Verification and Computation in Restricted Tile Automata

David Caballero
Department of Computer Science, University of Texas, Rio Grande Valley, TX, USA
david.caballero01@utrgv.edu

Timothy Gomez
Department of Computer Science, University of Texas, Rio Grande Valley, TX, USA
timothy.gomez01@utrgv.edu

Robert Schweller
Department of Computer Science, University of Texas, Rio Grande Valley, TX, USA
robert.schweller@utrgv.edu

Tim Wylie
Department of Computer Science, University of Texas, Rio Grande Valley, TX, USA
timothy.wylie@utrgv.edu

Abstract

Many models of self-assembly have been shown to be capable of performing computation. Tile Automata was recently introduced combining features of both Celluar Automata and the 2-Handed Model of self-assembly both capable of universal computation. In this work we study the complexity of Tile Automata utilizing features inherited from the two models mentioned above. We first present a construction for simulating Turing Machines that performs both covert and fuel efficient computation. We then explore the capabilities of limited Tile Automata systems such as 1-Dimensional systems (all assemblies are of height 1) and freezing Systems (tiles may not repeat states). Using these results we provide a connection between the problem of finding the largest uniquely producible assembly using n states and the busy beaver problem for non-freezing systems and provide a freezing system capable of uniquely assembling an assembly whose length is exponential in the number of states of the system. We finish by exploring the complexity of the Unique Assembly Verification problem in Tile Automata with different limitations such as freezing and systems without the power of detachment.

2012 ACM Subject Classification Theory of computation → Turing machines; Computer systems organization → Molecular computing; Theory of computation → Problems, reductions and completeness

Keywords and phrases Tile Automata, Turing Machines, Unique Assembly Verification

Digital Object Identifier 10.4230/LIPIcs.DNA.2020.10

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

1 Introduction

Self-assembly systems have quickly become an intense area of research due to fabrication simplicity [13], the ability to create systems at the DNA level [16], the control of nanobots [14], and the maturity of experimental techniques [12]. Self-assembly is a naturally occurring process where simple particles come together to form complex structures. These are computationally of interest since computing at the molecular level yields a lot of power.

There are several models of tile self-assembly, and they each strive to capture some aspect of self-assembling systems. A few of the better known models are the Abstract Tile Assembly Model (aTAM) [24], the 2-Handed Assembly Model (2HAM) [3], the Staged self-assembly model [10], and the Signal-passing Tile Assembly Model (STAM) [19]. There are several other models designed to model different aspects of DNA/RNA or laboratory conditions. A recent model of tile self-assembly, called *Tile Automata* [5], was introduced as an intentional mathematical abstraction designed to implement the key features of active algorithmic self-assembly while avoiding specifics tied to any one particular implementation (using state change rules and tile attachments/detachments based on local affinities between states). By abstracting away implementation details, TA strives to serve as a proving ground for exploring the power of active algorithmic self-assembly, along with providing a central *hub* through which various disparate models of self-assembly can be related by way of comparison to TA. One recent example of this type of application includes [2] in which TA is shown capable of simulating the *Amoebots* model [8] of programmable matter.

Given the goal of TA to connect many models of self assembly, in this paper we explore the computational power of limited Tile Automata systems such as versions of TA that do not allow detachment (not possible in some models). To facilitate this, we first show how to create general Turing Machines, and then we explore the complexity of a common question within self-assembly models: the unique assembly verification problem. If given a system, can the output be guaranteed? This is a natural problem that is polynomial in some models, yet uncomputable in others.

1.1 Previous Work

In his Ph.D. thesis, Winfree presented the Abstract Tile Assembly model (aTAM) and showed it was capable of universal computation by simulating a Turing Machine [24], and the computational power is explored in depth in other works such as [15]. The 2-Handed Assembly Model (2HAM) [3] introduced a more powerful model and is capable of fuel efficient computation [20] along with the Signal-passing Tile Assembly Model [19] which has tiles that can interact to turn glues on or off.

In [10, 25], the authors show a connection between finding the smallest Context Free Grammar and optimization problems in the Staged Assembly model. In the staged assembly model, it was show that while only using a constant number of tile types, a system can construct length-n lines using $\mathcal{O}(\log n)$ bins and mixes [9]. Repulsive forces have been shown to aid in constructing shapes at constant scale [18]. Further, by utilizing the temperature to encode information, shapes can be constructed with constant (or nearly) tile types [6, 22].

The Unique Assembly Verification problem asks if a given system uniquely produces a given assembly. In the aTAM this problem was shown to be solvable in polynomial time [1]. In the 2HAM this problem was shown to be in coNP with certain generalizations being coNP-Complete [3, 21]. In the staged assembly model, this problem is known to be coNP^{NP}-hard and conjectured to be PSPACE-Complete [23]. Adding the power of negative glues also vastly changes the complexity of this problem making in uncomputable in models that include it due to the ability for pieces of assemblies to break off [11]. However, adding negative glues but restricting the ability for assemblies to detach we still see an increase in difficulty with UAV in aTAM without detachment being coNP-complete [4].

The Tile Automata model was introduced in [5] merging ideas from Cellular Automata and Tile Self-Assembly. The authors showed that freezing tile automata (where a tile cannot repeat states) is capable of simulating non-freezing systems. This powerful model has also been shown to be capable of simulating models of programmable matter [2]. Cellular Automata has been shown to be Turing Complete even in 1-dimension [7].

Table 1 Given a Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_a, q_r, q_s)$, simulating Tile Automata systems are given in Theorems 3.4 and 3.5, respectively.

Turing Machine	Tile Automata System	States	Transition Rules
Determinisic	Non-Freezing 1D	$\mathcal{O}(\lvert Q\rvert\lvert\Gamma\rvert)$	$\mathcal{O}(\lvert\delta\rvert)$
Bounded Time	Freezing 1D	$\mathcal{O}(\lvert Q\rvert\lvert\Gamma\rvert TIME(M))$	$\mathcal{O}(\lvert\delta\rvert TIME(M)^2)$

Table 2 Results for the Unique Assembly Verification in Tile Automata. **Transition Rules** describes the types of transition rules allowed in the system. In Affinity Strengthening Systems all transition rules increase affinity so no detachment may occur. **Freezing** indicates whether the system is freezing where tiles cannot repeat states. **Result 1D** is the complexity of UAV in 1 Dimension and **Result 2D** is the complexity of 2 Dimensions. **Theorem** is where these can be found. *This result is only true when cycles in the production graph are allowed. All other results are true regardless of which definition is used.

Transition Rules	Freezing	1D Result	2D Result	Theorem
Affinity Strengthening	Freezing	coNP-hard	coNP$^{\text{NP}}$-Complete	Thms. 6.8, 6.7
Affinity Strengthening	Non-freezing	PSPACE-Complete	PSPACE-Complete	Thm. 6.3
General	Freezing	Open	Undecidable	Thm. 5.2*
General	Non-freezing	Undecidable	Undecidable	Thm. 5.1

1.2 Our Contributions

In Tile Automata, cases may occur where systems contain one terminal assembly but exhibit behavior that does not naturally seem to uniquely produce that assembly. We define unique assembly later, but note that the final requirement addresses a feature of Tile Automata and other models with detachment where there exist assemblies that are not terminal but are never part of the final assembly. Cycles in the production graph are not possible in many self-assembly models so we add this restriction. However many of our results work with or without this restriction, so we explore both cases.

In this work we explore Tile Automata systems that uniquely assemble n-length lines and the complexity of determining whether a system uniquely assembles a given assembly. We first present a Turing Machine simulation capable of covert and fuel-efficient computation. We use this construction to show a connection between the largest finite assembly problem and Busy Beaver Machines (Turing Machines that print a certain number of symbols using a minimum number of states). In the more restricted case of Freezing Systems we show we can construct n-length lines using $\mathcal{O}(n)$ states. Results are shown in Table 1.

We then explore the Unique Assembly Verification problem. An overview of the results are shown in Table 2. We show that UAV is uncomputable via Turing Machine simulation. We also extend this to 2-Dimensional freezing systems (this reduction results in a system with cycles). By removing the ability for assemblies to break apart we achieve a model closer to traditionally studied models. We restrict this by studying what we call *Affinity-Strengthening* systems where a state can never lose affinity by a transition. In this case, we show the UAV problem is PSPACE-Complete utilizing bounded-space Turing Machine simulation. When restricting the model to both Affinity Strengthening and Freezing we show membership in coNP$^{\text{NP}}$. We then provide reductions to show coNP$^{\text{NP}}$-completeness for 2-dimensional UAV and coNP-hardness in 1 dimension.

2 Model and Definitions

A Tile Automata system is a marriage between cellular automata and 2-handed self-assembly. Systems consist of a set of monomer tile states, along with local affinities between states denoting the strength of attraction between adjacent monomer tiles in those states. A set of local state-change rules are included for pairs of adjacent states. Assemblies (collections of edge-connected tiles) in the model are created from an initial set of starting assemblies by combining previously built assemblies given sufficient binding strength from the affinity function. Further, existing assemblies may change states of internal monomer tiles according to any applicable state change rules. An example system is shown in Figure 1.

2.1 States, tiles, and assemblies

Tiles and States. Consider an alphabet of *state types*[1] Σ. A tile t is an axis-aligned unit square centered at a point $L(t) \in \mathbb{Z}^2$. Further, tiles are assigned a state type from Σ, where $S(t)$ denotes the state type for a given tile t. We say two tiles t_1 and t_2 are of the same *tile type* if $S(t_1) = S(t_2)$.

Affinity Function. An *affinity function* takes as input an element in $\Sigma^2 \times D$, where $D = \{\perp, \vdash\}$, and outputs an element in $\mathbb{N}$. This output is referred to as the *affinity strength* between two states, given direction $d \in D$. Directions $\perp$ and $\vdash$ indicate above-below and side-by-side orientations of states, respectively.

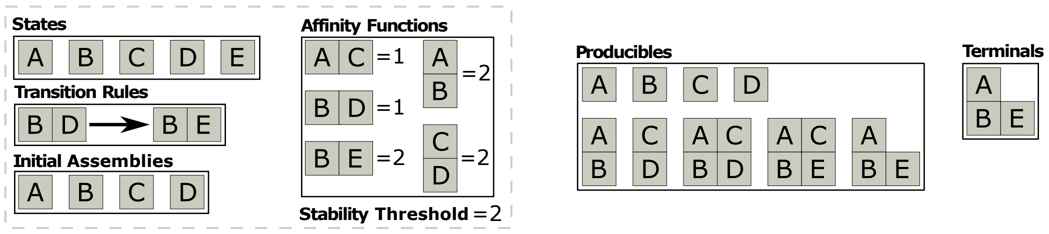
Transition Rules. Transition rules allow states to change based on their neighbors. A *transition rule* is a 5-tuple $(S_{1a}, S_{2a}, S_{1b}, S_{2b}, d)$ with each $S_{1a}, S_{2a}, S_{1b}, S_{2b} \in \Sigma$ and $d \in D = \{\perp, \vdash\}$. ($S_{1a}$ and S_{1b} being the left state or the top state.) Essentially, a transition rule says that if states S_{1a} and S_{2a} are adjacent to each other, with a given orientation d, they can transition to states S_{1b} and S_{2b} respectively.

Assemblies. A *positioned shape* is any subset of $\mathbb{Z}^2$. A *positioned assembly* is a set of tiles at unique coordinates in $\mathbb{Z}^2$, and the positioned shape of a positioned assembly $\mathcal{A}$ is the set of coordinates of those tiles, denoted as $\texttt{SHAPE}_{\mathcal{A}}$. For a positioned assembly $\mathcal{A}$, let $\mathcal{A}(x, y)$ denote the state type of the tile with location $(x, y) \in \mathbb{Z}^2$ in $\mathcal{A}$.

For a given positioned assembly $\mathcal{A}$ and affinity function Π, define the *bond graph* $G_{\mathcal{A}}$ to be the weighted grid graph in which:

- each tile of $\mathcal{A}$ is a vertex,
- no edge exists between non-adjacent tiles,
- the weight of an edge between adjacent tiles T_1 and T_2 with locations (x_1, y_1) and (x_2, y_2), respectively, is
  - $\Pi(S(T_1), S(T_2), \perp)$ if $y_1 > y_2$,
  - $\Pi(S(T_2), S(T_1), \perp)$ if $y_1 < y_2$,
  - $\Pi(S(T_1), S(T_2), \vdash)$ if $x_1 < x_2$,
  - $\Pi(S(T_2), S(T_1), \vdash)$ if $x_1 > x_2$.

[1] We note that Σ does not include an "empty" state. In tile self-assembly, unlike cellular automata, positions in $\mathbb{Z}^2$ may have no tile (and thus no state).



(a) Tile Automata System Γ. **(b)** The producibles and terminals of Γ.

Figure 1 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 to be terminal.

A positioned assembly $\mathcal{A}$ is said to be *τ-stable* for positive integer τ provided the bond graph $G_{\mathcal{A}}$ has min-cut at least τ.

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

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

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

Transitionable Assemblies. Consider some set of transition rules Δ. An assembly A is *transitionable*, with respect to Δ, into assembly B if and only if there exist $\mathcal{A} \in A$ and $\mathcal{B} \in B$ such that for some pair of adjacent tiles $t_i, t_j \in \mathcal{A}$:

- $\exists$ a pair of adjacent tiles $t_h, t_k \in \mathcal{B}$ with $L(t_i) = L(t_h)$ and $L(t_j) = L(t_k)$
- $\exists$ a transition rule $\delta \in \Delta$ s.t. $\delta = (S(t_i), S(t_j), S(t_h), S(t_k), \perp)$ or $\delta = (S(t_i), S(t_j), S(t_h), S(t_k), \vdash)$
- $\mathcal{A} - \{t_i, t_j\} = \mathcal{B} - \{t_h, t_k\}$

2.2 Tile Automata model (TA)

A *tile automata system* is a 5-tuple $(\Sigma, \Pi, \Lambda, \Delta, \tau)$ where Σ is an alphabet of state types, Π is an affinity function, Λ is a set of initial assemblies with each tile assigned a state from Σ, Δ is a set of transition rules for states in Σ, and $\tau \in \mathbb{N}$ is the *stability threshold*. When the affinity function and state types are implied, let (Λ, Δ, τ) denote a tile automata system. An example tile automata system can be seen in Figure 1.

▶ **Definition 2.1** (Tile Automata Producibility). *For a given tile automata system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$, the set of* producible assemblies *of Γ, denoted $\texttt{PROD}_\Gamma$, is defined recursively:*
- *(Base) $\Lambda \subseteq \texttt{PROD}_\Gamma$*
- *(Recursion) Any of the following:*
  - *(Combinations) For any $A, B \in \texttt{PROD}_\Gamma$ such that A and B are τ-combinable into C, then $C \in \texttt{PROD}_\Gamma$.*
  - *(Breaks) For any $C \in \texttt{PROD}_\Gamma$ such that C is τ-breakable into A and B, then $A, B \in \texttt{PROD}_\Gamma$.*
  - *(Transitions) For any $A \in \texttt{PROD}_\Gamma$ such that A is transitionable into B (with respect to Δ), then $B \in \texttt{PROD}_\Gamma$.*

For a system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$, we say $A \to_1^\Gamma B$ for assemblies A and B if A is τ-combinable with some producible assembly to form B, if A is transitionable into B (with respect to Δ), if A is τ-breakable into assembly B and some other assembly, or if $A = B$. Intuitively this means that A may grow into assembly B through one or fewer combinations, transitions, and breaks. We define the relation $\to^\Gamma$ to be the transitive closure of $\to_1^\Gamma$, i.e., $A \to^\Gamma B$ means that A may grow into B through a sequence of combinations, transitions, and/or breaks.

▶ **Definition 2.2** (Production Graph). *The production graph of a Tile Automata system Γ is a directed graph where each vertex corresponds to an assembly in $\texttt{PROD}_\Gamma$ and there exists a directed edge between assemblies A and B if $A \to^\Gamma B$.*

▶ **Definition 2.3** (Terminal Assemblies). *A producible assembly A of a tile automata system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$ is* terminal *provided A is not τ-combinable with any producible assembly of Γ, A is not τ-breakable, and A is not transitionable to any producible assembly of Γ. Let $\texttt{TERM}_\Gamma \subseteq \texttt{PROD}_\Gamma$ denote the set of producible assemblies of Γ which are terminal.*

▶ **Definition 2.4** (Freezing). *Consider a tile automata system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$ and a directed graph G constructed as follows:*
- *each state type $\sigma \in \Sigma$ is a vertex*
- *for any two state types $\alpha, \beta \in \Sigma$, an edge from α to β exists if and only if there exists a transition rule in Δ s.t. α transitions to β*

Γ *is said to be* freezing *if G is acyclic and* non-freezing *otherwise. Intuitively, a tile automata system is freezing if any one tile in the system can never return to a state which it held previously. This implies that any given tile in the system can only undergo a finite number of state transitions.*

▶ **Definition 2.5** (Affinity Strengthening). *An Affinity-Strengthening system is a Tile Automata system where all transition rules can only increase a states affinity with all other states so no detachments ever occur. Formally a tile automata system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$ is an Affinity Strengthening system if for each $s, s' \in \Sigma$ where s transitions to s', $\Delta(s,t) \leq \Delta(s',t) \forall t \in \Sigma$.*

▶ **Definition 2.6** (Bounded). *A tile automata system Γ is* bounded *if and only if there exists a $k \in \mathbb{Z}_{>0}$ such that for all $A \in \texttt{PROD}_\Gamma$, $|A| < k$.*

▶ **Definition 2.7** (Unique Assembly). *A Tile Automata system Γ* uniquely produces *an assembly A if*
- *A is the only assembly in $\texttt{TERM}_\Gamma$*
- *for all $B \in \texttt{PROD}_\Gamma$, $B \to^\Gamma A$.*
- *Γ is bounded.*
- *there does not exist a pair of assemblies $B, C \in \texttt{PROD}_\Gamma$, such that $B \to^\Gamma C \to^\Gamma B$.*[2]

[2] When we refer to Unique Assembly allowing cycles, this requirement is omitted.

3 One Dimensional Turing Machine

Since Tile Automata is a generalization of 2HAM and borrows from Cellular Automata it is expected that it is as powerful as both of these models. Here we present a construction that is capable of both covert and fuel-efficient computation. We present informal definitions of each of these. For rigorous definitions, we refer the reader to [20, 19] for fuel-efficiency, and [4] for covert computation.

▶ **Definition 3.1** (Simulation). *A Tile Automata system T is said to simulate a Turing Machine M, if for every producible assembly a of T can be mapped to a configuration m of M and any other producible assembly b such that $a \rightarrow_1^\Gamma b$, b either also maps to m or maps to another configuration m' such that m' is the next step of m. Finally, each terminal assembly of T maps to an output of M.*

▶ **Definition 3.2** (Covert Computation). *Given a Tile Automata system T that simulates a Turing Machine M, T covertly simulates M if for each output of M, there exits a single terminal assembly that maps to it.*

▶ **Definition 3.3** (Fuel Efficient Computation). *A fuel efficient Turing machine simulation in Tile Automata represents the tape of a Turing machine as one assembly, and requires that each computational step of the Turing machine occurs by way of the attachment of at most a constant number of assemblies of at most constant size. Thus, the simulation of n steps of a computation "uses up" at most $O(n)$ tiles worth of fuel.*

▶ **Theorem 3.4.** *For any Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_a, q_r, q_s)$, there exists a covert, fuel-efficient, 1-dimensional Tile Automata system $T = (\Sigma_{TA}, \Pi, \Lambda, \Delta)$[3] that can simulate M such that $|\Sigma_{TA}| = \mathcal{O}(|Q||\Gamma|)$ and $|\Delta| = \mathcal{O}(|\delta|)$.*

Proof. Given a Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_a, q_r, q_s)$, we construct the Tile Automata system $T = (\Sigma_{TA}, \Pi, \Lambda, \Delta)$ as follows.

States. Conceptually, we partition the set of states (Σ_{TA}) into three subsets for clarity: *head* states $\mathcal{H}$, *symbol* states $\mathcal{S}$, and *utility* states $\mathcal{W}$. Let $\mathcal{H} = \{h_{(q,s)} | q \in Q, s \in \Sigma\}$ and let $\mathcal{S} = \{\sigma_s | s \in \Sigma\}$ (Figure 2a). All states in $\mathcal{H}$ and $\mathcal{S}$ have affinity with all states in Σ_{TA}. There are eight states in $\mathcal{W}$: signal accept states, final accept states, signal reject states, final reject states, and four buffer states B_L, B'_L, B_R, and B'_R. The *signal accept state* has affinity with all states in Σ_{TA}, and the *final accept state* has affinity with all states other than itself and the four buffer states. The two reject states have corresponding affinity rules as those of the accept states. The buffer states ensure that no two assemblies attach during the computation. Each of the four buffer states have affinity with each state in $\mathcal{H}$ and $\mathcal{S}$. B_L and B_R have affinity with B'_L or B'_R respectively.

Transitions. We create a transition rule such that for each Tile Automata state $h_{(q,s)} \in \mathcal{H}$ and $\sigma_i \in \mathcal{S}$, the rule represents a step in M (Figure 2b). WLOG, assume an assembly A representing the a configuration of a Turing Machine M has the state $h_{(q,s)}$ with states, $\sigma_L, \sigma_R \in \mathcal{S}$ to the left and right of $h_{(q,s)}$, respectively. If the head of M moves right then the transition rule will take place between $h_{(q,s)}$ and σ_R. If the TM head moves left then the transition rule will be between σ_L and $h_{(q,s)}$. $h_{(q,s)}$ will transition into the state representing

[3] 1-Dimensional Tile Automata systems always have $\tau = 1$ so we omit that parameter from T

the symbol that is to be written on the tape in M after a state q reads symbol s. Either σ_L or σ_R would then transition into the state $h_{(q',\sigma_L)}$ or $h_{(q',\sigma_R)}$ respectively where q' is the new state of the head of M after reading s from state q. There also exists an additional transition rule if σ_L or σ_R is a buffer state. This will transition B_L or B_R to state B'_L or B'_R respectively. B'_L/B'_R transitions into the symbol state representing the blank symbol when it is to attached to state B_L/B_R.

Accept/Reject. For transitions where M enters the accept state, we create transition rules where both tiles enter the signal accept state. This state has transition rules with each other state transitioning that state into the signal accept state as well. If it transitions with a buffer state or the final accept state, both tiles enter the final accept state. The final accept state also transitions with every other state and both tiles become the final accept state. The reject states follow the same rules.

Input. We construct a Tile Automata system that runs M on a string x. We construct the system as described and create an initial assembly A that represents x. A will have a length of $|x| + 2$. The left most state of A will be B_L. (WLOG assume the head of M starts on the left most cell.) The next state of A will be $s_{(q,s)}$ where q is the initial state of M and s is the first symbol in x. The next states of A each represent the symbols in the string x in order. The rightmost state of A is B_R (Figures 2c, 2d).

The buffer states B_L and B_R are always an initial assembly and are used to extend the tape if the head attempts to move past the right edge. First, the head state causes B_R to transition to B'_R. With B'_R on the edge of the assembly a new B_R tile will attach. Once this attachment occurs B'_R transitions to the symbol state representing the blank symbol on the tape. Then the head state may transition with the blank symbol if needed. The same process occurs with B_L when the head attempts to move off the left end of the tape.

Terminal Assemblies. If M accepts the input x, then by the rules of our system the accept states will appear in our assembly. The signal accept state will be the first to appear and will propagate to the edges of the assembly. Once the signal accept state reaches the buffer states on the edge of the assembly they will transition into the final accept states. Any final accept state that is attached to any other state will make that tile into a final accept state. Any two final accept states that are next to each other do not have affinity and will detach. After the accept state appears in an assembly the only terminal assemblies that will exist are single final accept states. The same will occur if the machine rejects.

Since there are only two possible terminal assemblies, the final accept state and the final reject state, this construction performs covert computation. This computation is also fuel efficient since the only time a new assembly is attached is when the Turing Machine writes on a blank symbol at the edge of the tape, which can only occur once per computation step. ◀

3.1 Freezing Systems

Here we present modifications to the construction above for freezing 1-dimensional systems to perform bounded time computation.

▶ **Theorem 3.5.** *For any bounded-time Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_a, q_r, q_s)$, there exists a covert, fuel-efficient, 1-dimensional freezing Tile Automata system $T = (\Sigma_{TA}, \Pi, \Lambda, \Delta)$ that can simulate M such that. $|\Sigma_{TA}| = \mathcal{O}(|Q||\Gamma|TIME(M))$ and $|\Delta| = \mathcal{O}(|\delta|TIME(M)^2)$.*

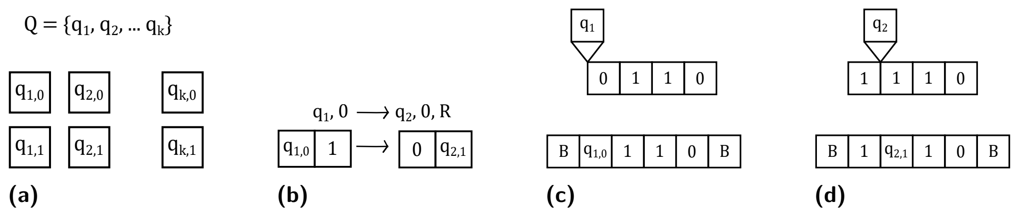


Figure 2 (a) Tile automata states (Below) created from the states of Turing Machine (Above) over a binary alphabet. (b) State change rules (Below) created from the Turing Machine transition rules (Above). (c) A Turing Machine (Above) configuration and the representative TA assembly (Below) . (d) The same Turing Machine (Above) after making one step and the assembly (Below) after the same step.

Proof. We modify the construction from Theorem 3.4. We have Σ_{TA} partitioned into three sets $\mathcal{H}$, $\mathcal{S}$, and $\mathcal{W}$. In a freezing system states can not be repeated, so for each state in $\mathcal{H}$ and $\mathcal{S}$ we create a number of states equal to the number of steps the Turing Machine M can take. Each head state will not only represent the state of the Turing machine and the symbol on the tape, but it will also represent how many steps the Turing Machine has taken. Each symbol state will represent the symbol on the tape and also the last step that it was modified. The head states will have a transition rule with each symbol state regardless of the last step that symbol was modified. When a head state transitions into a symbol state it will represent the step that the transition took place.

This increase in state-space ensures no tile will ever become the same state twice. Symbol states written at step x can only transition into a head state. The head state will always represent a step $y > x$. When the head state transitions back to a symbol state it will go to a symbol state written at state y. Since $x < y$, no tile will ever repeat states. ◀

4 Shapebuilding and the Largest Assembly Problem

Given a Tile Automata system with limited states, we examine how large of an assembly may be constructed. We first consider the case of one-dimensional assemblies and leverage Theorems 4.2 and 4.3 to show that the longest buildable line's length is related to the Busy Beaver function in general, and exponential in the case of freezing systems. We then consider the Largest Assembly problem, and apply Theorem 4.3 to show that this problem is uncomputable for general TA even in one-dimension.

4.1 General

The Busy Beaver function $BB(n)$, for any positive integer n, is the maximum number of symbols printable by a Turing Machine using n states.[4]

▶ **Definition 4.1** (String Representation). *An assembly A is said to represent a string x if there exists a mapping of the states in A to the symbols in x such that the n^{th} state of A maps to the n^{th} symbol of x for all $0 < n \leq |x|$*

▶ **Lemma 4.2.** *For any n-state 2-symbol (not including the blank symbol) Turing Machine M which produces an output x, there exists a $\mathcal{O}(n)$-state Tile Automata System T which uniquely assembles an assembly A, such that A represents x.*

[4] For this definition we consider Turing Machines using a binary alphabet.

Proof. We modify the construction from Theorem 3.4 so that once M halts the head state transitions into a symbol state. The resulting assembly will be terminal since symbol states do not transition with each other. This final assembly will consist of symbol states that each represent the symbols in x. The number of states used by T is $2n$ head states, 2 symbol states, and 4 buffer states which is bounded by $\mathcal{O}(n)$. Note there is no need for accept/reject states since the head state just turns into a symbol state when the TM halts. ◀

▶ **Theorem 4.3.** *For any positive integer n, there exists a 1-dimensional Tile Automata system that uniquely assembles a $BB(n)$-length line using $\mathcal{O}(n)$ states.*

Proof. Using Lemma 4.2 we can take any Busy Beaver Machine and create a Tile Automata system which uniquely produces an assembly the same size as the number of symbols printed on the tape. ◀

4.2 Freezing

For freezing Tile Automata systems, we can create systems that uniquely produce n-length lines and only require states that are logarithmic in the length of the line. For clarity we begin with a helping lemma.

▶ **Lemma 4.4.** *For all $n = 2^x$ for $x \in \mathbb{N}$, there exists a 1-dimensional freezing Tile Automata system that uniquely assembles an n length line using $\mathcal{O}(\log n)$ states.*

Proof. The cases for $x = 0, 1, 2$ are trivial. A system that uniquely builds a length 2^3 line is shown in Figure 3. The only initial states are 1_A and 1_B. The affinities are between adjacent states. The transition rules are highlighted in red which transition to make the next producible assembly depicted. Our unique terminal assembly is a length 2^3 line. We will show that by adding a constant number of states, transitions, and affinities to this system the length of the uniquely assembled line will double, and that this process can be repeated to uniquely assemble any length 2^n line.

For $n > 3$, Let T_n be the system that uniquely assembles a length 2^n line derived by recursively applying the following process to T_3 $n-3$ times. Assuming that T_n uniquely assembles a length 2^n line of the form $(1_A, n_D, \dots, n_D, n_A, n_B, n_F, \dots, n_F, 1_B)$, T_{n+1} is constructed as follows. First we add the non-initial states $n+1_A, \dots, n+1_F$, and a transition from (n_A, n_B) to both $(n+1_E, n_B)$ and $(n_A, n+1_C)$. We add six new transitions involving $n+1_C$ or $n+1_E$ which allow that state to propagate left/right respectively and transition to $n+1_D$ and $n+1_F$ respectively when the end to the line assembly is reached. There will be 6 additional transition rules added to allow states $n+1_D$ and $n+1_F$ to propagate in the opposite direction and eventually transition 1_A and 1_B to $n+1_B$ and $n+1_A$ respectively. Adding the affinity rule $(n+1_A, n+1_B)$ will allow the two length 2^n lines to bond uniquely assembling a length 2^{n+1} line. This new system uniquely produces a length 2^{n+1} line of the same form previously described, to which the process can be repeated to once again double the length of the unique assembly. ◀

▶ **Theorem 4.5.** *For all positive integers n, there exists a 1-dimensional freezing Tile Automata system that uniquely assembles an n length line using $\mathcal{O}(\log n)$ states.*

Proof. We modify the construction from Lemma 4.4 to build arbitrary length-n lines.

To build any length-n line using $\mathcal{O}(\log n)$ states we modify $T = T_{\lceil \log_2 n \rceil}$. Let b_i indicate the i^{th} least significant bit of n's binary expansion. For all $i > 2$ such that b_i is equal to 1 we add a transition rule from (i_A, i_B) to (i_L, i_L) in T. When these two states are adjacent

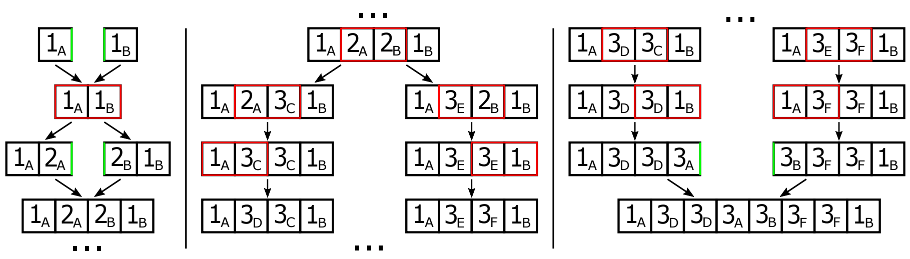


Figure 3 A system that uniquely builds a length 2^3 line. The only initial states are 1_A and 1_B. The affinities are between adjacent states. The transition rules are highlighted in red which transition to make the next producible depicted.

they exist in an assembled line of length 2^i. This transition "locks" this producible, stopping it from growing. Four more transition rules are added to allow this state to propagate to the ends of the line. Finally, we add a transitions between all i_L states and the states 1_B and 1_A, which are the endpoints of the lines. These endpoints transition to states that have affinity with the next largest locked producible on one side. If b_1 or b_2 is equal to 1 we add in an assembly of size $b_1 \times 1 + b_2 \times 2$ that connects to the last locked producible. ◀

4.3 Largest Finite Assembly Problem

Given a positive integer n, the Largest Finite Assembly Problem asks what is the largest assembly that can be uniquely assembled in a Tile Automata system using n states.

▶ **Theorem 4.6.** *The Largest Finite Assembly problem in Tile Automata is uncomputable.*

Proof. Let σ_n be the size of the largest assembly that can be constructed using n states. From Theorem 4.3, there must exists a system that can construct a line of length $BB(n)$ using $\mathcal{O}(n)$ states so $\sigma_{\mathcal{O}(n)} \geq BB(n)$. This means σ_n grows asymptotically as fast as the Busy Beaver function, which grows faster than any computable function. Thus, σ_n is uncomputable. ◀

5 Unique Assembly Verification

A well-studied problem in self-assembly is the Unique Assembly Verification problem. This asks whether a given system uniquely produces a given assembly. We show that the general problem is undecidable. Again, we consider two definitions of Unique Assembly one where systems with cycles are allowed in the production graph, and the other where they are not.

5.1 Undecidability

▶ **Theorem 5.1.** *Tile Automata Unique Assembly Verification is undecidable even in one dimension.*

Proof. Using Theorem 3.4 we reduce from the halting problem. Given a Turing Machine M we can construct a Tile Automata system Γ that simulates M. If M halts then there exists a single terminal assembly which is the final accept state tile. If M does not halt then there exists no terminal assemblies. This is true under both definitions of Uniquely Assembly since the only time there would exist a cycle in the production graph of Γ is if M ever revisited a configuration. If M revisits a configuration then M will not halt so our system will not uniquely assemble the final accept state tile. ◀

▶ **Theorem 5.2.** *Freezing 2-Dimensional Tile Automata Unique Assembly Verification is undecidable under the definition of Unique Assembly allowing cycles even when all assemblies are of constant height.*

Proof. To prove undecidability we reduce from UAV for 1-Dimensional Tile Automata systems (Theorem 5.1). Given an instance of UAV asking if a system Γ uniquely produces an assembly A we use the simulation provided in [5] to create a freezing Tile Automata system Γ'. By the definition of Γ' simulating Γ if $\texttt{TERM}_{\Gamma}$ only contains one terminal assembly A then $\texttt{TERM}'_{\Gamma}$ will only contain one assembly A' that maps to A.

The simulation utilizes constant scale macroblocks to represent tiles so the height of the assemblies in T will be constant height. This simulation also uses a token passing scheme that results in cycles in the production graph so this system will not uniquely produce assemblies if cycles are not allowed. ◀

6 Affinity Strengthening UAV

Many self-assembly models where UAV is well-studied do not have detachment (and are thus decidable). Here, we investigate versions of TA without this power and show hardness. We do this by exploring Affinity-Strengthening Tile Automata (ASTA). We start by considering the non-freezing case, then consider the added restriction of freezing.

6.1 Non-Freezing

▶ **Lemma 6.1.** *The Unique Assembly Verification problem in Affinity-Strengthening Tile Automata is in PSPACE.*

Proof. The UAV problem can be solved by the following co-nondeterministic algorithm. Given an Assembly A and an ASTA system T, nondeterministically build an assembly B of less than size $2|A|$ where $|A|$ is the size of the given assembly. We now have a branch for every producible assembly and we check the following about B in order. If any branch rejects, the whole algorithm rejects.

- If $B = A$, accept.
- If $|B| \geq |A|$, reject.
- If $B \neq A$ and B is terminal, reject.
- Continue nondeterministically performing construction steps (attachments and transitions) on B. If B is reached again, reject. If A is reached, accept.

Only assemblies up to size $2|A|$ can be checked since if any assembly exists larger than $2|A|$, it would have been built using at least one assembly of size greater than $|A|$, which would have already been rejected. We can also check if B is terminal using a nondeterministic subroutine by non-deterministically building a second assembly and checking if it can attach to B. Checking if an assembly is breakable or if it is transitionable can be done in polynomial time and space. The final step of the algorithm checks for cycles in the production graph. By the definition of unique assembly, $B \rightarrow^{\Gamma} A$, by continuing to perform construction steps on B we will eventually reach A. If we ever end up reaching B again we know that there exists a cycle in the production graph (cycle checking in a directed graph is in P).

This algorithm shows the UAV problem for Affinity-Strengthening Tile Automata is in coNPSPACE which equals PSPACE. For the case of unique assembly where cycles in the production graph are allowed, the last step of the algorithm is skipped. ◀

▶ **Lemma 6.2.** *The Unique Assembly Verification problem in Affinity-Strengthening Tile Automata is PSPACE-hard.*

Proof. We show UAV in Affinity-Strengthening TA is PSPACE-hard by describing how to reduce from any problem $L \in$ PSPACE. Consider a Turing Machine M that decides L. The construction from Theorem 3.4 can be modified to be an Affinity-Strengthening system that results in a system capable of performing bounded space computation (a Linear Bounded Automata, which is equivalent to parsing a context-sensitive grammar and is PSPACE-complete [17]). The only transition where a state loses affinity is from the signal accept and reject state to the final accept and reject state. We remove the final states from the system. This will result in two possible terminal assemblies one consisting of a buffer state, then accept states, then another buffer state, and the other being the same with reject states. We remove the buffer state from the set of initial assemblies. We change the length of the assembly representing the input to be the amount of space used by M.

Given a bounded space deterministic Turing machine and its input, construct a Tile Automata system that uniquely produces the assembly with accept states if and only if the Turing machine accepts. If the Turing Machine rejects, then the reject assembly will be the only terminal assembly. If the TM ever enters an infinite loop then there will exist a cycle in our system and there will not exist any terminal assemblies, so the TA system will not uniquely produce any assembly regardless of whether there exists a restriction on cycles. ◀

▶ **Theorem 6.3.** *The Unique Assembly Verification problem in Affinity-Strengthening Tile Automata is PSPACE-complete.*

Proof. Follows from Lemmas 6.1 and 6.2. ◀

6.2 Freezing

In this section we show the complexity of Unique Assembly Verification in a freezing Affinity-Strengthening Tile Automata system. In 2-dimensions, we show UAV is coNP$^{\text{NP}}$-Complete. We utilize the same reduction strategy as in [23]. We conclude by showing coNP-hardness for UAV in one dimension. Note that cycles cannot occur in Freezing Affinity-Strengthening Tile Automata, so we only consider one definition of Unique Assembly.

▶ **Definition 6.4** (∀∃3SAT). *Given a 3SAT formula $\phi(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$, is it true that for every assignment to variables $x_1, \ldots, x_k$, there exists an assignment to $x_{k+1}, \ldots, x_n$ such that $\phi(x_1, \ldots, x_n)$ is satisfied?*

▶ **Lemma 6.5.** *The Unique Assembly Verification problem in freezing Affinity-Strengthening Tile Automata is in coNP^{NP}.*

Proof. Take the construction and algorithm from Lemma 6.1, we prove that the running time is polynomial. When building an assembly B, since the system is freezing we know the time to build B is $|\Sigma||B|$ where $|\Sigma|$ is the number of states in the system. Since we reject if one branch rejects, this is a coNP algorithm.

We utilize one subroutine that is in coNP to check if B is terminal. This is done in polynomial time by nondeterministically building a second assembly and checking if they can attach. If there is an assembly that can attach to B, then the assembly is not terminal. Using the coNP algorithm and using the subroutines as oracles, this problem is in coNP$^{\text{NP}}$ ◀

▶ **Lemma 6.6.** *The Unique Assembly Verification problem in freezing Affinity-Strengthening Tile Automata is coNP^{NP}-Hard.*

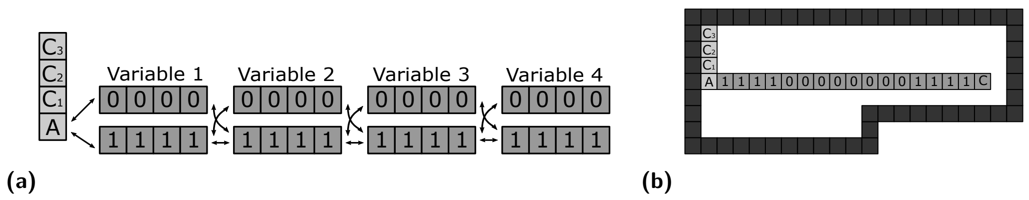


Figure 4 Part of the construction for Theorem 6.6. (a) The base assemblies are constructed nondeterministically. One is constructed for every possible variable assignment. (b) An example of a base assembly fitting into a frame. C_x binds cooperatively to C_{x-1} and the frame states.

Proof. Given an instance of ∀∃3-SAT, this reduction produces a $\tau = 2$ freezing ASTA system which uniquely assembles a target assembly if and only if the instance of ∀∃3-SAT is true. This system has stability threshold 2 to allow for *cooperative binding* in which two assemblies attach using affinities at two separate points, when one of the affinities alone would not be strong enough for this attachment to be stable.

Overview. We first create an 'L'-shaped base assembly contained in a larger frame (Figure 4b) that encodes a variable assignment. Rows of this assembly represent clauses and columns represent variables. Each clause is evaluated by cooperatively placing tiles that represent the assignment of the variable in its column, and whether the clause of its row is currently satisfied. Once the assignments are evaluated, additional tiles fill out the rest of the frame. If the assignment evaluates to false, then frame will be filled. If the assignment evaluates to true, then there will be remaining spaces representing the assignment to the variables in the first quantifier. We construct a test assembly for every possible assignment to thee variables that can attach into that space. Once an assembly has completely filled out its frame, all states inside transition into a target state and create our target assembly.

Base Assemblies. We construct a rectangular base assembly for every possible variable assignment to $x_1, \ldots, x_n$, with the rows of this assembly representing clauses and columns representing variables. There are two sets of initial states for each variable: one for 0, and one for 1. These sets of states attach to form length-4 line assemblies. The line assemblies have affinities with both the 0 and 1 line assemblies of the next variable. The nondeterministic nature of the model will ensure the creation of all possible combinations of these 0 and 1 line assemblies (Figure 4a). Given m clauses in our 3SAT formula, the TA system includes tiles with initial states $C_1, \ldots, C_m$. These states cooperatively attach to state A and a frame (Figure 4b). The frame ensures there is no unbounded growth. Tiles then cooperatively bind to fill out this structure. The affinities between these states and the variable line assemblies are encoded such that they evaluate if the variable assignment, represented by the base assembly, satisfies the 3SAT formula (Figure 5a). The row containing C_i evaluates whether the i^{th} clause is satisfied by the variable assignment of the base. U and S states cooperatively attach to fill out a row- U indicating the clause has not yet been satisfied, and S indicating that it has. This is done by "passing" the assignment of the variable line upwards with a specific encoding of the affinities. When an S state attaches, only S states can attach to its right side. This allows a Y state to attach at the end of the row if a previous clause was not already evaluated to be unsatisfied. If it is not satisfied, the rightmost state of that row will be N, which does not allow a Y state to attach above it.

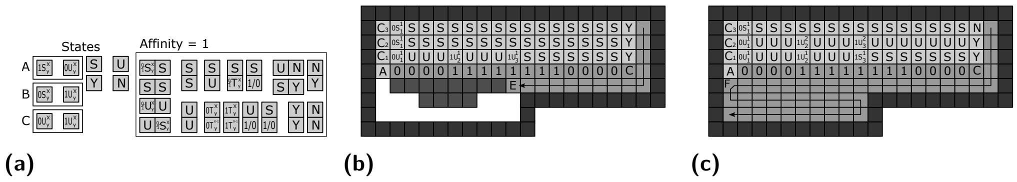


Figure 5 (a) Initial states needed to evaluate if the variable assignment satisfies the 3SAT formula. Choose 1 from A/B/C for each clause/variable combination. Choose A if 1 assigned to variable y satisfies the x^{th} clause, B if 0 satisfies, and C if the variable does not appear in that clause. T is a placeholder for U or S, depending on which was chosen for each clause/variable combination. (b) Example of a 4-variable, 3-clause base assembly that is marked as true (top right “Y”). The assembly grows downward, but interacts with the variable tile line to encode their variable assignment in the assembly’s geometry. (c) Example of a 4-variable 3-clause base assembly marked as false (top right “N”). The assembly grows to fill out the entire frame.

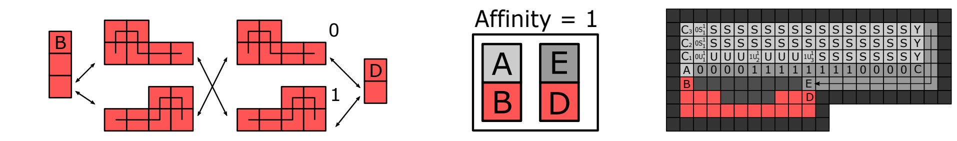
Once the rectangle is filled out an assembly will be marked as “True” or “False”, represented by the top right Y/N state in the construction. (Figure 5b, 5c). True assemblies grow downward, leaving a space between the base assembly and the frame. The shape of this space is an encoding of this assembly’s original variable assignment of $x_1, \ldots, x_k$ (Figure 5b). False assemblies also grow downward, but entirely fill out the frame of the base construction.

Test Assemblies. A set of test assemblies are also built using the same nondeterministic method used to create the base assemblies’ variable assignments. A test assembly is created for each assignment to variables $x_1, \ldots, x_k$ (Figure 6a). The geometry of a test assembly encodes this variable assignment in a complementary fashion to that of a “True” base assembly representing the same assignment to $x_1, \ldots, x_k$. This allows a test assembly to attach to a “True” base assembly with the same variable assignment to $x_1, \ldots, x_k$, but not to any other due to that causing overlapping geometry. The test assemblies cooperatively bind with two strength-1 affinities at two points (Figure 6b). A test assembly will only be terminal if there is no base assembly matching its variable assignment that was marked as “True”.

Transition to Uniform Assembly. If the solution to the instance of ∀∃3SAT is true, all assemblies eventually grow/transition to one unique target assembly. To achieve this, there are state transitions which allow every “True”/“False” flagged base assembly to grow into one uniform assembly. For base assemblies marked “True”, to which a test assembly attached, the states needed to cooperatively bind these test assemblies to base assemblies having a transition rule to transition to state T. For assemblies marked “False”, a transition to state T occurs when A and F (Figure 5c) are adjacent. Additional transition rules between state T and all other states (excluding the frame states) allow this state to propagate throughout the entire assembly. The transitions used are shown in Figure 7a. These transitions will change every state besides the frame states to state T. This is the target assembly for our created instance of ASTA UAV (Figure 7b).

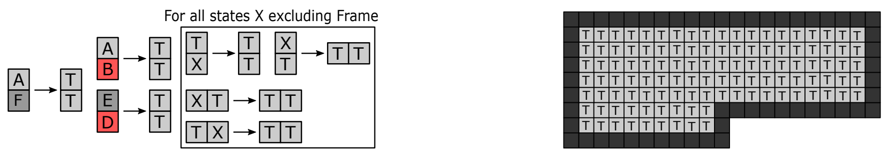
The only terminal assembly possibly produced that is not the target assembly is a test assembly representing a specific assignment to $x_1, \ldots, x_k$ that could not attach to an assignment assembly marked “True”, which represents the same variable assignment. Thus, the system only uniquely assembles the target assembly if the instance of ∀∃3SAT is true. ◀

▶ **Theorem 6.7.** *The Unique Assembly Verification problem in freezing Affinity-Strengthening Tile Automata is coNP^{NP}-Complete.*



(a) (b) (c)

Figure 6 (a) Test assemblies are nondeterministically built by allowing the possibility for each assignment of one variable construction to attach to either assignment of the next variable construction. (b) Affinities between test assemblies and base assemblies. (c) Example of a test assembly binding to a base assembly that encodes the same variable assignment of $x_1, \ldots, x_k$.



(a) (b)

Figure 7 (a) Transitions Utilized. All states will take the place of X, excluding those that are part of the frame. (b) Target Assembly after the T state has fully propagated through the assembly.

Proof. Follows from Lemmas 6.5 and 6.6. ◀

▶ **Theorem 6.8.** *The Unique Assembly Verification problem in freezing Affinity-Strengthening Tile Automata is coNP-hard in one dimension.*

Proof. We show Affinity Strengthening Freezing UAV is coNP-hard by describing how to reduce from any problem in coNP. Given a problem $L \in$ coNP take a nondeterministic Turing Machine M that decides L. From Theorem 3.5, we construct systems that simulate bounded-time Turing Machines. Since we are considering polynomial-time machines, the size of this Tile Automata system is also polynomial. We change the system to be Affinity Strengthening in the same way as in Lemma 6.2. Further, since the Tile Automata model includes nondeterminism in selecting possible transitions for an assembly, we can simulate nondeterministic Turing Machines. We simply have transition rules for each possible outcome.

Using the method described above we can simulate M on x. If any of the possible computation paths lead to M accepting, the assembly with the accept states will appear as a terminal assembly. If all possible computations path reject, the only terminal assembly will be the assembly with the reject states. ◀

7 Conclusion

In this paper we looked at a powerful new model of self-assembly that combines properties of both cellular automata and hierarchical self-assembly models. We showed that even extremely limited and simple constructions in Tile Automata are powerful and capable of arbitrary computation. We also showed how difficult it is to determine the output of these limited systems. This opens several directions for future work.

One direction is further exploring the assembly of length-n lines in freezing systems. Does there exist a bound on buildable length? Is the finite assembly problem in freezing or other restricted system decidable? Also attempting to construct lines in systems with additional restrictions such as limits on the number of transition rules per state.

For the UAV problem, we show that the general case is undecidable. However, the complexity of the problem in freezing 1-dimensional systems is open. If the problem of asking whether a system is bounded is decidable, then UAV is decidable by first identifying whether a system is bounded and then constructing the production graph and finding the terminal assemblies. The problem for freezing 2-dimensional systems with no cycles is also open.

Since Tile Automata can be seen as a generalization of 2HAM, our results can be compared to the open problem of UAV in that model which is known to be in coNP. The most restricted version of Tile Automata we explore is Affinity Strengthening and freezing, which is only one level of the polynomial hierarchy above other generalizations of 2HAM such as allowing tiles to go into 3-dimensions or allowing a variable temperature. Further limiting Tile Automata may provide more insight into the hardness of these problems.

References

1 Leonard M. Adleman, Qi Cheng, Ashish Goel, Ming-Deh A. Huang, David Kempe, Pablo Moisset de Espanés, and Paul W. K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 23–32, 2002.

2 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.

3 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.

4 Angel A. Cantu, Austin Luchsinger, Robert Schweller, and Tim Wylie. Covert Computation in Self-Assembled Circuits. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:14, 2019.

5 Cameron Chalk, Austin Luchsinger, Eric Martinez, Robert Schweller, Andrew Winslow, and Tim Wylie. Freezing simulates non-freezing tile automata. In *International Conference on DNA Computing and Molecular Programming*, pages 155–172. Springer, 2018.

6 Cameron Chalk, Austin Luchsinger, Robert Schweller, and Tim Wylie. Self-assembly of any shape with constant tile types using high temperature. In *Proc. of the 26th Annual European Symposium on Algorithms*, ESA'18, 2018.

7 Matthew Cook. Universality in elementary cellular automata. *Complex systems*, 15(1):1–40, 2004.

8 Joshua J. Daymude, Kristian Hinnenthal, Andréa W. Richa, and Christian Scheideler. Computing by programmable particles. In *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 615–681. Springer, Cham, 2019.

9 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.

10 Erik D. Demaine, Sarah Eisenstat, Mashhood Ishaque, and Andrew Winslow. One-dimensional staged self-assembly. In *Proceedings of the 17th international conference on DNA computing and molecular programming*, DNA'11, pages 100–114, 2011.

11 David Doty, Lila Kari, and Benoît Masson. Negative interactions in irreversible self-assembly. *Algorithmica*, 66(1):153–172, 2013.

12 Constantine Evans. *Crystals that Count! Physical Principles and Experimental Investigations of DNA Tile Self-Assembly.* PhD thesis, California Inst. of Tech., 2014.

13 Antonios G Kanaras, Zhenxin Wang, Andrew D Bates, Richard Cosstick, and Mathias Brust. Towards multistep nanostructure synthesis: Programmed enzymatic self-assembly of dna/gold systems. *Angewandte Chemie International Edition*, 42(2):191–194, 2003.

14 Ryuji Kawano. Synthetic ion channels and dna logic gates as components of molecular robots. *ChemPhysChem*, 19(4):359–366, 2018. `doi:10.1002/cphc.201700982`.

15 Alexandra Keenan, Robert Schweller, Michael Sherman, and Xingsi Zhong. Fast arithmetic in algorithmic self-assembly. *Natural Computing*, 15(1):115–128, March 2016.

16 Ceren Kimna and Oliver Lieleg. Engineering an orchestrated release avalanche from hydrogels using dna-nanotechnology. *Journal of Controlled Release*, April 2019. `doi:10.1016/j.jconrel.2019.04.028`.

17 Sige-Yuki Kuroda. Classes of languages and linear-bounded automata. *Information and Control*, 7(2):207–223, 1964. `doi:10.1016/S0019-9958(64)90120-2`.

18 Austin Luchsinger, Robert Schweller, and Tim Wylie. Self-assembly of shapes at constant scale using repulsive forces. *Natural Computing*, August 2018. `doi:10.1007/s11047-018-9707-9`.

19 Jennifer E. Padilla, Matthew J. Patitz, Raul Pena, Robert T. Schweller, Nadrian C. Seeman, Robert Sheline, Scott M. Summers, and Xingsi Zhong. Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes. In *Unconventional Computation and Natural Computation*, pages 174–185. Springer, 2013.

20 Robert Schweller and Michael Sherman. Fuel efficient computation in passive self-assembly. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'13, pages 1513–1525. SIAM, 2013.

21 Robert Schweller, Andrew Winslow, and Tim Wylie. Complexities for high-temperature two-handed tile self-assembly. In Robert Brijder and Lulu Qian, editors, *DNA Computing and Molecular Programming*, pages 98–109, Cham, 2017. Springer International Publishing.

22 Robert Schweller, Andrew Winslow, and Tim Wylie. Nearly constant tile complexity for any shape in two-handed tile assembly. *Algorithmica*, 81(8):3114–3135, 2019.

23 Robert Schweller, Andrew Winslow, and Tim Wylie. Verification in staged tile self-assembly. *Natural Computing*, 18(1):107–117, 2019.

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

25 Andrew Winslow. Staged self-assembly and polyomino context-free grammars. *Natural Computing*, 14(2):293–302, 2015.