 (Patterned Assemblies)**.** *We say a positioned assembly A' represents a pattern P if for each tile $t \in A'$, $c(t) = P(\rho_{A'}(t))$ and $dom(A') = dom(P)$. We say a positioned assembly B' represents a pattern P at scale $h \times w$ if it represents the scaled pattern P^{hw}.*

A system Γ uniquely assembles a pattern P if it uniquely assembles an assembly A, such that A contains a positioned assembly that represents P.

2.5 Tile Automata Restrictions

Here we define the relevant restrictions of Tile Automata. All but the last has been defined in previous work [1, 5, 8, 9]

Affinity Strengthening. *Affinity Strengthening* requires that any transition preserves affinities between tiles within assemblies. For each transition rule $(\sigma_a, \sigma_b) \rightarrow (\sigma_c, \sigma_d)$, $\Pi(\sigma_c, \sigma_d) = 1$. By limiting our focus to affinity strengthening systems, we do not need to consider the scenario where a stable assembly becomes unstable (and would fall apart).

Freezing. In a freezing system, a tile may not transition to any state more than once. Thus, if a tile with state σ_a transitions into another state σ_b, it is not allowed to transition back to σ_a.

Bonded. Transitions only occur between tiles that have affinity with each other.

Single-Transition Tile Automata system. Γ is a Single-Transition Tile Automata system if for all transitions rules $(S_{1a}, S_{2a}, S_{1b}, S_{2b}, d)$ either $S_{1a} = S_{1b}$ or $S_{2a} = S_{2b}$.

Bonded, Single-Transition allows us to skip a couple steps in the simulation in the STAM from [8].

Deterministic Transition Rules. A system has deterministic transitions rules if for all pairs of states S_1, S_2 and direction $d \in \{v, h\}$ there only exists one transition rule between the states in that direction.

Color-Locked. A tile automata system is *Color-Locked* if for every transition rule $\delta = (S_{1a}, S_{2a}, S_{1b}, S_{2b}, d) \in \Delta$, $c(S_{1a}) = c(S_{1b})$ and $c(S_{2a}) = c(S_{2b})$, i.e. tiles are not allowed to change their color.

This restriction allows for transitions to be independent of the color, we can imagine this the color being inherent to the tile. These restrictions all together can model a signal tile carrying a chemical marker that cannot change, and transitions only expose more binding sites.

3 Simulation of General 1D Staged

In this section, we show how to simulate all 1D staged systems with TA systems. First, we define what simulate means for these systems, followed by a high-level overview of our simulation, and then the details.

3.1 Simulation

Here, we utilize a simplified definition of simulation in which the set of final terminal assemblies, from the *target* staged system to be simulated, is exactly the same, under a mapping function, as the final terminal assemblies of the *source* TA system that is simulating it. This is a standard type of simulation used, and we omit technical definitions in this version. A stronger definition of simulation incorporates *dynamics*, in which assemblies may attach in the target system if and only if they attach in the source system. However, our approach focuses on simulating a restricted set of dynamics that are sufficient to ensure the production of all final (and partial) assemblies. We leave the problem of fully simulating the dynamics of a staged system as future work.

3.2 Overview

We create a Tile Automata system with initial tiles representing the initial tile types of the staged system. Each assembly in our Tile Automata system represents an assembly in a specific stage and bin. Each state is a pair consisting of a tile type t and a

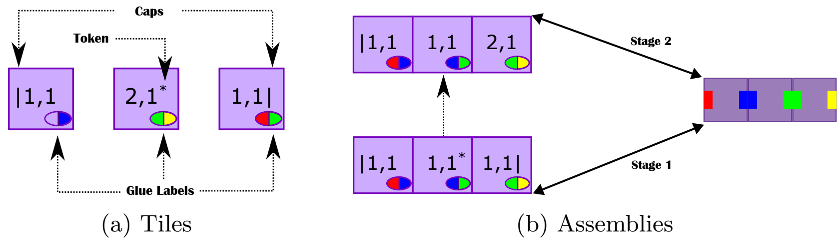


Fig. 4: (a) Each of our Tile Automata states conceptually represents two glue labels that say which tile type they map to (a glue may be null, as in the leftmost state). They may also contain features such as the left/right cap or the active state token. (b) Assemblies map based on the glue labels on the Tile Automata states. Multiple Tile Automata assemblies represent the same Staged assembly, but sometimes in different stages.

stage-bin label representing t in that specific stage and bin. Some states will have an *active state token*(*) used to track the progress of the Tile Automata assembly in the assembly tree. We simulate only left- or right-handed assembly trees based on the parity of the stage number. The logic for the transition rules is described in Algorithm 1 using a *Glue-Terminal Table*. Each Tile Automata assembly builds according to the assembly trees of the staged system by having the token "read" the glues to decide if an assembly is terminal in a bin and needs to transition to the next stage.

3.3 Glue-Terminal Table

For the simulation to work, we need to know the glues used in each bin of the target system because we cannot "read" the absence of a glue/assembly in self-assembly. However, we can use the Glue-Terminal Table to construct the transition rules. This table stores which glues correspond with each bin.

Definition 4 (Glue-Terminal Table)**.** *For a staged system $\Upsilon = (M_{r,b}, T)$, the Glue-Terminal table $GT((s,b),g)$ is a binary $|M_{r,b}| \times G$ table with rows labeled with stage-bin pairs and columns labeled with glues. The entry $GT((s,b),g)$ is true (Used) if there exists at least two producible assemblies in bin b that attach using glue g in stage s. If it is false (Term.), the glue is never used in bin b for stage s.*

This table can be computed recursively by checking the glues of the that are assemblies in the previous bin. Computing terminal assemblies can be done much easier since it's 1D.

3.4 States and Initial Tiles

A state in our Tile Automata system has the following properties: each state has the first two properties and the second two properties are optional. The first label has sb possible options, the second has t, and the rest only increase the state space by a constant factor. This results in an upper bound on the states used of $\mathcal{O}(sbt)$.

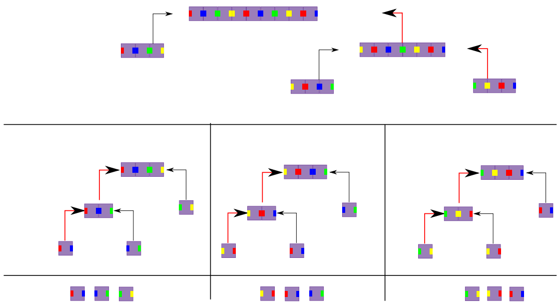
- **Stage-Bin Label.** Each state $(s,i)_t$ is labeled with a pair of integers (s,i) saying the state represents the i^{th} bin in stage s.

Algorithm 1 Algorithm to create transition rules for each pair of states in a Tile Automata system.

Data: Left state a and right state b, and glue-terminal table GT.
Result: Transition rule $(a, b) \rightarrow (a', b')$ if such a rule exists.
Let $L(\sigma)/R(\sigma)$ be the left/right glue label of the tile type σ maps to.
Let $\texttt{STAGE}(\sigma)$ be the stage σ is in. Let $\texttt{BIN}(\sigma)$ be the bin σ is in.
Let $\texttt{NEXT\_BIN}(\sigma)$ be the bin σ will be in the next stage.
Let $\texttt{HAS\_TOKEN}(\sigma)$ be *true* if σ contains a token, *false* otherwise.

if $R(a) \neq L(b)$ **then**
  Return null
if $\texttt{HAS\_TOKEN}(a) \wedge \texttt{STAGE}(a)$ is odd **then**
  if b has a right cap **then**
    if $GT((\texttt{STAGE}(b), \texttt{BIN}(b)), R(b)) =$ *Used* **then**
      $a' \leftarrow a - *$; $b' \leftarrow b + *$; $b' \leftarrow b' - |$
    else if $GT((\texttt{STAGE}(b) + 1, \texttt{NEXT\_BIN}(b)), R(b)) =$ *Used* **then**
      $a' \leftarrow a - *$; $b' \leftarrow b - |$ $\texttt{STAGE}(b') \leftarrow \texttt{STAGE}(b') + 1$; $\texttt{BIN}(b') \leftarrow \texttt{NEXT\_BIN}(b')$
    else
      $a' \leftarrow a$; $b' \leftarrow b$ $\texttt{STAGE}(a') \leftarrow \texttt{STAGE}(a') + 1$; $\texttt{BIN}(a') \leftarrow \texttt{NEXT\_BIN}(a')$ $\texttt{STAGE}(b') \leftarrow \texttt{STAGE}(b') + 1$; $\texttt{BIN}(b') \leftarrow \texttt{NEXT\_BIN}(b')$
  else
    $a' \leftarrow a - *$; $b' \leftarrow b + *$ $\texttt{STAGE}(b') \leftarrow \texttt{STAGE}(b') + 1$; $\texttt{BIN}(b') \leftarrow \texttt{NEXT\_BIN}(b')$
  Return $(a, b) \rightarrow (a', b')$
if $\texttt{HAS\_TOKEN}(b) \wedge \texttt{STAGE}(b)$ is even **then**
  if a has a left cap **then**
    if $GT((\texttt{STAGE}(a), \texttt{BIN}(a)), L(a)) =$ *Used* **then**
      $b' \leftarrow b - *$; $a' \leftarrow a + *$; $a' \leftarrow a' - |$
    else if $GT((\texttt{STAGE}(a) + 1, \texttt{NEXT\_BIN}(a)), L(a)) =$ *Used* **then**
      $b' \leftarrow b - *$; $a' \leftarrow a - |$ $\texttt{STAGE}(a') \leftarrow \texttt{STAGE}(a') + 1$; $\texttt{BIN}(a') \leftarrow \texttt{NEXT\_BIN}(a')$
    else
      $b' \leftarrow b$; $a' \leftarrow a$ $\texttt{STAGE}(b') \leftarrow \texttt{STAGE}(b') + 1$; $\texttt{BIN}(b') \leftarrow \texttt{NEXT\_BIN}(b')$ $\texttt{STAGE}(a') \leftarrow \texttt{STAGE}(a') + 1$; $\texttt{BIN}(a') \leftarrow \texttt{NEXT\_BIN}(a')$
  else
    $b' \leftarrow b - *$; $a' \leftarrow a + *$ $\texttt{STAGE}(a') \leftarrow \texttt{STAGE}(a') + 1$; $\texttt{BIN}(a') \leftarrow \texttt{NEXT\_BIN}(a')$
  Return $(a, b) \rightarrow (a', b')$

- **Glue Labels.** Each state $(s, i)_t$ represents a tile t from the staged system. We say this state has the glue labels of t when defining our affinity rules in Tile Automata. This label also defines our mapping from TA states to staged tiles in both directions.
- **Active State Token.** A state $(s, i)_t^*$ may have an Active State Token *. The token is used to enforce the left/right handed assembly trees by starting on one side of an assembly, and allowing attachment to other states with matching glue and stage-bin labels.



(a) Staged System

(s, b)	Red	Blue	Green	Yellow
$(1,1)$	Term	Used	Used	Term
$(1,2)$	Used	Used	Term	Term
$(1,3)$	Used	Term	Term	Used
$(2,1)$	Term	Term	Used	Used

(b) Glue-Terminal Table

Fig. 5: (a) Example Staged system to be simulated. (b) Glue-Terminal Table for shown staged system. In the table, s is the stage and b is the bin.

- **Caps.** A state may have a cap on one side, denoted $|s,i)_t$ or $(s,i|_t$. This means that on the side of the cap $|$, there are no affinity rules for that state. Until an assembly is ready to attach, it will have caps on its left and right most tiles.

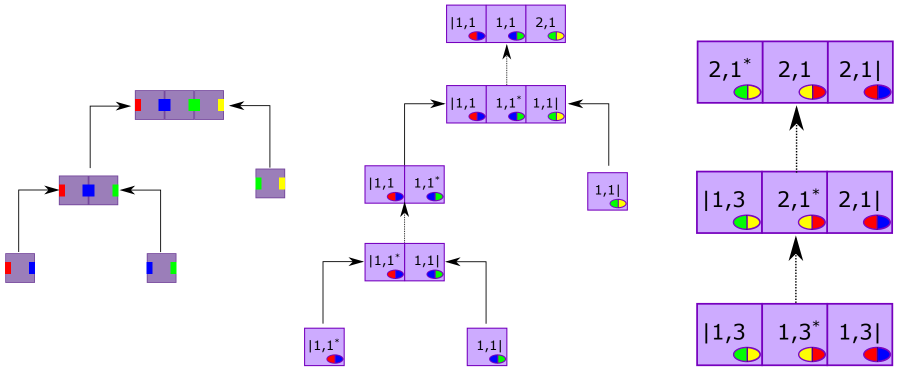
We create an initial state for each pair $b_{1,i}, t$ where $b_{1,i}$ is the i^{th} bin of the first stage and t is a tile input to that bin. If the left glue of the t is used in the $b_{1,i}$, then we include the state $(1, i_t|$, i.e., the right cap state. If the left glue is open, but the right glue is used, the tile is the first in a left-handed assembly tree. In this case, we include the token left cap state $|1, i^*_t)$.

If a tile is terminal in the first bin, we instead include an initial state representing the first bin where the state is consumed. For example, if a tile t is input to bin $(1,i)$ and is terminal, but its right glue is used in an attachment in bin $(2,j)$ (where there's an edge between $(1,i)$ and $(2,j)$), then we instead include an initial state $|2, j_t)$.

3.5 Bin Simulation

In any odd stage, we construct every terminal using a sequence of attachments representing a left-handed assembly tree. For even stages, we use a right-handed assembly tree. We control this with the token by defining our affinity rules such that every attachment occurs between one state with the token and one without a cap. We switch between the left and right handed trees to reduce the amount of times the token must walk back and forth on the assembly since the token ends on the opposite side each time.

We walk through an example of a bin in the first stage in Figure 6a. The token left cap state $|1, 1^*_t)$ attaches to the right cap state $(1, 1_{t'}|$ if t' attaches to the right



(a) $(1, 1)$ (b) $(1, 3)$

Fig. 6: (a) Example simulation of an assembly in stage 1. Notice the token moves leftward through the assembly as it builds to enforce a left handed assembly tree. (b) Transition for terminal assembly in bin $(1, 3)$. Since the rightmost glue is terminal in bin $(1, 3)$ the token changes the stage to 2 and starts moving left to remove the cap.

of t. These two states then transition. If the right glue of t' is used in the bin, the token moves to that state and removes the cap. This process can then repeat in the bin. Looking at the next tile t'', the right glue is unused, and thus, the assembly is terminal, and the transition should move it to the next stage, now changing directions as outlined in Figure 6b. The process for defining transitions is described in Algorithm 1; when given two states and the Glue-Transition table, a transition rule is returned if one would exist in the system. Note that this algorithm is non-deterministic as one bin may output to multiple bins in the next stage, so a pair of states may have multiple transition rules.

Theorem 1. *For any 1D staged system Υ with flexible glues, s stages, b bins, and t tile types, there exists a 1D Freezing Affinity-Strengthening Tile Automata system Γ with $\mathcal{O}(sbt)$ states that simulates Υ.*

Proof. Consider a staged system $\Upsilon = (M_{r,b}, T)$ with s stages, b bins and t tiles types. Tile Automata system $\Gamma = (\Sigma, \Pi, \Delta)$ which simulates Υ is defined and discussed below.

State complexity $\mathcal{O}(sbt)$. Each tile type in Υ requires a unique state in Γ for every bin in every stage, resulting in $s \cdot b \cdot t$ states. The additional state increase for the token and caps of each state is constant for a total of $\mathcal{O}(sbt)$ states.

Flexible Glues, Freezing and Affinity Strengthening. A state $\sigma_t \in \Sigma$ with tile type $t \in T$ has affinity with a state $\sigma'_t \in \Sigma$ with tile type $t' \in T$ if t attaches to t' in Υ. With the affinity function we can encode general glues so we can simulate flexible glues. For every transition rule $\delta \in \Delta$, δ does not alter the tile type a state represents since only the stage, bin, token, or cap are affected.

Every transition rule is freezing and either removes a cap, moves the token forward, or advances to the next stage. Once a state with a tile type t has lost its cap it can never regain it. In a single stage, the token may walk over each tile a maximum of 2 times as both sides of the assembly must be checked to decide if the assembly is terminal. Note that this token walk involves adding an additional distinct state so the tiles do not visit the same state twice.

Simulation. We prove this is a correct simulation by induction on the size of the assemblies. The initial assemblies cover our base case for single tiles in Λ. The tile input in the first stage in Υ ensures each included assembly is in Λ. For the recursive case, assume every assembly $A \in \texttt{PROD}_\Upsilon$ with $|A| < x$ is simulated. Let b be the bin in which A is produced. A must be produced using two assemblies B and C, each of size $< x$, which are also in bin b. From our assumption, B and C have assemblies representing them- $B', C' \in \texttt{PROD}_\Gamma(\Lambda)$. Since B and C are produced in the same bin and have matching assemblies B' and C' with matching tokens, they may combine into an assembly A'. A will represent A since it has the same labels. $\square$

3.6 Lines

Using Theorem 1, we provide an alternate proof from [5] of length-n lines with $\mathcal{O}(\log n)$ states.

Corollary 1. *For all $n \in \mathbb{N}$, there exists a freezing Tile Automata system that uniquely assembles a $1 \times n$ line in $\mathcal{O}(\log n)$ states.*

Proof. In [12], it is shown that there exists a staged assembly system that uniquely produces a $1 \times n$ line with 6 tile types, 7 bins, and $\mathcal{O}(\log n)$ stages. From theorem 1, there exists a Freezing Affinity-Strengthening Tile Automata system Γ with $\mathcal{O}(sbt)$ states that simulates any staged system Υ with s stages, b bins and t tile types. Therefore, simulating the staged assembly system from [12] can be done with $\mathcal{O}(\log n)$ states. $\square$

4 Freezing Affinity Strengthening

While the results in the previous section imply that you may implement Context Free-Grammar (CFGs) by simulating 1D Staged, here we provide a direct simulation of CFGs. This direct simulation has the advantage of being deterministic and single transition. An example CFG is shown in Figure 7, along with the corresponding TA system in Figure 8. In addition to the freezing and affinity strengthening constraints, this result achieves the feature that tiles never undergo a change in their color throughout the assembly process. We denote rules that adhere to this constraint as *color-locked* rules.

4.1 Context-Free Grammars

A **context-free grammar (CFG)** is a set of recursive rules used to generate patterns of strings in a given language. A CFG is defined as a quadruple $G = (V, \Upsilon, R, S)$. V represents a finite set of non-terminal symbols and Υ is a finite set of terminal symbols. The symbol R is the set of production rules and S is a special variable in V called the start symbol. Production rules R of CFGs are in the form $A \rightarrow BC|a$, with V in the left-hand side and V and/or Υ on the right- hand side. A CFG derives a string through recursively replacing nonterminal symbols with terminal and non-terminal symbols based on its production rules.

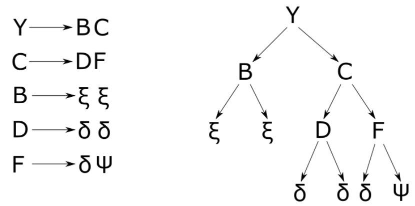


Fig. 7: A restricted context-free grammar (RCFG) G and its corresponding parse tree that produces a pattern P, $\xi\xi\delta\delta\delta\psi$. This is a deterministic grammar, producing only pattern P.

Minimum Context Free Grammars We define the size of a grammar G as the number of symbols in the right hand side rules. Let CF_P be the size of the smallest CFG that produces the singleton language $|P|$.

Restricted Context-Free Grammars (RCFG). In this work, we focus on the CFG class used in [13] which they name Restricted CFGs. These restricted grammars produce a singleton language, $|L(G)| = 1$ and thus are deterministic. This is the same concept of Context-Free Straight Line grammars from [4]. Each RCFG production rule R contains two symbols on its right-hand side. We can convert any other deterministic CFG to this form with only a constant factor size increase.

Figure 7 presents an example RCFG G and its parse tree that derives a pattern of symbols P, $\xi\xi\delta\delta\delta\psi$. The parse tree shows how internal nodes are non-terminal symbols and leaf nodes contain a terminal symbol whose in-order traversal derives the output string. Notice that since RCFG G is deterministic, each non-terminal symbol $N \in V$ has a unique subpattern $g(N)$ that is defined by taking N to be the start symbol S and applying the production rules. Here, the language or output pattern P of G can be denoted by $L(G) = g(S)$.

4.2 1D Patterned Assembly Construction

We describe our method of simulating a Restricted CFG G with Tile Automata to build a 1D patterned assembly that represents the pattern P derived from G.

Initial Tiles and Producibles. This Tile Automata system, Γ_G, begins with creating its initial tiles from the unique terminal symbols, Υ, in RCFG G. In Figure 7, the output pattern P derived from G has three unique terminal symbols ξ, δ, and ψ. Each unique Υ in G is mapped to a distinct color and remains locked to the symbol throughout the construction. From G's production rule parse tree, internal nodes have two child nodes consisting of two similar or different terminal symbols, Υ. Depending on the placement of the terminal symbols, the initial tiles are designated as L for left-hand side or R for right-hand side. Figure 8a depicts that an initial tile consists of an Υ symbol with its distinct color in an L or R state.

Following G's parse tree, the initial tiles can combine to build Γ_G's first set of producible assemblies. Grammar G's production rules can be encoded into system Γ_G by providing the affinity rules. If two terminal symbols in G connect to the same internal node in its parse tree, the initial tiles in Γ_G that represent the symbols combine to form a producible. The first set of producibles cannot bind to any other tile because

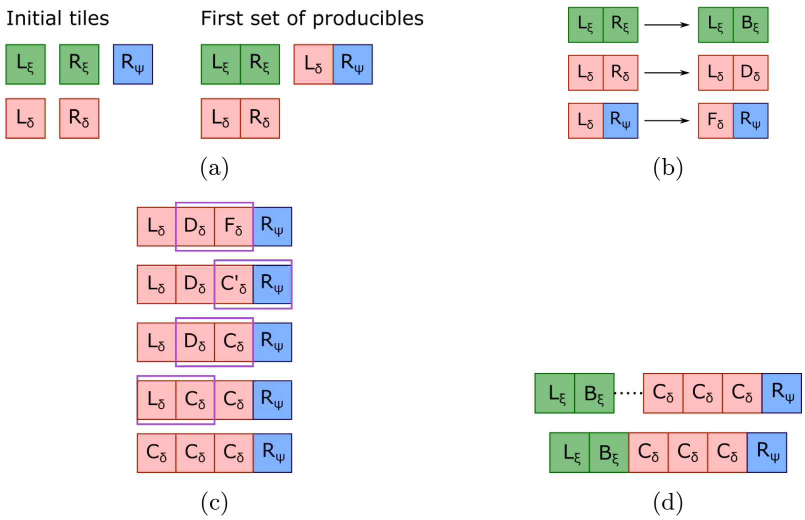


Fig. 8: Tile Automata system, Γ_G, assembling a 1D patterned assembly that represents the pattern P produced by the RCFG G shown in Figure 7. (a) Γ_G contains initial tiles from the unique terminal symbols of G. Grammar G's production rules are encoded in Γ_G as affinity rules, allowing initial tiles to form the first set of producibles. (b) Following G's rules, Γ_G's color-locked, one-sided transition rules are applied to the first set of producibles. (c) Subpattern assembly $L_\delta D_\delta F_\delta R_\psi$ transitions tiles towards captile R, marking visited tiles. Once the transitions reach captile R, we transition to the left of the subassembly to C_δ tiles, removing the marks along the way. (d) RCFG G production rule $Y \rightarrow BC$, directs Γ_G to combine B and C subassemblies to build the terminal patterned line assembly, representing pattern P from grammar G.

they are capped with L and R states, which we denote as *captiles*, and thus are stopped from growing, shown in Figure 8b. Note that these first producibles are subpatterns of P.

Uncapping Producibles. RCFG G production rules tell Γ_G how the first producible assemblies will combine to form larger subpatterns of P and ultimately represent the final patterned line assembly. In Γ_G, our first set of producibles are composed of L and R captiles. For these producibles to combine with each other, we apply one-sided, color-locked transition rules to uncap each producible, opening their left or right-hand side depending on the nonterminal symbols placement in grammar G's production rules. For example, in Figure 7 nonterminal C is composed of a D on the left-hand side and F on the right-hand side. In Figure 3.2b, the producible $L_\delta R_\psi$ represents G's terminal symbols $\delta\psi$ as well as nonterminal F. Because F sticks to D's right side, a one-sided transition rule is applied to producible $L_\delta R_\psi$ changing only the pink tile L_δ to a new tile F_δ, forming next producible $F_\delta R_\psi$. Here, the color-locked restriction in Γ_G applies because the new tile F_δ retains its color (pink) that is designated to the terminal symbol δ of P from G. This producible $F_\delta R_\psi$ is considered a right-handed subassembly because it is uncapped on its left side, allowing it to attach to the right-hand side of the producible that represents nonterminal D. The rest of Γ_G's first producibles transition according to G's production rules as shown in Figure 8b.

Transition Walk. Γ_G recursively applies G's production rules to build the other subassemblies needed to represent pattern P. Grammar G's production rule $C \rightarrow DF$ tells Γ_G that there is affinity between D and F, directing producibles $L_\delta D_\delta$ and $F_\delta R_\psi$ to combine and form a new subpattern assembly $\delta\delta\delta\psi$ of P, shown at the top of Figure 8c. In Lemma 1, we show how every nonterminal in G is represented as a subpattern assembly produced by Γ_G. Subpattern assembly $L_\delta D_\delta F_\delta R_\psi$, represents nonterminal C from G and is capped with captiles L and R. From G's production rules in Figure 7, nonterminal symbol Y is composed of B on the left-hand side and C on right-hand side. To uncap the left side of subpattern $L_\delta D_\delta F_\delta R_\psi$, a series of one-sided, color-locked transition rules are applied to turn each tile into a C_δ tile making the subassembly uniform, depicted in Figure 8c. The adjacent tiles that have transition rules between them are outlined in purple, with the resulting tiles shown in the subassembly below it.

We apply the method of "walking" across 1D assemblies from [5] to uncap left or right sides of subassemblies. Subpattern assembly $L_\delta D_\delta F_\delta R_\psi$ must have an opened left side to attach to subassembly B, so we first transition tiles towards the right side, marking visited tiles with a prime notation. Once the transitions reach captile R, we begin to transition to the left of the subassembly to C_δ tiles, removing the prime notations along the way. As shown in Figure 8c, once producibles D and F combine, a one-sided, color-locked transition rule applies changing the F_δ tile for a temporary C'_δ tile, where the prime marks the tile as visited. Next, the adjacent C'_δ and R_ψ tiles transition to remove the prime from the C'_δ tile, producing subpattern $L_\delta D_\delta C_\delta R_\psi$. Another transition is applied between adjacent tiles $D_\delta C_\delta$ to form the fourth subassembly in Figure 8c. Finally, one more transition occurs between $L_\delta C_\delta$ to produce subpattern $C_\delta C_\delta C_\delta R_\psi$.

Patterned Line Assembly. Figure 8d depicts the subpattern assemblies created by Γ_G that represent nonterminal symbols B and C. According to the affinity rules of Γ_G, subassemblies B and C combine to form terminal assembly Y. Subassemblies for B and C attach and terminal assembly Y is constructed and capped with captiles L and R on its sides. This new terminal assembly Y represents G's pattern P, with each distinct colored tile representing unique terminal symbols of pattern P.

Definition 5 (Nonterminal Pattern)**.** *For a nonterminal $N \in V$, let $g(N)$ be a substring derived when N is the start symbol of grammar G.*

Lemma 1. *Each producible assembly in Γ_G, created from a RCFG $G = (V, \Upsilon, R, S)$ represents a subpattern $g(N)$ for some symbol N in $V \bigcup \Upsilon$.*

Proof. We will prove by induction that any producible assembly B represents a subpattern $g(N)$ for some symbol N in $V \bigcup \Upsilon$.

For the base case, if B is an initial tile, then B represents some terminal symbol $N \in \Upsilon$. For the inductive step, if B is a larger assembly, then we show B represents a nonterminal $N \in V$. We define the following two recursive cases. B is built from combining subassemblies C and D, we can assume these assemblies represent symbols N_C and N_D respectively. We know from how we defined our affinity rules if C and D can combine then there is some rule $N \rightarrow N_C N_D$. Then B represents the pattern $g(N) =$

$g(N_C) \oplus g(N_D)$. B is producible via transition from an assembly C, B must represent the same subpattern as C since the transition rules do not change the color. □

Theorem 2. *For any pattern P, there exists a Freezing Tile Automata system Γ with deterministic single transition rules that uniquely assembles P with $\mathcal{O}(CF_P)$ states and 1×1 scale. This system is cycle-free and transition rules do not change the color of tiles.*

Proof. By definition, there exists a CFG G that produces P with $|G| = CF_P$. We construct the system Γ_G. From Lemma 1, each producible assembly B must represent a subpattern $g(N)$ for some symbol N. The only terminal of Γ is the assembly representing the start symbol S since all other assemblies either can attach to another assembly or can transition. □

5 Optimal Patterns in Tile Automata

In this section we show that general Tile Automata can obtain Kolmogorov optimal state complexity at 1×1 scale. These first results are achieved by applying the efficient binary string construction from [1], and allowing the additional tiles used by the assembly to fall off, thus leaving only the string. We can then utilize the Turing machine from to simulate a universal Turing Machine. The Turing Machine in was designed to accept/reject an input, so we modify the Turing Machine to print P on the tape and halt.

Lemma 2. *For any binary pattern X there exists an affinity strengthening Tile Automata system that uniquely constructs an assembly representing X at scale,*

- 4×2 *with* $\mathcal{O}(|X|^{\frac{1}{4}})$ *states,*
- 3×2 *with* $\mathcal{O}(|X|^{\frac{1}{3}})$ *states using single-transition rules, and*
- 2×1 *with* $\mathcal{O}(|X|^{\frac{1}{2}})$ *states using deterministic single-transition rules and is cycle free.*

Proof. These constructions are provided in [1] which shows that there exists a method to encode the bits of a string in the transition rules of the system. Each construction takes advantage of a feature not available in the stricter class of systems. The model shown in this paper however does have seeded growth but a simple extension shows this works with 2-handed production. □

Theorem 3. *For any pattern P, there exists a Tile Automata system Γ that uniquely assembles P with $\Theta(K_P^{\frac{1}{4}})$ states at 1×1 scale.*

Proof. Given a pattern P, we first consider a Turing machine M that will print P. Using the process described in [5], we create a system $\Gamma_M = (\Sigma, \Pi, \Lambda, \Delta, \tau)$ that simulates M. When M has completed printing P, the buffer states B_L and B_R need to detach. We take Σ and create a copy Σ_{SR} which we modify by removing the accept/reject states in favor of *final states*. For every state $\rho \in \Sigma_{SR}$ where ρ composes P, we create $\rho_F \in \Sigma_{SR}$ with affinity only for every other final state. Starting with the rightmost tile that composes P, we add transition rules that will transition each tile with state ρ into their final state equivalent ρ_F. Since these final states have no affinity with the buffer states, tiles with those buffer states, and any other state not considered a final state, will detach from the assembly. This detaching process begins with a transition rule between B_R and the rightmost tile with state ρ, turning ρ into ρ_F.

From Lemma 2, we encode Γ_M in a binary string $b(\Gamma_M)$ and use $b(\Gamma_M)$ to construct system Γ_S that uses $\Theta(K_P^{\frac{1}{4}})$ to assemble $b(\Gamma_M)$. [21] states there exists a universal Turing machine that uses linear space in the amount of space used by the machine being simulated. Γ will simulate a universal Turing machine with Γ_S being used to construct the input into Γ, giving us a system that uniquely assembles P with $\Theta(K_P^{\frac{1}{4}})$ states and 1×1 scale. $\square$

5.1 Deterministic Single Transition Turing Machine

The Turing machine from [5] utilizes transition rules that change both tiles in the same step. While [8] shows a way to simulate double rules with single rules, we present a slight modification to the Turing machine construction to make it utilize single rules.

Lemma 3. *For any pattern P, there exists a Tile Automata system Γ with deterministic single-transition rules that uniquely assembles P with $\mathcal{O}(K_P)$ states and 1×1 scale. This system is cycle free.*

Proof. We create a Turing machine M that will print P. Using Turing machine M, we use the process described in [5] to create a system $\Gamma_D = (\Sigma, \Pi, \Lambda, \Delta, \tau)$ that simulates M utilizing double-transition rules. We then modify Σ, Δ, and Π into single-transition rule versions Σ_{SR}, Δ_{SR}, and Π_{SR} as follows.

Σ_{SR} and Π_{SR} will initially be a copy of Σ and Π respectively, while Δ_{SR} is populated with every single-transition rule in Δ. For every double-transition rule $\delta = (\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}, d) \in \Delta$, we create an additional state $\omega \in \Sigma_{SR}$. The affinity strength of ω using Π_{SR} will be equal to the affinity strength of D using Π for all directions. We take δ and create 3 transition rules $\delta_{S1}, \delta_{S2}, \delta_{S3} \in \Delta_S$ defined below.

- $\delta_{S1} = (\mathsf{A}, \mathsf{B}, \mathsf{A}, \omega, d)$
- $\delta_{S2} = (\mathsf{A}, \omega, \mathsf{C}, \omega, d)$
- $\delta_{S3} = (\mathsf{C}, \omega, \mathsf{C}, \mathsf{D}, d)$

We use the *final states* described in the proof of Theorem 3 to modify Σ_{SR} in order to detach the buffer states. Using our modifications, we create a Tile Automata system $\Gamma = (\Sigma_{SR}, \Pi_{SR}, \Lambda, \Delta_{SR}, \tau)$ with deterministic single-transition rules that uniquely assembles P with $\mathcal{O}(K_P)$ states and 1×1 scale. $\square$

Using Lemma 2we can encode the input to a universal Turing machine with square root the number of states with deterministic single transition rules.

Theorem 4. *For any pattern P, there exists a Tile Automata system Γ with deterministic single transition rules that uniquely assembles P with $\mathcal{O}(K_P^{\frac{1}{2}})$ states and 1×1 scale. This system is cycle free.*

Proof. We make some modifications to the process used in the proof of Theorem 3 to satisfy the deterministic single-transition rules. We create Γ_M using the method described in the proof of Lemma 3 and encode the system in a binary string $b(\Gamma_M)$. Γ_S is created using $b(\Gamma_M)$ which will use $\mathcal{O}(K_P^{\frac{1}{2}})$ as shown in Lemma 2. Γ will simulate a universal Turing machine that uses the assembly built by Γ_S, giving us a system that uniquely assembles P with $\mathcal{O}(K_P^{\frac{1}{2}})$ states and 1×1 scale. □

Other methods for non-deterministic rules and with single and double rules give the following.

Theorem 5. *For any pattern P, there exists a Tile Automata system Γ with single transition rules that uniquely assembles P with $\Theta(K_P^{\frac{1}{3}})$ states and 1×1 scale.*

Proof. A deterministic single-rule TA system Γ_M can be constructed according to Lemma 3, and using an encoding $b(\Gamma_M)$, we make Γ_S which uses $\Theta(K_P^{\frac{1}{3}})$ states using Lemma 2 □

5.2 Freezing with Detachment

We do not directly consider Freezing and allowing detachment since the results of [9] shown that any non-freezing system can be simulated by a freezing system by replacing tiles. Also shown in the full version of [5] it was shown freezing Tile Automata with only height 2 assemblies can simulate a general Turing machine. The assembly can then fall apart to achieve 1×1 scale.

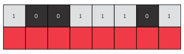
6 Affinity Strengthening

As shown in [5], Affinity Strengthening Tile Automata (ASTA) is capable of simulating Linear Bounded Automata (LBA) and that verification in ASTA is PSPACE-Complete. Thus, it makes sense to view this version of the model as the spaced-bounded version of Tile Automata, similar in power to LBAs or Context Sensitive Grammars. We select space-bounded Kolmogorov complexity as our method of bounding the state complexity since we can encode a string and simulate a Turing machine as in the previous section to get an upper bound. The concept of bounded Kolmogorov Complexity was explored in [14]. For these results, we consider building scaled patterns in which each pixel of the pattern is expanded to a $s \times O(1)$ box of pixels. Another way to view this upper bound is that for any algorithm α that outputs P in $f(|P|)$ space, we may construct an assembly representing P of size $\mathcal{O}(f(|P|)$, in $\mathcal{O}(|\alpha|)^{\frac{1}{4}}$ states, where $|\alpha|$ is the number of bits describing α for general Tile Automata. Similar bounds are shown for the other restrictions. It is interesting to point out that with a large enough scale factor we achieve Kolmogorov optimal bounds, including optimal scaled shape constructions as in [16].

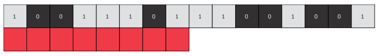
6.1 Space Bounded Kolmogorov Complexity

Definition 6 (Space Bounded Kolmogorov Complexity)**.** *Given a pattern P, and a function $f : \mathbb{N} \to \mathbb{N}$ that outputs the space used by a Turing machine, let $KS_P(f(|P|))$ be the length of the smallest string that, when input to a universal Turing machine M_K, halts with the pattern P on the tape in $f(|P|)$ space.*

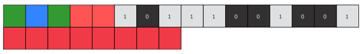
It was stated in [14] that there exists some optimal Turing machine, which we call M_K, that incurs only a constant multiplicative factor increase in the space used. We note for two space bounds $f(|P|)$ and $g(|P|)$, the value $KS_P(g(|P|)) \leq KS_P(f(|P|))$



(a) Assembly representing input string.



(b) Universal Turing machine running on input.



(c) The pattern is output on the tape.

(d) The pattern scales outward to fill the assembly.

Fig. 9: (a) It is possible to build assemblies representing binary strings with an efficient number of states. (b) We can then run a universal Turing machine on the input increasing the length of the assembly as needed. (c) The Turing machine will halt with the pattern output on the tape. (d) The pattern will then scale out to fill the assembly.

as using more space allows for more efficient computing of all pattern P, with $|P| < c$ for some constant c.

6.2 Construction

Figure 9a shows a sketch of the assembly for deterministic Tile Automata using the string constructions from [1] shown in Lemma 2. The single rule Turing machine can be modified to never break apart and only increase the tape length, similar to the PSPACE-hard reduction from [5]. Figure 9b shows an example Turing machine being run where the tape length is increased.

Once the pattern has been printed or assembled (Figure 9c), there are additional tiles in the assembly to deal with. However, since we cannot detach tiles, we scale the pattern. The first step is to expand the length of pattern. If we use s tape cells to print a pattern $|P|$, we scale each point in the pattern by $c \cdot |P|$. This is done with a simple algorithm implemented in the transition rules. Create a token state that starts at the leftmost state after the string is printed. Go to each 'pixel' and tell it expand once after first signaling the neighboring cells to move right (to prevent overwriting). We do this for each pixel in the pattern, push the other states, increase pixel size. The system repeats this process until all pixels of the pattern are fully expanded, and then they transition the tiles below them, which results in the patterned assembly of Figure 9d.

Theorem 6. *For any pattern P, scale factor $s > 0$, there exists an Affinity Strengthening Tile Automata system Γ with deterministic single transition rules that uniquely assembles P with $\mathcal{O}(KS_P(s|P|)^{\frac{1}{2}})$ states and $s \times 2$ scale. This system is cycle free.*

Proof. Let X be the string that when input to M, P is written to the tape in $s|P|$ space. Using the binary string building results from Lemma 2 we can encode X in $\mathcal{O}(|X|^{\frac{1}{2}})$ states. Then we run M using the single transition rule Turing machine described in the proof of Lemma 3. This will run and leave the pattern P on the tape states. Consider a second Turing machine M_{INC} scales up the pattern to fill the width of the tape. Each pixel is increased by the same amount. The states then copy the color to the state below it as well. This can be done in a constant number of states. The amount that each pattern scales by is $\frac{s|P|}{|P|} = |P| \cdot (s-1)$. □

Theorem 7. *For any pattern P, scale factor $s > 0$, there exists an Affinity Strengthening Tile Automata system Γ with single-transition rules that uniquely assembles P with $\mathcal{O}(KS_P(s|P|)^{\frac{1}{3}})$ states and $s \times 3$ scale.*

Proof. Again using the Single-Transition rule Turing machine from the proof of Lemma 3 and the string building result from Lemma 2, we can construct the input to the universal Turing machine M_K. The pattern P can be output in $s|P|$ space. We then scale up the pattern to fill the assembly. □

Theorem 8. *For any pattern P, scale factor $s > 0$, there exists an Affinity Strengthening Tile Automata system Γ that uniquely assembles P with $\mathcal{O}(KS_P(s|P|)^{\frac{1}{4}})$ states and $s \times 4$ scale.*

Proof. Lastly using the same method from Lemma 2 we can encode the input to the universal Turing machine in $|X|^{\frac{1}{4}}$ where $|X|$ is the length of the string. This results in an assembly of height 4 as resulting assembly will be of dimensions $|X| \times 4$. The string X can then be input to the Turing machine to print the pattern than scale up. □

7 Lower Bounds

We provide lower bounds for general Tile Automata under the three transition rule restrictions. We do this by showing a binary string encoding a Tile Automata system can be passed to a Turing machine to output a patterned assembly, from which the pattern P can be read and output. This means we cannot encode a system in less bits than the Kolmogorov Complexity K_P. We achieve similar bounds as [1] as we use the same system for binary string encoding.

For affinity strengthening we provide a lower bound based on the Space Bounded Kolmogorov Complexity defined in Section 6. As with the previous result, we show that a binary string encoding a system can be passed to a Turing machine that outputs the uniquely produced assembly representing the pattern P in $f(|P|)$ space. This means we cannot encode the system in less than $KS_P(f(|P|))$ bits. We give an upper bound of $f(n) = \mathcal{O}((s|P|)^2 \log^2 s|P|)$ in Lemma 4 to compute a pattern scaled by a factor of s. With this we base our lower bounds on $KS_P((s|P|)^2 \log^2 s|P|)$.

7.1 General

Theorem 9. *For any Pattern P over constant colors a Tile Automata system Γ that uniquely assembles P at any scale requires $\Omega(K_P^{\frac{1}{4}})$ states.*

Proof. A Tile Automata system $\Gamma = \{\Sigma, \Pi, \Lambda, \Delta, \tau = \mathcal{O}(1)\}$ can be encoded in $< c|\Sigma|^4$ bits for some constant c. We may store Π as a $|\Sigma| \times |\Sigma|$ table with each $\mathcal{O}(\log \tau)$ bit cell storing their binding strength which is at most τ. The initial tiles Λ can be encoded with a single bit for each state. Δ is the largest part of the encoding taking $2|\Sigma|^4$ bits . This can be stored as a 4D table where each cell contains two bits (v, h). The first bit at index $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ being whether or not the states (σ_1, σ_2) transition to (σ_3, σ_4) vertically and the second bit horizontally. The exact constant achieved is thus dependent on τ.

Consider a Turing machine M_{TA} that takes as input the binary description of a Tile Automata system Γ that uniquely assembles an assembly A and outputs the pattern of A as a string. We can assume M_{TA} can be described in constant bits. The producible assemblies of a Tile Automata system are recursively enumerable. Since we know that Γ uniquely produces P we know there exists a finite number of assemblies as well as the system must be bounded. This makes verifying the terminal assembly is decidable as there's only a finite number of possible Combinations, Breaks, and Transitions to check.

Let M_K be the fixed universal Turing machine to define K_P, assume there exists a system $\Gamma' = \{\Sigma', \Pi', \Lambda', \Delta', \tau' = \mathcal{O}(1)\}$ that uniquely produces the pattern P with $|\Sigma'| < (\frac{K_P}{c})^{\frac{1}{4}}$ states. Using our encoding method above encode Γ' as a binary string $b(\Gamma')$ with in $|b(\Gamma')| < K_P$. If we pass $b(\Gamma')$ along with an encoding of M_{TA} to the universal Turing Machine M_K it will simulate the algorithm and output the pattern P. This would mean that M_K can produce the pattern with less than $K_P + | < M_k > |$ bits which violates the Kolmogorov Complexity so this is not possible. $\square$

Theorem 10. *For any Pattern P over constant colors, a Tile Automata system Γ with single transition rules that uniquely assembles P at any scale requires $\Omega(K_P^{\frac{1}{3}})$ states.*

Proof. We use the same argument for this proof but show the system can be encoded more efficiently. We can store our transition rules in a $\mathcal{O}(|\Sigma|^3)$ bit table. This is a 3D table where each cells stores 4 bits. The first two indices representing the starting states and the third is the target state. There is only one state since single transition rules only change one rule at a time. The table stores 4 bits in order to store whether they transitions vertically or horizontally, and whether the first or second tile changes to the other state. $\square$

Theorem 11. *For any Pattern P over constant colors, a Tile Automata system Γ with deterministic transition rules that uniquely assembles P at any scale requires $\Omega\left((\frac{K_P}{\log K_P})^{\frac{1}{2}}\right)$ states.*

Proof. Deterministic rules can be encoded in $\mathcal{O}(|\Sigma|^2 \log |\Sigma|)$ bits. To achieve this, store the rules in a $|\Sigma| \times |\Sigma|$ table where each cell stores up to two other pairs of states which takes $\mathcal{O}(\log |\Sigma|)$ bits. We only need to store a constant number of pairs since each pair of states and orientation can only have a single rule. Note that this method can encode single or double transition rules with only a constant factor difference. Applying similar algebra as done for Theorem 9 we have $|\Sigma| = \Omega\left((\frac{K_P}{\log K_P})^{\frac{1}{2}}\right)$. $\square$

7.2 Affinity Strengthening

In [5] it was shown that the Unique Assembly Verification Problem (UAV) for affinity strengthening Tile Automata is solvable in PSPACE. In Lemma we show that given a binary string $b(|\Gamma|)$ describing a directed Affinity Strengthening Tile Automata system, we can produce a description of the uniquely produced assembly A in $\mathcal{O}(|A|^2 \log^2 |\Sigma|)$ space. We then apply this fact in Theorem 12 to get a state complexity lower bound based on bounded-space Kolmogorov complexity.

Lemma 4. *Given a binary string $b(\Gamma)$ describing a directed Tile Automata system Γ, there exists an algorithm that outputs the uniquely produced assembly $\texttt{TERM}_\Gamma = \{A\}$, in $\mathcal{O}(|A|^2 \log^2 |\Sigma|)$ space.*

Proof. This can be done by making multiple calls to a subroutine that solves the unique assembly verification problem (UAV) for affinity strengthening Tile Automata. For each integer starting at $i = 1$, call the algorithm for UAV on each assembly B of size $|B| = i$. If the UAV algorithm returns yes, then return B since $A = B$, i.e. B is the uniquely produced assembly. Storing one of these assembly take $\mathcal{O}(|A| \log |\Sigma|)$ bits and we only need to have stored one at a time and the largest assembly we store is $|A|$ size.

The exact details of the algorithm are shown in [5]. This algorithm only stores a constant number of assemblies at a time each of up to size $2|A|$. We can store an assembly in $|A| \log |\Sigma|$ bits thus giving our bound. ☐

Theorem 12. *For all Patterns P, scale factor $s > 0$, an Affinity Strengthening Tile Automata system Γ that uniquely assembles P at scale $n \times m$ for $nm = s$ requires $\Omega(KS_P((s|P|)^2 \log^2 s|P|)^{\frac{1}{4}})$ states.*

Proof. We can use the same method for encoding Γ into a binary string $b(\Gamma)$ as done in Theorem 9 to achieve $|b(\Gamma)| = \mathcal{O}(|\Sigma|^4)$. We can pass $b(\Gamma)$ along with an algorithm that outputs the pattern P produced by Γ to the universal Turing Machine M_K. With this we can bound the length of the string, $|b(\Gamma)| \geq KS_P(f(|P|))$ where $f(|P|)$ is the space taken by the algorithm to output P.

From Lemma 4 we know we can output a description of the uniquely produced assembly A in $\mathcal{O}(|A|^2 \log^2 |\Sigma|)$ space and the pattern can be read and output. A naive implementation can give $|\Sigma| \leq |A|$ by assigning each tile a unique state. The size of the assembly is $|A| = s|P|$, so we can bound the space by the scale factor s and the pattern size $|P|$ giving us $\mathcal{O}((s|P|)^2 \log^2 s|P|)$. We therefore get $|\Sigma| = \Omega\left(KS_P((s|P|)^2 \log^2 s|P|)^{\frac{1}{4}}\right)$. ☐

Theorem 13. *For all Patterns P, scale factor $s > 0$, an Affinity Strengthening Tile Automata system Γ with single transition rules that uniquely assembles P at scale $n \times m$ for $nm = s$, requires $\Omega(KS_M(P, s|P|^3)^{\frac{1}{3}})$ states.*

Proof. We may encode a system with single transition rules in $|\Sigma|^3$ bits so we get a bound of $|\Sigma| = \Omega\left(KS_P((s|P|)^2 \log^2 s|P|)^{\frac{1}{3}}\right)$. ☐

Theorem 14. *For all Patterns P, scale factor $s > 0$, an Affinity Strengthening Tile Automata system Γ with deterministic transition rules that uniquely assembles P at scale $n \times m$ for $nm = s$, requires $\Omega\left((\frac{KS(P,|P|^3)}{\log KS_M(P,|P|^3)})^{\frac{1}{2}}\right)$ states.*

Proof. A deterministic Tile Automata system can be encoded $\mathcal{O}(|\Sigma|^2 \log |\Sigma|)$ bits. By performing the same steps as in Theorem 11 we get $|\Sigma| = \Omega\left((\frac{KS(P,|P|^3)}{\log KS_M(P,|P|^3)})^{\frac{1}{2}}\right)$. ☐

8 Conclusion

In this paper we show how to convert any 1D staged assembly system to an equivalent 1D freezing Tile Automata system. We then show how this generalizes some previous results. We then show how a similar techinque can be used to implement CFGs to build patterns. We then described a set of upper and lower bounds for pattern building based on previous work. There are many interesting directions for future work.

- What is the most efficient method to compute the glue-terminal table?
- Can we improve the number of states needed in the TA simulation? Could it be reduced to $\mathcal{O}(st + bt)$ or even $\mathcal{O}(sg + bg)$ where g is the number of glues in the system? What is the lower bound?
- Does allowing for 1D scaling help achieve better bounds?
- Can 1D staged simulate 1D freezing Affinity-Strengthening Tile Automata? I.e., are they equivalent? If so, how many tiles, bins, and stages are needed?
- What challenges arise when attempting to generalize this to 2D? The glue-terminal table must not only store whether or not an assembly is terminal based on its glues, but also its geometry.
- What is the lower bound for building patterns in 1D freezing Affinity-Strengthening Tile Automata? Are there languages that Tile Automata can assemble more efficiently than staged?

Declarations

Ethical Approval

Not applicable.

Competing interests

There are no competing interests that we are aware of in reference to this paper.

Authors' contributions

These authors contributed equally to this work.

Funding

No external funding was received.

Availability of data and materials

Data Availability Statement: No Data associated in the manuscript.

References

[1] Alaniz RM, Caballero D, Cirlos SC, et al (2022) Building squares with optimal state complexity in restricted active self-assembly. In: Proc. of the Symposium on Algorithmic Foundations of Dynamic Networks, pp 6:1–6:18

[2] Alumbaugh JC, Daymude JJ, Demaine ED, et al (2019) Simulation of programmable matter systems using active tile-based self-assembly. In: DNA Computing and Molecular Programming, Cham, DNA'19, pp 140–158

[3] Barad G, Amarioarei A, Paun M, et al (2019) Simulation of one dimensional staged dna tile assembly by the signal-passing hierarchical tam. Procedia Computer Science 159:1918–1927

[4] Benz F, Kötzing T (2013) An effective heuristic for the smallest grammar problem. In: Proc. of the 15th Annual Conf. on Genetic and Evolutionary computation, pp 487–494

[5] Caballero D, Gomez T, Schweller R, et al (2020) Verification and Computation in Restricted Tile Automata. In: 26th Inter. Conf. on DNA Computing and Molecular Programming, pp 10:1–10:18

[6] Caballero D, Gomez T, Schweller R, et al (2021) Covert computation in staged self-assembly: Verification is pspace-complete. In: 29th Annual European Symposium on Algorithms, ESA'21, pp 23:1–23:18

[7] Cannon S, Demaine ED, Demaine ML, et al (2013) Two Hands Are Better Than One (up to constant factors): Self-Assembly In The 2HAM vs. aTAM. In: 30th Inter. Sym. on Theoretical Aspects of Computer Science, pp 172–184

[8] Cantu AA, Luchsinger A, Schweller R, et al (2020) Signal passing self-assembly simulates tile automata. In: 31st Inter. Sym. on Algorithms and Computation, ISAAC'20, pp 53:1–53:17

[9] Chalk C, Luchsinger A, Martinez E, et al (2018) Freezing simulates non-freezing tile automata. In: DNA Computing and Molecular Programming, Cham, pp 155–172

[10] Chalk C, Martinez E, Schweller R, et al (2018) Optimal staged self-assembly of general shapes. Algorithmica 80(4):1383–1409

[11] Chalk C, Martinez E, Schweller R, et al (2019) Optimal staged self-assembly of linear assemblies. Natural Computing 18(3):527–548

[12] Demaine ED, Demaine ML, Fekete SP, et al (2008) Staged self-assembly: nanomanufacture of arbitrary shapes with o (1) glues. Natural Computing 7(3):347–370

[13] Demaine ED, Eisenstat S, Ishaque M, et al (2011) One-dimensional staged self-assembly. In: Proceedings of the 17th international conference on DNA computing and molecular programming, DNA'11, pp 100–114

[14] Longpré L (1986) Resource bounded kolmogorov complexity, a link between computational complexity and information theory. Tech. rep., Cornell University

[15] Schweller R, Winslow A, Wylie T (2019) Verification in staged tile self-assembly. Natural Computing 18(1):107–117

[16] Soloveichik D, Winfree E (2007) Complexity of self-assembled shapes. SIAM Journal on Computing 36(6):1544–1569

[17] Thubagere AJ, Li W, Johnson RF, et al (2017) A cargo-sorting DNA robot. Science 357(6356):eaan6558

[18] Tikhomirov G, Petersen P, Qian L (2017) Fractal assembly of micrometre-scale dna origami arrays with arbitrary patterns. Nature 552(7683):67–71

[19] Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology

[20] Winslow A (2015) Staged self-assembly and polyomino context-free grammars. Natural Computing 14(2):293–302

[21] Woods D, Neary T (2009) The complexity of small universal turing machines: A survey. Theoretical Computer Science 410(4-5):443–450

[22] Woods D, Doty D, Myhrvold C, et al (2019) Diverse and robust molecular algorithms using reprogrammable dna self-assembly. Nature 567(7748):366–372